Security Implications of Insecure DNS Usage in the Internet

Zur Erlangung des akademischen Grades Doktor-Ingenieur (Dr.-Ing.)
Genehmigte Dissertation von Philipp Jeitner aus Mühlacker
Tag der Einreichung: 1. April 2022, Tag der Prüfung: 13. Juni 2022

1. Gutachten: Prof. Dr. Michael Waidner
2. Gutachten: Prof. Dr. Haya Shulman
3. Gutachten: Prof. Dr. Sebastian Schinzel
Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Fraunhofer SIT

ATHENE
Nationales Forschungszentrum für angewandte Cybersicherheit

Security Implications of Insecure DNS Usage in the Internet

Accepted doctoral thesis by Philipp Jeitner

1. Review: Prof. Dr. Michael Waidner
2. Review: Prof. Dr. Haya Shulman
3. Review: Prof. Dr. Sebastian Schinzel

Date of submission: 1. April 2022
Date of thesis defense: 13. Juni 2022

Darmstadt

Bitte zitieren Sie dieses Dokument als:
URN: urn:nbn:de:tuda-tuprints-215236
URL: http://tuprints.ulb.tu-darmstadt.de/21523

Dieses Dokument wird bereitgestellt von tuprints,
E-Publishing-Service der TU Darmstadt
http://tuprints.ulb.tu-darmstadt.de
tuprints@ulb.tu-darmstadt.de

Darmstadt, Technische Universität Darmstadt
Jahr der Veröffentlichung der Dissertation auf TUprints: 2022

Abstract

The Domain Name System (DNS) provides domain-to-address lookup-services used by almost all internet applications. Because of this ubiquitous use of the DNS, attacks against the DNS have become more and more critical. However, in the past, studies of DNS security have been mostly conducted against individual protocols and applications. In this thesis, we perform the first comprehensive evaluation of DNS-based attacks against a wide range of internet applications, ranging from time-synchronisation via NTP over internet resource management to security mechanisms. We show how to attack those applications by exploiting various weaknesses in the DNS. These attacks are based on both, already known weaknesses which are adapted to new attacks, as well as previously unknown attack vectors which have been found during the course of this thesis. We evaluate our attacks and provide the first taxonomy of DNS applications, to show how adversaries can systematically develop attacks exploiting the DNS. We analyze the attack surface created by our attacks in the internet and find that a significant number of applications and systems can be attacked. We work together with the developers of the vulnerable applications to develop patches and general countermeasures which can be applied by various parties to block our attacks. We also provide conceptual insights into the root causes allowing our attacks to help with the development of new applications and standards.

The findings of this thesis are published in in 4 full-paper publications and 2 posters at international academic conferences. Additionally, we disclose our finding to developers which has lead to the registration of 8 Common Vulnerabilities and Exposures identifiers (CVE IDs) and patches in 10 software implementations. To raise awareness, we also presented our findings at several community meetings and via invited articles.

Zusammenfassung

Das Domain Name System (DNS) bietet Dienste zum Auflösen von Domänennamen, welche von fast allen Internetanwendungen genutzt werden. Aufgrund dieser allgegenwärtigen Nutzung des DNS sind Angriffe auf das DNS immer kritischer geworden. In der Vergangenheit wurden Studien zur DNS-Sicherheit jedoch meist gegen einzelne Protokolle und Anwendungen durchgeführt. In dieser Arbeit führen wir die erste umfassende Bewertung von DNS-basierten Angriffen auf eine breite Palette von Internetanwendungen durch, die von der Zeitsynchronisierung über NTP und die Verwaltung von Internetressourcen bis hin zu Sicherheitsmechanismen reichen. Wir zeigen, wie man diese Anwendungen angreifen kann, indem man verschiedene Schwachstellen im DNS ausnutzt. Diese Angriffe basieren sowohl auf bereits bekannten Schwachstellen, die für neue Angriffe angepasst werden, als auch auf bisher unbekannten Angriffsvektoren, die im Laufe dieser Arbeit gefunden wurden. Wir bewerten unsere Angriffe und stellen die erste Taxonomie von DNS-Anwendungen auf, um zu zeigen, wie Angreifer systematisch Angriffe entwickeln können, die das DNS ausnutzen. Wir analysieren die Angriffsfläche, die durch unsere Angriffe im Internet geschaffen wird, und stellen fest, dass eine erhebliche Anzahl von Anwendungen und Systemen angegriffen werden kann. Wir arbeiten mit den Entwicklern der anfälligen Anwendungen zusammen, um Patches und allgemeine Gegenmaßnahmen zu entwickeln, die von verschiedenen Parteien eingesetzt werden können, um unsere Angriffe zu blockieren. Außerdem liefern wir konzeptionelle Einblicke in die Ursachen, die unsere Angriffe ermöglichen, um bei der Entwicklung neuer Anwendungen und Standards zu helfen.

Die Ergebnisse dieser Arbeit wurden in 4 Full-Paper-Publikationen und 2 Postern auf internationalen wissenschaftlichen Konferenzen veröffentlicht. Darüber hinaus legen wir unsere Erkenntnisse gegenüber Entwicklern offen, was zur Registrierung von 8 CVE-Kennungen (Common Vulnerabilities and Exposures) und Patches in 10 Softwareimplementierungen geführt hat. Um das Bewusstsein zu schärfen, haben wir unsere Ergebnisse auch auf mehreren Community-Treffen und in eingeladenen Artikeln vorgestellt.

Own contributions

This work is a cumulative thesis based on several academic publications. All of these publications focus on attacks against internet applications using the Domain Name System in different contexts. Out of these 6 publications, 4 [1]–[3], [6] were written only by the author of this thesis using the valuable support of his advisors Haya Shulman and Michael Waidner. The other 2 publications [4], [5] were written in additional co-authorship together with Tianxiang Dai, who is another doctoral student at Technical University of Darmstadt. In accordance to the doctoral regulations of the Technical University of Darmstadt, a detailed statement over the contributions of all authors to each individual publication is given in the respective chapter describing the publication. Additionally to the publications which are part of this thesis, the author of this work has published and submitted a number of additional publications which are listed in the References section of this thesis if they are already accepted by the respective publisher [7], [8].

Non-academic contributions

Additionally to the academic publications which are part of this work, there are several other contributions which are directly linked to these works and should also be considered as results of the overall contribution of this thesis.

First, during the work on this thesis, several vulnerabilities have been identified in different open- and closed-source software components. As part of a responsible disclosure process, these vulnerabilities were reported to the respective developers of those software components, which led to a significant number of patches and advisories [9]–[22]. Some of these patches and advisories are based on the direct input of the author of this thesis, while others have been written and developed by the developers independently using only the reported vulnerability description. Many of these vulnerabilities were assigned numbers in the Common Vulnerabilities and Exposures (CVE) system [23]–[30].

Furthermore, to raise awareness onto the vulnerabilities discovered in the works which make up this thesis, several community talks have been given [31]–[33] and invited articles have been written [34] by the author of this work as well as co-authors of the

respective publications.

Finally, the methodologies and tools developed during this thesis were also used to conduct a study of the IT resources of German political parties [35]–[37] in the context of the 2021 German federal elections. This study was conducted by the author of this work together with other employees of Fraunhofer SIT. The results were used to protect the parties against potential attacks aimed against them in an attempt to manipulate the election results.

Contents

1. Introduction

The Domain Name System (DNS) is the central system in the internet to provide hosts with information about the addresses of communication peers. Initially designed in the 1980s [38], it replaced manually maintained lists of hostnames and addresses [39]. As of today, the DNS has evolved into a complex system of servers used to support a variety of different applications and security mechanisms.

As the only system which provides a universally accepted source of naming information in the internet, the DNS is required by almost all internet applications to provide a mapping between domain names like `www.example.com` to Internet Protocol (IP) addresses like `203.0.113.45`. While it is technically possible for two services to communicate directly with the use of plain IP addresses as identifiers, such an approach requires the peers to exactly know the addresses of each other. Not only does this hinder the adoption of services significantly, as IP addresses are not easily remembered by users, but it also prevents easy relocation of services to different infrastructure, as relocating IP addresses from one infrastructure to another is a very complex and costly process.

Because of this ubiquitous use of DNS in almost all internet applications, the DNS has always been an interesting target for attackers. By attacking the DNS, attackers can modify the data returned by the DNS to make applications connect to malicious servers to eavesdrop on communication or to serve malware. Furthermore, because of the sheer number of DNS applications and servers, vulnerabilities in the DNS protocol design can be exploited to conduct attacks in many other different ways, such as Distributed Denial of Service (DDoS) attacks.

In the past, studies on DNS insecurities and their impact have been focused mostly on particular applications, such as web-pages served with the Hypertext Transport Protocol (HTTP), or did not dig into the relationships between the DNS and other protocols at all. While such an approach is useful to study the general weaknesses of the DNS and its protocol, it cannot provide insights into the impact of these weaknesses.

In this work, we provide the first comprehensive overview over the security implications of attacks conducted via the DNS for internet applications. To achieve this we study many different applications from different contexts and use cases, analyze their use case of DNS and develop end-to-end attacks against these applications, all while providing clear

motives why an attacker would conduct such an attack.

Our work shows that attacks conducted via the DNS can be executed against almost all internet applications with varying levels of impact, from simply making a service unusable to giving full control over the targeted system. We show that the attacks can be executed by weak, off-path attackers which only need to have connectivity to the target systems. We show that our attacks apply to a big portion of the internet systems as of today. We also develop and suggest countermeasures to block our attacks and provide conceptual insights to prevent the vulnerabilities allowing our attacks on the design level in future protocols.

1.1. Contributions

In this section we provide a summary of the various contributions of the thesis.

Impact-analysis against various different applications

While there are many works which study the security of the DNS in itself [40], [41] and it has been shown that practical attacks on DNS are possible [42]–[44], there are only a few works which dig into the relationships between DNS and other applications to analyze the actual practical impact of DNS-based attacks against those applications [45]. Similarly, attacks in-the-wild were typically targeted against websites [46]–[50] or the actual target applications are not publicly known [51]. However, to execute a meaningful attack, attackers require to execute such an attack in the use case of an application (i.e., a website of an account provider, a time-synchronisation mechanism, or a mail provider) which it wants to compromise. Depending on this use case, some of the requirements to execute an attack (i.e., triggering DNS queries), can be very easy to accomplish or pose a significant hurdle for the attacker to succeed with the attack.

In this thesis, we provide the first systematic study of DNS-based attacks against a collection of popular applications and security mechanisms. We develop a taxonomy of an application's properties which are important for conducting DNS-based attacks. We show that DNS-based attacks can be exploited for a much wider range of attacks than just hijacking websites: Among others, we show how to shift time on systems using the Network Time protocol (NTP) [1] - even against a newly proposed, proven-secure mechanism [2], how to hijack internet resources such as Network Blocks, Certificates and Cloud Systems [4] or to circumvent security mechanisms such as Resource Public Key Infrastructure (RPKI) and Firewall filters [5]. Our goal is to show how the different use

cases in which the DNS is used by different types of applications change the way these applications can be attacked.

Developing new attack methodologies

To conduct practical attacks against applications, attackers do not only have to attack the DNS, but also need to bring the application to a state where it actually makes use of the malicious information. In our publications, we develop new attack methodologies to overcome various hurdles to conduct practical attacks.

We develop new techniques for triggering DNS queries in a victim application to break its established associations and trigger DNS lookups even in the case when the application is already started and would not normally make use of the DNS [1], [5]. We show how to gather the required information to locate the victim servers and trigger DNS lookups in web applications in case the attacker does not have any previous knowledge other than which resource he or she attempts to hijack [4].

Furthermore, we identify and develop a new attack vector against applications using the DNS which is independent of traditional cache poisoning methodologies [6] and such not affected by commonly used protection mechanisms.

Finally, we analyze these new attack methodologies in terms of their required attacker capabilities and properties in victim systems. This enables us to answer the question how practical it is to launch such attacks for different types of attackers.

Analyzing attack surface

To estimate the number of systems vulnerable to our attacks in the internet, we conduct extensive measurements of all properties required in a system for a successful attack. This includes multiple studies of a wide variety of DNS nameservers and resolver populations [1], [4]–[6], as well as a study of the clients and servers of the various applications which were analyzed for DNS-based attacks [1], [4], [5].

To conduct our measurements, we develop new methodologies for collecting data remotely and implement them in software. This allows us to conduct measurements against a wide range of internet systems without causing any harm to those systems or its users. We develop techniques for testing rate-limiting in NTP servers [1], presence of a record in the cache of DNS resolvers [1], DNSSEC-validation and fragmentation-support in DNS resolvers [1], [4], [5] and misinterpretations in DNS forwarders [6] among others.

We further conduct source-code analysis and lab-evaluation of our attacks and practically verify our attacks against several open source components, such as NTP implementations [1], RADIUS-severs, and libraries [6]. We also provide the first taxonomy of different

cache poisoning methodologies against various applications and show which methodology is the most applicable, stealthy, and effective [5].

Proposing and developing countermeasures

Along with the description of our attacks [1], [4]–[6], we also propose countermeasures and mitigations against our attacks which can be applied on different technical levels and by different actors to block and/or complicate the attacks. We also propose new systems [3] which are designed to prevent our attacks on a systematic level and which can be deployed without require deploying complex security mechanisms systems, such as DNSSEC.

Furthermore, we work together with the developers of vulnerable applications to fix application-level vulnerabilities by reporting those vulnerabilities as part of a vulnerable disclosure process [15]–[19] and initiate discussions in standardization bodies on how to mitigate our attacks on a more conceptual level via presentations at non-academic conferences [31]–[33] and invited articles [34].

Providing conceptual insights

Finally, we provide conceptual insights into the design decisions which allowed the vulnerabilities found in our work. We show how mismatched expectations have led to developers not implementing checks and how unrealistic assumptions in security designs can lead to vulnerabilities even in proven-secure systems. As such, our work shows that many of these relationships between different protocols and applications are not well understood, which can lead to the vulnerabilities presented in this work.

1.2. Organisation

This Thesis is structured as follows:

In Chapter 2, we give an overview of the Domain Name System (DNS) and review existing attacks. We then follow up with a description of the individual studies which make up this thesis:

In Chapter 3, we begin with a previously unknown application of Off-path DNS cache poisoning attacks against the Network Time Protocol (NTP). We then continue in Chapter 4 with a study on how to weaponize DNS cache poisoning attacks to gain control over critical internet resources like network blocks, certificates, and cloud systems. Afterwards, we provide the first extensive study on how to apply DNS Cache Poisoning attacks to attack

numerous additional applications in Chapter 5, including an analysis of the various practical hurdles with each application. This study provides a taxonomy which is built upon the insights from the first two studies. In Chapter 6, we show a previously unknown way to attack internet applications through DNS-based attacks without the need of poisoning a DNS cache. In this study, we provide a systematic analysis of injection-attacks against various applications and show that such attacks endanger a wide variety of different applications, since almost none of them consider this as an attack vector.

Afterwards, in Chapter 7, we provide a summary of the impact of our several studies, which lists countermeasures, security patches, and our efforts in raising awareness to our attacks. Finally, we conclude this thesis in Chapter 8.

2. Background

In this chapter, we give an overview of the Domain Name System (DNS) in Section 2.1. We then review previous attacks which exploited the weaknesses of DNS in Section 2.2.

2.1. The Domain Name System

The Domain Name System (DNS) [38], [52] is the core internet system for the organisation of names in the internet. The DNS provides the facilities to organise the naming of internet hosts and resources in a single tree under the DNS root, which is stored as a distributed database. To implement lookups of names in this DNS tree, the DNS uses two different roles of servers: (authoritative) nameservers, which store the data for a specific part of the DNS tree (called a zone), and (recursive) resolvers, which follow the delegations between nameservers to find the correct nameserver which is authoritative for a specific zone, ask it for a requested data point (called a DNS record) and provide it to the requesting client, such as a web-browser.

2.1.1. The DNS tree

To allow efficient lookups without the need for a centralized directory service, the data stored inside the DNS is organized in a tree. Each node in the DNS tree is identified by a domain name, which consists of a series of labels, where each label presents one layer in the DNS tree. Domain names are ordered backwards, meaning that the labels corresponding to the upper levels in the DNS tree come last. To allow for easy extensibility, each DNS sub-tree (zone) can be delegated to one or more different authoritative nameservers, which store the data for all domain names under this zone, if they are not delegated further to other nameservers. These design decisions shape how the ownership of names in the internet works upon today: To purchase a domain name `example.com` to allow users access to its services, an organization will typically register a second-level domain under one of the top-level domains (i.e., `com.`) under the DNS root. But instead of entering the information for the newly purchased domain directly into this parent zone, only a

delegation to a nameserver responsible for handling requests for `example.com` will be stored. Resolvers which wish to fetch information from this zone will then be redirected to the newly set-up nameserver serving `example.com`.

2.1.2. Usage in applications

Because of its ease of use and the critical service it provides, the DNS is used by almost all applications in the internet. Its most commonly known use case is the mapping of internet hostnames to addresses. For example, when a user at host A wishes to open the website at host B, he or she will typically only know the hostname of B (e.g. `example.com`), but not its internet protocol (IP) address. To find the address, A asks the DNS resolver configured on the system for the address associated with the domain name identical to the hostname of B, via a DNS lookup of type `A` for `example.com`. The DNS resolver knows the nameservers for the DNS root and will ask its way down from there to where the name of B is located in the DNS tree. Once succeeded, it will provide the address of B to A, so A can finally connect to B.

While the location of websites is a significant use case of the DNS, its authority over internet domain names reaches much wider: For example, to locate the server responsible for handling mail for a mail address `alice@example.com`, the sending system uses the DNS via a DNS lookup of type `MX` for the domain `example.com`. As another example, many firewalls rely on the DNS to allow or block communication to destinations which are identified by domain name, but whose addresses change frequently. Time-synchronisation over the Network Time Protocol (NTP) relies on the DNS to create a pool of servers to connect to and even the BGP routing system in the internet relies on the DNS for the implementation of one of its security features: Resource Public Key Infrastructure (RPKI) uses the DNS to locate the repositories storing the certificates which certify which network is allowed to advertise routes to a given network block.

2.2. Attacks in the DNS

Because of its core position in the internet as the authority over all domain names, the DNS is a common target to attacks. In the following, we provide an overview of the most important attacks which can be executed with or against the DNS [40], [53], [54].

Denial-of-service. Because of its connection-less communication model over the User Datagram Protocol (UDP) [55], DNS servers have become a constant source of Denial-of-Service (DoS) attacks [56]. These so-called reflection attacks [57] exploit the fact that

DNS servers cannot validate the source of an incoming request and thus will send back their answer to any address specified as part of the request's source address field. While this can be used to hide the source of an attack, amplification attacks additionally aim to amplify the amount of traffic which is sent to the victim by exploiting the fact that DNS responses are typically bigger than their corresponding queries [58]. This increases the amount of traffic an attacker is able to generate in a Denial-of-Service attack, whose goal it is to block a certain host or network from being reachable.

Rebinding attacks. DNS Rebinding [59] is an attack technique where the attacker changes the address of its own domain names during a web-browsing session of the victim. This allows the attacker to circumvent the Same-Origin-Policy [60] enforced by modern browsers, which would normally prevent the attackers website from accessing any third party websites, including systems inside the internal network of the victim. Since the Same-Origin-Policy is enforced upon the domain name level, changes in addresses are not registered and the attacker can therefore communicate with devices inside the victim's internal network by luring the victim onto his own website and then changing the websites domain's address to the address of the machine in the internal network.

Tunneling. DNS tunneling [61], [62] is a technique to transport arbitrary information through the DNS. As internet connections are often filtered, this can be used by attackers to exfiltrate data through an otherwise filtered connection or even to obtain free internet connectivity at public wifi hotspots [62].

Reconnaissance and Zone Enumeration. Because of its ubiquitous use, attackers can also use the DNS to map networks and systems in order to find vulnerable systems in a target network. In the past, DNS severs often allowed to extract full zonefiles by unauthorized zone transfer [63] or DNSSEC-based zone enumeration [64]. Since zone-walking vulnerabilities have been addressed with the development of better DNSSEC versions [65] and nameservers do not allow unauthenticated zone-transfers anymore, newer approaches for zone-walking have been developed [66] and with faster internet-connectivity, brute-force attempts for mapping a DNS zone have gained more attention [67]–[69].

Cache poisoning attacks. Because of its authority over all domain names in the internet, one of the most significant attacks against the DNS is an attack on the data authenticity provided to clients which use the DNS to locate other internet hosts and fetch other types of data. In DNS cache poisoning attacks, the goal of the attacker is to replace the data

inside the DNS records requested by a client. Most typically, this is used to redirect these clients to an IP address of the attacker instead of the legitimate servers of that domain. When forwarding the information to the legitimate server, this allows the attacker to gain a Man-in-the-Middle position between the client and server to eavesdrop or modify any unencrypted communication between those two.

Injection attacks. Additionally to the known attacks listed in this section, we identified a novel attack vector which allows attacking application with DNS by tunneling malicious injection payloads through DNS records. In our study [6], which is described in detail in Chapter 6, we find that most applications do not conduct validations of inputs received via the DNS, other than direct user input. This allows various types of injection attacks as we describe in Section 6.

Most of the attacks described in this list are well understood, affect one specific type of application or are not specific to DNS. For example, reflection attacks can be done similarly in other connection-less protocols like NTP [70], DNS rebinding only affects interfaces of local systems [59], and DNS Tunneling and reconnaissance are often not considered attacks, rather methods to circumvent specific security mechanisms [71].

In this work, we therefore focus on the applicability of DNS Cache poisoning to attack internet applications. Additionally to this, in Chapter 6 we describe a novel attack vector using DNS for injection attacks which can also be applied to different applications.

2.2.1. History of cache poisoning attacks

To speed up DNS lookups, DNS resolvers do not only implement the recursive lookup functionality described in Section 2.1, but also implement a cache, meaning that they store the results of those lookups for a defined period of time. This cache can be used to answer queries for records which where already fetched in the recent past. This results in a significant speed increase for website loading times and reduces the load on authoritative nameservers, as they only have to provide commonly queried records when they expire from the caches of the querying resolvers.

In DNS cache poisoning attacks, the attacker's goal is to inject a specially crafted malicious DNS record in the cache of its victim, thereby "poisoning" the resolver's cache. When a victim client asks for the address of the target domain, whose DNS record has been injected, the client will then get the injected data, typically a record mapping the target domain to the address of the attacker. Since the malicious record is cached, the attack affects all the users of the poisoned resolver.

The theoretical possibility of DNS cache poisoning attacks is known since the 1990s [72], [73]. However, practicality of such attacks has first been demonstrated by Kaminsky in 2008 [42], who showed how to conduct DNS cache poisoning by brute-forcing the challenge-response fields of a DNS response (the 16-bit transaction ID field) by triggering enough queries to randomly selected sub-domains. Following to this attack, resolvers have been reinforced with patches for increasing the entropy of the challenge-response fields by employing UDP source port randomization which renders the attack impractical as an attacker now has to guess 32 instead of 16 bits of randomness.

After that, DNS cache poisoning attacks have become more of an accessory attack to increase the stealthiness of of BGP hijacking attacks [74]–[76]. BGP hijacking attacks can be used to hijack a victim's network traffic directly, but only hijacking the DNS traffic and conducting a cache poisoning attack improves efficiency and stealthiness, as it allows for selectively hijacking only the victim's traffic and releasing the BGP hijack after a short period of time in an attempt to disguise it as a misconfiguration.

Later, other methodologies for conducting off-path DNS cache poisoning attacks have been developed [43] by copying the challenge-response values from the benign response sent by the legitimate nameserver with the help of IP fragmentation. The idea was to force the namserver to sent its response in 2 fragments, one containing the challenge-response fields and one containing the actual data. The design of IPv4's fragmentation mechanism allows such an attack to be conducted by only having to guess the 16 bit IP packet identifier (IPID), an attack technique which has been used against other protocols than DNS as well [77]–[79].

Another method for conducting off-path DNS poisoning attacks has been shown in 2020 [44] by abusing a side-channel in the mechanism which handles ICMP errors in modern operating systems (OSes). This side-channel can be abused to conduct a port scan of the target resolver during a DNS query, which allows the attacker to infer the random UDP source port number. This reduces the entropy of the DNS challenge-response fields down to 16 bit which makes an attack as shown in 2008 [42] possible again.

In our work [6], presented in Chapter 6, we show yet another methodology for conducting off-path DNS cache poisoning attacks. Other than to guess or work around network protocol mechanisms to protect against off-path packet injection, our approach works by exploiting ambiguities in the DNS standard which results in resolvers misinterpreting the domain names of otherwise normal DNS responses. We show how an attacker can exploit this misinterpretation by providing special domain names which are misinterpreted by the target resolver for the domain name of a victim.

2.2.2. Attacks against DNS in the wild

In the past, there were several reports of DNS cache poisoning attempts in the wild [46]–[51], [80]–[82]. These attacks were mostly conducted using (short-lived) BGP-hijacks or by compromising of the nameservers themselves.

2.2.3. Defence mechanisms

Similar to other standards which were developed in the beginning of the internet [83], [84], the DNS standard originally did not include any means of protection against attacks which aim to compromise the authenticity of the data transferred with it [85]. As explained in Section 2.2, this leads to a wide variety of possible attacks against almost all applications in the internet.

DNSSEC. As a response to these threats, the IETF standardised the Domain Name System Security Extensions (DNSSEC) [64]. DNSSEC aims to provide the DNS with guarantees for data authenticity and integrity (but not confidentiality), by adding cryptographic signatures to all DNS records. These signatures are part of a Public-Key-Infrastructure (PKI) with its root at the DNS root key [64].

However, DNSSEC is not widely deployed in the internet. Recent studies in 2017 found that only 1% of the internet domains are signed and that only 12% of resolvers enforce validation of the cryptographic signatures [86]. In our studies, we have shown that, while DNSSEC deployment has increased, most of the internet still is not protected. We found that less than 5% of the domains are signed [5, Table 4], and that only 28.6% of the resolvers validate DNSSEC signatures [1].

As to why deployment of such a critical security systems is so slow, Chung et al. [41] showed that the deployment of DNSSEC is often error-prone, i.e., while many domains support signed records, the DS record which delegates trust from the upper zone to the private key of a child-zone is often missing. Thus the chain of trust is broken and cannot be validated.

Furthermore, even if it is broadly adopted, DNSSEC cannot prevent all types of cache poisoning attacks: For example, stub-resolvers and forwarders typically do not validate DNSSEC due to implementation complexity and problems in upstream resolvers [87], DNSSEC-signed zones are often using weak cryptographic algorithms and short keys [41], [88], and even when DNSSEC is validated, one of our studies showed that cache poisoning can still happen by exploiting misinterpretation-issues in commonly used forwarder implementations [6].

Challenge-response authentication. Challenge-response authentication is the basic defense against off-path cache poisoning attacks. DNS clients randomly select a 16 bit DNS transaction identifier (TXID) and 16 Bit UDP source port which have to be matched by the DNS server in its answer. As has been shown, these fields alone do no suffice to protect even against off-path attacks, as the respective values can be inferred with side channels [44] or copied from legitimate answers [43]. There are some proposals to increase the amount of randomness in the DNS requests with nameserver randomisation [89], 0x20 encoding [90], or DNS cookies [91]. All of these attempts have in common that they cannot fully prevent an attack, as they require cooperation of the DNS client and server for compatibility reasons (which can by disturbed by the attacker), or they do not provide enough randomness to completely block an attack.

Cache hardening. In addition to making a target resolver accept a response injected from off-path, the attacker has to make sure that the malicious records provided in the response are actually cached. A comparative evaluation of different cache poisoning payloads is given in [92]. While there have been some attempts in blocking some of these payloads by not caching the information, such attempts can only block an attack in specific scenarios where an attacker is restricted in the payloads that he or she can send, but they do not present a complete solution against DNS cache poisoning.

Transport encryption. Recent proposals have been made to encrypt DNS packets via the use of standard encryption layers like Transport Layer Security (TLS) [93], [94]. While they are not directly targeted towards preventing off-path cache poisoning, they can prevent most off-path attacks. However, both current proposals, DNS-over-HTTPS (DoH) and DNS-over-TLS (DoT), do only protect the packets on the path from one server to another and do not provide a secure, standardized method for backwards-compatible deployment.

3. Attacking NTP with DNS

In this section we present an attack against the Network Time Protocol (NTP). Other than previous attacks against NTP, this attack is based on attacking the DNS, which is used by typical NTP clients to locate servers to synchronise time. This section is based on three publications which where published at the 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), which are listed below:

[1] P. Jeitner, H. Shulman, and M. Waidner, "The impact of dns insecurity on time," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, CORE A, 2020, pp. 266–277. DOI: 10.1109/DSN48063.2020.00043.

[2] P. Jeitner, H. Shulman, and M. Waidner, "Pitfalls of provably secure systems in internet the case of chronos-ntp," in *2020 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*, Poster, 2020, pp. 49–50. DOI: 10.1109/DSN-S50200.2020.00027.

[3] P. Jeitner, H. Shulman, and M. Waidner, "Secure consensus generation with distributed doh," in *2020 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*, Poster, 2020, pp. 41–42. DOI: 10.1109/DSN-S50200.2020.00023.

In the following, we start with a description of related work and attacks against NTP in Section 3.1. We then present the main publication [1] in Section 3.2, and continue with two additional publications [2], [3] which discuss an extension of the attack as well as a proposed counter-mechanism in Sections 3.3 and 3.4.

Contributions of individual authors

All publication in this chapter were written by the author of this thesis using the valuable support of his advisors Haya Shulman and Michael Waidner. In the following, a detailed summary of the contributions of each author is given.

The Impact of DNS Insecurity on Time

The paper *The Impact of DNS Insecurity on Time* was published as a full research paper at the *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. It constitutes a joint work of Philipp Jeitner, Haya Shulman and Michael Waidner.

As Philipp Jeitner's supervisor, Haya Shulman proposed the initial concept. Philipp Jeitner developed the attack methodology, analyzed different NTP clients, conducted all measurements on resolvers and nameservers and performed the data analysis. The paper was written by Philipp Jeitner and Haya Shulman together, where Haya Shulman helped formulating the introduction and conclusion sections while Philipp Jeitner contributed the technical content of the paper. Michael Waidner was a general advisor of this work and contributed with continuous feedback during the writing process. The paper was presented at the conference by Philipp Jeitner.

All authors agree with the use of this paper as part of Philipp Jeitner's cumulative dissertation.

Pitfalls of Provably Secure Systems in Internet: The Case of Chronos-NTP

The paper *Pitfalls of Provably Secure Systems in Internet: The Case of Chronos-NTP* was published as a fast abstract paper in the *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN) - Supplemental Volume*. It constitutes a joint work of Philipp Jeitner, Haya Shulman and Michael Waidner.

Philipp Jeitner developed the attack against Chronos NTP client based on the attack in the full DSN'20 paper *The Impact of DNS Insecurity on Time*. The paper was written by Philipp Jeitner and Haya Shulman together, where Haya Shulman helped formulating the introduction and conclusion sections while Philipp Jeitner contributed the technical content of the paper. Michael Waidner was a general advisor of this work and contributed with continuous feedback during the writing process. The paper was presented at the conference by Philipp Jeitner.

All authors agree with the use of this paper as part of Philipp Jeitner's cumulative dissertation.

Secure Consensus Generation with Distributed DoH

The paper *Secure Consensus Generation with Distributed DoH* was published as a fast abstract paper in the *2020 50th Annual IEEE/IFIP International Conference on Dependable*

Systems and Networks (DSN) - Supplemental Volume. It constitutes a joint work of Philipp Jeitner, Haya Shulman and Michael Waidner.

Philipp Jeitner developed the general concept of the majority scheme and contributed the mathematical analysis. The paper was written by Philipp Jeitner and Haya Shulman together, where Haya Shulman helped formulating the introduction and conclusion sections while Philipp Jeitner contributed the technical content of the paper. Michael Waidner was a general advisor of this work and contributed with continuous feedback during the writing process. The paper was presented at the conference by Philipp Jeitner.

All authors agree with the use of this paper as part of Philipp Jeitner's cumulative dissertation.

3.1. The Network Time Protocol

The Network Time Protocol (NTP) [95] is an internet protocol to synchronize the time between different systems. In its most commonly used operation mode, NTP uses a typical client/server design, where the client wants to synchronize time with one (in case of SNTP) or multiple different servers (to enhance precision). In this operation mode, the client (A) sends a simple request to the server (B), asking it for the current time. The server responds with two timestamps, the time when the request is received ($T2$) and when the response is sent ($T3$). The client records the time when the request is sent to the server ($T1$), as well as when the answer is received ($T4$). These four timestamps are then used to calculate the offset θ between the time of the client A ($T(A)$) and the server B ($T(B)$) [95]:

$$\theta = T(B) - T(A) = \frac{(T2 - T1) + (T3 - T4)}{2}$$

In its simplest operation mode, also called Simple Network Time Protocol (SNTP) [96], this value is then used by the client to adjust its clock by θ. To enhance precision and counter very simple attacks, a full NTP client will calculate multiple values for θ across multiple transactions and servers, and will filter out outliers. Most commonly, these servers are selected from the NTP pool [97], which consists of a number of 4.000 servers whose addresses can be queried randomly with a DNS query to `pool.ntp.org`.

NTP Security. In its typical client/server pool operation mode, NTP does not support authentication or encryption of packets. While some security modes for NTP exist, these modes either require static configuration of keys [98] or were proven insecure [99]. Newer recommendations to secure NTP, such as Network Time Security (NTS) [100]

and Roughtime [101] are currently in development, but none of these mechanisms has been deployed by larger NTP operators [97]. As such, attacking NTP clients can be easily achieved by on-path attackers simply by changing the timestamps of the server ($T2$ and $T3$) inside the transmitted packets.

Off-path attacks against NTP. In 2016, Malhotra et al. [79] showed that off-path attacks against NTP are theoretically possible by exploiting IP fragmentation, just as it has been shown for DNS [43]. However, due to the low packet sizes of NTP, attackers need to make servers fragment their packets to very low Minimum Transmission Unit (MTU) sizes, which is typically not practical as shown by our studies [1]. Furthermore, a meaningful attack against NTP requires a specific IP fragment reassembly strategy which is not typically used in the wild [79]. In our study [1], we show how to attack NTP by employing the same basic attack idea (copying of challenge-response parameters from legitimate IP fragments). However, instead of attacking the NTP transaction directly, we attack the DNS lookup issued by the NTP clients to find servers. Since this lookup is only done once, and DNS packets can get much bigger in size, this attack is more practical and applies to much more servers.

Configuration interface attacks. As shown in [102], NTP servers can often also be attacked by queries against their configuration interface, which is often left open unsecured. Recent patches in *ntpd* (the de-facto standard NTP implementation) were developed to close the vulnerability, but a study conducted as part of our work [1] showed that about 5% of the servers in the NTP pool are still vulnerable to this attack.

Denial-of-service. Additionally to attacks against the NTP clients itself, NTP servers were commonly exploited as reflectors in Distributed Denial of Service (DDoS) attacks [58], [103]. Similar to the DNS case described in Section 2.2, such attacks exploit the fact that NTP is a connection-less protocol, so servers will answer requests to packets with spoofed source address fields.

3.2. Attacking NTP with DNS

In our study [1], we show how an off-path attacker can attack NTP clients. Instead of attacking the NTP transaction directly, our attack relies on attacking the DNS to redirect the NTP client to different NTP servers controlled by the attacker. In this study, we show how the attacker can trigger such DNS lookups even when the NTP client has already

been started. We analyze the impact of our attacks against different kinds of NTP client implementations, measure the vulnerable population of systems in the internet, and provide a theoretical analysis of how probable it is that a given NTP client using the NTP pool is vulnerable at any given time.

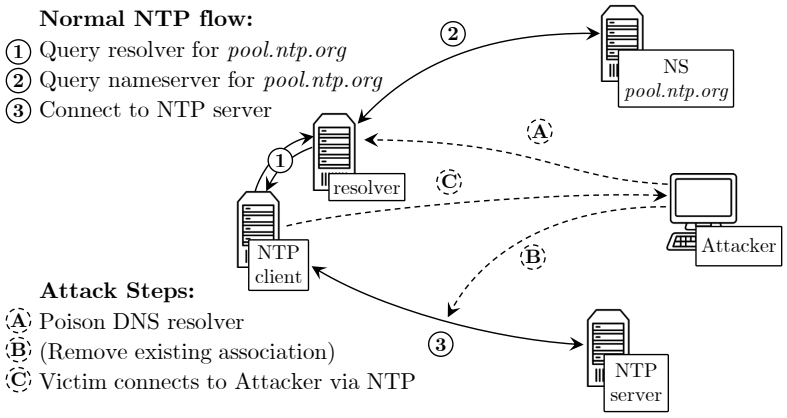


Figure 3.1.: Attacking NTP with DNS [1, Figure 1]

In the past, studies have already shown how to exploit NTPs rate-limiting mechanism to slow down requests to a specific server with Kiss of Death (KoD) packets [104]. In our study, we show that this mechanism employed by servers to reduce the effects of Denial-of-Service attacks can also be used to make clients switch to other NTP servers. We show how our attack works in Figure 3.1: In step (A), the attacker poisons the NTP client's DNS resolver using the fragmentation-method discovered by [43] as described in Section 2.2.1. Then the attacker removes existing associations to the servers used in Step (B), which causes the client to switch to the attacker-provided servers in step (C).

Triggering DNS queries. To make the client break its existing associations to NTP servers, the attacker floods the server with spoofed requests with the source address of the victim client, which causes the server to stop responding to the client completely. After a while, this causes the client to flag the server as non-responsive and switch to the next server. We show that this attack can be repeated until all of the used servers are flagged. At this point, the client will issue a new DNS lookup to the DNS server pool domain `pool.ntp.org` to find new servers.

Finding the correct server. We show that the attacker can find the server(s) which are used by the client by different means: First, when the client also operates as a server to provide time to other clients, the attacker can simply probe the client by sending an NTP request. The IP address of the primary upstream server used will be included in the response. Second, we show that the size of the NTP pool and the rate-limits enforced by these servers are small enough to simply execute the attack against all servers in the NTP pool.

Attacking DNS for time-shifting attacks. To poison the DNS server used by the client, the attacker needs to poison the resolver used by the NTP client. To measure the amount of resolvers vulnerable to such an attack in the internet, we conduct a study of open resolvers and resolvers of advertisement network (ad-net) clients in the internet for which we use a special method of testing if a record is cached in a given resolver. We find that 91% of the resolvers accept fragmented DNS responses and as such are vulnerable to the attack. Second, we find that for 13.8% of the cases, the attacker can initiate DNS cache poisoning using a third party application which significantly reduces the difficulty of the attack.

Impact against applications. We also evaluate our attack against different NTP implementations, *ntpd*, *openntpd* and *chrony*. We find that once the attack starts, it will take from 17 to 84 minutes once the victim client will start shifting its time towards the attacker provided value. We also calculate the probability of a typical client being in a vulnerable state: To succeed with the attack, more than $\frac{1}{2}$ of the servers used by the client must use rate-limiting. We show that for a typical scenario of a client using the NTP pool, this probability is 15.3% at any given time.

3.3. Adapting the attack against Chronos

In our poster [2], we show how to extend our attack to a proven-secure, improved version of NTP called Chronos [105], [106] which was designed after the attack discovered in [79]. It is designed to defend against both, off-path and on-path attacks, by improving the standard NTP server selection algorithm together with a significantly increased number of servers. The developers of Chronos show that Chronos can defend against attacks assuming a honest majority in the servers of the NTP pool.

In our work, we show that this assumption does not hold in practice, as Chronos uses the typical DNS-method when generating its local server list. We show that, since this depends only on a single nameserver which provides the references to all subsequent servers,

compromising the connection to this single server is sufficient to gain the upper hand over the rest of the NTP servers in the list. We also show that this attack is actually easier than an attack against a normal NTP client. Because the Chronos server list generation method does 24 queries over a day and joins the results of all responses together, an attacker has to succeed with his attack only once out of 12 times in the first 12 hours of the Chronos pool generation time-period.

3.4. Securing pool generation with majority voting

To provide a countermeasure against our attack on Chronos in [2], we design a modified pool generation method for Chronos in [3]. Our proposed mechanism works by adapting the idea of Chronos of guaranteeing security by utilizing a bigger server pool onto the DNS layer. Our mechanism works by using a number of n distinct DNS resolvers and distributing the lookups to generate the server pool over all these servers. To gain control over a majority of servers in the pool, as it is required for an attack against Chronos, the attacker would thereby need to attack the same majority of DNS resolvers out of these n. This is because we modify the pool generation algorithm such that every DNS resolver will always provide the exact same amount of NTP servers to the pool. To prevent against attacks on the link to the resolvers, our approach relies on resolvers providing service over DNS-over-HTTPS (DoH) [93], which as of today is offered by a wide range of public resolver operators.

4. Taking Over Internet Resources

In this Section, we present a study of DNS-cache poisoning attacks against the infrastructure of internet-resource providers, to hijack the accounts at those providers and exploit the resources. This Section is based on our paper [4] which was presented at the 30th USENIX Security Symposium 2021:

[4] T. Dai, P. Jeitner, H. Shulman, and M. Waidner, "The hijackers guide to the galaxy: Off-path taking over internet resources," in *30th USENIX Security Symposium (USENIX Security 21)*, CORE A*, USENIX Association, Aug. 2021. [Online]. Available: `https://www.usenix.org/conference/usenixsecurity21/presentation/dai`.

Contributions of individual authors

The paper *The Hijackers Guide To The Galaxy: Off-Path Taking Over Internet Resources* was published as a full research paper at the *30th USENIX Security Symposium (USENIX Security 21)*. It constitutes a joint work of Tianxiang Dai, Philipp Jeitner, Haya Shulman, and Michael Waidner.

Haya Shulman proposed the initial concept and structured the paper. Haya Shulman also wrote the Introduction, Background and Conclusions, while Philipp Jeitner wrote the rest. After the initial round of reviews, Haya Shulman and Philipp Jeitner decided on what to change about the paper and to add new attack methodologies and other types of Internet resource providers. Tianxiang Dai and Philipp Jeitner designed, implemented and analysed the measurements together. More specifically, on the provider and resolver side, Tianxiang Dai developed the testing infrastructure and performed the SADDNS measurement for all. He did the resolver tests on RIRs and CAs, while Philipp Jeitner did the resolver tests on Registrars and IaaS providers. On the customer and nameserver side, Tianxiang Dai performed all the measurements. The data analysis was done by Tianxiang Dai and Philipp Jeitner together, where Philipp Jeitner did most of the analysis. Philipp Jeitner contributed the list of countermeasures. Tianxiang Dai also implemented a proof-of-concept attack and demonstrated it on one test account of a chosen RIR. Michael

Waidner was a general advisor of this work and contributed with continuous feedback during all phases of the paper writing process. The paper was presented at the conference by Philipp Jeitner. The work was later presented at the community events NANOG 83 and RIPE 83 by Tianxiang Dai, and also as a guest post at APNIC Blog.

All authors agree with the use of their joint paper as part of Philipp Jeitner's and Tianxiang Dai's cumulative dissertation, considering a contribution of 60% from Philipp Jeitner and 40% from Tianxiang Dai.

4.1. Internet resource management

Like other digital assets, internet resources like IP addresses and domains are managed at the web-services of providers which allocate those resources. In this work, we analyze the management infrastructure of internet resources like IP addresses held at Regional Internet Registries (RIRs) [107], domains held at domain registrars, virtual machines held at infrastructure as a service (IaaS) providers, and Certificates held at Certification authorities (CAs). These providers allow the resources to be allocated, changed and sold to others – depending on the type of resource. The management of these resources is not different to the management of any other asset in any other account: The user logs in at the provider's website with his username and password, and can then execute actions which perform changes on those resources, i.e., allocate new virtual machines or transfer a domain to a new owner.

4.2. Taking Over Internet Resources

In our work, we show how to hijack those internet resources by hijacking the accounts under which these resources are managed. To accomplish this, we attack the password recovery functionality present for almost all internet accounts today: When the user forgets his password, he can set a new password by requesting a special email from the provider. This email contains a token which allows the user to set a new password. Since the email is sent to the address stored in the providers database, only the legitimate user receives the token and can reset the password.

However, since communication between email servers is essentially unauthenticated, as attackers can simply downgrade connections to not use TLS [108], these password recovery emails can be hijacked by any adversary on the path between the email servers they traverse. In our work, we show that such an attack is practical even for off-path attackers, as an attacker can hijack the communication between the provider and customer's mail

server by poisoning the DNS resolver of the provider. An overview of this attack is shown in Figure 4.1: In step (1), the attacker poisons the provider's DNS resolver and replaces the address of the mail server responsible for the customer's email address (either `MX` or `A` record) with his own. In step (2), the attacker triggers the password recovery for the customer's account at the provider. Since the resolver is poisoned, this results in the password recovery email being sent to the attacker. In step (3), the attacker resets the password of the customer's account, logs in and exploits the resources.

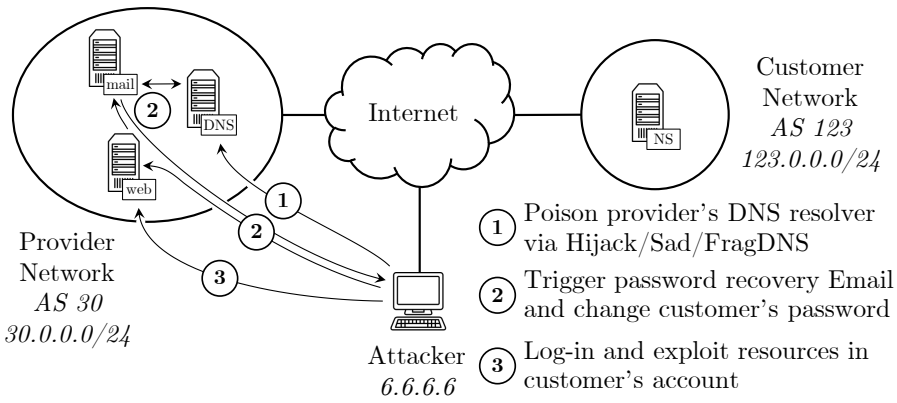


Figure 4.1.: Taking Over Internet Resources [4, Figure 1]

Performing cache poisoning against the provider's resolver. To show how the DNS resolver of the provider can be poisoned, we evaluate three different off-path cache poisoning methodologies against the infrastructures of 32 different providers: (1) IP fragmentation injection as discovered by [43], (2) Off-path side-channel interference of UDP source port [44], and (3) BGP hijacking of the connection between the resolver and nameserver [76]. Our results show that all providers are vulnerable against at least one of those methodologies and between 17% to 56% of the customers accounts are vulnerable, based on the properties of the nameservers responsible for these account's domains. We also show how the attacker can identify the target mail servers and resolvers and measure the time between a password-recovery request and a DNS lookup at the nameservers to show that this time span is short enough for the attacker to predict the timing of the DNS query.

Triggering password recovery. To trigger the password recovery, the attacker needs to enter information about the customer's account at the provider's website. We study

the password recovery at 32 providers, and find that in the case of RIRs and domain registrars, this information is often publicly accessible at the provider's WHOIS databases. Furthermore, we find that even if the information is hidden, many of those accounts use well-known usernames and email addresses which are easily predictable by the attacker.

Exploiting the account. We also show what possibilities an attacker has once he has received the password recovery email and has access to the account. We provide a classification of different types of actions, from compromising the victims virtual infrastructure over issuing malicious certificates to selling the IP address space of the victim for monetary gain. We show that the attacker can often execute an attack completely unnoticed, since countermeasures designed to communicate changes in the account can be disabled once the attacker gained access and raising alarms with manual checks is unlikely due to the high number of transactions and limited personnel these providers have. We also show that the default security measures at most providers are insufficient: While most providers offer some form of 2-factor-authentication, this is not mandatory and therefore not used by many customers.

5. A taxonomy of attacks against different applications

In this section, we present our study on the applicability of DNS cache poisoning attacks against different applications. In this study, we use our methodologies of implementing attacks against applications from our previous studies [1], [4] (see Sections 3 and 4) and apply them to a broader set of applications, which allows us to define a comprehensive taxonomy of the important aspects of applications when analysing DNS cache poisoning attacks. This study [5] was presented at the 2021 ACM SIGCOMM Conference (SIGCOMM '21):

[5] T. Dai, P. Jeitner, H. Shulman, and M. Waidner, "From ip to transport and beyond: Cross-layer attacks against applications," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, ser. SIGCOMM '21, CORE A*, Virtual Event, USA: Association for Computing Machinery, 2021, pp. 836–849, ISBN: 9781450383837. DOI: 10.1145/3452296.3472933. [Online]. Available: https://doi.org/10.1145/3452296.3472933.

Contributions of individual authors

The paper *From IP to Transport and beyond: Cross-Layer Attacks against Applications* was published as a full research paper at the *ACM SIGCOMM 2021 Conference*. It constitutes a joint work of Tianxiang Dai, Philipp Jeitner, Haya Shulman and Michael Waidner.

Haya Shulman proposed the initial concept and wrote Introduction, Background and Conclusions, while Philipp Jeitner wrote the rest. After the initial reviews, Haya Shulman and Philipp Jeitner decided on what to change about the paper and to add new attack methodologies. Tianxiang Dai and Philipp Jeitner performed the analysis of applications, more specifically Tianxiang Dai did this for XMPP, Email and PKI and Philipp Jeitner for RADIUS, NTP, Bitcoin, Tunneling and Intermediate Devices. Tianxiang Dai and Philipp Jeitner designed, implemented and analysed the measurements together. More specifically, on the resolver side, Tianxiang Dai developed the testing infrastructure and performed

the SADDNS measurement for all. Tianxiang Dai did the resolver tests for CAs, VoIP, Email, and Open Resolvers, while Philipp Jeitner did the resolver tests on Eduroam, CDNs, AdNet and NTP. On the nameserver side, Tianxiang Dai performed all the measurements. The data analysis was done by Tianxiang Dai and Philipp Jeitner together, where Philipp Jeitner did most of the analysis. Tianxiang Dai also built the online tool which allows everyone in the Internet to test their own resolvers. Michael Waidner was a general advisor of this work and contributed with continuous feedback during all phases of the paper writing process. The paper was presented at the conference by Philipp Jeitner.

All authors agree with the use of their joint paper as part of Philipp Jeitner's and Tianxiang Dai's cumulative dissertation, considering a contribution of 70% from Philipp Jeitner and 30% from Tianxiang Dai.

5.1. Cross-layer attacks

In the past, attacks against networks have become more sophisticated, spanning multiple layers of networking protocols. In such a cross-layer attack [109], an attacker performs an attack against one or multiple layers to perform an attack against another layer. Because such attacks have become more common recently [49], there is a rising effort to analyze and understand such sophisticated attacks, i.e. [76], [110], [111]. In 2020, Sun et al. [112] analyzed the applicability of BGP prefix hijacks against multiple applications: Tor (the onion router), Domain Validation, and Bitcoin. In our work, we present a similar analysis for DNS cache poisoning attacks. We provide a taxonomy of multiple methodologies to perform cache poisoning and analyze its applicability to a variety of different applications.

5.1.1. Taxonomy of vulnerable applications

We study the potential results of DNS cache poisoning attacks against different applications in a systematic study. For our study, we first classify DNS-using applications into 3 different usage types: location, federation, and authorization. We then gather a list of various applications for each usage type that use different protocols in different use cases, from RADIUS [113] servers using the DNS to discover remote authentication servers [114] to firewalls, which allow to set-up filtering rules based upon DNS names. All those applications are listed in Table 5.1.

Using this dataset of applications, we analyze them according to different properties relevant for DNS cache poisoning which we previously identified using the knowledge from our previous studies [1], [4] (see Sections 3 and 4). We identify the following

Category	Protocol	Use Case	query name	known	query trigger method	Record Type	DNS used for loc.	fed.	auth.	Methodologies Hijack	SadDNS	Frag	Cache Poisoning impact
Authentication	Radius	Peer discovery	target	✓[1]	direct	NAPTR, SRV, A	✓	✓		✓	✓	✓	DoS: no network access
Online Chat	XMPP	Chat+VoIP	target	✓[1]	bounce	A, SRV	✓	✓		✓	✓	✓	Hijack: eavesdropping
Email	SMTP	Mail	target	✓[1]	direct/bounce	A, MX	✓	✓		✓	✓	✓	Hijack: eavesdropping
	SPF,DMARC	Anti-Spam	target	✓[1]	authentication	TXT			✓	✓	✓	✓	Downgrade: spoofing
	DKIM	Integrity Checking	target	✓[1]	direct/bounce	TXT			✓	✓	✓	✓	Downgrade: spoofing
Web	HTTP	Web sites	target	✓[1]	direct	A	✓			✓	✓	✓	Hijack: eavesdropping
	SMTP	Password recovery	target	✓[1]	direct	A, MX, TXT	✓			✓	✓	✓	Hijack: account hijack
Sync	NTP	Time synchronisation	known	✓	connection DoS	A	✓			✓	✗	✓[2]	Hijack: change time
Crypto-currency	Bitcoin	Peer discovery	known	✓	waiting	A	✓			✓	✗	✗	Hijack: fake blockchain
Tunnelling	OpenVPN	VPN	config	✗	connection DoS	A	✓			✓	✓[2]	✓[2]	DoS: no VPN aceess
	IKE	VPN	config	✗	connection DoS	A	✓			✓	✓[2]	✓[2]	DoS: no VPN aceess
	IKE	Opportunistic Enc.	target	✓[1]	bounce	IPSECKEY	✓		✓	✓	✓[2]	✓[2]	Hijack: eavesdropping
PKI	DV	Domain Validation	target	✓[1]	authentication	A, MX, TXT	✓		✓	✓	✗	✗	Hijack: fraud. certificate
	OCSP	Revocation checking	target	✓[1]	direct	A			✓	✓	✓	✓	Downgrade: no check
	RPKI	Repository sync.	known	✓	waiting	A	✓			✓	✗	✗	Downgrade: no ROV
Intermediate devices	–	Firewall filters	config	✗	waiting	A	✓			✓	✓[2]	✓[2]	Downgrade: no filters
	HTTP/...	Loadbalancers	config	✗	on-demand	A	✓			✓	✓[2]	✓[2]	Hijack: eavesdropping
	HTTP	CDN's	config	✗	on-demand	A	✓			✓	✗	✓[2]	Hijack: eavesdropping
	DNS	ANAME/ALIAS[?]	config	✗	on-demand	A	✓			✓	✓[2]	✓[2]	Hijack: eavesdropping
	HTTP/Socks	Proxies	target	✓[1]	direct	A	✓			✓	✓	✓	Hijack: eavesdropping

[1]: Depends on the attack scenario. [2]: Requires a third-party application to trigger queries.

Table 5.1.: Applications analysed for DNS Cache poisoning [5, Table 1]

properties of applications which are significant for an attack: (1) Trigger-ability of DNS queries from the position of a remote attacker, (2) DNS query types triggered by the application, and (3) additional security measures like mandatory use of Transport Layer Security (TLS) [115] when using the response from the DNS. Using these properties, we define different triggering methods which can be used by an attacker depending on the application. These triggering methods influence the chance of success of a DNS cache poisoning attempt against an application. We show that depending on the cache poisoning methodology used (see Section 5.2), some triggering methods are infeasible because they do not allow to trigger a big enough amount of queries. However, we also show that while cache poisoning cannot be executed directly with the application under attack, in practice, an attacker can often find a third-party application or system which shares the same resolver as the application under attack, and such can be used to trigger queries to execute the attack.

Finally, we classify the impact of a successfully poisoned cache against our dataset of applications and show that all applications are vulnerable to cache poisoning attacks: We found different impacts, ranging from Denial-of-Service (DoS) attacks in case connections are secured with TLS over hijacking attacks to security downgrade attacks. We also show that authentication and encryption cannot always prevent an attack: For example, in both, RPKI [116] and OCSP [117], cache poisoning can be used to prevent connectivity to repositories storing information to validate the authenticity of information (BGP announcements and X.509 certificates) which lead the affected application to revert to no

validation, effectively causing a security downgrade.

	BGP Hijack		SadDNS	Fragmentation	
	sub-	same-		any IPID	global IPID
Applicability					
Vuln. resolvers	70%	80%	11%	91%	
Vuln. domains	or 53%	or 70%	and 12%	and 4%	and 1%
Effectiveness					
Hitrate	100%		0.2%	0.1%	20%
Queries needed	1		497	1024	5
Total traffic (pkts)	2		987K	65K	325
Stealthiness					
Visibility	very visible	visible	stealthy, but locally detectable (Packet flood)		very stealthy
Additional requirements					
Additional requirements	none		none	max(resolver EDNS size) < min(nameserver MTU)	

Table 5.2.: Comparison of cache poisoning methodologies [5, Table 6]

5.2. Taxonomy of cache poisoning methodologies

In our study, we analyze three different methodologies for executing cache poisoning attacks from the position of an off-path attacker: (1) IP fragmentation injection as discovered by [43], (2) Off-path side-channel interference of UDP source port [44], and (3) BGP hijacking of the connection between the resolver and nameserver [76]. Our goal is to answer the question which methodology is the most effective for an attacker when attacking different applications. Moreover we want to identify and compare secondary metrics, like stealthiness and efficiency. To achieve these goals, we measure the requirements of all three cache poisoning methodologies across datasets of potential victim resolvers and nameservers of the applications in our dataset (see Section 5.1.1). Furthermore, we perform simulations of the expected success rates and the amount of attack traffic needed for different query-triggering methodologies identified during our application analysis (See Section 5.1.1).

Our results show that there exists a trade-off between stealthiness and applicability in the analyzed methods which we compare in Table 5.2: While cache poisoning via BGP hijacking is highly applicable (assuming the attacker has the capabilities to do so), it is also very visible to all networks to which the malicious BGP advertisements are forwarded. On the other hand, cache poisoning using the IP fragmentation method [43] is not applicable

against as many victims, but can be executed very stealthily without the danger of raising any alarms.

6. A new attack vector: Injection attacks over DNS

In this Chapter, we present our last study which analyzes the impact of a newly found attack vector to perform attacks against applications. In this study, we show how to attack applications through malicious payloads carried through normal DNS responses to exploit injection vulnerabilities in applications when receiving input from the DNS. This study [6] was presented at the 30th USENIX Security Symposium 2021:

[6] P. Jeitner and H. Shulman, "Injection attacks reloaded: Tunnelling malicious payloads over DNS," in *30th USENIX Security Symposium (USENIX Security 21)*, CORE A*, USENIX Association, Aug. 2021. [Online]. Available: `https://www.usenix.org/conference/usenixsecurity21/presentation/jeitner`.

Contributions of individual authors

The paper *Injection Attacks Reloaded: Tunnelling Malicious Payloads over DNS* was published as a full research paper at the *30th USENIX Security Symposium (USENIX Security 21)*. It constitutes a joint work of Philipp Jeitner and Haya Shulman.

Philipp Jeitner found the first vulnerabilities which lead to the idea for the paper. The paper concept was developed by Philipp Jeitner together with Haya Shulman as his supervisor. Philipp Jeitner developed all the attacks, performed the analysis of all software components and conducted the measurements of remote systems in the paper. The paper was written by Philipp Jeitner and Haya Shulman together, where Haya Shulman helped formulating the introduction and conclusion while Philipp Jeitner contributed the technical content of the paper. The paper was presented at the conference by Philipp Jeitner, who also presented it as part of the community event NANOG 83 and authored a guest article about it at APNIC blog. Philipp Jeitner also managed the communication with the developers of the applications and DNS implementations which were found vulnerable in this work as part of several responsible disclosure processes.

All authors agree with the use of this paper as part of Philipp Jeitner's cumulative dissertation.

6.1. Injection attacks

Injection vulnerabilities [118] are one of the primary attacks vectors abused for remote exploits. In an injection attack, an attacker exploits a vulnerability in a remote system which allows him to inject malicious data into another context than it was intended by the developer, for example an SQL command [119], Javascript code [120], LDAP queries [121], or network protocol headers [122]–[124]. The attack is based on the fact that the application creating this context is neither validating the attacker's input properly (if applicable), nor does it apply proper countermeasures (i.e., escaping) which leads to the attacker's input being interpreted differently than it was intended by the developer of the application. This allows the attacker to change the meaning of the resulting data into something else, allowing the attacker to execute actions in the name of the application which he would normally not be able to do. In this sense, this classification includes buffer overflow attacks since they also depend on misinterpretation of resulting data (program stack or heap) caused by missing checks by the application (buffer length), even though they are often considered to be a separate category of vulnerability [125].

Preventing injection attacks. Injection attacks are most effectively prevented by proper input validation and escaping. However, this requires application developers to be aware of the risks of injection attacks and to implement checks in every application.

To tackle this issue, several proposals have been made to mitigate injections attacks on a more conceptual level: Frameworks for web applications which analyze inputs and outputs of the application [119], [126], [127] and detect potential attacks based on this analysis. Such approaches rely on the fact that the communication done by a typical web application is relatively simple: Web applications often only receive input from one source (the user) and only send commands or data to another source (typically a database). Furthermore, web applications are typically stateless, so security frameworks can assume that any outputs of the application must be a result of the input request triggering the execution of the application. While these assumptions allow for an efficient design of such security frameworks, they are not very well suited for cases where applications receive inputs from multiple sources, and where the outputs cannot be easily attributed to a single input, such as in the case of DNS requests.

6.2. DNS-based injection attacks

In our paper [6], we show for the first time how the DNS can be exploited for injection attacks. We provide a comprehensive analysis of the components involved in a DNS lookup

chain and show that none of these components applies validation of data provided by DNS nameservers to applications. We analyze this newly identified attack vector against different types of applications with different methodologies: fuzzing, source code review, and dynamic (black box) execution. We show how attackers can systematically create attack payloads targeting applications and how to execute attacks by triggering the queries from the application under attack. Our attacks exploit the fact that DNS resolvers do not filter out any suspicious data in DNS records and that applications do not implement checks because of developers being unaware of this new attack vector.

Analysis of the DNS lookup chain. In a typical DNS lookup use case, shown in Figure 6.1, applications trigger queries to DNS nameservers to fetch various types of data from the DNS. The nameservers provide this data in the form of DNS records which are processed by DNS resolvers, stub resolvers, and finally the application itself. We perform an analysis of the DNS and associated standards [39], [52], [128], [129] and analyze the expected vs. actual behaviour of the three main components involved in a typical DNS lookup chain, excluding the nameserver which is assumed to be controlled by the attacker:

- DNS resolvers are expected to handle DNS records transparently to allow adding new types of DNS records types for new DNS use cases easily. In our tests against real-world resolvers, we find that most resolvers satisfy this requirement, but some resolvers fail to process certain special cases because of a naive decoding logic which leads to a misinterpretation of the names inside the DNS packet. We show how this misinterpretation can be exploited for easily executable cache poisoning attacks by crafting malicious names in Section 6.3.1.
- Stub resolvers are expected to validate that domain names included in DNS packets are also valid *hostnames* per RFC952 before passing them to applications. However, in our study of common stub resolver implementations, only one out of ten implementations did implement these checks and furthermore, seven out of ten included the same misinterpretation errors which we also found in DNS resolvers in the wild. This lack of validation means that applications are exposed to input injection attacks when using these stub resolver implementations.
- Applications do not perform input validation of inputs received from the DNS, even though this data is untrusted and should be treated exactly like any other input received directly from a remote party. We perform a study of different applications from different DNS use cases and find that none of these applications perform any input validation on data received from the DNS – in contrast to data received from remote peers directly, which is validated by those applications. Furthermore, we find eight applications where this missing input validation leads to exploitable vulnerabilities.

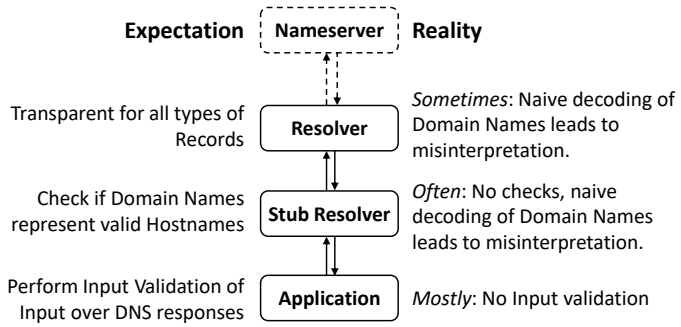


Figure 6.1.: Expected vs. actual behaviour in the DNS lookup chain [6, Figure 2]

6.3. Attacks against applications

We perform a study on 16 different applications (shown in Table 6.1) from four different DNS use cases: Address lookups, Discovery, reverse lookups, and authentication. Our analysis shows that none of the analysed applications validate input received from the DNS. To analyze if this missing validation leads to exploitable vulnerabilities, we further analyze the use case for the input received from the DNS (i.e., connecting to a server, placing it in a cache, or displaying information to the user). Our analysis shows that whether missing DNS input validation leads to a vulnerability depends mostly of the context in which the DNS lookup is made, for what purpose it is used, and in which types of data structures it is stored. For example, we find exploitable vulnerabilities in administrative interfaces of routers (OpenWRT) where the unfiltered DNS input is included in an HTML page, which leads to a cross-site-scripting vulnerability and allows the attacker to gain full control over the vulnerable router. We also discover a configuration injection vulnerability in a script included with radsecproxy, a RADIUS [113] proxy which is typically used in Eduroam to route authentication requests to the authentication servers of remote organisations, when their users are roaming in another organisations network. In conclusion, our analysis shows that DNS-based input validation vulnerabilities exist in a variety of different applications, ranging from simple administrative tools like ping over complex administrative interfaces like in OpenWRT to security mechanism implementations like libspf2. The fact that such vulnerabilities are spread out in applications which otherwise typically implement input validation of data received directly from remote systems shows that many developers are not aware of the risks of dealing with DNS inputs.

DNS Use-Case	Application	Trigger	Set	Uses libc	Validates	Input use	Attack found
		Query					
Address lookups (A, CNAME)	Chrome	js,html		yes	no	cache	no
	Firefox	js,html		yes	no	cache	no
	Opera	js,html		yes	no	cache	no
	Edge	js,html		yes	no	cache	no
	unscd	client app		yes	no	cache	no
	java	client app		both	no	cache	no
	ping(win32)	✗	✗	yes	no	display	yes
discovery (MX, SRV, NAPTR)	openjdk	login	✗	no	no	create URL	yes
	ldapsearch	login	✗	no	no	create URL	no
	radsecproxy	login		no	no	configure	yes
Reverse lookups (PTR)	ping(linux)	✗	✗	yes	no	display	yes
	trace(linux)	✗	✗	yes	no	display	yes
	OpenWRT	✗	ping	yes	no	display	yes
	openssh	login		yes	no	display,log	yes
Authentication (TXT, TLSA)	policyd-spf	SMTP		no	no	text protocol	no
	libspf2	SMTP		no	-	parse	yes
All	Resolvers	client app		no	some	cache	yes

Table 6.1.: Analysed software and tools [6, Table 1]

6.3.1. Performing DNS cache poisoning with misinterpretation

Additionally to our attacks against applications, we also find new attacks against DNS resolvers, which allow for easily conductible cache poisoning attacks. Other than traditional cache poisoning attacks which aim to fool the DNS resolver into accepting a spoofed DNS response sent by the attacker in the name of a legitimate nameserver (see Section 2.2.1), our attack is based on misinterpretation of the data received from a normal DNS response.

As we discuss in Section 6.2, some DNS resolvers misinterpret special characters, namely zero-bytes (`"\000"`) and period signs (`"\."`) included in domain name labels due to naive decoding of DNS record data. We show how such a misinterpretation can happen in Figure 6.2: In Step 1&2, the DNS resolver decompresses and decodes a DNS domain name from the wire-format into a text string. This process involves copying the individual labels which are stored in wire-format to the result string, and joining them with period signs (`"."`). Since the DNS standard does not disallow any special characters in domain names, this can lead to a misinterpretation when a label contains period signs itself. When the record is decoded naively, the in-label period sign is misinterpreted for a label-separator, which results in a different domain name than actually present in the packet. A similar misinterpretation happens with zero-bytes (`"\000"`), which are commonly interpreted as string-terminators in many programming languages.

We show how to exploit these misinterpretations to execute attacks. Essentially, the attacker triggers a query to an attacker-controlled domain, which is typically possible in

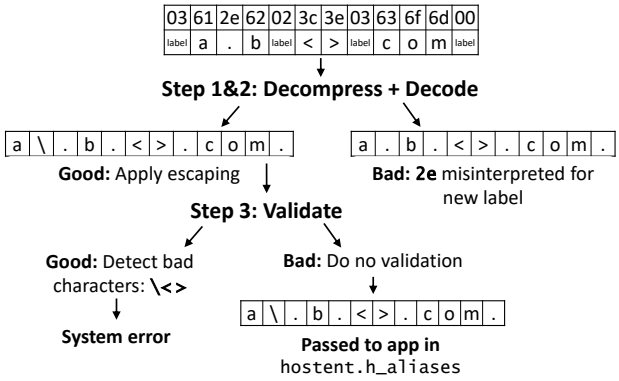


Figure 6.2.: DNS record processing in resolver libraries [6, Figure 7]

many applications, like triggering a query to a malicious website or advertisement in the context of a browser. The response from the attacker's nameserver includes a specially crafted domain name (e.g., `victim.com\000.attacker.com`) which is designed to be misinterpreted by the faulty resolver. The misinterpreted data corresponds to a domain name which is not under the control of the attacker (e.g., `victim.com`). When the resolver is vulnerable, it will misinterpret the domain name and store the attacker-provided IP address for this domain inside its cache. Now, when a client asks for the target domain (`victim.com`), it will receive the maliciously injected address of the attacker from the resolver's cache.

We evaluate this attack (both with zero-byte and period) against various resolver implementations. Our results show that while most common resolver implementations are not vulnerable to the attack, there is still an alarmingly high number of vulnerable devices in the internet, as we found that 8% out of 1.3 million tested open resolvers in the internet are vulnerable to at least one of the attacks.

7. Impact and Countermeasures

In this section, we present countermeasures and mitigations which can be implemented by various parties to block or complicate our attacks in Section 7.1. Additionally to these proposed countermeasures, we list our efforts in eliminating those vulnerabilities by working together with developers in Section 7.2. In Section 7.3, we present our efforts in raising public awareness to these attacks by giving industry talks, providing publicly usable test-tools and using our tools in a vulnerability scan of critical systems in the scope of the German federal elections 2021. Finally, in Section 7.4, we provide conceptual insights which can help to prevent similar vulnerabilities in future application and protocol designs.

7.1. Countermeasures

In this section, we discuss countermeasures and mitigations to block or complicate our attacks. These measures aim to mitigate the attacks, but they do not aim to remove the vulnerability which makes the attack possible itself, e.g., that applications do not implement input validation or resolvers which allow the UDP source port to be guessed via side-channels. In such cases, before these mitigations are implemented, systems should be patched to remove those vulnerabilities. We list our efforts in helping to develop fixes for these vulnerabilities in Section 7.2.

7.1.1. Countermeasures in Applications

Applications and developers can implement several techniques to complicate our attacks, which are listed below.

Third-party authentication. Typical cache poisoning attacks aim to redirect a client to an attacker-provided server to eavesdrop on communications or inject false information. As the DNS is only used to find the correct communication partner in such a use case, third-party authentication, as done via TLS, can mitigate these attacks by authenticating the communication partner.

However, third-party authentication can only protect the confidentiality and integrity of the communication, but not its availability, meaning that attackers can still exploit cache poisoning to make a service unavailable. In cases in which this service is essential to make other security-related decisions, such as in the resource public key infrastructure (RPKI), such an attack can lead to a security-downgrade. This is because the information source for making a security decision is not available [5].

Furthermore, third-party authentication is not available for all use cases of the DNS. For example, the SPF and DMARC email sender validation mechanisms store authorization information directly in the DNS, which can be attacked by cache poisoning as well. Newer approaches like MTA-STS [130] aim to circumvent this by delegating the information storage to a third-party, TLS-protected server. However, they are still vulnerable when cache poisoning occurs in the discovery phase of the mechanism [130, Section 10.2].

Finally, even when using third-party authentication with TLS, the security of the connection still relies on the DNS, as TLS certificates are typically generated by domain validation which by itself can be attacked via cache poisoning [110].

Separating shared systems. Most of our attacks are based on the fact that a single system (e.g., a DNS cache or DNS using application like radsecproxy) is shared by different users. Because of this, duplicating this shared system into multiple instances can mitigate our attacks as the attacker is no longer capable of causing meaningful harm to real users because he or she is locked to its own instance and cannot manipulate the state of other instances. However, implementing such an approach can come at the cost of lower performance, as DNS caches are designed to increase performance by caching requests of multiple users. Thus, if caches are separated, this caching might no longer have the same performance benefits. To reduce the performance impact, this concept can be scaled in terms of how many users share the same system, i.e., instead of instancing a DNS resolver for each user, an operator might decide to only run 2 sets of systems, one for high-privileged and one for low-privileged users. This concept can be applied to DNS caches as well as services which use DNS, such as Radius proxies or SMTP servers, which were the target of our attacks in Section 6.

Higher-level mitigations. In some cases, applications can implement mitigations on higher levels. For example, when the goal of the attacker is to hijack an account as in Section 4, implementing two-factor authentication and notifications of changes in the account can block an attack or enable the legitimate user to revert any malicious changes quickly, despite not preventing the attacker from getting access to the account credentials. Since such types of mitigations are application-specific, we refer to the respective section

about mitigations of the individual works of this thesis [1], [4]–[6].

7.1.2. Countermeasures in DNS Infrastructure

In this section, we describe countermeasures which can be implemented in the DNS Infrastructure.

DNSSEC. All of our attacks which are based on cache poisoning can – and should – ideally be prevented by validating the integrity and authenticity of the data clients receive from resolvers with DNSSEC. DNSSEC protects the authenticity of the data with cryptographic signatures. However, DNSSEC deployment has been slow in the past, with less then 1% of the domains being signed and only 19.14% to 28.94% of resolvers validating signatures in our studies [5], see Section 2.2.3.

Furthermore, DNSSEC validation is typically only implemented in recursive resolvers, but not in forwarders or stub resolvers as these have to rely on given upstream resolvers which might not provide the required DNSSEC records needed for validation [87]. This is problematic when (malicious) modifications to the record data happen after the DNSSEC validation has been performed in the recursive resolver. The data from our study [6] suggests that the misinterpretation which allows the period and zero-byte cache-injection attacks from Section 6 is caused primarily by DNS implementations in DNS forwarders. As neither these forwarders nor its clients typically validate DNSSEC but only rely on the validation performed by upstream resolvers, this deployment of DNSSEC does not prevent our attack, as the validation occurs before the modification of the data happens.

However, it should be noted that "end-to-end" DNSSEC validation, meaning that DNSSEC signatures are validated in end-systems (i.e., stub-resolvers), is technically possible and would prevent our cache poisoning attacks from Chapter 6 if it is implemented correctly.

Other security mechanisms. Additionally to DNSSEC, some specific attacks can be prevented by other security mechanisms on the infrastructure level. For example, BGP hijacking might be mitigated by the deployment of Resource Public Key Infrastructure (RPKI). However, because of several deployment barriers, most of the internet is not protected by RPKI and most networks do not enforce its validation either [131]–[134].

Furthermore, off-path attacks against the DNS (via Fragmentation or ICMP-based side channels) might be blocked by including more randomization in the DNS responses, effectively increasing the entropy of the challenge-response fields the attacker has to guess. This might be achieved by the use of response order randomization [1], 0x20 encoding [135] or DNS cookies [91]. However, these techniques have deployment barriers which

can limit the strictness with which they can be enforced, or they only increase the required resources for an attack – but not completely prevent it.

Filtering untypical requests and responses. Many of our attacks depend on vulnerable processing of untypical requests or responses in the DNS equipment. Examples of such untypical requests and responses are fragmented DNS packets, DNS requests to closed UDP ports, or DNS packets with untypical characters in domain names. Such packets (or the answers generated by them) can be filtered by intermediate systems, such as firewalls, to make the specific attack impossible by preventing the malicious DNS data to reach its target. However, filtering such packets always comes at the cost of (potential) false positives – effectively blocking benign applications from working correctly.

For example, blocking fragmented DNS packets may cause systems to loose connectivity, especially with the deployments of DNSSEC, which increases the size of DNS packets [136]. Furthermore, blocking ICMP messages makes debugging incorrect configurations harder and filtering DNS packets whose payloads contain untypical characters may cause problems with certain applications which depend on these characters being supported – be it applications which already exist, such as SRV service discovery [137] (requiring support for underscore `"_"`), or future applications which are not developed yet. To limit the backfire caused by such mitigations, they should ideally be only deployed in parallel to discussion in the respective standardization bodies, so that existing standards which rely on the blocked functionality are changed and future standards are developed without dependencies to the blocked functionality [138].

In case of the filtering of untypical characters in DNS packets, we provide a proof-of-concept implementation of a DNS proxy which filters such packets at `https://xdi-attack.net`.

7.2. Securing vulnerable software

Additionally to our efforts in proposing mitigations, we have worked with developers of affected software components to patch vulnerabilities which have been found as part of this thesis and improve the resilience of those components against attacks. All vulnerabilities were disclosed as part of a responsible disclosure process. The patches, advisories and Common Vulnerabilities and Exposures identifiers (CVE IDs) which are the result of this process are listed below:

Java. *Vulnerability in the Java SE product of Oracle Java SE.* CVE-2021-2432 [24], advisory available at [17].

OpenWRT. *There is missing input validation of hostnames displayed in OpenWRT before 19.07.8.* CVE-2021-32019 [25], advisory available at [18].

radsecproxy. *Missing input validation in radsecproxy's 'naptr-eduroam.sh' and 'radsec-dynsrv.sh' scripts can lead to configuration injection via crafted radsec peer discovery DNS records.* CVE-2021-32642 [26], advisory available at [19], patch available at [11].

libspf2. *Stack buffer overflow in libspf2 versions below 1.2.11.* CVE-2021-20314 [23], advisory available at [21], patch available at [14].

dietlibc. `__dns_decodename` *will now reject invalid incoming names.* Patch available at [9].

golang. *Go before 1.15.12 and 1.16.x before 1.16.5 allows injection.* CVE-2021-33195 [27], patchnotes available at [16].

NetBSD. *Default to check-names for safety.* Patch available at [10].

c-ares. *Missing input validation on hostnames returned by DNS servers.* CVE-2021-3672 [28], advisory available at [15], patch available at [12].

nodejs. *Improper handling of untypical characters in domain names.* CVE-2021-22931 [29], advisory available at [20].

uclibc-ng. *Incorrect handling of special characters in domain names in uclibc and uclibc-ng.* CVE-2021-43523 [30], advisory available at [22], patch available at [13].

7.3. Raising awareness

Additionally to our attempts in fixing vulnerable implementations, we are raising awareness to the risks of the attacks identified in this work by giving talks at industry conferences [31]–[33] and writing invited articles [34] in an effort to increase the deployment of security mechanisms and mitigations to block the attacks. We have also reached out to standardization bodies in an attempt to standardize the desired behaviour of stub resolvers to block the injection attacks identified in [6].

To allow third parties to test their networks against our attacks, we also provide publicly available test tools at `https://crosslayerattacks.sit.fraunhofer.de/` and `https://xdi-attack.net/`. These allow users to test their systems for the same attack-enabling properties as we tested in the measurements sections of our papers [5], [6].

Finally, we used our methodologies and tools developed during the work to evaluate our attacks to conduct a study of German political parties [35]–[37] in the context of the 2021 German federal elections. This study was conducted by the author of this work together with other employees of Fraunhofer SIT. The results were used to protect the parties against potential attacks aimed against them in an attempt to manipulate the election results.

7.4. Conceptual insights

On a conceptual level, our several works demonstrate the "weakest link" problem when securing internet applications: We have shown how to attack systems and mechanisms by subverting the assumptions made when they were designed – such as honest majorities [2] or assumptions about which components should implement checks in a DNS resolution chain [6]. As such, our works show that many of these implicit dependencies between different components are not well understood, which leads to the vulnerabilities presented in this work.

Furthermore, our work has shown the drawbacks of general internet principles, such as transparency and layering: While these principles have been great for developing new applications and can be seen as one of the reasons for the huge success of the internet as of today, they also come at some risk. For example, applications not expecting transparency might not perform validation on the input they receive. This is amplified by the use of layering, where an upper layer (e.g., an application) does consume services of lower layers (e.g., the DNS) to provide functionality. Because the developers of the upper layer are not aware of the inner workings of the lower layer, and lower layer developers cannot make any assumptions on the upper layer, this can lead to a mismatch of expectations between those applications. This leads to vulnerabilities because each layer expects security-critical validation steps to be performed by the other layer.

8. Conclusions

In this thesis, we have shown how to attack internet applications using the DNS. We have analyzed the DNS usage of various internet applications and created a taxonomy of the DNS usage of applications. We used this taxonomy to show how to systematically develop attacks against these applications. We have developed end-to end attacks against the users of these applications which exploit weaknesses in the DNS protocol as well as DNS usage of the applications themselves and showed that attackers can exploit the DNS to achieve a wider range of different attacks, from shifting time over hijacking internet resources to gaining full control over a target system. We have evaluated our attacks against real-world systems by conducting internet-wide scans against DNS servers as well as applications and showed that a significant amount of users and systems is vulnerable. We have developed and proposed countermeasures to block our attacks and helped developers to patch their applications against our attacks by following a responsible disclosure process. Finally, we provided conceptual insights which can help to make future protocols and applications immune to such attacks by design.

Our work has peen published in a series of 4 full papers [1], [4]–[6] and 2 posters [2], [3] at various high-ranked international conferences. Additionally, the disclosure process of the vulnerabilities found has led to the registration of 8 CVE IDs [23]–[30] and patches in 10 different software implementations [9]–[22]. Finally, to raise awareness to our attacks, we have given presentations at multiple community meetings [31]–[33] and have written invited articles [34].

References

Publications which are part of this thesis

[1] P. Jeitner, H. Shulman, and M. Waidner, "The impact of dns insecurity on time," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, CORE A, 2020, pp. 266–277. DOI: 10.1109/DSN48063.2020.00043.

[2] P. Jeitner, H. Shulman, and M. Waidner, "Pitfalls of provably secure systems in internet the case of chronos-ntp," in *2020 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*, Poster, 2020, pp. 49–50. DOI: 10.1109/DSN-S50200.2020.00027.

[3] P. Jeitner, H. Shulman, and M. Waidner, "Secure consensus generation with distributed doh," in *2020 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*, Poster, 2020, pp. 41–42. DOI: 10.1109/DSN-S50200.2020.00023.

[4] T. Dai, P. Jeitner, H. Shulman, and M. Waidner, "The hijackers guide to the galaxy: Off-path taking over internet resources," in *30th USENIX Security Symposium (USENIX Security 21)*, CORE A*, USENIX Association, Aug. 2021. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/dai.

[5] T. Dai, P. Jeitner, H. Shulman, and M. Waidner, "From ip to transport and beyond: Cross-layer attacks against applications," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, ser. SIGCOMM '21, CORE A*, Virtual Event, USA: Association for Computing Machinery, 2021, pp. 836–849, ISBN: 9781450383837. DOI: 10.1145/3452296.3472933. [Online]. Available: https://doi.org/10.1145/3452296.3472933.

[6] P. Jeitner and H. Shulman, “Injection attacks reloaded: Tunnelling malicious payloads over DNS,” in *30th USENIX Security Symposium (USENIX Security 21)*, CORE A*, USENIX Association, Aug. 2021. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/jeitner.

Other publications of the author of this thesis

[7] T. Hlavacek, P. Jeitner, D. Mirdita, H. Shulman, and M. Waidner, “Stalloris: RPKI downgrade attack,” in *31st USENIX Security Symposium (USENIX Security 22)*, CORE A*, Boston, MA: USENIX Association, Aug. 2022. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/hlavacek.

[8] P. Jeitner, H. Shulman, L. Teichmann, and M. Waidner, “XDRI attacks - and - how to enhance resilience of residential routers,” in *31st USENIX Security Symposium (USENIX Security 22)*, CORE A*, Boston, MA: USENIX Association, Aug. 2022. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/jeitner.

Patches and advisories in software

[9] dietlibc, *__dns_decodename will now reject invalid incoming names*, 2021. [Online]. Available: https://bazaar.launchpad.net/~mirabilos/dietlibc/MAIN/revision/2482.

[10] NetBSD, *Default to check-names for safety*, 2021. [Online]. Available: https://github.com/NetBSD/src/commit/e54a0531e4b16a8d1959782.

[11] F. Mauchle, *add result validation to dyndisc example scripts*, 2021. [Online]. Available: https://github.com/radsecproxy/radsecproxy/commit/ab7a2ea42a75d5ad3421e4365f63cbdcb08fb7af.

[12] B. House, *ares_expand_name() should escape more characters*, 2021. [Online]. Available: https://github.com/c-ares/c-ares/compare/809d5e8..44c009b.patch.

[13] W. Brodkorb, *Merge remote-tracking branch ’evolvis/dns-things’*, 2021. [Online]. Available: https://github.com/wbx-github/uclibc-ng/commit/0894e879f091a6a281c02b8da1c0606f4f2dcfb8.

[14] Shevek, *spf_compile.c: Correct size of ds_avail.* 2021. [Online]. Available: https://github.com/shevek/libspf2/commit/c37b7c13c30e225183899364b9f2efdfa85552ef.

[15] Project c-ares Security Advisory, *Missing input validation on hostnames returned by DNS servers*, 2021. [Online]. Available: https://c-ares.haxx.se/adv_20210810.html.

[16] *Go 1.16.5 and Go 1.15.13 are released*, 2021. [Online]. Available: https://groups.google.com/g/golang-announce/c/RgCMkAEQjSI.

[17] *Oracle Critical Patch Update Advisory - July 2021*, 2021. [Online]. Available: https://www.oracle.com/security-alerts/cpujul2021.html.

[18] *Security Advisory 2021-08-01-1 - XSS via missing input validation of host names displayed*, 2021. [Online]. Available: https://openwrt.org/advisory/2021-08-01-1.

[19] F. Mauchle, *Missing input validation in dynamic discovery example scripts*, 2021. [Online]. Available: https://github.com/radsecproxy/radsecproxy/security/advisories/GHSA-56gw-9rj9-55rc.

[20] M. Dawson, *August 2021 Security Releases*, 2021. [Online]. Available: https://nodejs.org/en/blog/vulnerability/aug-2021-security-releases/.

[21] P. Jeitner, *Remote stack buffer overflow in libspf2*, 2021. [Online]. Available: https://www.openwall.com/lists/oss-security/2021/08/11/6.

[22] P. Jeitner, *Incorrect handling of special characters in domain names in uclibc and uclibc-ng*, 2021. [Online]. Available: https://www.openwall.com/lists/oss-security/2021/11/09/1.

CVEs assigned due to the works in this thesis

[23] MITRE, *CVE-2021-20314: Stack buffer overflow in libspf2 versions below 1.2.11 when processing certain SPF macros*, 2021. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-20314.

[24] MITRE, *CVE-2021-2432: Vulnerability in the Java SE product of Oracle Java SE*, 2021. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-2432.

[25] MITRE, *CVE-2021-32019: There is missing input validation of host names displayed in OpenWrt before 19.07.8*, 2021. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-32019.

[26] MITRE, *CVE-2021-32642: Missing input validation in radsecproxy's 'naptr-eduroam.sh' and 'radsec-dynsrv.sh' scripts can lead to configuration injection via crafted radsec peer discovery DNS records*, 2021. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-32642.

[27] MITRE, *CVE-2021-33195: Go before 1.15.12 and 1.16.x before 1.16.5 allows injection*, 2021. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-33195.

[28] MITRE, *CVE-2021-3672: Missing input validation check of host names returned by DNS in c-ares*, 2021. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-3672.

[29] MITRE, *CVE-2021-22931: Node.js before 16.6.0, 14.17.4, and 12.22.4 is vulnerable to Remote Code Execution, XSS, Application crashes due to missing input validation of host names*, 2021. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-22931.

[30] MITRE, *CVE-2021-43523: Incorrect handling of special characters in uClibc and uClibc-ng before 1.0.39*. 2021. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-43523.

Non-academic talks and articles

[31] T. Dai, *The Hijackers Guide To The Galaxy: Off-Path Taking Over Internet Resources*, NANOG 83, 2021. [Online]. Available: https://www.youtube.com/watch?v=6Pic7wm7agM.

[32] P. Jeitner, *Injection Attacks Reloaded: Tunnelling Malicious Payloads over DNS*, NANOG 83, 2021. [Online]. Available: https://www.youtube.com/watch?v=G5xFSvb7awY.

[33] T. Dai, *The Hijackers Guide To The Galaxy: Off-Path Taking Over Internet Resources*, RIPE 83, Video: https://ripe83.ripe.net/archives/video/667/, 2021. [Online]. Available: https://ripe83.ripe.net/wp-content/uploads/presentations/47-RIPE83_Dai.pdf.

[34] P. Jeitner, *Resurrection of injection attacks*, APNIC Blog, 2022. [Online]. Available: https://blog.apnic.net/2022/02/22/resurrection-of-injection-attacks/.

Media coverage of political party study

[35] M. Bergermann and N. Husmann, "Experten offenbaren, wie leicht Deutschlands Parteien Opfer von Hackern werden können," *WirtschaftsWoche*, 2021. [Online]. Available: https://www.wiwo.de/politik/deutschland/dringender-handlungsbedarf-experten-offenbaren-wie-leicht-deutschlands-parteien-opfer-von-hackern-werden-koennen/27401302.html.

[36] A. Biselli, "Parteien müssen ihre IT-Sicherheit stärken," *golem.de*, 2021. [Online]. Available: https://www.golem.de/news/bundestagswahl-parteien-muessen-ihre-it-sicherheit-staerken-2107-157970.html.

[37] *Fraunhofer-Studie zur IT-Sicherheit der Parteien*, 2021. [Online]. Available: https://www.athene-center.de/en/news/news/fraunhofer-studie-zur-it-sicherheit-der-parteien-1424.

Other references

[38] P. Mockapetris, "Domain names - concepts and facilities," RFC Editor, STD 13, Nov. 1987. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1034.txt.

[39] K. Harrenstien, M. Stahl, and E. Feinler, "Dod internet host table specification," RFC Editor, RFC 952, Oct. 1985.

[40] D. Atkins and R. Austein, "Threat analysis of the domain name system (dns)," RFC Editor, RFC 3833, Aug. 2004.

[41] T. Chung, R. van Rijswijk-Deij, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson, "Understanding the role of registrars in dnssec deployment," in *Proceedings of the 2017 Internet Measurement Conference*, 2017, pp. 369–383.

[42] D. Kaminsky, "It's the End of the Cache As We Know It," in *Black Hat conference*, http://www.blackhat.com/presentations/bh-jp-08/bh-jp-08-Kaminsky/BlackHat-Japan-08-Kaminsky-DNS08-BlackOps.pdf, Aug. 2008.

[43] A. Herzberg and H. Shulman, "Fragmentation Considered Poisonous: Or one-domain-to-rule-them-all.org," in *IEEE CNS 2013. The Conference on Communications and Network Security, Washington, D.C., U.S.*, IEEE, Oct. 2013.

[44] K. Man, Z. Qian, Z. Wang, X. Zheng, Y. Huang, and H. Duan, "DNS Cache Poisoning Attack Reloaded: Revolutions with Side Channels," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020.

[45] M. Brandt, T. Dai, A. Klein, H. Shulman, and M. Waidner, "Domain Validation++ For MitM-Resilient PKI," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2018, pp. 2060–2076.

[46] *'Unprecedented' DNS Hijacking Attacks Linked to Iran*, 2019. [Online]. Available: https://threatpost.com/unprecedented-dns-hijacking-attacks-linked-to-iran/140737/.

[47] *Hacked or spoofed: Digging into the malaysia airlines website incident*, https://www.trendmicro.com/vinfo/us/security/news/cyber-attacks/hacked-or-spoofed-digging-into-the-malaysia-airlines-website-compromise, Accessed: 2021-1-19, 2015.

[48] *Webnic registrar blamed for hijack of lenovo, google domains*, https://krebsonsecurity.com/2015/02/webnic-registrar-blamed-for-hijack-of-lenovo-google-domains/, Accessed: 2021-1-19, 2015.

[49] S. Goldberg, *The myetherwallet.com hijack and why it's risky to hold cryptocurrency in a webapp*, 2018. [Online]. Available: https://medium.com/@goldbe/the-myetherwallet-com-hijack-and-why-its-risky-to-hold-cryptocurrency-in-a-webapp-261131fad278.

[50] *Global dns hijacking campaign: Dns record manipulation at scale*, https://www.fireeye.com/blog/threat-research/2019/01/global-dns-hijacking-campaign-dns-record-manipulation-at-scale.html, Accessed: 2021-1-19, 2019.

[51] *Sea turtle keeps on swimming, finds new victims, dns hijacking techniques*, https://blog.talosintelligence.com/2019/07/sea-turtle-keeps-on-swimming.html, Accessed: 2021-01-19, 2019.

[52] P. Mockapetris, "Domain names - implementation and specification," RFC Editor, STD 13, Nov. 1987. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1035.txt.

[53] N. Chatzis, "Motivation for behaviour-based dns security: A taxonomy of dns-related internet threats," in *The International Conference on Emerging Security Information, Systems, and Technologies (SECUREWARE 2007)*, 2007, pp. 36–41. DOI: 10.1109/SECUREWARE.2007.4385307.

[54] C. Marrison, "Understanding the threats to dns and how to secure it," *Network Security*, vol. 2015, no. 10, pp. 8–10, 2015, ISSN: 1353-4858. DOI: 10.1016/S1353-4858(15)30090-8. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1353485815300908.

[55] J. Postel, "User datagram protocol," RFC Editor, STD 6, Aug. 1980. [Online]. Available: http://www.rfc-editor.org/rfc/rfc768.txt.

[56] M. Prince, *The DDoS That Almost Broke the Internet*, CloudFlare Blog, Apr. 2013.

[57] V. Paxson, "An Analysis of Using Reflectors for Distributed Denial-of-Service Attacks," *Computer Communication Review*, vol. 31, no. 3, pp. 38–47, 2001. [Online]. Available: http://doi.acm.org/10.1145/505659.505664.

[58] C. Rossow, "Amplification Hell: Revisiting Network Protocols for DDoS Abuse," in *Proceedings of the Network and Distributed System Security (NDSS) Symposium*, Feb. 2014.

[59] C. Jackson, A. Barth, A. Bortz, W. Shao, and D. Boneh, "Protecting browsers from dns rebinding attacks," *ACM Transactions on the Web (TWEB)*, vol. 3, no. 1, pp. 1–26, 2009.

[60] A. Barth, "The web origin concept," RFC Editor, RFC 6454, Dec. 2011. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6454.txt.

[61] CmdrTaco, *Ip tunneling through nameservers*, https://slashdot.org/story/00/09/10/2230242/ip-tunneling-through-nameservers, 2000.

[62] D. Kaminsky, *Black ops of dns*, http://s3.amazonaws.com/dmk/Black_Ops_DNS_BH.ppt, 2004.

[63] R. Siamwalla, R. Sharma, and S. Keshav, "Discovering internet topology," *Unpublished manuscript*, 1998.

[64] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "Dns security introduction and requirements," RFC Editor, RFC 4033, Mar. 2005. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4033.txt.

[65] B. Laurie, G. Sisson, R. Arends, and D. Blacka, "Dns security (dnssec) hashed authenticated denial of existence," RFC Editor, RFC 5155, Mar. 2008. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5155.txt.

[66] M. Wander, L. Schwittmann, C. Boelmann, and T. Weis, "Gpu-based nsec3 hash breaking," in *Network Computing and Applications (NCA), 2014 IEEE 13th International Symposium on*, IEEE, 2014, pp. 137–144.

[67] J. Oberheide, M. Karir, and Z. M. Mao, "Characterizing dark dns behavior," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer, 2007, pp. 140–156.

[68] *Sublist3r: Fast subdomains enumeration tool for penetration testers*, `https://github.com/aboul3la/Sublist3r`, 2015.

[69] *Dnsrecon: Dns enumeration script*, `https://github.com/darkoperator/dnsrecon`, 2010.

[70] J. J. Gondim, R. de Oliveira Albuquerque, and A. L. Sandoval Orozco, "Mirror saturation in amplified reflection distributed denial of service: A case of study using snmp, ssdp, ntp and dns protocols," *Future Generation Computer Systems*, vol. 108, pp. 68–81, 2020, ISSN: 0167-739X. DOI: `10.1016/j.future.2020.01.024`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0167739X19322745`.

[71] T. van Leijenhorst, K.-W. Chin, and D. Lowe, "On the viability and performance of dns tunneling," in *Conference on Information Technology and Applications*, 2008, pp. 560–566.

[72] P. Vixie, "DNS and BIND security issues," in *Proceedings of the 5th Symposium on UNIX Security*, Berkeley, CA, USA: USENIX Association, Jun. 1995, pp. 209–216, ISBN: 1-880446-70-7.

[73] D. J. Bernstein, *DNS Forgery*, `http://cr.yp.to/djbdns/forgery.html`, Nov. 2002.

[74] *New Threat: Targeted Internet Traffic Misdirection*, `http://www.renesys.com/2013/11/mitm-internet-hijacking`.

[75] H. Ballani, P. Francis, and X. Zhang, "A Study of Prefix Hijacking and Interception in the Internet," in *Proceedings of ACM SIGCOMM*, 2007.

[76] H. Birge-Lee, Y. Sun, A. Edmundson, J. Rexford, and P. Mittal, "Bamboozling certificate authorities with BGP," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[77] Y. Gilad and A. Herzberg, "Off-path tcp injection attacks," *ACM Transactions on Information and System Security (TISSEC)*, vol. 16, no. 4, p. 13, 2014.

[78] H. Shulman and M. Waidner, "Towards security of internet naming infrastructure," in *European Symposium on Research in Computer Security*, Springer, 2015, pp. 3–22.

[79] A. Malhotra and S. Goldberg, "Attacking NTP's Authenticated Broadcast Mode," en, *ACM SIGCOMM Computer Communication Review*, vol. 46, no. 1, pp. 12–17, May 2016, ISSN: 01464833. DOI: 10.1145/2935634.2935637. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2935634.2935637 (visited on 02/06/2019).

[80] D. Madory, *Recent Routing Incidents: Using BGP to Hijack DNS and more*, 2018. [Online]. Available: https://www.lacnic.net/innovaportal/file/3207/1/dougmadory_lacnic_30_rosario.pdf.

[81] *Dnspionage campaign targets middle east*, https://blog.talosintelligence.com/2018/11/dnspionage-campaign-targets-middle-east.html, Accessed: 2021-01-19, 2018.

[82] *Security incident on november 13, 2020*, https://blog.liquid.com/security-incident-november-13-2020, Accessed: 2021-01-19, 2020.

[83] J. B. Postel, "Simple mail transfer protocol," RFC Editor, STD 10, Aug. 1982. [Online]. Available: http://www.rfc-editor.org/rfc/rfc821.txt.

[84] T. Berners-Lee, R. T. Fielding, and H. F. Nielsen, "Hypertext transfer protocol – http/1.0," RFC Editor, RFC 1945, May 1996. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1945.txt.

[85] D. E. E. 3rd and C. W. Kaufman, "Domain name system security extensions," RFC Editor, RFC 2065, Jan. 1997. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2065.txt.

[86] T. Chung, R. van Rijswijk-Deij, B. Chandrasekaran, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson, "A longitudinal, end-to-end view of the DNSSEC ecosystem," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 1307–1322.

[87] W. Hardaker, O. Gudmundsson, and S. Krishnaswamy, "Dnssec roadblock avoidance," RFC Editor, BCP 207, Nov. 2016.

[88] H. Shulman and M. Waidner, "One key to sign them all considered vulnerable: Evaluation of dnssec in the internet.," in *NSDI*, 2017.

[89] A. Hubert and R. van Mook, "Measures for making dns more resilient against forged answers," RFC Editor, RFC 5452, Jan. 2009.

[90] D. Dagon, M. Antonakakis, P. Vixie, T. Jinmei, and W. Lee, "Increased DNS forgery resistance through 0x20-bit encoding: Security via leet queries," in *ACM Conference on Computer and Communications Security*, P. Ning, P. F. Syverson, and S. Jha, Eds., ACM, 2008, pp. 211–222, ISBN: 978-1-59593-810-7. [Online]. Available: http://doi.acm.org/10.1145/1455770.1455798.

[91] D. E. E. 3rd and M. P. Andrews, *Domain Name System (DNS) Cookies*, RFC 7873, May 2016. DOI: 10.17487/RFC7873. [Online]. Available: https://rfc-editor.org/rfc/rfc7873.txt.

[92] A. Klein, H. Shulman, and M. Waidner, "Internet-Wide Study of DNS Cache Injections," in *INFOCOM*, 2017.

[93] P. Hoffman and P. McManus, "Dns queries over https (doh)," RFC Editor, RFC 8484, Oct. 2018.

[94] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. Hoffman, *Rfc 7858-specification for dns over transport layer security (tls)*, 2016.

[95] D. Mills, J. Martin, J. Burbank, and W. Kasch, "Network time protocol version 4: Protocol and algorithms specification," RFC Editor, RFC 5905, Jun. 2010. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5905.txt.

[96] D. Mills, "Simple network time protocol (sntp) version 4 for ipv4, ipv6 and osi," RFC Editor, RFC 4330, Jan. 2006. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4330.txt.

[97] *www.ntppool.org*, https://www.ntppool.org/en/, accessed 19-10-2021.

[98] D. Reilly, H. Stenn, and D. Sibold, "Network time protocol best current practices," RFC Editor, BCP 223, Jul. 2019.

[99] S. Röttger, *Analysis of the ntp autokey procedures*, https://web.archive.org/web/20150103044649/https://zero-entropy.de/autokey_analysis.pdf, 2012.

[100] D. Franke, D. Sibold, K. Teichel, M. Dansarie, and R. Sundblad, "Network time security for the network time protocol," RFC Editor, RFC 8915, Sep. 2020.

[101] A. Malhotra, A. Langley, W. Ladd, and M. Dansarie, "Roughtime," IETF Secretariat, Internet-Draft draft-ietf-ntp-roughtime-05, May 2021. [Online]. Available: https://www.ietf.org/archive/id/draft-ietf-ntp-roughtime-05.txt.

[102] A. Malhotra, M. Van Gundy, M. Varia, H. Kennedy, J. Gardner, and S. Goldberg, "The Security of NTP's Datagram Protocol," en, in *Financial Cryptography and Data Security*, A. Kiayias, Ed., vol. 10322, Cham: Springer International Publishing, 2017, pp. 405–423, ISBN: 978-3-319-70971-0 978-3-319-70972-7. DOI: 10.1007/978-3-319-70972-7_23. [Online]. Available: http://link.springer.com/10.1007/978-3-319-70972-7_23 (visited on 02/06/2019).

[103] J. Czyz, M. Kallitsis, M. Gharaibeh, C. Papadopoulos, M. Bailey, and M. Karir, "Taming the 800 pound gorilla: The rise and decline of ntp ddos attacks," in *Proceedings of the 2014 Conference on Internet Measurement Conference*, ACM, 2014, pp. 435–448.

[104] A. Malhotra, I. E. Cohen, E. Brakke, and S. Goldberg, "Attacking the Network Time Protocol," en, in *Proceedings 2016 Network and Distributed System Security Symposium*, San Diego, CA: Internet Society, 2016, ISBN: 978-1-891562-41-9. DOI: 10.14722/ndss.2016.23090. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/sites/25/2017/09/attacking-network-time-protocol.pdf (visited on 02/06/2019).

[105] O. Deutsch, N. R. Schiff, D. Dolev, and M. Schapira, "Preventing (Network) Time Travel with Chronos," in *Proceedings 2018 Network and Distributed System Security Symposium*, San Diego, CA, 2018, ISBN: 978-1-891562-49-5.

[106] N. Rozen-Schiff, D. Dolev, T. Mizrahi, and M. Schapira, "A secure selection and filtering mechanism for the network time protocol," IETF Secretariat, Internet-Draft draft-ietf-ntp-chronos-03, Aug. 2021. [Online]. Available: https://www.ietf.org/archive/id/draft-ietf-ntp-chronos-03.txt.

[107] R. Housley, J. Curran, G. Huston, and D. Conrad, "The internet numbers registry system," RFC Editor, RFC 7020, Aug. 2013. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7020.txt.

[108] V. Dukhovni and W. Hardaker, "Smtp security via opportunistic dns-based authentication of named entities (dane) transport layer security (tls)," RFC Editor, RFC 7672, Oct. 2015.

[109] A. Klein, "Cross layer attacks and how to use them (for dns cache poisoning, device tracking and more)," in *2021 IEEE Symposium on Security and Privacy (SP)*, 2021, pp. 1179–1196. DOI: 10.1109/SP40001.2021.00054.

[110] M. Brandt, T. Dai, A. Klein, H. Shulman, and M. Waidner, "Domain Validation++ For MitM-Resilient PKI," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18, New York, NY, USA: ACM, 2018, pp. 2060–2076, ISBN: 978-1-4503-5693-0. DOI: 10.1145/3243734.3243790. [Online]. Available: http://doi.acm.org/10.1145/3243734.3243790 (visited on 02/04/2019).

[111] J. Selvi, "Bypassing http strict transport security," *Black Hat Europe*, 2014.

[112] Y. Sun, M. Apostolaki, H. Birge-Lee, L. Vanbever, J. Rexford, M. Chiang, and P. Mittal, "Securing internet applications from routing attacks," *Commun. ACM*, vol. 64, no. 6, pp. 86–96, 2021. DOI: 10.1145/3429775. [Online]. Available: https://doi.org/10.1145/3429775.

[113] C. Rigney, S. Willens, A. Rubens, and W. Simpson, "Remote authentication dial in user service (radius)," RFC Editor, RFC 2865, Jun. 2000. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2865.txt.

[114] S. Winter and M. McCauley, "Dynamic Peer Discovery for RADIUS/TLS and RADIUS/DTLS Based on the Network Access Identifier (NAI)," RFC Editor, RFC 7585, Oct. 2015.

[115] E. Rescorla, "The transport layer security (tls) protocol version 1.3," RFC Editor, RFC 8446, Aug. 2018.

[116] M. Lepinski and S. Kent, "An infrastructure to support secure internet routing," RFC Editor, RFC 6480, Feb. 2012. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6480.txt.

[117] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams, "X.509 internet public key infrastructure online certificate status protocol - ocsp," RFC Editor, RFC 6960, Jun. 2013. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6960.txt.

[118] *A03 Injection - OWASP Top 10:2021*, https://owasp.org/Top10/A03_2021-Injection/.

[119] Z. Su and G. Wassermann, "The essence of command injection attacks in web applications," *Acm Sigplan Notices*, vol. 41, no. 1, pp. 372–382, 2006.

[120] Y. Nadji, P. Saxena, and D. Song, "Document structure integrity: A robust basis for cross-site scripting defense.," in *NDSS*, vol. 20, 2009.

[121] J. M. Alonso, R. Bordon, M. Beltran, and A. Guzman, "LDAP injection techniques," in *2008 11th IEEE Singapore International Conference on Communication Systems*, ISSN: null, Nov. 2008, pp. 980–986. DOI: 10.1109/ICCS.2008.4737330.

[122] M. Johns and J. Winter, "RequestRodeo: Client side protection against session riding," in *Proceedings of the OWASP Europe 2006 Conference*, 2006.

[123] S. P. Chandramouli, P.-M. Bajan, C. Kruegel, G. Vigna, Z. Zhao, A. Doupé, and G.-J. Ahn, "Measuring e-mail header injections on the world wide web," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, 2018, pp. 1647–1656.

[124] T. Terada, "Smtp injection via recipient email addresses," *MBSD White Paper (December 2015)*, 2015.

[125] *Next Steps - OWASP Top 10:2021*, https://owasp.org/Top10/A11_2021-Next_Steps/.

[126] M. Dalton, C. Kozyrakis, and N. Zeldovich, "Nemesis: Preventing Authentication & [and] Access Control Vulnerabilities in Web Applications," 2009.

[127] R. Sekar, "An efficient black-box technique for defeating web application attacks.," in *NDSS*, 2009.

[128] "Standard for Information Technology–Portable Operating System Interface (POSIX (R)) Base Specifications, Issue 7," *IEEE Std 1003.1, 2016 Edition (incorporates IEEE Std 1003.1-2008, IEEE Std 1003.1-2008/Cor 1-2013, and IEEE Std 1003.1-2008/Cor 2-2016)*, pp. 1–3957, Sep. 2016, ISSN: null. DOI: 10.1109/IEEESTD.2016.7582338.

[129] A. Gustafsson, "Handling of unknown dns resource record (rr) types," RFC Editor, RFC 3597, Sep. 2003.

[130] D. Margolis, M. Risher, B. Ramakrishnan, A. Brotman, and J. Jones, "Smtp mta strict transport security (mta-sts)," RFC Editor, RFC 8461, Sep. 2018.

[131] Y. Gilad, A. Cohen, A. Herzberg, M. Schapira, and H. Shulman, "Are we there yet? on rpki's deployment and security.," in *NDSS*, 2017.

[132] T. Hlavacek, A. Herzberg, H. Shulman, and M. Waidner, "Practical experience: Methodologies for measuring route origin validation," in *48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2018, Luxembourg City, Luxembourg, June 25-28, 2018*, 2018, pp. 634–641. DOI: 10.1109/DSN.2018.00070. [Online]. Available: https://doi.org/10.1109/DSN.2018.00070.

[133] A. Reuter, R. Bush, Í. Cunha, E. Katz-Bassett, T. C. Schmidt, and M. Wählisch, "Towards a rigorous methodology for measuring adoption of RPKI route validation and filtering," *CoRR*, vol. abs/1706.04263, 2017. arXiv: 1706.04263. [Online]. Available: http://arxiv.org/abs/1706.04263.

[134] T. Hlavacek, I. Cunha, Y. Gilad, A. Herzberg, E. Katz-Bassett, M. Schapira, and H. Shulman, "Disco: Sidestepping rpki's deployment barriers," in *Network and Distributed System Security Symposium (NDSS)*, 2020.

[135] P. A. Vixie and D. Dagon, "Use of Bit 0x20 in DNS Labels to Improve Transaction Identity," Internet Engineering Task Force, Internet-Draft draft-vixie-dnsext-dns0x20-00, Mar. 2008, Work in Progress, 8 pp. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-vixie-dnsext-dns0x20-00.

[136] K. Fujiwara and P. A. Vixie, "Fragmentation Avoidance in DNS," Internet Engineering Task Force, Internet-Draft draft-ietf-dnsop-avoid-fragmentation-06, Dec. 2021, Work in Progress, 12 pp. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-dnsop-avoid-fragmentation-06.

[137] A. Gulbrandsen and D. L. Esibov, *A DNS RR for specifying the location of services (DNS SRV)*, RFC 2782, Feb. 2000. DOI: 10.17487/RFC2782. [Online]. Available: https://rfc-editor.org/rfc/rfc2782.txt.

[138] S. W. Brim and B. E. Carpenter, *Middleboxes: Taxonomy and Issues*, RFC 3234, Feb. 2002. DOI: 10.17487/RFC3234. [Online]. Available: https://rfc-editor.org/rfc/rfc3234.txt.

A. The Impact of DNS Insecurity on Time

[1] P. Jeitner, H. Shulman, and M. Waidner, “The impact of dns insecurity on time,” in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, CORE A, 2020, pp. 266–277. DOI: 10.1109/DSN48063.2020.00043.

The Impact of DNS Insecurity on Time

Philipp Jeitner*, Haya Shulman*, Michael Waidner*†
*Technical University of Darmstadt, †Fraunhofer Institute for Secure Information Technology SIT

***Abstract*— We demonstrate the first practical off-path time shifting attacks against NTP as well as against Man-in-the-Middle (MitM) secure Chronos-enhanced NTP. Our attacks exploit the insecurity of DNS allowing us to redirect the NTP clients to attacker controlled servers. We perform large scale measurements of the attack surface in NTP clients and demonstrate the threats to NTP due to vulnerable DNS.**

I. Introduction

Network Time Protocol (NTP) is one of the core Internet protocols meant to synchronise time on Internet systems. Due to its critical role in the Internet, NTP has a long history of attacks. Recently [1] demonstrated a proof of concept attack against NTP that allows off-path attackers to shift time on victim clients. The idea of [1] was to exploit overlapping IPv4 fragments, whereby a shifted time provided in attacker's fragments would overwrite the time provided by the real NTP server in a response sent to the NTP client. However, the limiting factors of the attack are: (1) the attack requires that the NTP servers agree to fragment the NTP responses to IP packets of 68 byte Maximum Transmission Unit (MTU) - NTP servers do not fragment to such low MTU, (2) the attacker must create and synchronise two spoofed fragments concurrently - this is difficult to achieve even in lab conditions and (3) the client must agree to shift time - since only a response from one NTP server is shifted while the rest return correct responses, the client would in most cases ignore it. Indeed, the authors found that only $0,0008\%$ of the NTP servers in the Internet could potentially be vulnerable to the attack.

Chronos. Nevertheless, due to the critical role that NTP plays in the Internet there was an immediate followup work, which devised enhancements to NTP, called Chronos [2], to block that attack. Chronos is also on a standardisation track of the IETF [3]. Chronos leverages ideas from distributed computing on clock synchronisation in the presence of Byzantine adversaries and is designed to provide security even against strong Man-in-the-Middle (MitM) attackers and corrupted NTP servers. To enhance the security of NTP with Chronos only the NTP clients need to be modified. In contrast to "plain" NTP which queries few (typically up to 4) NTP servers, Chronos enhanced client queries time from multiple NTP servers, applies a secure algorithm for eliminating suspicious responses and averages the time over the valid responses. The authors demonstrate that in order to shift time on Chronos enhanced NTP client by 100ms a strong MitM attacker would need 20 years of effort. *In this work, in addition to demonstrating practical off-path time-shifting attacks against plain NTP, we also show off-path attacks against Chronos enhanced NTP.*

Security in isolation. Chronos [2], [3] was designed to guarantee the security of NTP in isolation assuming an ideal environment and not taking into account other systems and protocols, like inter-domain routing, Domain Name System (DNS), Network Address Translation (NAT) devices. In the Internet, where multiple systems are running in concert, such a model does not encompass all the possible threats and attack vectors which can be exploited to subvert the security of a system. Indeed, as we demonstrate in this work, vulnerabilities in these systems can adversely affect the security of NTP.

The authors in [2] *implicitly* assume that the attacks on NTP are launched when the NTP client queries the NTP servers in the Internet for time. They model the attacker to be either a MitM that changes some responses to contain shifted time or an attacker that corrupts some of the NTP servers so that they provide invalid time. The security of Chronos is guaranteed if the majority of the queried NTP servers provide correct time. The idea is that all the responses are fed to a sophisticated algorithm which calculates time based on all the responses, the algorithm identifies and discards "bad" time and only considers responses with valid time. But, what happens when all, or a majority of, the responses are incorrect? In this case, Chronos will not be able to identify and discard bad responses. Luckily, even the most powerful MitM attacker cannot launch an attack to concurrently change the responses from *all or a majority of* the queried NTP servers - this is the idea underlying the design of Chronos.

The "achilles heel" of Chronos. To generate a pool of NTP servers from which it then polls the time, the Chronos client periodically (every hour) queries DNS for domain `pool.ntp.org`. The idea is to gradually collect hundreds of NTP servers, from which the Chronos client will be selecting the NTP servers at random and will be querying them for time. Then applying a sophisticating algorithm for filtering bad responses. We show that the achilles heel of Chronos are the multiple DNS queries.

Our attack. The idea of our attack is to leverage the cache poisoning vulnerability in DNS, and to inject malicious DNS records redirecting the DNS queries for `pool.ntp.org` to an attacker controlled host. The attack consists of these phases (illustrated in Figure 1): the attacker needs to predict (or to trigger) a query from the victim DNS resolver to the nameserver `pool.ntp.org` sent in step ② (we explain this in Section IV). The attacker performs DNS cache poisoning in step Â (Section III). The attacker causes the victim NTP client to drop the existing associations to real NTP servers in

2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)
Accepted version © 2020 IEEE. https://doi.org/10.1109/DSN48063.2020.00043

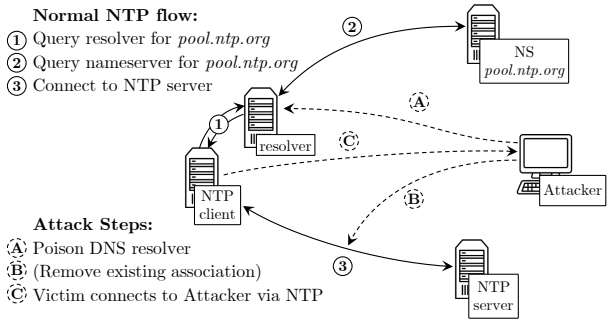


Fig. 1. Attack overview

step Ⓑ (Section IV-B). Finally, the victim NTP client queries the attacker's NTP server and receives shifted time in step Ⓒ.

These attacker controlled NTP servers provide incorrect time when receiving queries from NTP clients. Hence allowing the attacker to ensure that all (or a majority of) the responses from the malicious NTP servers are bad.

Our attack also applies against Chronos enhanced NTP client. In fact the multiple DNS requests issued by Chronos make the attack even easier to launch than against plain NTP. We demonstrate that these attacks can be launched even with an *off-path attacker*. Such attacks are even easier to launch with stronger attackers, such as those performing BGP prefix hijacking [4]–[8].

Countermeasures. DNSSEC [RFC4033-RFC4035] would prevent the attacks, however, our measurements show that most NTP domains are not signed and most DNS resolvers do not perform validation. Only one domain (`time.cloudflare.com`) is signed with DNSSEC, and only between 19.14% and 28.94% of the clients (depending on the geolocation) validate DNSSEC.

Ethical considerations. Our attacks were tested against remote networks reliably, yet were ethically compliant. We measured and evaluated vulnerabilities in the DNS caches of the subjects of our study and measured which DNS caches are used by NTP servers on those networks, but did not hijack their traffic nor Internet resources and neither did we place incorrect DNS records for Internet domains that are not under our control in the caches of our subjects. Specifically, to avoid harming Internet customers and domains, we set up victim domains, which were used by us for evaluating the attacks, and evaluating the vulnerabilities. This ensured that the "victim" networks would not use the spoofed records for any "practical" purpose. We kept an extremely low attack volume and did not generate excessive traffic to avoid any interference with the operations of those services or load on them.

Our research shows that the NTP ecosystem is vulnerable to *practical* off-path cache poisoning attacks and worse, that the recent proposal Chronos for improving NTP security is even more susceptible to our attacks than the plain NTP. We are disclosing the vulnerabilities along with recommendations for countermeasures.

Our contributions.

• In this work we improve over the attacks presented in [1] and demonstrate the first off-path attacks against Chronos-enhanced NTP that was proven secure even against the strong MitM attackers.

• *New attack methodologies.* We develop new methods to launch the attacks when the attacker does not control the timing of the queries and the timing is difficult to predict: (1) recurring IP defragmentation cache poisoning until the attack succeeds, (2) utilising other systems using the same DNS resolver for initiating the attack and issuing the query.

• *Adaptation of general cache poisoning techniques for attacks against NTP.* For our cache poisoning attacks we apply defragmentation cache poisoning attacks presented in [9], and adapt them for attacks against NTP. The idea is to inject a spoofed second fragment that is reassembled with the first fragment provided by the real nameserver. This allows to match the challenge response parameters without having to guess them because they are in the first fragment, while injecting malicious payload in the second fragment. We provide evaluation of these techniques against the NTP ecosystem. During run-time the NTP client already has associations to other NTP servers and hence will not use the injected record with the new NTP server. We develop a technique which causes the NTP client to break established associations to NTP servers and *pin* the client to the NTP server provided by the attacker. To do this, we exploit the rate limiting defence supported by the vast majority of NTP servers in the Internet.

• *We measure the attack surface introduced by our techniques on NTP ecosystem.* We perform extensive measurements to identify the NTP clients that are vulnerable to our off-path attacks. This essentially means NTP clients that use DNS resolvers that are vulnerable to cache poisoning attacks.

• *Conceptually our work demonstrates the "weakest link" problem in the Internet.* Our work provides insights to the gap between theoretical proofs of security done in isolated environments with assumptions which, unfortunately, do not hold in practice. This exposes the systems, even when proven secure, to unexpected attacks. As we show in this work, NTP heavily depends on DNS for its security, hence security of NTP cannot be established in isolation from DNS. Unfortunately, as is demonstrated by the research community as well as attacks that were detected in the Internet, DNS is extremely vulnerable to DNS cache poisoning.

• We provide more information about this project at http://ntp-attack.sit.fraunhofer.de.

Organisation. We review Related Work in Section II. In Section III we explain how to exploit IP fragmentation for launching DNS cache poisoning attacks and discuss the hurdles which the attacker needs to overcome. In Section IV we explain the adaptations of the attack for attacking NTP: specifically, triggering the DNS request and then causing the victim NTP client to use the NTP server provided by the attacker. In Section V we evaluate the attack against different implementations of NTP clients. In Sections VII and VIII we provide our measurements of the attack surface of DNS and NTP servers as well as the DNS resolvers. In Section IX we provide recommendations for countermeasures. We conclude this work in Section X.

2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)
Accepted version © 2020 IEEE. https://doi.org/10.1109/DSN48063.2020.00043

II. Related work

NTP Security. Network Time Protocol (NTP) is one of the Internet's oldest protocols that was designed in the 80s. NTP has a long history of attacks. NTP was typically exploited as a reflector to launch Distributed Denial of Service (DDoS) attacks against victims in the Internet, [10], [11].

NTP can also be attacked via non-standard queries and via Config interface, [12]. Although the recent patches introduced to NTPv4.2.8p9 were meant to close the vulnerabilities, however, our measurements performed in this work indicate that at least 5% of NTP servers still have open Config.

The researchers also investigated the implications of shifting time on security of computer systems and applications and integrity of timing information [13], [14]. Additionally, evaluations were performed of on-path (Man-in-the-Middle) attacks for shifting time on NTP clients, along with analysis of the potential impact of the attacks, [15], [16].

Recently [1] demonstrated that it is theoretically possible to shift time with an off-path attacker, however, their attack assumes that the attacker can reduce the responses from NTP servers to a very low MTU of MTU=68, which is unrealistic, and hence the authors could only present a proof of concept attack in a lab setup. Furthermore, the assumption of [1] that the victim NTP clients support a specific fragments' reassembly strategy could not be verified in practice: it requires generating two fragments that overlap with the NTP response of the NTP server and overwrite a few locations in the NTP packet. The attack also requires that the client accepts the response with the bad time - in practice clients query a number of NTP servers and ignore servers that provide bad time.

Our work is similar to [1] as we also apply fragmentation in order to cause the victim NTP client accept an incorrect time in a response to its NTP request. However, in contrast to [1] we do not attack the time-responses from NTP server, but we target the DNS responses from the nameserver. Our attack is also more practical in that it does not require sending two overlapping fragments - which is difficult to match correctly in practice, but we need to send a single (second) fragment which contains the malicious payload and redirects the victim NTP client to query the "wrong" NTP servers. Specifically, instead of providing an incorrect time directly, we first redirect the victim NTP client to "query" the attacker's NTP servers, which in turn provide shifted time. Our attacks are also more effective: we do not attack a response from a single NTP server, but provide shifted responses from *all* the NTP servers that the client queried. This allows us to ensure that the client definitely accepts the shifted time.

In a recent work, NTP 'prime-the-pump' [17] used a Kiss of Death (KoD) packet to the victim NTP client, to cause it to slow down requests to that NTP server. In contrast, we abuse the rate limiting mechanism of NTP to cause the client to "forget" the server and replace it with an attacker controlled one via a malicious DNS record that we inject into a resolver's cache that the victim NTP client is using.

Although NTP support cryptographic authentication, [18] in practice NTP is generally not cryptographically protected [1].

There were also proposals for probing time from distributed locations to provide security against MitM attackers, [19], [20]. These proposals are not deployed in the Internet due to the significant changes that adoption requires. Recently, [2] proposed to generate NTP-servers redundancy by collecting a sufficiently large set of NTP servers through multiple DNS queries. Chronos was also proven to provide security against even strong MitM attackers. Nevertheless, as we show, exactly the multiple DNS queries make Chronos even more susceptible to our off-path attacks than the plain NTP.

DNS Security. DNS cache poisoning attacks were known since the 90s, and evaluated in the lab were considered mostly theoretical [21], [22]. In 2008 Kaminsky [23] demonstrated the first practical DNS cache poisoning attack. To enforce a query the attacker would request a random subdomain of the target victim domain. In the spoofed responses the attacker adds a new malicious nameserver for the victim domain, which is added to the cache if the attack succeeds. Following Kaminsky attack many systems were patched to support the best practices in [RFC5452], [24]. The main patches were challenge-response authentication based on randomisation of source port and transaction identifier (TXID) values in DNS packets, each of which is 16 bits.

After deployment of source and TXID randomisation off-path attacks became a theoretical threat. To launch successful cache poisoning attacks, the attackers have to be MitM, i.e., have to be able to see the queries.

However, in 2011 [9] demonstrated practical off-path DNS cache poisoning attacks that use fragmented IPv4 packets. The idea is to inject a spoofed fragment into IP defragmentation cache that is to be reassembled with the real fragment from the nameserver, and so to inject malicious payload into the DNS response. Fragmentation is a popularly exploited attack vector for attacks against different systems and protocols [1], [25]–[27].

III. DNS Poisoning via Fragments Replacement

The basic component that we use to launch a DNS cache poisoning attack is to manipulate a DNS response sent from the nameserver to the victim resolver via replacement of IPv4 fragments. We exploit the replacement of fragments to overwrite part of the payload in a real DNS response from the nameserver with malicious values. This allows the off-path attacker to inject malicious DNS records into the cache of a victim DNS resolver without having to guess the values in the challenge-response fields (port, TXID), [**?**], [28] of the DNS response (those are transmitted in the first fragment).

We next list the components needed for launching a DNS cache poisoning attack.

1) IP fragmentation: DNS responses typically do not exceed 1500 bytes and hence do not fragment. To cause the nameserver to send a fragmented DNS response the attacker sends to the nameserver a *ICMP Destination Unreachable Fragmentation Needed* error message (type 3, code 4) with DF bit set. The error tells the nameserver that the Maximum Transmission Unit (MTU) to the destination is smaller. Upon

2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)
Accepted version © 2020 IEEE. https://doi.org/10.1109/DSN48063.2020.00043

receiving the ICMP error the nameserver registers the MTU indicated in the ICMP error and sends the DNS response in IP fragments that do not exceed the MTU indicated in the ICMP error.

2) IP defragmentation cache poisoning: Assume that the nameserver sends a response in two fragments. The attacker generates a second fragment that is identical to the second fragment of the nameserver except for the new malicious records - IP addresses or nameserver records - that the attacker adds. These records map the NTP servers to hosts controlled by the attacker.

To cause the victim resolver to reassemble the spoofed second fragment with the first fragment of the nameserver the attacker has to ensure that both fragments contain the same source and destination IP addresses, the same IPID value[1], and that both are fragments (i.e., `more fragments` flag is set in the first fragment and is zero in the second (last) fragment). The fragments are then reassembled based on the offset values.

To predict the IPID that will be assigned by the nameserver to the response sent to the victim DNS resolver, the attacker sends a few queries to the nameserver. This allows to measure the current IPID value and the rate at which the value is incremented. The attacker uses that information to extrapolate the value that will be used in a DNS response that will be sent by the nameserver to the victim resolver. We use the same IPID prediction techniques as those developed earlier [9], [29]. This will be the IPID value that the attacker will set in its spoofed second fragment sent to the victim DNS resolver. Notice that if the rate at which the nameserver receives queries is high, and IPID increments are high, the attacker can send multiple fragments, to cover a set of possible IPID values. Most stringent operating systems, such as Windows and patched Linux, allow 100 and 64 identical fragments respectively, each with a different IPID value.

3) Calculating UDP checksum: If the spoofed second IP fragment is correctly constructed, it will be reassembled with the first IP fragment of the nameserver and the UDP payload will be passed on to the transport layer and then to the DNS software. These layers will perform subsequent checks, including the UDP checksum, the packet length, and correctness of the DNS records. The next challenge is matching the UDP checksum.

Since the off-path attacker modifies part of the payload of the original IP packet sent by the nameserver it has to ensure the UDP checksum matches the one of the original IP packet. The UDP checksum value resides in the first fragment in UDP header and cannot be altered by the attacker. The attacker needs to "fix" the checksum by adjusting some other two bytes in the second fragment.

UDP checksum calculation is performed as follows: a ones' complement sum is calculated on all the 16-bit values in the packet and then the ones' complement (i.e., invert all bits) is taken of that value to the checksum field. This means that with knowledge about the original second fragment f_2, the attacker can ensure a matching checksum by crafting the modified fragment f_2' so that the ones' complement sum $sum_1(f_2') = sum_1(f_2)$. This is possible for example by measuring the change in ones' complement sum between f_2 and the preliminary modified fragment ${f_2}^*$ and subtracting it from unimportant 16-bit values, therefore creating $f_2' = {f_2}^* - (sum_1({f_2}^*) - sum_1(f_2))$.

[1]IPID, [RFC6864], is used to identify all fragments belonging to the same original IP packet. IPID values can often be predicted.

IV. Redirection to Attacker's NTP Servers

In this section we explain under what conditions the attacker can trigger DNS requests for launching the DNS cache poisoning attack and injecting a spoofed DNS record mapping to a malicious NTP host. Then, how the attacker can cause the client to use the malicious record. We explain that at Boot time the victim NTP client will use the injected record directly since it does not have any other associations. At run-time, we demonstrate how to exploit rate limiting supported by NTP servers to cause the NTP client to break its associations to NTP servers in order to use the newly injected record of the attacker.

A. Attacking NTP during boot-time

The attack on NTP during boot-time is shown in Figure 2. After startup, the NTP client immediately queries it's DNS resolver to find NTP servers to associate to. If the attacker was able to poison the DNS resolver's cache before, the resolver will return the IP address (*6.6.6.6*) controlled by the attacker in the DNS response and the NTP client associates with the NTP server at that address, thus takes time from the attacker.

There are three ways to initiate the cache poisoning attack at boot-time: (1) by predicting the time of the query via side channels, (2) by causing other systems that use the same DNS resolver, such as Email or open resolvers, to issue the query, or (3) by periodically planting the spoofed (second) fragment every 30 seconds in the IP defragmentation cache of the victim resolver until the NTP client issues the query and the response is reassembled with the spoofed fragment waiting in the defragmentation cache. We measured frag. reassembly timeout of 60 to 120 seconds on Windows distributions and 30 seconds on Linux; [RFC2460] specifies the reassembly timeout of 60 seconds. This approach requires a low attack volume which can be completed with only one low bandwidth attacking host. Furthermore, the TTL of `pool.ntp.org` A record is only 150 sec. Namely, the resolver will frequently issue queries for `pool.ntp.org`, at most every 150 sec. This means that the attacker needs to maintain the spoofed fragment in the IP defrag. cache for at most 150 sec, which in the worst case requires 150/30 = 5 spoofed (second) fragments per attack. Referring to option (2), in Section VIII-B3 we show that DNS resolvers are indeed often shared between different systems, so that the system used to poison the cache can be different from the system targeted by the attack.

B. Attacking NTP during run-time

During run-time, the NTP client already has a list of associations with NTP servers which addresses are resolved.

2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)
Accepted version © 2020 IEEE. https://doi.org/10.1109/DSN48063.2020.00043

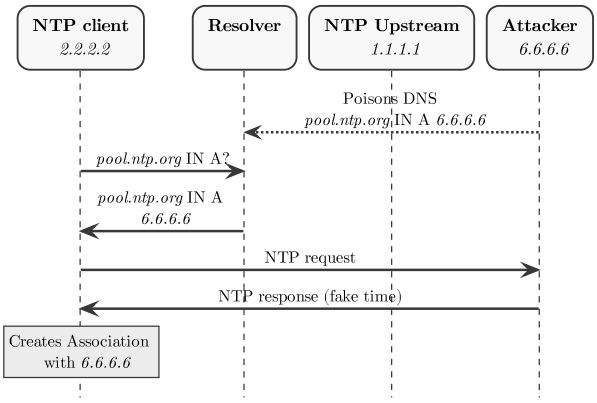


Fig. 2. Boot-time attack on NTP.

This means that poisoning the resolver's cache during run-time will not directly lead the NTP client to associate to the attacker's NTP server. However, we show how the attacker can still trigger a DNS query and make the client connect to the attacker controlled server. This attack is shown in Figure 3. First, the attacker needs to remove the existing association to the server at 1.1.1.1, which will cause the client to replace the server with another one by querying the DNS resolver and therefore, trigger the DNS query.

1) Removing the existing association: To remove the existing association with the NTP server at 1.1.1.1, the attacker needs to disrupt the communication between the NTP client and server. This way, the NTP server will appear unreachable to the NTP client, and therefore lead the client to replace the unreachable server via DNS. While the easiest way to disrupt this communication would be a Denial-of-Service attack on the NTP server, this method may require a huge amount of resources and won't go unnoticed for long. Instead, we show how to facilitate the rate-limiting mechanism used by many NTP servers to make the NTP server appear unreachable to the client even though it is not.

2) Exploiting rate limiting of NTP: Many public NTP servers in the internet enable rate-limiting to limit the server load caused by defective NTP clients or the impact on reflection attacks. Similar to DNS, NTP servers can be used as reflectors of Denial-of-Service attacks which use spoofed source IP addresses to hide the attacker's real address. The rate-limiting mechanism in the default implementation of NTP works by monitoring the amount of NTP queries from a single IP and denies service, ie. stop answering requests for some time, if the time-span between two NTP queries is too low. In previous works, Malhotra et al. [1] already showed how this mechanism can be used to force an NTP client to take time from another NTP server by triggering 'Kiss-o-death' packets, which are NTP responses generated to indicate that the client should reduce it's query interval and are generated just before the server starts rate-limiting the NTP client. Instead of relying on the client to reduce it's query interval, our attack relies on the server-side rate-limiting which will cause the server to stop replying to NTP requests at all. In Figure 3, this is shown as the attacker keeps sending spoofed NTP requests with the NTP

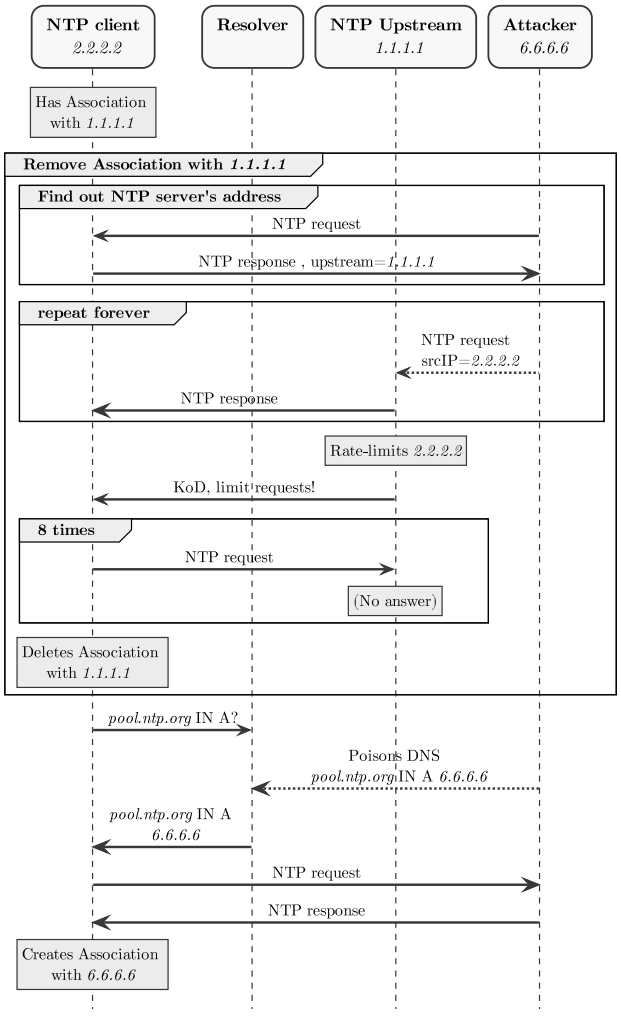


Fig. 3. Run-time attack on NTP.

client's source IP address so that the server finally rate-limits the client and stops answering any subsequent requests. After that, the NTP client will keep trying to query the NTP server several times after it gives up and marks the association as unusable. Depending on the client, it will then search for a new server immediately via DNS or switch to another one if it uses multiple associations simultaneously. Consequently, to succeed with the attack the attacker may need to remove multiple existing associations of the NTP client using the method presented here.

To find out the address of the NTP server(s) used by the client, which is needed to create the spoofed NTP requests, the attacker has multiple options:

a) Generating a list of IP addresses: The attacker queries the DNS system for the domain name configured in the NTP client and create a list of possible upstream NTP server addresses. For the `pool.ntp.org` domain this list consists of 2000 to 3000 servers, which is small enough for the attacker to successfully disrupt existing associations with any of these servers by abusing NTP's rate limiting mechanism.

b) NTP Upstream server address leakage: If the target NTP client is also acting as a NTP server (eg. NTPd at

2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)
Accepted version © 2020 IEEE. https://doi.org/10.1109/DSN48063.2020.00043

default configuration), the attacker can use the 'id' field in the NTP response to identify the currently used upstream server's IP address. In this case, the attacker only learns one server address at a time and needs to wait until the upstream server is changed until it learns the next server's IP address. This is the variant shown in Figure 3.

c) Querying the servers configuration interface: Some servers running NTPd expose the NTP configuration interface into the Internet which can be used to learn both, the configured DNS hostnames as well as all currently used upstream server IP addresses. Furthermore, [12] showed that this interface can also be exploited for other attacks. Nevertheless, we found that 5.3% of the NTP servers in the `pool.ntp.org` pool still respond to configuration queries during our measurements in section VII-A.

3) Conducting the cache poisoning attack: To place the malicious DNS record in the vicitm resolver's cache, the attacker can employ the same methods as discussed in Section IV-A. Predicting the timing of the query in this case is easier as the query is triggered by the attacker himself when the client re-queries its own resolver after it has deleted the existing association. However, utilising other systems to place the malicious record in the cache might still be easier as those systems might allow the attacker to avoid or override cached records and evict records from the cache by selecting a custom query domain which is not possible with NTP itself as the query domain cannot by influenced by the attacker.

V. Evaluation of Plain NTP Clients

In this section we evaluate which NTP clients are vulnerable to boot-time attacks and which are vulnerable to run-time attacks. We then provide an analysis of successful time-shifting attacks by causing the victim NTP client to query the malicious NTP server of the attacker and accept from it a shifted time.

A. DNS Lookup Behaviour of NTP Implementations

We evaluate applicability of our attacks to popular NTP clients. Our evaluation is performed in both attack scenarios presented in Section IV: at boot-time and at run-time. The evaluation is based on analysis of DNS lookup behaviour of the NTP implementations using: source-code and lab evaluation of the run-time attacks against the NTP clients with DNS poisoning. The clients support either NTP or Simple NTP (SNTP). The main difference between those protocols is that SNTP uses a single NTP server whereas NTP can use multiple NTP servers. In the second column (from left) in Table I we list the different NTP client implementations. In the third column we list the distribution of different NTP clients and NTP servers in `pool.ntp.org`. In the last two columns we report which attack setup (boot-time or run-time) applies to which NTP client implementation. As listed, all NTP implementations are vulnerable to a boot-time-attack and 4 NTP implementations are vulnerable to a run-time attack. These 4 out of 7 NTP implementations make up at least 45% of clients of `pool.ntp.org`.

TABLE I
Attack scenarios for popular NTP clients.

	Client	*pool.ntp.org* Usage [30]	Attacks	
			boot-time	run-time
NTP	NTPd	26.4%	✓	✓
	openntpd	4.4%	✓	✗
	chrony	4.8%	✓	✓
SNTP	ntpdate	20.0%	✓	n/a
	Android	14.0%	✓	✓
	ntpclient	1.2%	✓	✗
	systemd	*not listed*	✓	✓

1) Boot-time attacks: Our analysis shows that all types of NTP clients are vulnerable to cache poisoning at boot-time when performing DNS lookups, as there is no viable mitigation mechanism against the attack itself. Some clients enforce limits to how much or how fast time can be shifted from the local system clock, however, these limits are often not used in the default configuration or are explicitly not enforced at boot-time, because of the assumption that the system clock may be way off at the time the system starts because of dead Real-time-clock batteries etc. The only exception to this we saw for traditional NTP clients is the openntpd client, which has the option to partially authenticate the time set via using the `Date`-header of a HTTP response gathered using a TLS-protected connection to a configurable webserver. This is not enabled by default.

2) Run-time attacks: Next, we show and evaluate the vulnerability against run-time attacks of those NTP implementations. Using documentation and lab tests, we discovered that only 4 of our clients are vulnerable to run-time attacks: NTPd, chrony, Android and systemd-timesyncd. openntpd and ntpclient do not support DNS queries during run-time at all, so hindering communication with the used servers will just disable time synchronisation until the client is restarted. A special case is ntpdate, which is a command-line utility which only synchronises time once and then exits, so that the run-time attack does not apply here as well. However, this utility is often used as part of a regularly run cronjob, so boot-time attacks against this client can be done any time the program is invoked. We practically evaluated all potentially vulnerable clients except Android, which was discarded from the list because we found no device which actually used the built-in NTP client, all available devices used the mobile network to synchronise time instead. However, from source code reviews of code, we can defer that since the built-in NTP client is always invoked by hostname[2], DNS lookups must be triggered every NTP query if not answered from a local DNS cache.

For the other 3 clients we performed DNS poisoning attacks using a DNS resolver reconfigured after the clients had done their initial boot-time DNS lookups. We then used a custom application to send spoofed NTP mode 3 queries to all 4 of our labs NTP servers to trigger the rate-limiting mechanism and stop communication with our client. After some time, all 3 clients re-queried the DNS resolver to find new servers and switched over to our attacker-NTP server which provided time shifted by -500 seconds. All 3 clients started to adjust

[2]Source code: https://android.googlesource.com/platform/frameworks/base/+/master/core/java/android/util/NtpTrustedTime.java

2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)
Accepted version © 2020 IEEE. https://doi.org/10.1109/DSN48063.2020.00043

TABLE II
RUN-TIME ATTACK DURATION AGAINST DIFFERENT CLIENTS

Client	Scenario	Attack duration
NTPd	P_2	47 minutes
NTPd	P_1	17 minutes
openntpd	P_1	84 minutes
chrony	P_1	57 minutes

their system clocks after some time, the exact timing values are listed in Table II. For ntpd, we did this evaluation two times, one time assuming the attacker knows the upstream NTP servers' addresses in advance (Scenario P_1) and one time where the attacker discovers these addresses one-at-a-time by querying the client via mode NTP 3 queries (Scenario P_2).

B. Probability Analysis of Run-time Attack

To successfully shift time on a NTP client, an attacker needs to replace a majority of NTP associations this client uses. In this section we give a probabilistic analysis of whether an attacker is able to do this on a running NTP client, depending on his knowledge of the used upstream server's IP addresses.

1) Scenario 1: In this scenario, we have a client which has a list of upstream servers, which we remove one-after-another by attacking the connection between client and server. This is the case if the attacker cannot know the upstream servers' addresses upfront but instead discovers it via querying the client as shown in Figure 3. We assume that the client has chosen it's upstream servers randomly from the NTP pool, so the probability that each of the servers in the list does rate limiting is p_{rate}. Therefore the probability of successfully removing n servers by exploiting the server's rate-limiting mechanism is $P_1(n) = p_{rate}^n$

2) Scenario 2: In the second scenario, we assume that the attacker has knowledge about the clients upstream servers upfront and therefore can choose the servers which shall be removed from the list. This can be achieved when the server leaks configuration information or simply by attacking potential connections to all servers in `pool.ntp.org`. Assuming a server list of size m, we calculate the probability of a successful attack as the probability that n or more servers out of m do rate-limiting:

$$P_2(m,n) = \sum_{i=n}^{m} \overbrace{\underbrace{\binom{m}{i}}_{\substack{\text{Number of possibilities to choose} \\ \text{i out of m servers}}} \cdot p_{rate}^{i} \cdot p_{\overline{rate}}^{m-i}}^{\text{Probability that exactly i out of m servers do rate limiting}}$$

If $n = m$, this is the same as p_{rate}^n. As said, an attacker needs to replace a majority of the used upstream servers to successfully shift time, therefore we have $n \geq \lceil \frac{m}{2} \rceil$. For $p_{rate} = 38\%$ as indicated by a scan of all servers in `pool.ntp.org` described in Section VII-A we list the probabilities to conduct a successful attack in Table III.

3) Common client configurations: What probabilistic scenario applies to what client depends in its configuration and implementation behaviour. For example, ntpd in most default configurations is configured to have 4 persistent pool associations which are used to create server associations

TABLE III
PROBABILITIES THAT AN NTP CLIENT IS IN A VULNERABLE STATE DEPENDING ON THE NUMBER OF USED ASSOCIATIONS m.

m	$n = \max(\lceil \frac{m}{2} \rceil, m-2)$	$P_1(n)$	$P_2(m,n)$
1	1	38.0%	38.0%
2	2	14.4%	14.4%
3	2	14.4%	32.4%
4	3	5.5%	15.7%
5	3	5.5%	28.4%
6	**4**	**2.1%**	**15.3%**
7	5	0.8%	7.8%
8	6	0.3%	3.9%
9	7	0.1%	1.8%

using DNS lookups when needed. The maximum long-time number of associations is 10 (`NTP_MAXCLOCK`), so together with the pool-associations, a default ntpd client will have $m = 6$ NTP upstream servers and will only query for new associations during run-time if this number is reduced below 3 (`NTP_MINCLOCK`). This means the attacker needs to remove $n = m - 2 = 4$ servers to trigger a DNS query. Notably, neither `NTP_MAXCLOCK` nor `NTP_MINCLOCK` can be configured, the only variable which can be changed is the number of configured pool associations, which effectively *lowers* the amount of active server associations as they count towards the `NTP_MAXCLOCK` limit. The probability for a run-time attack against ntpd is therefore $P_1(4)$ or $P_2(6,4)$.

The default configuration for systemd-timesyncd includes only a single ntp pool domain. As systemd-timesyncd is a SNTP client, it holds only a single association to one NTP server but caches the list of servers from the last DNS query, which by default contains 3 more server addresses additional to the one used. As these servers will be queried before a DNS query is triggered, the attacker is required to attack associations to all of them, giving a run-time attack probability of $P_1(4)$, since all of these 4 servers need to be removed. If more domains are configured, `systemd-timesyncd` will round-robin through all of them but since no responses are cached except for the first domain, this will still trigger a DNS query.

VI. EVALUATION OF CHRONOS NTP CLIENT

Chronos [2] is a special NTP client proposed to prevent time-shifting attacks by sampling time from a bigger set of NTP servers than traditional NTP implementations. It is also available as an internet standard draft document [31]. While Chronos uses a proven secure method for setting the time using a subset of a big pool of NTP nameservers, it is not stated exactly how this server pool is generated, which is important since the security guarantees of Chronos vanishes if the attacker is able to control more than $\frac{2}{3}$ of the NTP servers in the pool. The original proposal describes the pool generation works by querying the `pool.ntp.org` domain every hour for 24 hours and use the union of all gathered IP-addresses as the server pool. Because the nameservers of `pool.ntp.org` normally give 4 IP-addresses per DNS query, this results in a maximum of 96 servers. However, since the DNS namserver controls the servers Chronos considers during its algorithm, an attacker attacking the DNS instead of NTP can feed many

2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)
Accepted version © 2020 IEEE. https://doi.org/10.1109/DSN48063.2020.00043

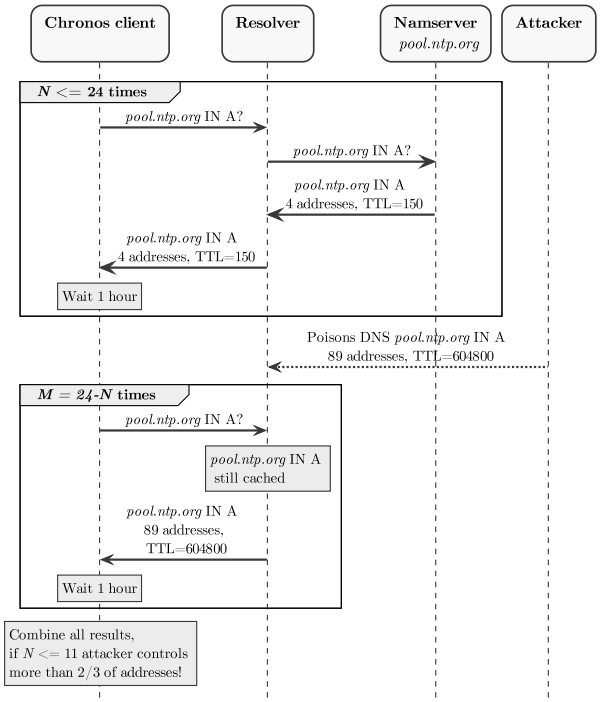


Fig. 4. DNS poisoning attack on chronos.

of its own IP-Addresses into Chronos' server pool, thereby overwhelming the honest servers. In the following we point out two weaknesses of the proposed server-pool generation mechanism and after that, describe a potential DNS poisoning attack on Chronos.

A. Predictability of the hourly DNS lookup

While not explicitly stated in the Chronos proposal, the mechanism of querying the DNS hourly may give an attacker the possibility of knowing the DNS queries' timing, since it is done hourly for 24 hours. If the attacker can predict queries because they are only done at full-hour, or because he was able to observe previous queries, this facilitates off-path DNS poisoning by removing the necessity to trigger the query himself.

B. Combining the results of all DNS responses

The Chronos proposal queries the DNS multiple times to gather a big enough pool of servers. This queries are spread over 24 hours, mainly to prevent gathering the same NTP servers multiple times because of caching or round-robin mechanisms at the nameservers of `pool.ntp.org`. However, Chronos does not make any efforts to prevent even a single malicious DNS response to influence the outcome of the whole pool-generation process, neither by checking the values of the response's Time-to-live value or the number of addresses given.

C. DNS poisoning attack against Chronos

Using the second of these two discovered weaknesses, we now present an enhanced DNS poisoning attack against Chronos which requires poisoning the DNS only once in this 24-hour period, shown in Figure 4. First we allow that the client has already done a number of N of the 24 DNS requests before the attack starts, this could have been the result of the attacker failing to perform the attack a number of times. These queries where answered by the nameservers for `pool.ntp.org`, which give 4 IP addresses per DNS request. This results in a number of $4N$ honest nameservers already in the generated server pool before the attack starts. Then, the attacker poisons the DNS resolver with as many addresses he can fit into a single DNS response (up to 89 for a single, non-fragmented UDP response) and sets the TTL to a value bigger than 24 hours, to cause any subsequent DNS requests to be answered from cache instead of the original nameserver. With this attack, the attacker adds 89 attacker-controlled IP addresses to the server pool and effectively ends the server pool-generation algorithm since subsequent DNS queries will always be answered with the cached, attacker-given records from the DNS resolver. The pool does now contain $4N$ honest and 89 malicious servers. To fulfil the requirement on a successful attack on Chronos, the attacker needs to control $\frac{2}{3}$ of the servers in the pool, so we get the maximum number for N from $\frac{2}{3} \cdot (89 + 4N) \leq 89$ and conclude that the attacker succeeds with his attack if the cache poisoning succeeds before or during the 12th DNS transaction ($N <= 11$). This means that the chances of a successful attack against Chronos are actually *higher* than against a traditional NTP client during boot-time, since the attacker effectively has 12 tries in 24 hours to succeed with the attack.

VII. Measuring NTP Servers Attack Surface

In this section we measure the presence of server-side properties required for our attack to work: Rate-limiting support in NTP servers and fragmentation support in the NTP domain's nameservers.

A. Rate Limiting of `pool.ntp.org` NTP Servers

For successfully attacking NTP during run-time, the attacker needs to remove enough existing associations using NTP's rate-limiting mechanism. For this to work, the server actually needs to use rate-limiting. To find out how many servers do this, we conduct a study of all servers in *pool.ntp.org*. We first query the *pool.ntp.org* namservers multiple times for all existing NTP country zones (<*countrycode*>.*pool.ntp.org*) and gather the union of all result IP addresses. This way we gathered a list of 2432 NTP servers. We then query each of those NTP servers 64 times, once per second and check for (1) KoD-packet from the NTP server or (2) the server stops responding after some queries, which indicates rate-limiting. To prevent false-positives because of packet loss as well as because some servers will answer a small fraction of queries, even during the client is rate-limited, we do this by taking the number of responses during the first and second half of the test and mark a server as rate-limiting if the first half has more than 8 additional responses compared to the second half. Using this method, we find that 780 (33%) servers send KoD-messages and 904 (38%) servers stopped sending responses during the

2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)
Accepted version © 2020 IEEE. https://doi.org/10.1109/DSN48063.2020.00043

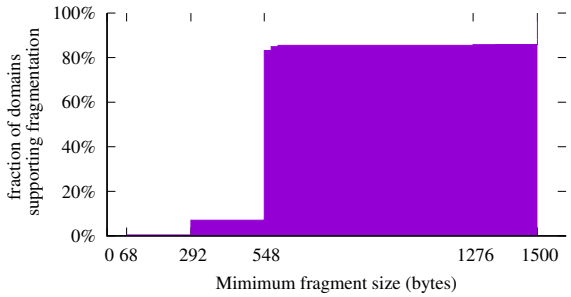


Fig. 5. Cumulative distribution of fragment sizes emitted by popular 1M domains which do not support DNSSEC.

second half of our test. We therefore conclude that around 38% of servers in pool.ntp.org do rate-limiting, as sending KoD's is a clear indicator, but not every server sends a KoD message before rate-limiting the client.

B. Fragmentation support of NTP Nameservers

For DNS poisoning via IP fragment injection to work, nameservers have to support path MTU discovery (PMTUD) and fragment DNS responses on arrival of ICMP fragmentation needed messages. We tested this behaviour on nameservers of `pool.ntp.org` and found that 16 out of 30 DNS nameservers fragment packets to fragments below 548 bytes on receiving ICMP fragmentation needed. None of those 30 namesevers supports DNSSEC for the `pool.ntp.org` domain.

Furthermore, we evaluated PMTUD support of popular 1M domains including different NTP domains, not only those under `pool.ntp.org`. In this measurement, we gathered data of 877,071 nameservers considering DNSSEC, fragmentation support and size of fragments emitted after arrival of ICMP fragmentation needed messages. We found that 7.66% of domains do not support DNSSEC but emit fragmented DNS responses which makes them vulnerable to DNS cache-poisoning attacks via injection of IP fragments. Considering the minimum fragment size emitted by those domain's nameservers, see Fig. 5, we found that most of those nameservers (83.2%) fragment DNS responses down to a size of 548 bytes and 7.05% even down to 292 bytes. This means that clients who use those NTP servers are vulnerable to the attack.

Whether the required response size for the DNS message to be fragmented is reached depends on the response generated by the nameserver, but attackers can employ several techniques to increase the size of a message. For example, if the domain which is queried is (partially) under control of the attacker, attackers may trigger queries to exceptionally long sub-domains under the victim domain which is under attack. This is the case if a third-party system is used to trigger the DNS query as discussed in Section IV-A.

VIII. Measuring DNS Resolvers Attack Surface

We measured DNS resolvers using an ad network and open DNS resolvers. In what follows we explain the study and report on our measurement results of vulnerable clients.

TABLE IV
`pool.ntp.org` CACHING STATE IN TESTED OPEN RESOLVERS

Query	Cached in	Absolute Resolvers	
		Cached	Not Cached
`pool.ntp.org IN NS`	58.28%	376,594	269,618
`pool.ntp.org IN A`	69.41%	448,521	197,691
`0.pool.ntp.org IN A`	63.92%	413,046	233,166
`1.pool.ntp.org IN A`	61.28%	395,971	250,241
`2.pool.ntp.org IN A`	61.55%	397,751	248,461
`3.pool.ntp.org IN A`	58.58%	378,525	267,687

A. Vulnerable Open DNS Resolvers

To measure the client-side attack surface of our attack, we conduct a study of all Open DNS resolvers in the Censys DNS responder dataset [32]. For attackers in a MitM position between DNS resolver and namserver, all clients are vulnerable to our attack, because the `pool.ntp.org` namservers do not offer DNSSEC support. However, for off-path attackers without the possibility to get in path, e.g., by BGP-hijacking, attackers can still try to to fragmentation based DNS poisoning [9]. This requires the resolver to accept fragmented DNS responses, a property which we will explore here.

1) Finding resolvers used by NTP clients: First, we need to find those open resolvers which are used by NTP clients. We do this by employing a technique which can detect if a DNS resolver has cached certain DNS records by sending DNS queries with the Recursion Desired (RD) Bit set to zero [33]. This is done with all records listed in Table IV. We conclude that a resolver is used by NTP clients if any of those records is cached, since these domains generally serve no other purpose than NTP server discovery.

To ensure the cache testing method works as expected, for each resolver, we test if they respect the RD bit by first, querying a known non-cached domain and second, querying a known-cached domain which has been placed into the cache by a previously issued query with RD=1.

Using this method we probed 1,583,045 resolvers for cached records (1,674,103 resolvers in the dataset did not respond to any query), and verified the technique to be working 646,212 times. The results are shown in Table IV. Out of the measured resolvers, 448,521 (69%) resolvers had the `A` record for `pool.ntp.org` cached. To further verify that our cache testing method indeed works, we provide TTL data for all these cached records in Figure 6 and show that the values are uniformly distributed, as one would expect for cached records.

2) Measuring fragmentation support: We then perform a generic scan of open resolvers for fragmentation acceptance by querying a purposely created domain we control, where the nameserver always responds to DNS requests with fragmented packets, even if the size is way below the maximum MTU of the path to the tested resolver. Out of the resolvers which had any of the `pool.ntp.org` records cached, 32% accepted fragmented DNS responses and 68% did not, which is roughly the same distribution as over the whole population of open resolvers (31% fragmented DNS response acceptance).

2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)
Accepted version © 2020 IEEE. https://doi.org/10.1109/DSN48063.2020.00043

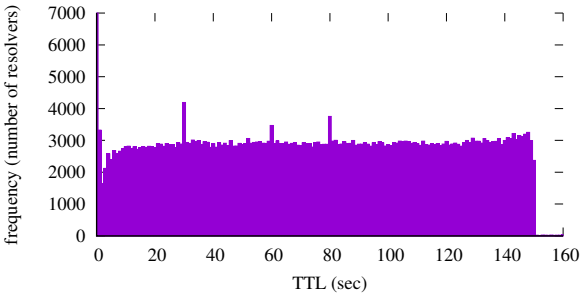


Fig. 6. TTL values of cached NTP pool records in open resolvers

B. Measurement via Ad-Network of Vulnerable DNS Resolvers

We conducted a study of resolvers used by Web clients in the Internet, as many Web clients will also run NTP clients. For this purpose, we created an automated test website which was served by an advertisement network. This study is designed to test fragmentation support of those resolvers which is a requirement for off-path DNS-poisoning.

1) Testing fragmentation support through Javascript: To conduct the study, we used an ad network which serves our test website as an 'popunder'-ad. This causes our test website to be loaded in a background browser window or tab. The test website then uses a script to load a series of images from different tests such that each image is to verify the successful DNS lookup of its corresponding domain, similar to the study done by Lian et al. in 2013 to identify DNSSEC deployment [34]. We conducted the following tests where T represents a randomly generated token to differentiate clients and avoid caching. Moreover, our nameserver fragmented the responses irrespective of any path-MTU-discovery results:

- `T.baseline.domain.com` - normal A record
- `T.ftiny.domain.com` - response fragmented to 68 bytes
- `T.fsmall.domain.com` - response fragmented to 296 bytes
- `T.fmedium.domain.com` - response fragmented to 580 bytes
- `T.fbig.domain.com` - response fragmented to 1280 bytes
- `sigfail.domain.com` - incorrectly DNSSEC-signed record
- `sigright.domain.com` - correctly DNSSEC-signed record

We used a customised nameserver for our domain to guarantee that responses for queries about the {ftiny, fsmall, fmedium, fbig} domains are always in at least two fragments of their corresponding MTU size. After the test is finished, the javascript records if the images are loaded via onsucess()/onerror() event handlers and finally push back the results to the server. To filter out invalid results, we removed any results where the client had the page open for less than 30 seconds, to prevent errors because of timeouts at the resolver level. We also remove results which failed the *baseline* or *sigright tests* as this tests are designed as baseline tests to verify that the testing method works.

2) Results: In our first run of the study we published our test website using the ad network without targeting any specific regions. In this run, which we call dataset 1, we got valid results from 5847 unique clients. Because our first run gave only 120 results for clients in the North America region, we conducted a second run only targeting clients in the United States and Canada, which we call dataset 2.

Results for both datasets are displayed in Table V, which group clients by region as well as device type (determined using the HTTP user-agent header). Furthermore, by correlating the randomised T values with nameserver logs, we figured that 791 clients uses resolver operated by Google LLC, all of which filter fragments of sizes below 'big'. The column 'Accepts any fragment size' shows results of clients which used resolvers accepting at least one fragmented response. Depending on the attacker's ability to trigger a large enough fragmented response at the nameserver, these clients are vulnerable to off-path defragmentation DNS poisoning.

Fragment acceptance does not change dramatically with the fragment size, many resolvers (64%) accept even the tiniest possible fragment size with MTU=68, this number increases to 77% for medium fragments and 86% for big fragments. This distribution across various fragment sizes behaves similar in for all regions and device groups. DNSSEC validation ranges between 19.14% and 28.94%, a meaningful increase over the 3% measured by Lian et al. in 2013 [34].

3) Finding shared DNS resolvers: The attacker can utilise different systems for initiating the cache poisoning attack and injecting a malicious record into the victim DNS resolver's cache, and is not limited to attacking the query issued by the victim NTP client. Popular systems present in almost any networks are Email servers or web clients/proxies; for instance, Email issues queries to DNS resolver upon receipt of new Emails for performing domain-based anti-spam validation. We perform an Internet scale measurement for resolvers which are used by Email and web as well as by NTP clients. Then we cause the queries via Email or web clients for attacking NTP.

For web clients we leverage an ad-network causing the clients to issue queries through their DNS resolvers to our nameserver and for Email we send Email messages to the measured domains, which in turn cause the DNS resolvers to issue queries to our nameservers. First, we sent direct DNS queries to all resolvers to figure out if they are perhaps open resolvers. Secondly, we conduct a small portscan for SMTP servers in the /24 networks of all resolvers and web clients to find SMTP servers share the same DNS resolver. We found 6416 SMTP servers for all of which we sent test-emails causing bounces to our own nameserver, which lets us determine the DNS resolver used by the SMTP server. Finally we overlap the list of resolvers used by SMTP servers with the list of resolvers used by ad clients. Out of 18,668 resolvers used by web clients:

- 16,088 (86.2%) are only used by web clients,
- 2,116 (11.3%) are used by web clients and SMTP servers,
- 426 (2.3%) are open resolvers,
- 38 (0.2%) are open and used by both web and SMTP.

Combining SMTP and open resolver results, we find that an attacker could trigger queries for at least 2580 (13.8%) of resolvers using either SMTP or direct queries. This should be seen as a lower bound, because our very imperfect method of finding the SMTP servers used to trigger the queries. If such servers exist, but are located outside of the /24 network ranges

2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)
Accepted version © 2020 IEEE. https://doi.org/10.1109/DSN48063.2020.00043

TABLE V
RESULTS OF CLIENT RESOLVER STUDY USING ADS

	Accepts tiny fragments (68 B)		Accepts any fragment size		total	data set
Asia	1845	58.22%	2863	90.34%	3169	1
Africa	222	73.27%	290	95.71%	303	1
Europe	1010	72.66%	1277	91.87%	1390	1
Northern America	1352	58.43%	1757	75.93%	2314	2
Latin America	572	68.26%	759	90.57%	838	1
ALL	3742	64.00%	5320	90.99%	5847	1
Without Google	3439	68.02%	4555	90.09%	5056	1
PC	1664	60.80%	2447	89.40%	2737	1
Mobile,Tablet	2077	66.83%	2871	92.37%	3108	1

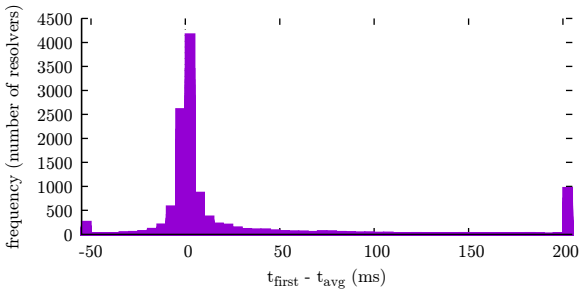


Fig. 7. Distribution of latency difference when querying open resolvers for `pool.ntp.org IN NS`. Values below -50 ms and above 200 ms are summed up on the sides.

we scanned, they are not considered in this study.

To find web clients' resolvers who also serve NTP clients, we develop another an additional cache-testing method. This method works by using the timing side-channel which is exposed by the latency of a query: If a record is not cached by the resolver, subsequent queries for the same domain should be faster then the first query because they can be answered from cache instead of asking the remote nameserver(s). This can be automatically tested by comparing the latency difference t_{first} of the first query and the average t_{avg} of subsequent queries to a threshold T and call the record cached, if $t_{first} - t_{avg} < T$. However, because we cannot control other variables which have an impact of the query latency, such as RTT to the tested resolver and the fact that some resolvers may have NS-records for parent zones like $ntp.org$ cached and others have not, there is no obvious value to choose T.

To find out if the method works, we executed it on the open resolver dataset first. We find that there is no obviously spot-able distribution of the timing latency $t_{first} - t_{avg}$ into two groups (cached and non-cached) and therefore no way to reasonably choose a value for T.

We conduct that executing such a test successfully requires further evaluation and multiple test passes including eviction of the cached records to control the variances in round-trip-time and other factors influencing the timing of the query. As this may substantially reduce the performance of affected resolvers and Web clients, we resigned from testing caching status of `pool.ntp.org` against web clients' resolvers and instead assume that many resolvers in this dataset are indeed used by NTP clients, as NTP clients are part of most modern operating systems.

IX. COUNTERMEASURES AND MITIGATIONS

In this section, we discuss the state of security of DNS and NTP. Essentially currently both are vulnerable and the dependency of NTP on DNS makes our off-path attacks possible. As an immediate countermeasure we recommend not to use DNS for NTP and instead to use a list of static IP addresses for NTP servers or to deploy distributed proposals like [35].

DNS Security. DNSSEC could make the cache poisoning attacks practically impossible. However, studies showed less than 10% of the DNS resolvers validate DNS responses, [34], [36], [37]. Our own validation with Ad-Net showed that DNSSEC validation depends on the geo-location and ranges between 19.14% and 28.94%. On the other hand, only about 1% of the domains are signed with DNSSEC and in our measurement we found *only one* signed NTP domain `time.cloudflare.com`, so even if the resolvers performed strict validation this would currently not help. Furthermore, recent studies found problems in DNSSEC keys generation making many signed domains vulnerable despite DNSSEC, [38]–[40].

NTP Security. Different recommendations for securing NTP were proposed, such as TLS for NTP (NTS) [41], or Roughtime [42] with proof of misbehaviour (wrong time) to report bad servers. However, none is deployed or used, mostly due to the significant changes to the NTP ecosystem that they require, or assumptions which cannot be fulfilled in practice.

X. CONCLUSIONS

Our attacks demonstrate that the NTP ecosystem is still vulnerable to off-path attacks. In this work we improve over the attacks in [1] by demonstrating more effective and practical attacks. We show that, as long as DNS is insecure, our attacks apply even to the security enhanced NTP client with Chronos [2]. Launching off-path time-shifting attacks against NTP is challenging: (1) the attacker must trigger a query or be able to predict the timing when the query is triggered and (2) once poisoning is successful, the attacker needs to cause the NTP client to use the new malicious mapping. We developed techniques which allow to handle both these challenges and performed Internet wide measurements to show the applicability of our methodologies.

Although the vulnerabilities that we exploit are not in NTP, our attacks demonstrate the risks of building security on insecure Internet foundations, and the risks of analysing security in isolated or ideal environment which does not reflect the situation on the real Internet.

REFERENCES

[1] A. Malhotra and S. Goldberg, "Attacking NTP's Authenticated Broadcast Mode," *ACM SIGCOMM Computer Communication Review*, vol. 46, no. 1, pp. 12–17, May 2016. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2935634.2935637

[2] O. Deutsch, N. R. Schiff, D. Dolev, and M. Schapira, "Preventing (Network) Time Travel with Chronos," in *Proceedings 2018 Network and Distributed System Security Symposium*, San Diego, CA, 2018.

2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)
Accepted version © 2020 IEEE. https://doi.org/10.1109/DSN48063.2020.00043

[3] N. S. Rozen, D. Dolev, T. Mizrahi, and M. Schapira, "A Secure Selection and Filtering Mechanism for the Network Time Protocol." [Online]. Available: https://tools.ietf.org/html/draft-schiff-ntp-chronos-03

[4] H. Ballani, P. Francis, and X. Zhang, "A study of prefix hijacking and interception in the internet," in *SIGCOMM*. ACM, 2007, pp. 265–276.

[5] K. R. B. Butler, T. R. Farley, P. D. McDaniel, and J. Rexford, "A survey of BGP security issues and solutions," *Proceedings of the IEEE*, vol. 98, no. 1, pp. 100–122, 2010.

[6] G. Huston, M. Rossi, and G. J. Armitage, "Securing BGP - A literature survey," *IEEE Communications Surveys and Tutorials*, vol. 13, no. 2, pp. 199–222, 2011.

[7] DYN, "Pakistan hijacks youtube," 24 February 2008, https://dyn.com/blog/pakistan-hijacks-youtube-1/.

[8] J. Cowie, "China's 18-minute mystery," 18 November 2010, https://dyn.com/blog/chinas-18-minute-mystery/.

[9] A. Herzberg and H. Shulman, "Fragmentation Considered Poisonous: or one-domain-to-rule-them-all.org," in *IEEE CNS 2013. The Conference on Communications and Network Security, Washington, D.C., U.S.* IEEE, October 2013.

[10] J. Czyz, M. Kallitsis, M. Gharaibeh, C. Papadopoulos, M. Bailey, and M. Karir, "Taming the 800 pound gorilla: The rise and decline of ntp ddos attacks," in *Proceedings of the 2014 Conference on Internet Measurement Conference*. ACM, 2014, pp. 435–448.

[11] C. Rossow, "Amplification Hell: Revisiting Network Protocols for DDoS Abuse," in *Proceedings of the Network and Distributed System Security (NDSS) Symposium*, February 2014.

[12] A. Malhotra, M. Van Gundy, M. Varia, H. Kennedy, J. Gardner, and S. Goldberg, "The Security of NTP's Datagram Protocol," in *Financial Cryptography and Data Security*, A. Kiayias, Ed. Cham: Springer International Publishing, 2017, vol. 10322, pp. 405–423.

[13] D. L. Mills, *Computer network time synchronization: the network time protocol on earth and in space*. CRC press, 2016.

[14] T. Mizrahi, "Security requirements of time protocols in packet switched networks," Internet Requests for Comments, RFC Editor, RFC 7384, October 2014. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7384.txt

[15] J. Selvi, "Bypassing http strict transport security," *Black Hat Europe*, 2014.

[16] ——, "Breaking ssl using time synchronisation attacks," *DEF CON Hacking Conference*, 2015.

[17] A. Malhotra, I. E. Cohen, E. Brakke, and S. Goldberg, "Attacking the Network Time Protocol," in *Proceedings 2016 Network and Distributed System Security Symposium*. San Diego, CA: Internet Society, 2016. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/sites/25/2017/09/attacking-network-time-protocol.pdf

[18] B. Dowling, D. Stebila, and G. Zaverucha, "Authenticated network time synchronization," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 823–840.

[19] T. Mizrahi, "Slave diversity: Using multiple paths to improve the accuracy of clock synchronization protocols," in *2012 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication Proceedings*. IEEE, 2012, pp. 1–6.

[20] A. Shpiner, Y. Revah, and T. Mizrahi, "Multi-path time protocols," in *2013 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS) Proceedings*. IEEE, 2013, pp. 1–6.

[21] P. Vixie, "DNS and BIND security issues," in *Proceedings of the 5th Symposium on UNIX Security*. Berkeley, CA, USA: USENIX Association, jun 1995, pp. 209–216.

[22] D. J. Bernstein, "DNS Forgery," Internet publication at http://cr.yp.to/djbdns/forgery.html, November 2002.

[23] D. Kaminsky, "It's the End of the Cache As We Know It," in *Black Hat conference*, August 2008, http://www.blackhat.com/presentations/bh-jp-08/bh-jp-08-Kaminsky/BlackHat-Japan-08-Kaminsky-DNS08-BlackOps.pdf.

[24] A. Hubert and R. Van Mook, "Measures for making dns more resilient against forged answers," RFC 5452, January, Tech. Rep., 2009.

[25] Y. Gilad and A. Herzberg, "Off-path tcp injection attacks," *ACM Transactions on Information and System Security (TISSEC)*, vol. 16, no. 4, p. 13, 2014.

[26] H. Shulman and M. Waidner, "Towards security of internet naming infrastructure," in *European Symposium on Research in Computer Security*. Springer, 2015, pp. 3–22.

[27] M. Brandt, T. Dai, A. Klein, H. Shulman, and M. Waidner, "Domain Validation++ For MitM-Resilient PKI," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 2060–2076.

[28] A. Herzberg and H. Shulman, "Antidotes for dns poisoning by off-path adversaries," in *2012 Seventh International Conference on Availability, Reliability and Security*. IEEE, 2012, pp. 262–267.

[29] Y. Gilad and A. Herzberg, "Fragmentation Considered Vulnerable," *ACM Transactions on Information and System Security (TISSEC)*, vol. 15, no. 4, pp. 16:1–16:31, April 2013, a preliminary version appeared in WOOT 2011.

[30] T. Rytilahti, D. Tatang, J. Köpper, and T. Holz, "Masters of Time: An Overview of the NTP Ecosystem," in *2018 IEEE European Symposium on Security and Privacy (EuroS P)*, Apr. 2018, pp. 122–136.

[31] N. Schiff, D. Dolev, T. Mizrahi, and M. Schapira, "A secure selection and filtering mechanism for the network time protocol version 4," Working Draft, IETF Secretariat, Internet-Draft draft-schiff-ntp-chronos-03, September 2019. [Online]. Available: http://www.ietf.org/internet-drafts/draft-schiff-ntp-chronos-03.txt

[32] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman, "A search engine backed by Internet-wide scanning," in *22nd ACM Conference on Computer and Communications Security*, Oct. 2015.

[33] C. E. Wills, M. Mikhailov, and H. Shang, "Inferring relative popularity of internet applications by actively querying dns caches," in *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 78–90. [Online]. Available: https://doi.org/10.1145/948205.948216

[34] W. Lian, E. Rescorla, H. Shacham, and S. Savage, "Measuring the Practical Impact of DNSSEC Deployment," in *Proceedings of USENIX Security*, 2013.

[35] P. Jeitner, H. Shulman, and M. Waidner, "Secure Consensus Generation with Distributed DoH," in *50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops, DSN Workshops 2020, Valencia, Spain, June, 2020*.

[36] A. Herzberg and H. Shulman, "Towards adoption of dnssec: Availability and security challenges." *IACR Cryptol. ePrint Arch.*, vol. 2013, p. 254, 2013.

[37] K. Fukuda, S. Sato, and T. Mitamura, "A technique for counting dnssec validators," in *2013 Proceedings IEEE INFOCOM*. IEEE, 2013, pp. 80–84.

[38] T. Dai, H. Shulman, and M. Waidner, "Dnssec misconfigurations in popular domains," in *International Conference on Cryptology and Network Security*. Springer, 2016, pp. 651–660.

[39] H. Shulman and M. Waidner, "One Key to Sign Them All Considered Vulnerable: Evaluation of DNSSEC in Signed Domains," in *The 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX, 2017.

[40] T. Chung, R. van Rijswijk-Deij, B. Chandrasekaran, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson, "A longitudinal, end-to-end view of the dnssec ecosystem," in *USENIX Security*, 2017.

[41] D. Franke, D. Sibold, K. Teichel, M. Dansarie, and R. Sundblad, "Network time security for the network time protocol," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-ntp-using-nts-for-ntp-20, July 2019. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-ntp-using-nts-for-ntp-20.txt

[42] A. Malhotra, A. Langley, and W. Ladd, "Roughtime," Working Draft, IETF Secretariat, Internet-Draft draft-roughtime-aanchal-03, July 2019. [Online]. Available: http://www.ietf.org/internet-drafts/draft-roughtime-aanchal-03.txt

2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)
Accepted version © 2020 IEEE. https://doi.org/10.1109/DSN48063.2020.00043

B. Pitfalls of Provably Secure Systems in Internet The Case of Chronos-NTP

[2] P. Jeitner, H. Shulman, and M. Waidner, "Pitfalls of provably secure systems in internet the case of chronos-ntp," in *2020 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*, Poster, 2020, pp. 49–50. DOI: 10.1109/DSN-S50200.2020.00027.

Pitfalls of Provably Secure Systems in Internet The Case of Chronos-NTP

Philipp Jeitner*, Haya Shulman*, Michael Waidner*†
*Technical University of Darmstadt, †Fraunhofer Institute for Secure Information Technology SIT

***Abstract*—The critical role that Network Time Protocol (NTP) plays in the Internet led to multiple efforts to secure it against time-shifting attacks. A recent proposal for enhancing the security of NTP with Chronos against on-path attackers seems the most promising one and is on a standardisation track of the IETF. In this work we demonstrate off-path attacks against Chronos enhanced NTP clients. The weak link is a central security feature of Chronos: The server pool generation mechanism using DNS. We show that the insecurity of DNS allows to subvert the security of Chronos making the time-shifting attacks against Chronos-NTP even easier than attacks against plain NTP.**

I. Introduction

Network Time Protocol (NTP) is one of the core Internet protocols, used to synchronise time on Internet systems. Recently [1] demonstrated a proof of concept time shifting attacks against NTP. The idea was to exploit overlapping IPv4 fragments, whereby a shifted time provided in the attacker's fragment would overwrite the time provided by the real NTP server in a response sent to the client. However, this attack is limited because it depends on various factors not typically present in the Internet, mainly minimum MTU support down to 68 bytes and an IP defragmentation algorithm not used by any current operating system.

Due to the critical role that NTP plays in the Internet there was an immediate followup work, which devised enhancements to NTP, called Chronos [2], to block the attack. Chronos leverages ideas from distributed computing on clock synchronisation in the presence of Byzantine adversaries and is designed to provide security even against strong Man-in-the-Middle (MitM) attackers and corrupted NTP servers.

Our contribution: In this work, we present an DNS-poisoning based attack on NTP, which is not only more practical than attacking NTP directly, but works even against the Chronos enhanced NTP clients. This attack is part of research on security of NTP with DNS [3].

II. DNS Cache-Poisoning Attacks

DNS cache poisoning [4]–[6] allows off-path attackers to gain MitM capabilities for some target victim domain. This allows the attacker to intercept any communication with the victim domain, such as web, email, FTP or in this case – NTP. Off-path cache poisoning attacks were demonstrated using BGP hijacking and defragmentation poisoning.

BGP hijacking places the attacker in a MitM position for the victim network [7], [8].

A. Attacking DNS with De-fragmentation Poisoning

IP Defragmentation cache-poisoning allows off-path attackers to hijack the DNS without the necessity to control BGP routers for prefix hijacking. In 2011 [4] demonstrated practical DNS cache poisoning attacks that use fragmented IPv4 packets. The idea is to inject a spoofed fragment into the IP defragmentation cache that is to be reassembled with the real fragment from the nameserver.

In [3] we evaluated fragmentation support of *pool.ntp.org* nameservers as well as DNS resolvers the internet. We found that 16 out of 30 nameservers for *pool.ntp.org* fragment DNS responses down to a MTU of 548 bytes while not supporting DNSSEC. There are also challenges with DNSSEC deployment [9]–[11]. Furthermore, using an ad-network based study we found that 90% of resolvers accept fragments of some size and 64% even the tiniest possible fragment size of 68 bytes MTU.

Additionally we found that DNS resolvers are often shared by different systems in the internet. This simplifies fragmentation-based DNS poisoning because the attacker can execute the DNS-poisoning and time-shifting attacks independently, picking a protocol which allows for easy triggering of DNS queries and yields bigger DNS responses to get over the minimum MTU size of the namserver. For 14% of DNS resolvers used by web clients attackers we were able to trigger queries via either SMTP servers or open resolvers.

III. Chronos – Security Enhanced NTP

Chronos is a formally verified, backwards-compatible NTP client which aims to prevent attacks against NTP using distributed-computing techniques [2]. In contrast to traditional NTP which queries few (typically up to 4) NTP servers, Chronos queries time from multiple NTP servers, applies a secure algorithm for eliminating suspicious responses and averages the time over the valid responses. The authors demonstrate that in order to shift time on a Chronos NTP client by 100ms a strong MitM attacker would need 20 years of effort.

Chronos *implicitly* assumes that the attacks on NTP are launched when the NTP client queries the NTP servers in the Internet for time. It's attacker model is that of a MitM attacker that changes some responses to contain shifted time or an attacker that corrupts some of the NTP servers. The security of Chronos is guaranteed if the majority of the queried NTP servers provide correct time. The idea is that all the responses are fed to an algorithm which calculates time by discarding

2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S)
Accepted version © 2020 IEEE. https://doi.org/10.1109/DSN-S50200.2020.00027

the bottom and top third of responses and sets the time to the average value of the surviving time samples.

Therefore, compared to traditional NTP, Chronos changes NTP clients in two ways. First, by using a bigger pool of upstream NTP servers and second, by replacing the clock select algorithm with one which is formally verified to be secure against a minority of malicious responses.

IV. Attacking Chronos with DNS

In this work we identify the "achilles heel" of Chronos: generating a pool of NTP servers which satisfies it's assumption of a honest majority of servers. The idea is to collect a set of roughly hundred NTP servers, from which NTP servers are selected at random and queried for time samples. This pool is generated by querying the *pool.ntp.org* domain hourly for 24 hours, yielding a number of 96 NTP servers when each DNS response contains 4 NTP servers as in the case of *pool.ntp.org*.

Chronos' server pool generation method is flawed in a way which ironically simplifies a DNS-based attack compared to traditional NTP clients. Because the DNS is queried 24 times, the attacker has potentially up to 24 tries to succeed with an cache-poisoning attack instead of just once in case of a traditional NTP client. We show that if the cache-poisoning attack succeeds until or during the 12th DNS request, the attacks still controls more than $\frac{2}{3}$ of the addresses included in the server pool, the definite maximum of servers required for a time-shifting attack on Chronos. How the cache poisoning is done – via BGP hijacking or fragment injection – and whether the query is triggered by Chronos directly of via a third party system like SMTP is not important for this attack to work.

What simplifies the attack on Chronos is the fact that other than the benign *pool.ntp.org* namservers, (1) the attacker can fit more addresses in a single DNS query (up to 89 for a single non-fragmented DNS response) and (2) set the DNS Time-to-live (TTL) to a value bigger than 24 hours, to cause any subsequent queries to be answered from cache, effectively not adding any new server to the pool. When this is done as shown in Figure 1, the resulting pool will consist of up to $4 \cdot 11 = 44$ benign and 89 malicious NTP servers which is a $\frac{2}{3}$ majority for the attacker. To succeed with this attack, the attacker therefore only needs to successfully attack the DNS once out of 12 queries during the first 11 hours of Chronos server pool generation mechanism.

V. Conclusions

Our attack demonstrates the risks of building security on insecure Internet foundations like DNS, and the risks of analysing security in isolation. The Chronos server pool-generation method can be improved to limit the impact of our attack – by not allowing more than 4 addresses in a single DNS reply and discarding responses with high TTL values. However, even with these mitigations, the dependency on the insecure DNS still remains and allows way easier attacks than Chronos was originally designed for – e.g., when the attacker manages to hijack the victim's DNS for a period of 24 hours. To devise secure solutions based on DNS we recommend proposals for generating distributed consensus in a secure way, such as that proposed in [12].

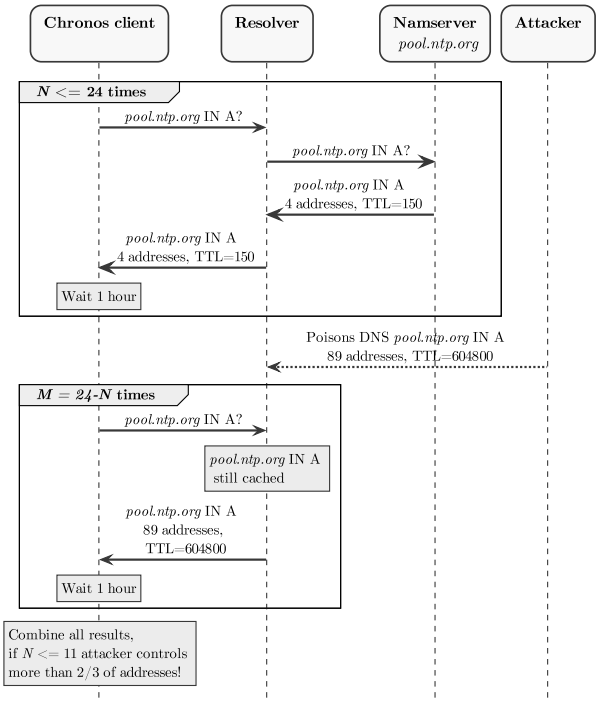


Fig. 1. DNS poisoning attack on chronos.

References

[1] A. Malhotra and S. Goldberg, "Attacking NTP's Authenticated Broadcast Mode," *ACM SIGCOMM Computer Communication Review*, vol. 46, no. 1, pp. 12–17, May 2016.

[2] O. Deutsch, N. R. Schiff, D. Dolev, and M. Schapira, "Preventing (Network) Time Travel with Chronos," in *Proceedings 2018 Network and Distributed System Security Symposium*, San Diego, CA, 2018.

[3] P. Jeitner, H. Shulman, and M. Waidner, "The Impact of DNS Insecurity on Time," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2020, (Forthcoming).

[4] A. Herzberg and H. Shulman, "Fragmentation Considered Poisonous: or one-domain-to-rule-them-all.org," in *IEEE CNS 2013. The Conference on Communications and Network Security, Washington, D.C., U.S.* IEEE, October 2013.

[5] H. Shulman and M. Waidner, "Towards security of internet naming infrastructure," in *European Symposium on Research in Computer Security*. Springer, Cham, 2015, pp. 3–22.

[6] M. Brandt, T. Dai, A. Klein, H. Shulman, and M. Waidner, "Domain Validation++ For MitM-Resilient PKI," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 2060–2076.

[7] D. Madory, "Recent Routing Incidents: Using BGP to Hijack DNS and more," 2018. [Online]. Available: https://www.lacnic.net/innovaportal/file/3207/1/dougmadory_lacnic_30_rosario.pdf

[8] S. Goldberg, "The myetherwallet.com hijack and why it's risky to hold cryptocurrency in a webapp," 2018. [Online]. Available: https://medium.com/@goldbe/the-myetherwallet-com-hijack-and-why-its-risky-to-hold-cryptocurrency-in-a-webapp-261131fad278

[9] A. Herzberg and H. Shulman, "Towards adoption of dnssec: Availability and security challenges." *IACR Cryptol. ePrint Arch.*, vol. 2013, p. 254, 2013.

[10] T. Dai, H. Shulman, and M. Waidner, "Dnssec misconfigurations in popular domains," in *International Conference on Cryptology and Network Security*. Springer, Cham, 2016, pp. 651–660.

2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S)

[11] H. Shulman and M. Waidner, "One key to sign them all considered vulnerable: Evaluation of {DNSSEC} in the internet," in *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, 2017, pp. 131–144.

[12] P. Jeitner, H. Shulman, and M. Waidner, "Secure Consensus Generation with Distributed DoH," in *50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops, DSN Workshops 2020, Valencia, Spain, June, 2020*.

2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S)

C. Secure Consensus Generation with Distributed DoH

[3] P. Jeitner, H. Shulman, and M. Waidner, "Secure consensus generation with distributed doh," in *2020 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*, Poster, 2020, pp. 41–42. DOI: 10.1109/DSN-S50200.2020.00023.

Secure Consensus Generation with Distributed DoH

Philipp Jeitner*, Haya Shulman† and Michael Waidner*†
*Technical University of Darmstadt, philipp.jeitner@sit.tu-darmstadt.de
†Fraunhofer Institute for Secure Information Technology SIT, {haya.shulman,michael.waidner}@sit.fraunhofer.de

***Abstract*—Many applications and protocols depend on the ability to generate a pool of servers to conduct majority-based consensus mechanisms and often this is done by doing plain DNS queries. A recent off-path attack [1] against NTP and security enhanced NTP with Chronos [2] showed that relying on DNS for generating the pool of NTP servers introduces a weak link. In this work, we propose a secure, backward-compatible address pool generation method using distributed DNS-over-HTTPS (DoH) resolvers which is aimed to prevent such attacks against server pool generation.**

I. Introduction

Majority-based consensus mechanisms ensure security by comparing the responses from a set of servers taken from a larger pool. The security guarantee is given by statistical analysis and assumptions on the distribution of benign and malicious servers in that pool. To generate the pool, some applications rely on lists of servers signed by central authorities (e.g., Tor [3]) but others like Chronos [2] and most Cryptocurrencies [4] just rely on the DNS. Recently [5], [6], [1] showed that this reliance on DNS introduces a weak link and allows an attacker to attack DNS instead of the application protocol. Specifically [1] showed that Chronos is vulnerable to off-path attacks, by subverting DNS security, while Chronos [2] guarantees security against strong on-path attackers. In this work we propose an approach for enhancing the DNS layer by distributing the queries for Network Time Protocol (NTP) servers to a set of resolvers. Our proposal, in tandem with Chronos, guarantees security to the NTP ecosystem.

Previous proposals on distributing DNS [7], [8], focused on the DNS nameservers, our goal in this work is to distribute the DNS resolvers. We propose a majority based protocol to securely generate a pool of servers via DNS which does not require any changes to the existing protocols nor infrastructure. Our proposal leverages majority decision supported by secure channels to a list of trusted DNS-over-HTTPS (DoH) [9] resolvers. The security is guaranteed since distributing the resolvers ensures that the queries are sent from different locations in the Internet. Hence even if some of the paths or servers are corrupted or controlled by the attacker, the requests from the paths that are not under attacker's control guarantee security. The security of our proposal assumes that the realistic attackers are limited in their capabilities: the attacker can control some of the servers and some of the links in the Internet but not all the Internet servers and links. This is a realistic assumption and applies even to the strong government sponsored attackers.

Our proposed system is shown in Figure 1. The mechanism works by querying the NTP pool domain through a list of distributed DoH resolvers (Step 2 in Figure 1) and combines the results to generate a server address pool containing a fraction of at least x (e.g. $\frac{1}{2}$) benign servers assuming that an attacker can only attack a fraction of x of the DoH resolvers.

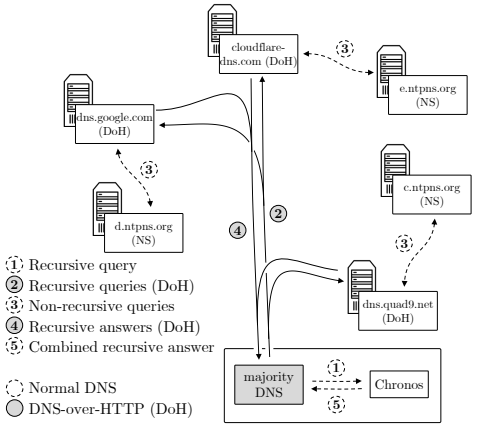


Fig. 1. System overview

II. Secure Server Pool Generation

The operation mode targeted in this work is that of applications aiming to generate a trustworthy pool of servers, the application can connect to. For the application to be secure, this pool must include a fraction of at least x benign servers. This is the operation mode Chronos [2] (and to some degree, traditional Network Time Protocol (NTP) clients) operate in. In this operation mode, the application is aware that not all servers in the generated pool are trustworthy, but the number of these is limited to be lower than the fraction of $1 - x$ of the total servers which addresses are included in the DNS response. As this operation mode is specific to server pool generation, it does only support address lookups[1]. Assuming a fraction x of secure DoH resolvers, this property can be fulfilled by querying all DoH resolvers, truncating the list of results from each resolver to the number of results from the resolver with the shortest list and then returning the the combination of these truncated lists to the application. This process is shown in Algorithm 1.

Ensuring that *all* of the servers in a returned DNS query are benign can be performed via a classic majority-vote on each of the returned addresses, e.g., the majority DNS resolver only includes an address in the final response, if it is given by a

[1]If Dual-stack operation needs to be supported, it depends on the application, whether the property of a honest majority of servers needs to be fulfilled for the union of A and AAAA records or for both sets individually.

2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S)
Accepted version © 2020 IEEE. https://doi.org/10.1109/DSN-S50200.2020.00023

Algorithm 1: Secure server pool generation lookup

```
Input: domain (query domain)
Input: resolvers (list of DoH resolvers)
Input: x (Fraction of assumed non-attacked resolvers, eg. 1/2)
results = [] , lengths = [] , addresspool = []
for res in resolvers do
    r = query(res, domain)
    results.append(r)
    lengths.append(||r||)
end
truncatelength = min(lengths[)
for r in results do
    addresspool.add(truncate(r, truncatelength))
end
return addresspool
```

majority of the DoH resolvers. This is not required for Chronos since it can handle a minority of malicious servers by itself.

We propose to deploy our mechanism without changing the DNS infrastructure, offering a standard-compatible DNS-resolver interface. The design is in Algorithm 1.

III. Security Analysis

In this section we give an analysis of the probability that responses from a fraction of x DoH resolvers can be trusted assuming that (1) all resolvers are benign and (2) a probability that an attacker can successfully perform an attack against any of the resolvers independently with a probability of p_{attack}.

a) Fraction of DoH resolvers x required for a successful attack: For a successful attack against the DNS-using application, we assume the attacker needs to control at least y (e.g., $\frac{1}{2}$) of the addresses in the server pool used by that application. Assuming a number of used DoH resolvers N and the length of the shortest[2] response list K, we get a total number of $N \cdot K$ addresses, where every resolver controls only K. To control more than y addresses the attacker therefore needs to attack at least x DoH resolvers fulfilling $yK \leq xK$, which gives us $x \geq y$. The attacker therefore is required to successfully attack at least $x = y$ of the DoH resolvers used.

b) Probability of attacking at least x of DoH resolvers: Assuming a number of used DoH resolvers N, the probability of attacking a fraction of at least x is p^M_{attack} with $M \leq \lceil xN \rceil$. Even when the only 3 DoH resolvers are used, this means that the probability of a successful attack which requires a malicious majority ($x \geq \frac{2}{3}$) is reduced significantly (p^2_{attack}). Furthermore, by increasing the number of used DoH resolvers, a successful attack becomes exponentially less probable, effectively giving the same type of asymptotic advantage over an attacker which is achieved by increasing the key size in a traditional cryptosystem.

IV. Discussion and Future Work

For our approach to work, the application must handle multiple instances of the same address in the response as individual servers. Without this assumption, an attacker might overwhelm a majority of DoH resolvers if those resolvers do not give mutually exclusive sets of responses. Furthermore, our proposal is only meant to provide security on the DNS layer, namely we guarantee that even if some of the responses from the DNS resolvers can be attacked, the majority response is correct, i.e., the NTP servers returned in a response are benign. This may however not always be the case: attackers can try to join the NTP pool themselves and operate malicious NTP servers. Hence, for the overall NTP ecosystem to maintain security a distributed mechanism on the NTP layer should also be used, such as the Chronos proposal [2].

[2]We use the shortest list, because this prevents attacks where the attacker seeks to overwhelm resolvers by including more responses than usual (See attack against Chronos [1]). This comes at the cost of allowing DoS attacks when the attacker includes no responses at all in his poisonous response.

V. Conclusion

In this work we propose a distributed generation of NTP server pool using multiple DoH DNS resolvers. We show that using our proposal mitigates the off-path attacks against plain NTP as well as against Chronos enhanced NTP [1], and guarantees security even against realistic on-path Man-in-the-Middle (MitM) attackers that control some (but not all) of the Internet paths and DNS resolvers. Our proposal is easy to integrate and does not require changes to the DNS infrastructure, it is backward compatible with standard DNS nameservers.

References

[1] P. Jeitner, H. Shulman, and M. Waidner, "The Impact of DNS Insecurity on Time," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2020, (Forthcoming).

[2] O. Deutsch, N. R. Schiff, D. Dolev, and M. Schapira, "Preventing (Network) Time Travel with Chronos," in *Proceedings 2018 Network and Distributed System Security Symposium*, San Diego, CA, 2018.

[3] The Tor Project. Tor directory protocol, version 3. https://github.com/torproject/torspec/blob/master/dir-spec.txt, accessed 2020-02-10.

[4] A. F. Loe and E. A. Quaglia, "You shall not join: A measurement study of cryptocurrency peer-to-peer bootstrapping techniques," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 2231–2247. [Online]. Available: https://doi.org/10.1145/3319535.3345649

[5] A. Herzberg and H. Shulman, "Fragmentation Considered Poisonous: or one-domain-to-rule-them-all.org," in *IEEE CNS 2013. The Conference on Communications and Network Security, Washington, D.C., U.S.* IEEE, 2013.

[6] M. Brandt, T. Dai, A. Klein, H. Shulman, and M. Waidner, "Domain Validation++ For MitM-Resilient PKI," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 2060–2076.

[7] C. Cachin and A. Samar, "Secure distributed dns," in *International Conference on Dependable Systems and Networks, 2004*. IEEE, 2004, pp. 423–432.

[8] E. S.-J. Swildens, R. D. Day *et al.*, "Domain name resolution using a distributed dns network," May 25 2010, uS Patent 7,725,602.

[9] P. Hoffman and P. McManus, "Dns queries over https (doh)," Internet Requests for Comments, RFC Editor, RFC 8484, October 2018.

2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S)

D. The Hijackers Guide To The Galaxy: Off-Path Taking Over Internet Resources

[4] T. Dai, P. Jeitner, H. Shulman, and M. Waidner, "The hijackers guide to the galaxy: Off-path taking over internet resources," in *30th USENIX Security Symposium (USENIX Security 21)*, CORE A*, USENIX Association, Aug. 2021. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/dai.

The *Hijackers* Guide To The Galaxy: *Off-Path* Taking Over Internet Resources

Tianxiang Dai*, Philipp Jeitner*†, Haya Shulman* and Michael Waidner*†

*Fraunhofer Institute for Secure Information Technology SIT
†Technical University of Darmstadt

Abstract

Internet resources form the basic fabric of the digital society. They provide the fundamental platform for digital services and assets, e.g., for critical infrastructures, financial services, government. Whoever controls that fabric effectively controls the digital society.

In this work we demonstrate that the current practices of Internet resources management, of IP addresses, domains, certificates and virtual platforms are insecure. Over long periods of time adversaries can maintain control over Internet resources which they do not own and perform stealthy manipulations, leading to devastating attacks. We show that network adversaries can take over and manipulate at least 68% of the assigned IPv4 address space as well as 31% of the top Alexa domains. We demonstrate such attacks by hijacking the accounts associated with the digital resources.

For hijacking the accounts we launch off-path DNS cache poisoning attacks, to redirect the password recovery link to the adversarial hosts. We then demonstrate that the adversaries can manipulate the resources associated with these accounts. We find all the tested providers vulnerable to our attacks.

We recommend mitigations for blocking the attacks that we present in this work. Nevertheless, the countermeasures cannot solve the fundamental problem - the management of the Internet resources should be revised to ensure that applying transactions cannot be done so easily and stealthily as is currently possible.

1 Introduction

Internet resources form the cornerstone of modern societies. The daily activities and services are increasingly digitalised, from critical infrastructures to medical services and child care. The society relies on the control over its Internet resources for availability and stability of digital services and assets. Due to their importance, Internet resources pose a lucrative target for adversaries.

Internet resources are at risk. In this work we explore the security of the Internet management systems of basic digital assets: IP addresses management with Regional Internet Registries (RIRs) [RFC7020], domains with domain registrars, virtual machine resources with infrastructure as a service (IaaS) providers and certification with Certificate Authorities (CAs), see the list in Table 1. These providers manage the allocation, registration and operation of the Internet resources for their customers. We study how easy it is for network adversaries to take over the accounts of these resource providers and then exploit the resources associated with the compromised accounts.

We show that the current practices of Internet resources management are insecure. Adversaries can take control over digital assets of customers and maintain control over them for long periods of time without being detected. Although such attacks are appealing for strong nation state adversaries and security agencies, we demonstrate that even weak *off-path* network adversaries can, through a series of protocol manipulations, take over the accounts of customers and thereby control the Internet resources associated with them.

Adversaries can hijack accounts. The idea behind our attacks is the following: the adversary poisons the cache of the DNS resolver of a resource provider, injecting a malicious record mapping the Email server of the victim customer to an adversarial host. The adversary invokes password recovery procedure. The Email server of the provider sends the password reset link to the IP address of the adversary. Adversary resets the password and hijacks the victim account. We demonstrate how the adversary can perform manipulations over the resources associated with the hijacked accounts.

Manipulation of the digital resources. The SSO (Single Sign On) accounts of the RIRs pose the highest risk: hijacking an SSO account allows a weak adversary to take over ASes and IP blocks allocated to the victim. Furthermore, through the hijacked account the adversary can make manipulations not only in the control plane of the Internet infrastructure but also in the Internet Routing Registries (IRR) and in Internet Addressing Resource Registries. Such modifications in the IRR can among others also facilitate extremely effective BGP prefix hijacks. Specifically, IRR records are prerequisite for

2021 30th USENIX Security Symposium
Accepted version. https://www.usenix.org/conference/usenixsecurity21/presentation/dai

BGP hijack attacks - without proper records in the IRR the attacker cannot convince benign upstream providers to accept and propagate the fraudulent BGP announcements in the input filters on the BGP sessions. Adversaries without the ability to modify the IRR, have to use less vigilant and generally poorly managed networks as upstream providers or have to utilise path manipulation attacks [35] - both restricting the success rate and the stealthiness of the attack. Our adversary can, by modifying the records in the IRR, cause well managed and reputed upstream providers to unwittingly propagate the malicious BGP announcements. Hence making BGP prefix hijacks more effective than the typical control plane BGP prefix hijacks while at the same time more difficult to identify. To maintain control over the victim Local Internet Registries (LIRs) resources over long periods of time the adversary implants itself in the system with elevated privileges.

We also show that hijacking an account under a CA allows an adversary to revoke certificates, renew a certificate or issue new certificates for domains registered under the hijacked account. Renewal of certificates allows to associate a new key-pair with the certificate. Nevertheless some CAs do not perform validation of certificate renewal requests issued from registered accounts.

By hijacking the accounts of domain registrars, the adversary can manipulate records in victim domains, e.g., to launch phishing attacks. Finally, hijacking accounts of IaaS providers enables the attackers to take over virtual machines and the resources that run on those virtual machines, including databases, applications and computations.

Disclosure and ethics. Our attacks were tested against providers and customers reliably, yet were ethically compliant. To avoid harming Internet users, we set up victim domains and registered victim accounts which were used by us for carrying out the attacks and for evaluating the vulnerabilities. This ensured that the providers would not use the spoofed records for any "real" purpose. In addition to evaluating the attacks with the "victim" accounts that we set up, we also evaluated our exploits of hijacked accounts against one large ISP under RIPE NCC and attacked the real domain of that ISP in coordination with that ISP. We are disclosing the results of our work to the providers.

Contributions. We provide the first demonstration of off-path attacks exploiting hijacked accounts under popular providers and show that adversaries can perform manipulations in the resources assigned to the accounts over long time periods without being detected.

Organisation. In Section 2 we review DNS cache poisoning and related work. In Section 3 we provide an overview of our study. In Section 4 we list methodologies for off-path DNS cache poisoning attacks. In Section 5 we evaluate the cache poisoning methodologies for taking over customers accounts in different providers in our dataset. Then, in Section 6, we demonstrate how the adversaries can manipulate digital resources assigned to the accounts they control. In Section 7 we explain the fraction of the digital resources (IP address' blocks and domains) that are at immediate risk due to being associated with vulnerable accounts. We recommend countermeasures in Section 8 and conclude in Section 9.

2 DNS Cache Poisoning Overview

DNS. Domain Name System (DNS), [RFC1035], performs lookup of services in the Internet. Recursive caching DNS resolvers receive DNS requests for services in different domains and send queries to the nameservers authoritative for those domains. The nameservers respond with the corresponding DNS records. The DNS records in responses are cached by the DNS resolvers and are provided to clients and servers which use that resolver. Subsequent requests for that domain are responded from the cache. For instance, to send an Email to `alice@example.info` the Email server of Bob will ask the DNS resolver for the IP address of the Email exchanger in domain `example.info`. The resolver asks the nameservers in domain `example.info` for an IP address and a hostname (`A` and `MX` records) of the Email exchanger and receives:

```
example.info IN MX mail.example.info
mail.example.info A 1.2.3.4
```

The resolver will send to the Email server of Bob the IP address of the Email exchanger of Alice and will also cache the records from the response for answering future queries for `MX` and `A` in `example.info` domain.

DNS Cache Poisoning. In a DNS cache poisoning attack the goal of the adversary is to redirect clients of some resolver to an IP address of the adversary for queries in a target victim domain. To do that, the adversary sends a DNS response from a spoofed source IP address, which belongs to the nameserver of the victim domain, with malicious DNS records mapping the victim domain to an IP address of the adversary. For instance, to intercept the Emails sent to Alice the adversary injects a DNS record mapping the Email exchanger of Alice to an adversarial host. If the resolver accepts and caches the malicious record, its cache becomes poisoned.

```
example.info IN MX mail.example.info
mail.example.info A 6.6.6.6
```

The added value of DNS cache poisoning attacks is that they have a local impact, affecting not the entire Internet but only the victim network and hence allow for extremely stealthy attacks, which can go undetected over long time periods. There is more and more evidence of DNS cache poisoning in the wild and the attacks are becoming increasingly sophisticated. In the recent cache poisoning attacks in the wild the adversaries attempt to intercept DNS packets by launching short-lived BGP (Border Gateway Protocol) prefix hijacks [34]. In such attacks, the adversary advertises a BGP announcement hijacking the prefix of a victim for a short time

only to hijack the target DNS packet and then releases the hijack [15]. This allows the attacker to poison the DNS cache of a victim resolver and then intercept all the communication between the victim resolver and the target domain. Recent research projects showed that the CAs (Certificate Authorities) and the bitcoin infrastructures were not resilient to prefix hijacks [6, 8, 9].

History of DNS Cache Poisoning. Launching cache poisoning in practice is however hard. We explain the evolution of cache poisoning attacks and the mitigations. In 1995 Vixie pointed out to the cache poisoning vulnerability and suggested to randomise the UDP source ports in DNS requests [45]. In 2002 Bernstein also warned that relying on randomising Transaction ID (TXID) alone is vulnerable [7]. Indeed, in 2007 [29] identified a vulnerability in Bind9 and in Windows DNS resolvers [30] allowing off-path attackers to reduce the entropy introduced by the TXID randomisation. In 2008 Kaminsky [26] presented a practical cache poisoning attack even against truly randomised TXID. Following Kaminsky attack DNS resolvers were patched against cache poisoning [RFC5452] by randomising the UDP source ports in queries. Nevertheless, shortly after different approaches were developed to bypass the source port and the TXID randomisation for launching off-path cache poisoning attacks. In 2012 [17] showed that off-path adversaries can use side-channels to infer the source ports in DNS requests. In 2015 [41] showed how to attack resolvers behind upstream forwarders. This work was subsequently extended by [47] with poisoning the forwarding devices. A followup work demonstrated such cache poisoning attacks also against stub resolvers [5]. [33] showed how to use ICMP errors to infer the UDP source ports selected by DNS resolvers. Recently [31] showed how to use side channels to predict the ports due to vulnerable PRNG in Linux kernel. In 2013 [18] provided the first feasibility result for launching cache poisoning by exploiting IPv4 fragmentation. IPv4 fragmentation based attacks were applied to shift time on NTP servers [11, 23, 32], these attacks are not practical anymore since the nameservers in NTP domains were patched to avoid fragmentation. The study in [11] used fragmentation based cache poisoning for bypassing domain validation with CAs. However, most CAs patched the vulnerabilities which [11] exploited to attack domain validation, e.g., Let'sEncrypt blocked fragmentation. Let'sEncrypt also deployed domain validation from multiple vantage points [9, 25], which makes the previous off-path attacks [8, 11] impractical.

In addition to other attacks in this work, we also show another way to attack the CAs, by taking over customers' accounts with the CAs and not by bypassing domain validation. As we show this allows even more effective attacks that were presented in [11]: (1) when controlling a compromised account the adversary can renew existing certificates to use a new key-pair. Since some CAs do not apply domain validation during certificates' renewal this attack allows to issue fraudulent certificates without the need to attack DV. Furthermore, in our work we use a number of off-path DNS cache methodologies from [14] to take over accounts with providers.

Cache poisoning attacks could be prevented with DNSSEC [RFC6840] [46] which uses cryptographic signatures to authenticate the records in DNS responses. However, DNSSEC is not widely deployed. Less than 1% of the second level domains (e.g., 1M-top Alexa) domains are signed, and most resolvers do not validate DNSSEC signatures, e.g., [12] found only 12% in 2017. Our measurements show that the DNSSEC deployment in our datasets is not better: the resolvers of 19 out of 35 tested providers do not validate DNSSEC signatures (see Table 2) and less than 5% of the customers' domains are signed. Deploying DNSSEC was showen to be cumbersome and error-prone [13]. Even when widely deployed DNSSEC may not always provide security: a few research projects identified vulnerabilities and misconfigurations in DNSSEC deployments in popular registrars [12, 42]. However, even correctly deployed DNSSEC does not prevent recent cache poisoning attacks [24]. The idea behind these attacks is to encode injections into the DNS records in DNS responses. When the resolver parses the records, a misinterpretation occurs, such that when the record is stored a different domain name is used. Since DNSSEC validation is applied prior to the misinterpretation, the validation of DNSSEC succeeds, and the DNS cache poisoning occurs afterwards. Preventing these attacks requires validating or escaping records from DNS lookups.

Recent proposals for encryption of DNS traffic, such as DNS over HTTPS [21] and DNS over TLS [22], although vulnerable to traffic analysis [40,43], may also enhance resilience to cache poisoning. These mechanisms are not yet in use by the nameservers in the domains that we tested. Nevertheless, even if they become adopted, they will not protect the entire resolution path, but only one link on which the transport is protected and hence will not completely prevent DNS cache poisoning attacks.

3 Attack Overview

In our study we explore the security of the services which provide access to and management of the key digital assets in the Internet: domains, IP prefixes and ASes, virtual machines and certificates. In Table 1 we list the resources, as well as the public service providers of these resources, that we studied in this work. Access and management of these digital resources is performed with the accounts that the providers offer to the customers via their web portals. In this section we provide an overview of our study from the perspective of the adversary for hijacking the accounts of the customers under different resource providers.

Find the target. Assume for example that the adversary wishes to hijack the DNS servers hosted on a victim prefix `205.251.192.0/18` – this was a real attack launched against

an LIR Amazon route53 in April 2018. First, the adversary needs to find an account to which these resources are assigned and through which these resources can be managed. Then, the adversary needs to find the username associated with that account. In Section 5.2 we show how to find the needed information: the owner, the public service provider, the Email which is associated with the account through which the digital resources can be managed. In the case of our example, the prefix is allocated by ARIN to an LIR with OrgId `AMAZON-4`, aka Amazon.com, Inc. and has 3 origin ASNs (Autonomous System Numbers) registered: AS16509, AS39111 and AS7224. We thereby learn that the responsible RIR for Amazon is ARIN and that Amazon has an LIR agreement with ARIN. We also find the Email address `ipmanagement@amazon.com` used by Amazon for managing its resources via the SSO account with ARIN.

Poison DNS of public service provider. The adversary uses one of the methodologies in Section 4 to launch off-path DNS cache poisoning attack against the DNS resolver of the service provider ARIN. During the attack the adversary injects a malicious DNS record mapping the Email server of domain `amazon.com` to the IP addresses controlled by the adversary (step ①, Figure 1). As a result, the Emails sent by ARIN to Amazon will be received by the adversary.

Hijack victim account. The adversary triggers password recovery for Email `ipmanagement@amazon.com`. This Email is associated with the SSO account at ARIN. In order to send the password recovery link, the Email server at ARIN needs the IP address of the Email server of Amazon. The resolver at ARIN already has a corresponding record in the cache, which it provides to the Email server. This IP address was injected by the adversary earlier in step ①. ARIN sends the Email with password recovery instructions to the adversary (step ②, Figure 1). The attacker resets the password and takes control over the account. We experimentally evaluate such attacks against the providers and their customers in our dataset in Section 5 for details.

Manipulate the resources. The adversary manipulates the resources assigned to the victim account, say of Amazon, and can sell the IP prefixes and ASes owned by Amazon (step ③, Figure 1). In Section 6 we describe the exploits we evaluated against the resources assigned to our victim accounts. We show that among others, the attacker can create additional accounts for itself with arbitrary privileges, and hence even if the real owner resets the password back, the attacker still maintains control over the resources. In some cases these manipulations generate notification Emails to the Email address associated with the resources. This Email address is however hijacked by the adversary, hence the adversary receives the notifications. As a result the attack will not be detected and can stay under the radar over a long period of time.

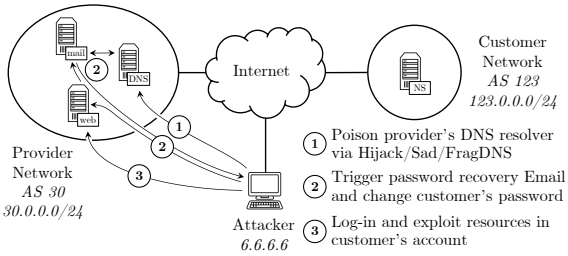


Figure 1: Attack overview

4 Off-Path DNS Cache Poisoning

The key contribution in our work is to show that once an adversary controls an account with a resource provider, it can in an easy and stealthy way manipulate the digital resources associated with that account. But, how easy is it to take over accounts? We show how to take over accounts by injecting a poisoned DNS record into the caches in DNS resolvers of providers. When the adversary triggers the password recovery procedure for the victim account, the reset email is sent to the adversarial host at the IP address in the injected DNS record.

How easy is it to launch off-path DNS cache poisoning? In this section we use methodologies from [14] to launch off-path DNS cache poisoning attacks: BGP prefix hijacks [8], side-channels [33], and IPv4 defragmentation cache poisoning [18]. We do not consider attack methodologies which are effective only against specific operating systems, say due to poor random number generators. We implement cache poisoning attacks using these methodologies and evaluate them against the providers and the customers in our dataset. We describe the experimental setup in Section 4.1. We explain our study methodology in Section 4.2. Then in Sections 4.3, 4.4 and 4.5 we present the DNS cache poisoning methodologies and the experimental evaluations against the targets in our dataset.

4.1 Setup

To test our attacks experimentally in the Internet we setup a victim AS. To purchase the victim AS we registered a secondary LIR account with RIPE NCC for our organisation (which has a primary account with RIPE NCC). We purchased a /22 prefix for our AS for 20,000USD. We connected our AS to DE-CIX internet exchange point in Frankfurt. This AS hosts the servers which we use for our evaluation of the attacks.

We set up an Unbound 1.6.7 DNS resolver on Linux 4.14.11, whose cache we poison with the records of the customer domains. We registered a victim domain and set up two nameservers in our domain and an Email server. We use our victim domain to register accounts with the services that we test in this work. We call this domain the victim customer domain. We also set up a border router which represents our attacker. The attacker's BGP router issues bogus BGP an-

nouncements that claim the prefix assigned to our victim AS. This allows us to evaluate the viability of attacks with BGP prefix hijacks against our domains hosted on our victim AS without affecting services and domains not under our control and without affecting the global BGP routing table in the Internet.

To evaluate cache poisoning attacks with side-channels we configure the nameservers in our domain to support rate-limiting and the DNS resolver to issue ICMP errors. To evaluate fragmentation based cache poisoning attacks we configure nameservers in our domain to reduce the MTU according to the value in ICMP fragmentation needed messages. The nameservers in our victim domain use a globally incremental IPID counter.

4.2 Study Methodology

Our experimental evaluation of the attacks is performed reliably yet without disrupting the functionality of the customers. To achieve this we evaluate the attacks in two steps: (1) We evaluate vulnerabilities to cache poisoning in providers. For this we set up victim domains and register victim accounts with the providers. We experimentally test the attack methodologies against providers by poisoning their DNS caches with malicious records mapping the Email server in our victim domain to the adversarial hosts that we control. We then hijack our victim accounts by triggering the password recovery procedures and changing their passwords. This enables us to validate vulnerabilities to cache poisoning yet without risking that the providers use poisoned records for genuine customers. The ability to take over the accounts of the real customers depends not only on vulnerabilities in providers' infrastructure but also on properties in customers' domains. (2) Hence, in this step we set up a victim DNS resolver and poison its cache with malicious records mapping the genuine customer domains to our adversarial hosts. The combination of both evaluations against the providers and against the customers enables us to estimate the extent of the vulnerable accounts that can be hijacked.

4.3 BGP Prefix Hijack

BGP (Border Gateway Protocol) allows ASes to compute the paths to any Internet destination. Since BGP is currently not protected, adversaries can send bogus BGP announcements to hijack victim prefixes, hence intercept the communication of victim ASes that accept the malicious BGP announcements. In our attacks we hijack the prefix of our AS: once in the evaluation against providers to intercept the responses from our nameservers sent to the DNS resolvers of the providers and then again during the evaluation of the customers, to intercept requests from our DNS resolver to the customers' domains. After our AS accepts the bogus BGP announcement, all the communication between the servers on our AS and the servers of the targets in our dataset traverse our adversarial BGP router.

We launch short-lived hijacks. Such hijacks are common [37] and allow the attacker to stay below the radar [4, 16]. It is believed that short-lived traffic shifts are caused by the configuration errors (that are quickly caught and fixed) and since they do not have impact on network load or connectivity, they are largely ignored [10, 27, 28]. We evaluate our attacks using short-lived same prefix hijacks and sub-prefix of the victim prefix.

Our experimental evaluation reflects a common BGP hijacking attacker: the attacker controls a BGP router or an AS, and issues BGP announcements hijacking the same-prefix or a sub-prefix of a victim AS in the Internet.

4.3.1 Attack evaluation against providers

The adversary announces to our victim AS a prefix of the network of the provider where the target DNS resolver is located. The bogus BGP announcement is sent *only* on the interface that is connected to our AS and is not sent to other destinations in the Internet. As a result, the responses from the nameservers of our victim domain are sent to the adversarial host instead of the DNS resolver of the provider. The adversary initiates password recovery procedure for an account of our victim customer domain. This triggers a DNS request to our victim domain. The corresponding nameserver sends a response, which is instead redirected to the adversary's host. The adversary manipulates the response, and injects a DNS record that maps the Email server of our victim domain to the IP address of the adversary. The response is then sent to the provider and the BGP hijack is released. The DNS resolver caches the response and returns it to the Email server, which sends the password recovery link to the IP address of our adversary. The adversary resets the password and takes control over the account.

4.3.2 Attack evaluation against customers

The adversary announces to our victim AS prefixes of the networks that host the nameservers in the target customers' domain. The bogus BGP announcements are sent *only* on the interface that is connected to our AS and not to other destinations in the Internet. As a result, the DNS requests from the DNS resolver on our victim AS are sent to the adversarial host instead of the nameservers of the customer' domain. The attacker releases the hijacked prefix, and additionally crafts a spoofed DNS response to our DNS resolver mapping the IP address of the adversary to the Email server of the victim customer's domain. The records from the DNS response are cached by our resolver.

4.4 Side-channel Port Inference

SadDNS off-path attack [33] uses an ICMP side channel to guess the UDP source port used by the victim resolver in the query to the target nameserver. This reduces the entropy in a DNS request from 32 bit (DNS TXID & UDP port) to 16 bit. The adversary then uses brute-force to match the TXID by sending spoofed packets for each possible TXID value to the resolver.

4.4.1 Attack evaluation against providers

We verify the existence of the ICMP global-rate limit: we send a single UDP probe to the resolver to verify that it emits ICMP port unreachable messages. Then, we send a burst of 50 spoofed UDP packets to closed ports at the resolver and follow up with a single non-spoofed UDP packet and observe if an ICMP port unreachable message is received by our sender. If the ICMP global rate-limit is present no message will be received because the global rate-limit is already reached.

The adversary initiates password recovery procedure with a provider for an account of our victim customer domain. This triggers a DNS request to our victim domain. The adversary mutes the nameservers on our victim AS, to prevent the response from being sent to the resolver of the provider, then runs the procedure for inferring the source port in the DNS request. Once the source port is found, it sends 2^{16} spoofed responses for each possible TXID values with malicious DNS records in payload. The records map the nameservers of the victim domain to attacker controlled IP addresses. If the response is accepted by the resolver of the public service, it is cached and used by the service for sending an Email with the password or the reset link. The attacker now controls the account.

4.4.2 Attack evaluation against customers

We configure our DNS resolver to send ICMP errors on closed ports. We use our own implementation of the SadDNS port scanning application with binary search and attempt to poison the resolver with a malicious record pointing the domain of the customer to our adversarial host. Due to a high failure rate, evaluation of each tuple (resolver, domain) takes up to 30 minutes, hence evaluating SadDNS on all the domains in our dataset is not practical. We therefore perform the measurement on a dozen randomly selected customers in our dataset. Our implementation performs the complete attack from triggering the queries to muting the nameservers and scanning the ports (using the ICMP side-channel) and in the last step sending the spoofed DNS responses with malicious records.

The high failure rate of the SadDNS attack is due to the fact that most of the queries do not generate a useful attack window, since the resolver times out after less than a second. The attacker can further improve this via manual attack by analysing the back-off strategies of the target resolver. The timeout of the resolver is implementation dependent, e.g., the timeout value of Unbound is a dynamically computed value based on RTT to the nameserver, while Bind uses 0.8 seconds. The DNS software increases the timeout value after each retransmission.

4.5 Injection into IP-Defragmentation Cache

The off-path adversary uses a spoofed IPv4 fragment to manipulate the fragmented response from the nameserver, [18]. The idea is to send a spoofed fragment which is reassembled with the first genuine fragment of the nameserver. The adversary replaces the second fragment of the nameserver with its malicious fragment, hence overwriting some parts of payload of a DNS response with new (attacker's injected) content. As a result, the reassembled IP packet contains the legitimate DNS records sent by the genuine nameserver with the malicious records from the fragment sent by the adversary. Since the challenge values (port, TXID) are in the first fragment of the response from the nameserver, they remain intact, and hence correct.

4.5.1 Attack evaluation against providers

We evaluate FragDNS attack against the resolvers of the providers with our victim domain. For our nameservers we use a custom application that we developed, which always emits fragmented responses padded to a certain size to reach the tested fragment size limit. The nameservers are configured to send `CNAME` records in the first fragmented response. As a result, when the resolver of the provider receives a fragmented response and reassembles it, the DNS software will issue a subsequent query for the `CNAME`-alias. This allows us to verify that the spoofed fragment arrived at the resolver and was reassembled correctly and cached, which is an indicator that the cache poisoning via fragmentation attack succeeded. Throughout the attack we use our adversarial host to trigger password recovery procedures and to inject malicious DNS records into the caches of the providers, mapping the Email servers in our domains to the IP addresses allocated to our adversary.

The adversary sends two spoofed fragments (for each nameserver's IP address) to the resolver of the public service. The fragments are identical except for the source IP addresses: one is sent with a source IP address of one nameserver and the other is with the source IP address of the other nameserver. The fragments are constructed so that they match the first fragment in the response that will have been sent by our nameserver. In the payload the fragments contain malicious DNS records mapping the Email server to the IP address of the adversary. The adversary initiates password recovery for our victim account. This triggers a DNS request to one of the nameservers in our victim domain. We do not know in advance which nameserver that will be, and hence initially send

two spoofed fragments (for each nameserver). The response from the nameserver is sent in two fragments. Once the first fragment reaches the IP layer at the resolver of the provider it is reassembled with one of the second fragments of the adversary (it is already waiting in IP defragmentation cache). The reassembled packet is checked for UDP checksum and if valid, passed on to the DNS software on the application layer. If the records from the DNS packet are cached by the resolver of the public service, the password recovery link will be sent to the host controlled by the attacker.

When to send the spoofed 'second' fragment? The standard [RFC791] recommends caching the IP fragments in IP defragmentation cache for 15 seconds. If there is no matching fragment after 15 seconds, the fragment is removed from IP defragmentation cache. The actual caching time exceeds 15 seconds in most implementations. For instance in Linux `/proc/sys/net/ipv4/ipfrag_time` is set to 30 seconds. For attack to succeed the total time between the moment that the fragment enters the IP defragmentation cache at the provider's resolver and the moment at which the first fragment from the genuine nameserver arrives should not exceed 15 seconds (to ensure that the attack is effective not only against resolvers running on Linux but also against standard compliant operating systems). We show that in practice 15 seconds suffice to launch the attack. We measure the latency from the moment that we trigger the password recovery procedure via the web interface of the provider and the moment that a DNS request from the resolver of the provider arrives at our nameservers. The measurements for different providers are plotted in Figure 2. As can be seen, except for two providers, all the latencies are below 30 seconds. The results for the attack window across all the providers, plotted in Figure 2 show that the latencies are stable, and are within the interval which provides for successful attacks. For instance, for AFRINIC RIR the attacker learns that after issuing the password recovery procedure the DNS query will be sent to the victim nameserver at a predictable time interval (between 0.1 and 0.2 seconds).

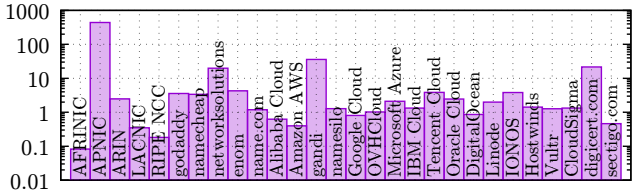


Figure 2: Avg. latency (in seconds) between registration and resolver query, excluding outliers outside $\pm 1\sigma$.

4.5.2 Attack evaluation against customers

Our evaluation is performed with the domain of a victim customer against our DNS resolver. The DNS resolver is configured to allow fragmentation. We look-up the nameservers in the domain of the customer and check if we can force them to fragment responses: (1) for each nameserver, our DNS resolver sends requests to the nameserver and receives responses. (2) From the adversarial host we send to these nameservers *ICMP fragmentation needed* errors indicating Packet Too Big (PTB) for the source IP address of our DNS resolver. (3) We send DNS requests from our resolver and check if the responses arrive fragmented according to the MTU indicated in the ICMP errors.

We then run FragDNS attack against the nameservers that fragment DNS responses following our ICMP PTB errors: (4) The adversarial host crafts spoofed second fragments, one for each nameserver in customer's domain. Since the adversary does not know to which nameserver the resolver will send a DNS request (the nameserver selection depends on DNS resolver software) it will send spoofed second fragments for each of the nameserver in that domain. Each fragment contains an identical payload: a malicious DNS record that maps the Email server of the customer domain to the IP address of our adversary. Each fragment has a different spoofed source IP address corresponding to each of the nameserver in the target domain. The adversary sends all these fragments to our DNS resolver. (5) The adversary causes our DNS resolver to issue a DNS request for a `MX` record in victim customer's domain. The nameserver which received the request responds with a fragmented DNS packet. The first fragment is reassembled with the matching second fragment that is waiting in the IP defragmentation cache. (6) The adversary receives a DNS response from our resolver. If the Email server in the response is mapped to the IP address of our adversary, then the attack succeeded.

5 Hijacking Accounts

In this section we evaluate DNS poisoning attacks against the providers and the customers using the methodologies in Section 4.

After collecting the target providers and their customers in Section 5.1, we analyse the password recovery mechanism at each provider in Section 5.2. Then we collect the DNS resolvers at those providers in Section 5.3.1. We evaluate off-path cache poisoning attacks against the DNS resolvers of providers in Section 5.3.2. Finally we measure the percentage of vulnerable customers of those providers in Section 5.4.

5.1 Datasets

In our measurements and attacks' evaluations we use two datasets: of providers and of their customers.

Providers. The providers that we study are listed in Table 1. For each class of resource providers (RIRs, Registrars, IaaS providers, CAs) we select a set of most popular examples. Our methodology for selecting the providers is: (1) all the five RIRs, (2) we scan the `whois` data of 100K-top Alexa domains and select the top 15 registrars according to the number of

domains each registrar is managing, (3) to select the IaaS providers, we use market share data and supplement it with additional selected providers[1], (5) we select the top 5 CAs which cover 97% of the market share[2], all other CAs have less than 1% market share.

For registrars and IaaS providers these datasets include providers which we could not test, because they do not allow creation of user accounts. For example, publicdomainregistry does not offer accounts to end-users directly, but only manages domain registration for webhosters. Providers where we could not register accounts are: tucows.com, publicdomainregistry, cscglobal, markmonitor, Rackspace cloud, CenturyLink Cloud and Joyent Triton.

We obtain a list of 32 resource providers which use 1,006 resolvers for sending Email (back-end IP addresses) on 44 ASes associated with 130 prefixes. Some resource providers use only a small amount of Email servers and resolvers on their own networks, while other providers use large pools of Email servers and resolvers provided by third-party Email services like Mailchimp and Sendgrid. We list this technical information in Table 2.

Customers. We extract account information for customers of RIRs and domain registrars from `whois` databases. We parse the Email addresses in `whois` records to extract the domains of the customers and query the nameservers responsible for those domains.

Because of data protection settings, not all `whois` records contain Email addresses, or only contain masked Email addresses which point to a registrar's Email proxy. We were able to find Email addresses for 74.62% of the ASes from RIR `whois` databases and for 10.60% of the domains owners in 100K-top Alexa list from domain registrar `whois` databases (see Table 3). We collected 94,997 user accounts hosted in 59,322 domains and 69,935 nameservers.

We were not able to retrieve user account information for IaaS accounts and CAs as this is not possible ethically in an automated way. An adversary can obtain this information, e.g., by enumerating usernames as described in Section 5.2.

Our dataset of domain registrars is also representative for other types of resources hosted under that domain. Organisations which own domains also own cloud resources at IaaS providers and certificates at CAs and use the same domain for their Email addresses and therefore are vulnerable to the same attack at those providers.

[1] https://www.srgresearch.com/articles/quarterly-cloud-spending-blows-past-30b-incremental-growth-continues-rise, https://stackify.com/top-iaas-providers/, https://www.g2.com/categories/infrastructure-as-a-service-iaas

[2] https://w3techs.com/technologies/history_overview/ssl_certificate: this market share data lists most of Let'sEncrypt certificates as issued by IdenTrust as Let'sEncrypt certificates are cross signed by IdenTrust. We do not test Let'sEncrypt itself because it does not offer traditional user accounts and therefore does not support password recovery.

5.2 Collecting Accounts' Information

The first step in our attack is to trigger the password recovery procedure at the provider. This step requires collecting information of the target customer whose account the attacker attempts to hijack, such as the Email account required to log into the target account, a username or a handle. We study for each service provider which information is needed for password recovery and how to collect that information for our targets; the data is summarised in Table 1. We found that the customers' Email addresses can often be retrieved from the public `whois` records. We were able to extract the Email addresses associated with the accounts at the providers for 41% of the customers in our study. For instance, the Emails for the SSO accounts of 74.62% of the LIRs (i.e., the customers of RIRs) can be retrieved via `whois`.

For victim customers whose details cannot be publicly accessible via `whois` we find the required information with manual research and dictionary attack. To carry out the dictionary attack we used the observations we derived from our data collection from public sources: the data we collected through `whois` shows that more than 24% of the Email addresses use one of ten well-known username parts, like `domains@email.info`, `hostmaster@email.info`, etc., which enables an informed attacker to find the Email addresses in less than ten attempts when these details are not publicly available through `whois`. We apply dictionary attack to also recover other details: for example, our study shows that about 1 in 10 LIRs (customers of RIRs) use usernames that are identical to the Email address that is registered in the `whois` records; e.g., username `operator` is associated with Email address `operator@email.info`.

5.3 Attacking Providers

The adversary needs to poison the DNS cache of the provider, by injecting a record into the resolver's cache that maps the domain of the provider to the adversarial IP addresses. We therefore collect the IP addresses of the DNS resolvers of the providers.

5.3.1 Identify the target DNS resolvers

In order to poison the DNS cache of the provider the adversary needs to find the IP addresses of the DNS resolvers which are used for looking up the Email servers of the customers during requests for password recovery.

We register accounts with the providers via the web portal of each provider. For our evaluation we register 20 accounts with each provider, each account is associated with a unique domain that we registered for that purpose. We use these registered accounts to learn about the infrastructure of the provider. We trigger the password recovery procedure for our registered accounts. To stay under the radar we limit the amount of password recovery requests to ten for each account.

Type	Provider	Details needed for PW recovery	Public-known	Captcha	Fragment	SadDNS	BGP hijack
RIRs	AFRINIC	NIC-handle	✓	✗	✓	✗	✓
	APNIC	Email	✓	✗	✓	✗	✗
	ARIN	Email, Username	✗	✓	✓	-	✗
	LACNIC	Username	✓	✗	✓	✗	✓
	RIPE NCC	Email	✓	✓	✓	✗	✓
Domain registrars	godaddy	Email, Domain name	✓	✗	✓	-	✓
	namecheap	Email	✓	✓	✓	✗	✓
	networksolutions	Email	✓	✗	✓	✗	✓
	enom.com	Login ID, Sec. question	✗	✓	✓	✗	✓
	name.com	Username[1]	✓	✗	✓	-	✓
	Alibaba Cloud	Username, 2-FA	✗	✓	✓	✗	✓
	Amazon AWS	Email	✓	✓	✓	✗	✓
	gandi.net	Email	✗	✗	✓	✗	✓
	namesilo.com	Email, Sec. question	✗	✓	✓	✗	✓
	Google Cloud	Last password, 2-FA	✗	✓	✓	✗	✓
	ovh.com	Email	✓	✗	✓	✗	✓
Cloud management (IaaS)	Amazon AWS	Email	Typically no for all IaaS providers	✓	✓	✗	✓
	Microsoft Azure	Email		✗	✗	✗	✓
	Alibaba Cloud	Username, 2-FA		✗	✓	✗	✓
	Google Cloud	Last password, 2-FA		✗	✓	✗	✓
	IBM Cloud	Email ('id')		✗	✓	✓	✓
	Tencent Cloud	Email		✓	✓	✗	✓
	Oracle Cloud	Email		✗	✗	✗	✓
	DigitalOcean	Email		✗	✓	✗	✓
	Linode	Username[1]		✗	✓	✓	✓
	IONOS	Email, id or domain		✗	✓	✓	✓
	Hostwinds	Email		✗	✗	✗	✓
	OVHcloud	Email		✓	✓	✗	✓
	Vultr	Email		✓	✓	✓	✓
	CloudSigma	Email		✗	✓	-	✓
CAs	IdenTrust	Account number	✗	✗	✓	-	✓
	DigiCert	Username[1]	✗	✗	✓	✗	✓
	Sectigo	Email	✗	✗	✓	✗	✓
	GoDaddy	Username[1], customer No.[1]	✗	✗	✓	-	✓
	GlobalSign	Username	✗	✗	✓	-	✓

-: No response. [1]: Can be retrieved using domain name/Email.

Table 1: Password recovery at each provider.

The Email server of the provider requests the DNS resolver to look up the `MX` and `A` records for our Email exchanger - this is required in order to send the password, or the link to reset the password. We monitor the requests arriving at the nameservers of our domains and collect the IP addresses which sent DNS requests for records in domains under which we registered our accounts. These IP addresses belong to the DNS resolvers used by the providers. We repeat this for each provider on our list in Table 2.

For every provider, we list in Table 2 the service providers of the Email servers and the DNS resolvers (by mapping the observed IP addresses to ASNs). Additionally, we also performed measurements if the resolvers of the providers in our dataset support DNSSEC and the default EDNS size in DNS requests.

5.3.2 Poison providers' DNS caches

To understand the vulnerabilities to cache poisoning across the providers, we evaluate the DNS cache poisoning methodologies against the DNS resolvers of the providers in our dataset. Our evaluations are done as described in Section 4 using the victim domains that we set up and the accounts that we registered. During the evaluations the adversary triggers password recovery procedure and applies the DNS cache poisoning methodologies (one during each test) to inject into the DNS cache of the provider malicious records mapping the Email servers of our victim domains to the hosts controlled by the adversary. In this section we report on the results of our evaluations and the extent of the vulnerabilities among the providers.

HijackDNS. To infer the scope of providers vulnerable to the attack in Section 4.3 we perform Internet measurements checking for vulnerabilities that allow sub-prefix hijacks. Since many networks filter BGP advertisements with prefixes more specific than /24, we consider an IP address vulnerable if it lies inside a network block whose advertised size is larger than /24. We therefore map all resolvers' IP addresses to network blocks. Then, to obtain insights about the sizes of the announced BGP prefixes for providers' network blocks with resolvers we use the BGPStream of CAIDA [2] and retrieve the BGP updates and the routing data from the global BGP routing table from RIPE RIS [38] and the RouteViews collectors [44]. We analyse the BGP announcements seen in public collectors for identifying networks vulnerable to sub-prefix hijacks by studying the advertised prefix sizes. The dataset used for the analysis of the vulnerable sub-prefixes was collected by us in January 2021. Our analysis in Table 2 shows that the networks of 29 providers are vulnerable to sub-prefix hijacks.

To understand the viability of same-prefix hijack attacks we perform experimental simulations using the target providers and customers in our dataset. For creating the topological map of the AS-relationship dataset of the customer domains and the providers in our dataset we use CAIDA [3]. We simulate the attacks using a simulator developed in [19]. We evaluate HijackDNS attack for each provider with respect to customer domains of the corresponding provider and an AS level adversary on an Internet topology. In our simulations we consider attacks from 1000 randomly selected ASes against the domains of the customers and providers. The adversary can succeed at the attack against 80% of the Alexa customer domains with 60% success probability. One of the reasons for the high success probability is the concentration of the nameservers in few ASes: 10% of the ASes host 80% of the nameservers in Alexa domains and 1% of ASes host 80% of the domains. The customers of the LIRs are slightly more resilient since they mostly use at least two nameservers on different prefixes. This means that to succeed the attacker would need to hijack both. Furthermore, the distribution of

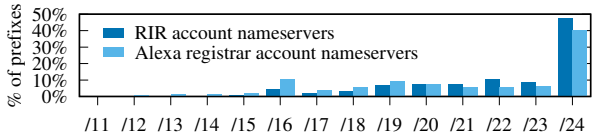


Figure 3: IP prefix distribution of customer accounts' nameservers

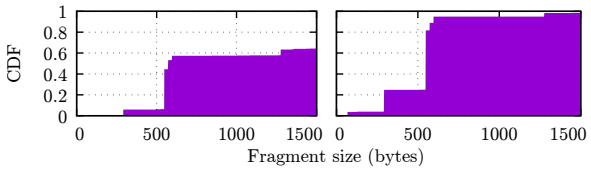


Figure 4: Cumulative distribution of lowest fragment size of nameservers (left) and domains (right) after sending ICMP PTB.

the nameservers across ASes is more uniform in contrast to Alexa domains.

SadDNS & FragDNS. To test vulnerabilities in the providers to SadDNS and FragDNS we perform the evaluations in Sections 4.4 and 4.5. Out of 31 tested providers, 28 (90%) are vulnerable to FragDNS attack and four of the providers are vulnerable to SadDNS attack. Vulnerabilities for each provider are listed in Table 1.

5.4 Measurements of Vulnerable Customers

The success of the attack against a specific victim customer depends not only on the vulnerabilities in the DNS resolver of the provider but also on the properties in the domain of the customer. For instance, say a DNS resolver of some provider is vulnerable to FragDNS attack but the nameservers of the customer's domain do not fragment UDP packets and packets that are too large are transmitted over TCP. In that case, the FragDNS attack is not effective against that customer. To understand the extent of the vulnerabilities in customers we evaluate the attack methodologies in Section 4 against the DNS resolvers that we own and control using the responses from the domains of the customers in our dataset. Using results from this evaluation we can reliably determine if the attack methodology is effective against a customer or not.

The results of our experimental evaluations of attacks in Section 4 and measurements of the customers' domains and their nameservers are summarised in Table 3.

HijackDNS. We analyse the prefixes of the customers similarly to Section 5.3.2. The results are plotted in Figure 3. Our findings are that more than 60% of the domains have all their nameservers on prefixes less than /24. Furthermore, above 20% of the domains host all the nameservers of each domain on a single prefix, as a result, by hijacking one prefix the adversary immediately hijacks the entire domain. Out of these, 17% host all the nameservers on a prefix that is less than /24. 10% of the domains have a single nameserver in the domain. To conclude: more than 40% of the domains are vulnerable to HijackDNS attack via sub-prefix hijack.

SadDNS. Based on the implementation in Section 4.4 we develop an automated simulation of the SadDNS attack, and run it on our dataset of customer domains to compute the success probability of SadDNS attack against our victim DNS resolver. When running the attack for domains that have the required properties (e.g., support rate limiting), poisoning succeeds after an average of 471s (min 39s, max 779s) which is comparable to the original SadDNS results (avg 504s, min 13s, max 1404s, [33]). Our test implementation triggers 497 queries on average for each domain, which is strongly correlated with the attack duration due to the fact that we do not trigger more than two queries per second; the resolvers return SRVFAIL when receiving more than two queries per attack iteration. By inverting this number we get a hitrate of 0.2%.

Our results of an automated evaluation of SadDNS show that 8,469 accounts from the RIRs dataset and 11% of the accounts from the Alexa dataset could be hijacked via the SadDNS method.

FragDNS. We measured the victim customers' nameservers for support of ICMP errors and fragmentation. We send a DNS request to the nameserver for `ANY` type DNS record. After a DNS response, we follow with an ICMP PTB error, that indicates different MTU values, and repeat the request. We check if the response arrived in fragments according to the MTU value indicated in the ICMP error message. We performed evaluations with the following MTU values of 1280, 576, 296 and 68 bytes.

Figure 4 (Left) shows cumulative distribution of fragmented packet size we received after sending ICMP PTB. Right side shows the percentage of the domains where at least one nameserver supported a MTU smaller than the plotted size. As can be seen, for more than 90% of the domains with PMTUD-configured nameservers, at least one nameserver is willing to reduce the fragment size (in response to ICMP PTB) to almost 548 bytes, for roughly 35% domains to 296 bytes and for 10% to 68 bytes. This essentially allows to inflict fragmentation to any size needed. Our evaluations in Section 4.5 indicate that 11964 RIR customer accounts (13.6%) and 2352 Alexa domain holder accounts (22.2%) are vulnerable to the FragDNS attack.

We analyse the attacker's success probability of crafting the spoofed second fragment with the correct UDP checksum and the correct IPID value. To compute the success rate to hit the correct UDP checksum we performed the following evaluation. For each customer domain in our dataset, we query the nameservers of each domain multiple times sending the same DNS request (with the domain of the customer's email and type MX), and check if the DNS responses from the nameservers contain the same DNS records and the same order of DNS records, during each iteration. The computation of the UDP checksum for each domain is described in pseudocode in Algorithm 1. Our evaluation shows that for 1748 domains

	Provider Rank		Mail service provided by	Resolver #	Resolver service provided by	Seen Via Signup	Seen Via PW Rec.	Accept Fragment 1500	1280	576	292	68	BGP prefix-size	DNSSEC do	DNSSEC validate	EDNS size
RIRs	-	AFRINIC	Self	3	Self	✓	✓	✓	✓	✓	✓	✓	/23	✓	✓	4096
	-	APNIC	Self	1	Self	✓	✓	✓	✓	✓	✓	✓	/24	✓	✓	4096
	-	ARIN	Self	4	Self	✓	✓	✓	✓	✓	✓	✓	/24	✓	✓	4096
	-	LACNIC	Self	1	Self	✓	✓	✓	✓	✓	✓	✓	/22	✓	✓	1280
	-	RIPE NCC	Self	3	Self	✓	✓	✓	✓	✓	✓	✓	/12-/23	✓	✓	4096
Registrars	1	godaddy	Self	3	Self	✓	✓	✓	✓	✓	✓	✓	/19-/21	✓	✗	4096
	2	namecheap	SendGrid	64	SendGrid	✓	✓	✗	✗	✓	✓	✓	/12-/23	✓	✗	1232
	3	networksolutions	Self	1	Self	(3)	✓	-	-	-	✓	✓	/20	✗	(1)	512 (2)
	6	enom	Self	17	Self, Google	✓	✓	✓	✓	✓	✓	✓	/20	✓	✗	4096
	9	name.com	Self (AWS)	8	Self (AWS)	✓	✓	✓	✓	✓	✓	✓	/12	✓	✗	4096
	10	Alibaba cloud	Self	11	Self	✓	✓	✓	✓	✓	✓	✓	/16-/21	✓	✗	4096
	11	AWS	Self	46	Self	✓	✓	✓	✓	✓	✓	✓	/12-/21	✓	✗	4096
	12	gandi	Self	3	Self	✓	✓	✓	✓	✓	✓	✓	/23	✓	✓	4096
	13	namesilo	Self	2	Self	✓	(1)	-	-	-	✓	✓	/16-/19	✗	✗	512 (2)
	14	Google Cloud	Self	120	Self	✓	(1)	-	✓	✓	✗	✗	/16-/22	✗	✗	1232
	15	OVHCloud	Self	4	Self	✓	✓	✓	✓	✓	✓	✓	/18-/24	✓	✓	4096
IaaS Providers	1	Amazon AWS	Self	46	Self	✓	✓	✓	✓	✓	✓	✓	/12-21	✓	✗	4096
	2	Microsoft Azure	outlook.com	373	outlook.com	✓	✓	-	-	-	✗	✗	/13-19	✗	✗	512 (2)
	3	Alibaba Cloud	Self	11	Self	✓	✓	✓	✓	✓	✓	✓	/16-/21	✓	✗	4096
	4	Google Cloud	Self	120	Self	✓	(1)	-	✓	✓	✗	✗	/16-/22	✗	✗	1232
	5	IBM Cloud	SendGrid	51	SendGrid	✓	(1)	✗	✗	✓	✓	✓	/12-/23	✓	✗	1232
	(7)	Tencent Cloud	Self	13	Self	✓	✓	✓	✓	✓	✓	✓	/12-/19	✓	✗	4096
	(8)	Oracle Cloud	Self	9	Self	✓	(1)	✗	✗	✗	✗	✗	/17-/23	✓	✓	1372
	-	DigitalOcean	Mailchimp	8	Mailchimp	✓	✓	✓	✓	✓	✓	✓	/17-/22	✓	✗	4096
	-	Linode	Self	2	Self	✓	✓	✓	✓	✓	✓	✓	/17	✓	✓	4096
	-	IONOS	Self	2	Self	✓	(1)	-	-	-	✓	✓	/16	✓	✓	1220
	-	Hostwinds	Postmark	15	OpenDNS	(3)	✓	✗	✗	✗	✗	✗	/19-/21	✓	✓	1410
	-	OVHCloud	Self	4	Self	✓	✓	✓	✓	✓	✓	✓	/18-/24	✓	✓	4096
	-	Vultr	Self	8	Self	✓	✓	✓	✓	✓	✓	✓	/18-/20	✓	✓	4096
	-	CloudSigma	Mailchimp	6	Mailchimp	✓	✓	✓	✓	✓	✓	✓	/17-/22	✓	✗	4096
CAs	1	IdenTrust	Trend Micro	114	Trend Micro	✓	(1)	✓	✓	✓	✓	✓	/15	✓	(1)	4096
	2	digicert.com	Self	137	Self	✓	✓	✓	✓	✓	✓	✓	/16-/22	✓	(1)	4096
	3	sectigo.com	SendGrid	10	SendGrid	(3)	✓	✗	✗	✓	✓	✓	/12-/23	✓	✗	1232
	4	godaddy	Self	3	Self	✓	✓	✓	✓	✓	✓	✓	/19-/21	✓	✗	4096
	5	globalsign.com	(1)	35	Google	✓	(1)	✓	✓	✓	✗	✗	/20	✓	✓	4096

Table 2: Measurement study of provider's DNS resolvers and Email servers. (1): Could not test. (2): No EDNS. (3): No Email after sign-up. -: Does not apply.

Account Provider	# Resources Total	# Resources found e-mail	# Accounts	Vulnerable to BGP sub	BGP same	Sad-DNS	FragDNS any	FragDNS global
Scanned resources				**Vulnerable Accounts**				
RIRs	92,857	69,287 75%	87,547	47,840 56%	n/a n/a	8,469 11%	14,136 17%	1,193 1.5%
Registrars	100,000	10,597 11%	7,450	3,308 45%	n/a n/a	666 10%	1,560 21%	85 1.2%
Both	192,857	79,884 41%	94,997	51,148 56%	n/a 80%	9,135 11%	15,696 17%	1,278 1.4%
Vulnerable resources								
IP Addresses				81%	n/a	30%	51%	21%
AS Numbers				60%	n/a	12%	20%	3%
Domains				47%	n/a	10%	27%	1%
Attack success probability								
Success probability				100%	60%	0.2%	0.1%	20%

Table 3: Customer-side vulnerability data

Algorithm 1: Predictability of records in responses.

```
for each (domain, nameserver) do
    initialise set of different DNS responses as empty;
    for batch = 1,2,...,25 do
        for iteration = 1,2,3,4 do
            send the same DNS request;
            if new response arrived then
                add the new response to the response set;
            end
        end
        if no new responses in last batch then
            break;
        end
    end
    record number of different DNS responses;
end
```

(62%), nameservers always return the same DNS response (with the same records and sorted in the same order); see Figure 5. For our measurement of the IPID allocation methods supported by the nameservers of the customers we use the following methodology. We issue queries from two hosts (with different IP addresses). Data per nameserver is listed in

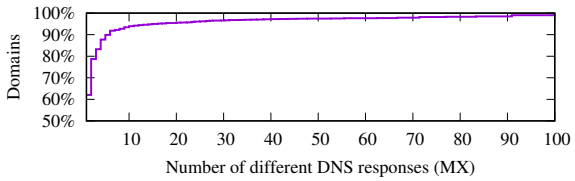


Figure 5: CDF of number of observed DNS MX responses per customer email address domain (each nameserver was queried 100 times).

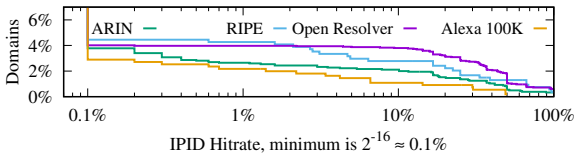


Figure 6: Reverse CDF to correctly guess the IPID for all customers' domains.

Table 4. Our measurements show that 290 vulnerable nameservers (4.88%) use a globally incremental IPID assignment. The computation of the IPID allocation for each domain are described in pseudocode in Algorithm 2.

Algorithm 2: IPID allocation in nameservers.

```
for each (domain, nameserver) do
    for batch = 1,2,3,4 do
        send DNS request from Prober_1;
        record IPID in DNS response as IPID_{2*i-1};
        send DNS request from Prober_2;
        record IPID in DNS response as IPID_{2*i};
    end
    if IPID_i, i = 1,2,...,8 is incrementing then
        globally incrementing;
    end
    if IPID_i, i = 1,3,5,7 or IPID_i, i = 2,4,6,8 is incrementing then
        per-dest incrementing;
    end
    if IPID_i == 0, i = 1,2,...,8 then
        zero;
    else
        random and other;
    end
end
```

	Per-Dest	Global	Zero	Random and other	N/A	Total
All	64.58% 45308	8.31% 5829	4.89% 3434	11.92% 8364	10.30% 7223	100% 70158
Frag	53.96% 3206	4.88% 290	13.75% 817	23.67% 1406	3.74% 222	100% 5941

Table 4: IPID allocation of all nameservers and of fragmenting nameservers.

We automate the attack in Section 4.5 and execute the entire FragDNS attack against all the vulnerable customer domains, by injecting malicious records mapping Email servers of customers to an IP address of our adversarial host. Our evaluation combines the data we collected on DNS records in responses (randomisation of the DNS records or of their order in responses) and the IPID allocation of the nameservers. We also used Algorithm 2 to estimate the IPID increment rate, by recording the timestamp of each response and calculating the average increment rate of IPID value. We then extrapolate the value of IPID and calculate the probability of our adversary to correctly place at least one out of 64 fragments[3] with the matching IPID in the resolver's defragmentation cache. We use different values for IPID increment rate and delay between the query, which probes the IPID value, and the IPID value that was de-facto assigned to the DNS response by the nameserver. Results are plotted in Figure 6. For example, the IPID prediction success rate is over 10% for roughly 3% of RIPE, 2% of ARIN and 1% of 100K-top Alexa customers. Success rates for ARIN and 100K-top Alexa customers are lower mostly because of the higher latencies of those, see Figure 2. For nameservers which do not use globally incremental IPID, we assume a hitrate of $64/2^{16}$ which is achieved by just randomly guessing the IPID.

The probability to compute the correct checksum is capped at a minimum of $1/2^{16}$ in case of nameservers which generate responses with different records or with random ordering of records. Finally the probabilities to correctly compute both, the IPID value and the order of records to get the correct UDP checksum, are multiplied resulting in the combined hitrate.

Our automated attack against all the customers shows that around 2% of the domains (5 for RIPE, 17 for ARIN) have a success probability higher than 10%. Furthermore, for about 20% of the domains, success probability is over 0.1% which is a consequence of non-predictable IPID allocation and the stable DNS records in responses generated by these domains. When the DNS response can be predicted, even with a random IPID allocation method, an attacker has a hitrate of about $64/2^{16} \approx 0.1\%$. At this hitrate, when the attacker performs the attack multiple times, the probability to conduct the attack successfully at least once is 50% at around 700 repetitions.

Our automated evaluation provides a lower bound for successful attacks against a randomly chosen domain – this is a worst case analysis since it also considers domains which are much more difficult to attack, e.g., since they use servers with random IPID allocation, servers with high traffic rate, and servers which return different number and order of records in responses. Adjusting the attack parameters manually against a given victim customer domain results in a much higher attack rate. Furthermore, against many customer domains with low traffic volume, incremental IPID values and fixed number and order of DNS records, the attacker can reach above 90% success rate.

6 Manipulation of Digital Resources

In this section we demonstrate exploits that the adversary can perform when controlling an account of a (victim) customer. Most of the actions are similar across the providers, even providers of different infrastructure, such as RIRs and the

[3] 64 fragments is the minimal size of the IP-defragmentation cache.

Additonal Validation	Attack		RIRs	Registrars	IaaS	CAs	Outcome / Attacker use
RIRs	Account transfer/delegation		✓	✓	✓	✗	permanent control
No	Changing the account details		✓	✓	✓	✓	permanent control
RIRs	Close the account permanently		✓	✓	✓	✓	DoS
No	Disabling Email alerts		✓	✓*	✗	✓*	remain stealthy
RIRs	Resource transfer		✓	✓	✓	✗	permanent control
			✓	✓	✗	✗	sell resources
No	Resource return / deletion		✓	✓	✓	✓	DoS
CAs	Purchase new resources		✓	✓	✓	✓	financial Damage
			✓	✓	✓	✓	anonymous usage
No	Control / Modify Resources	Whois DB	✓	✓	✗	✗	facilitates hijacking
		VMs	✗	✗	✓	✗	various
		NS records	✗	✓	✗	✗	traffic hijacking
No	Create new ROAs/certificates		✓	✗	✗	✓	facilitates hijacking
No	Create invalid ROAs		✓	✗	✗	✗	DoS
No	Revoke certificates		✗	✗	✗	✓	DoS

Table 5: Actions an attacker can carry out after hijacking a customer's account. *The Email address where the alert is sent can be changed. Additional validation requires either that additional documents are sent, or in case of issuing a new TLS certificate, that a domain validation must be passed.

domain registrars. Hence, we in details explain our demonstration of the exploits by taking over a victim LIR account, and then briefly describe the exploits we evaluated by taking over our victim accounts with the other providers. For our demonstration we select RIPE NCC RIR, GoDaddy domain registrar, Microsoft Azure IaaS provider and DigiCert CA. In order to evaluate the exploits using an account of a network operator, we cooperate with a large customer under RIPE NCC. We cooperate with that LIR and use a real account that has an *operator/administrator* role[4]. For domain registrars, IaaS and CAs, we used our own accounts which were used to buy test resources to test the possibilities of the account. We summarise the exploits for different providers in Table 5.

For our evaluation of the exploits, we first carry out our attacks in Section 4 to take over the victim accounts, and then carry out the exploits. We do this in order to understand what notifications are sent to the genuine account owners during such attacks, what actions can be performed, and which are prevented.

6.1 Regional Internet Registries

We show that adversary, controlling an SSO account of a victim LIR, can manipulate all the Internet resources associated with that LIR, e.g., the IPv4 and IPv6 addresses, ASNs, reverse DNS names to IP addresses mappings. The amount of resources managed by LIRs can vary enormously. There are small LIRs that manage just own AS, one IPv6 prefix and one or a very few IPv4 prefixes of minimal allocation size. There are also large LIRs managing vast PA address pools (Provider Aggregatable Addresses) for multiple clients. For instance, RIPE NCC has cumulative allocation of 587 202 560 IPv4 addresses.

The adversary impersonating an administrator with the RIPE NCC SSO account holder can initiate different actions that lead to disruption or degradation of the services that are tied to the IP resources managed by the victim LIR. The adversary can even initiate transfer of IPv4 addresses that belong to the victim LIR to obtain direct financial benefit from that process. Our experimental evaluation with an SSO account under RIPE NCC RIR shows that the actions of the attacker do not trigger alerts and can be detected when the LIRs realises that its digital resources are gone. The access to RIPE NCC SSO account with *operator* or *administrator* roles for the victim's LIR opens to a range of possible exploits. We explain selected exploits with an example victim LIR under RIPE NCC below (also summarised in Table 5); the attacks similarly apply to the other RIRs.

RPKI administration. Attacker creates/deletes/modifies Route Origin Authorizations (ROAs) in hosted RPKI system. This has two purposes: (1) to disrupt the propagation of the legitimate BGP updates of the resources managed by the victim LIR and (2) to facilitate BGP hijacking by authorising attacker's ASN to originate any subset of IP prefixes that are managed by the victim LIR. Networks which have deployed RPKI and perform filtering of BGP announcements with ROV will not trigger any alerts when the attacker issues a BGP announcement for a sub-prefix with a valid (yet fraudulent) ROA. We consider creation of ROA with origin set to ASN0 ("always drop" as per [RFC7607]) for a specific prefix within the resource pool managed by the victim LIR to be a special case of the malicious ROA intended to disrupt routing and cause DoS for the services tied to the IP addresses in question. Our measurements found that currently the Route-Origin Validation (ROV) is far from being universally deployed, with only 2190 ASes filtering invalid BGP announcements. Nevertheless, this is an increase in contrast to measurements from a few years ago, which found 71 ASes to validate ROV [20, 36]. Even with 2000 validating ASes, this type of attack is likely to cause only minor disruption in service availability and will remain unnoticed for extended period of time.

RIPE DB modifications. Attacker manipulates records in RIPE DB - the Internet addressing resource registry of the region and Internet Routing Registry (IRR) in one converged database. Modification of records in resource registry allows impersonation of the victim LIR's representatives in order to transfer resources from the victim LIR to unsuspecting recipient.

IRR records are prerequisite for BGP hijacking attacks, because without proper records in IRR the attacker would not be able to persuade any well-managed upstream provider that is consistent with AS operation best practices to accept the

[4]RIPE NCC Single Sign-On (SSO) accounts are general authentication mechanism for all web-based services provided by RIPE NCC that include customer portal and other harmless services - for example RIPE Meeting facilitation.

fraudulent BGP announcements in the input filters on the BGP sessions. Attackers without the ability to modify IRR have to use less vigilant and generally poorly managed networks as upstream providers or have to utilise path manipulation attacks - both restrict success rate and stealthiness of the attack.

Creating the IRR records contradictory to the state in BGP is a way to partially disrupt route propagation and thus traffic. It can also de-stabilise network and significantly complicate network operation for the legitimate administrators. Route servers in majority of Internet Exchanges and major networks use IRR data in automatically generated filters that are applied on incoming BGP announcements. As a result of the contradicting IRR records these networks will drop or de-prefer the announcements from the legitimate resource holder. Moreover, well-managed networks keep manually generated import filters on small-scale BGP sessions for both peering and downstream customers. When a new session is set up or when a new prefix is about to be propagated from the neighbouring AS that is subject to filtering, the administrators manually check the IRR and resource registry to verify that the announcement is legitimate. Failing to have the proper records in IRR and in resource registry leads to refused BGP peerings, excessively strict BGP filters and therefore to dropped routes and overall degradation of the Internet connectivity for the victim network.

Initiating IPv4 addressing resource transfer. Attacker sells the resources managed by the victim LIR. The potential gain from successfully completed attacks of this type is determined by the amount of the addresses managed by the LIR and the expected monetary value of IPv4 addressing resources. After the IPv4 regular pool depletion on 4 September 2012 each LIR is eligible for allocation of a single */22* (1024 IPv4 addresses) prefix as per current IPv4 Address Allocation and Assignment Policies for the RIPE NCC Service Region (ripe-720) [1]. We performed IPv4 /22 prefix transfer of an LIR under RIPE NCC which has not triggered alerts. This is not surprising since the IPv4 addresses' transfer is performed regularly by the RIRs, e.g., RIPE NCC and ARIN perform thousands of transfers per year, see statistics on IP transfer (PI and PA) we collected from the RIRs in Figure 7.

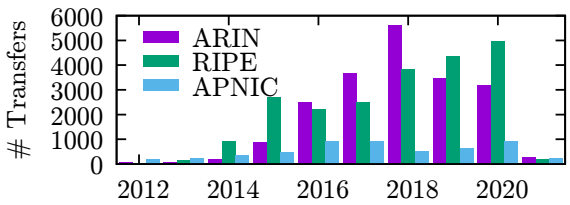


Figure 7: IPv4 resource transfers per year (PI & PA).

To sell the IPv4 addresses belonging to the victim LIR, we needed to perform the following:

(1) modify the relevant LIR contacts (Emails, phone numbers) in the RIR database and the IRR to receive the communication from the RIR intended to the genuine LIR, the buyer as well as the other parties relevant to the IP resource transfer and prevent the victim to learn about the resource transfer process. This is performed via the victim SSO account which the attacker took over.

(2) find the buyer for the resources and to impersonate the victim representatives to successfully close an IPv4 resource transfer contract; we used the compromised SSO account to sell and transfer the resources to an LIR that we set up for that purpose. In practice, the attacker can also collude with a malicious adversary, which will perform the transaction and afterwards will legally own the resources. The victim will need to prove to not have authorised the payment. In fact the resolution outcome of such a case is not clear since the RIRs have not faced this attack before.

(3) release the IPv4 prefix from the BGP routing table. This needs to be done so that the buyer believes that the resources are free and are being legitimately sold. This is done by sending an Email (via the Email address that the attacker modified in IRR and registry DB) to the upstream provider of the victim. The list of the upstream providers is available and can be obtained from the IRR. In the Email the attacker instructs the upstream provider to update the input filters and drop the network (that the attacker wishes to sell). Such requests are common, and do not require authentication or verification of requester's identity (e.g., with PGP/PKI) and the *sole source of truth that is checked prior applying the filters is RIPE DB* which, as we mentioned, the attacker can update via the SSO account.

Notice that if the buyer is colluding with the attacker - this step is not needed.

(4) Email a scanned IPv4 resource transfer contract to RIPE NCC. The contract has to match the company details of the victim and contain all formalities and certifications appropriate to the legal system in which the contract has been made. Moreover the contract has to be supplemented by extract from chamber of commerce or appropriate commercial registry that makes it possible to establish that the contract is signed by the eligible persons on both sides. This is simple to forge - the attacker can find and copy the signatures of the owner LIR online and paste them into the scanned contract document that the attacker prepares.

Extract from chamber of commerce is also simple to forge - for instance, in Czech Republic (CZE) the attacker has to go to any post office, pay 1 EUR and get either paper version with a stamp or PDF with PKI signature of state-operated CA. In any case, since the attacker only needs to send a scanned version of the document to RIPE, the attacker can get the document for any company and adjust it using Photoshop and it is accepted.

Finally, notice that the RIRs have limited personnel which, depending on the RIR, may need to deal with tens of transfers per day, see Figure 7. As a result, the adversary may often manage to sell the resources without raising alarms even

when not satisfying these simple four steps. For instance, RIPE NCC has just 24 employees responsible for IP address distribution[5] and there are more than 20 transfers per day, see Figure 7.

User and role management. Attacker that controls an account with *administrator* role assigns other newly created users either *operator* or *administrator* roles for the victim LIR. This effectively hides the activities of the attacker for long periods of time even though the legitimate holder is actively operating the LIR.

Modification of the LIR contacts and details. There are two sets of Email and postal addresses and phone numbers related to the LIR - the first set is published in IRR and RIPE DB and it is tied to the resources and published to facilitate operation of the network and solving technical problems and those contacts are also used by the RIPE DB software itself for generating notifications about changes in RIPE DB. The second set contains the designated LIR contact information, namely the primary point of contact for the LIR and a contact for billing-related matters. Moreover, there is a postal address, that may differ from legal address of the LIR company and it can be modified in LIR portal. The attacker can redirect or change the LIR contact information to avoid detection by the victim LIR staff when activities that result in notifications or follow-up Emails are to be executed. Modifying LIR contacts will also make any attempt to rectify damages caused by the attack, when detected, harder.

Termination of LIR membership. Attacker initiates termination of LIR membership with the RIPE NCC by submitting a forged termination request via a written notice sent by Email. Forgery cases are not new and have already been seen in the past[6].

Modification of LIR organisation name, legal address and VAT number. The attacker steals the LIR and all its IP addressing resources by pretending a transfer of ownership to other company. A scan of (the forged) contract of company acquisition has to be attached to the request in electronic form.

Requesting new or voluntarily returning IP addressing resources. The attacker requests or returns IP addressing resources from or to RIPE NCC. If the LIR is eligible for allocation of any scarce resources, the attacker obtains a new IP prefix that is not used in default-free zone (DFZ) and thus fulfils the prerequisite for transfer of not being announced. However, according to the current policy the newly obtained allocation of scarce resources (IPv4 addresses, 16-bit ASN) can not be transferred within 24 months from the allocation. The attacker can nonetheless hijack the resources for own purposes immediately and attempt to sell the resources after the grace period.

[5] https://www.ripe.net/about-us/staff/structure/registration-services
[6] https://mailman.nanog.org/pipermail/nanog/2011-August/039379.html

6.2 Domain Registrars

Domain registrars handle the ownership of domains in name of the customer. We map the 100K-top Alexa domains to registrars with whom these domains are registered in Table 1. We demonstrate exploits that the attacker can perform when taking over an account with GoDaddy[7]. The adversary can change the nameservers' IP addresses, which allows it to hijack the victim customer domain. This can be exploited to redirect clients to phishing websites. The adversary can delegate account access to itself or perform intra-and inter-registrar[8] domain transfer. The adversary can change the Email forwarding settings of the account which would allow it to hijack the Emails forwarded to the owner of the Email address. The adversary can also delete the domains associated with the compromised account and even close the account. The account owners can enable two-factor authentication for manipulation of resources associated with their account. However, this is not enabled by default, and is up to each customer to enable it.

6.3 Infrastructure as a Service

We evaluated the exploits that the adversary can carry out on resources associated with accounts at cloud providers. The adversary can manipulate virtual resources associated with the account, such as virtual machines, network interfaces, disk. The adversary can also exploit these resources to carry out attacks against victims in the Internet or victims located on the same cloud platform, e.g., via side channels [39]. In addition the adversary can create new accounts with owner privileges or transfer subscription to another Azure account.

6.4 Certificate Authorities

When controlling an account of a customer with a CA the adversary can revoke certificates and reissue existing certificates that were issued under that account. This allows to change the key-pair associated with a certificate. Nevertheless, some CAs, do not enforce any validation on reissuing certificates, in contrast to validation (domain validation, organisation validation or extended validation) that is enforced when requesting to issue a certificate for the first time. Therefore, instead of attacking domain validation procedure of the CAs it is more profitable to hijack a victim customer account and change the keys associated with the certificates for domains that the adversary targets. DigiCert and GoDaddy do not perform additional validation on requests to reissue certificates. Sectigo, GlobalSign, IdenTrust validate all requests to reissue certificates.

[7] The other domain registrars allow similar actions.
[8] It takes 60 days for an inter-registrar transfer to finalise.

7 Vulnerable Digital Resources

The large fraction of the accounts under different providers that can be hijacked is alarming. Even more disturbing are the exploits that the adversaries can do with the resources assigned to the accounts. *What is the extent of the Internet resources that are at immediate risk due to the vulnerable providers and vulnerable customers?*

To answer this question we perform a correlation between the accounts that our study found to be vulnerable to hijacks via any of the attack methodologies in Section 4 and the digital resources (domain names, IP addresses and ASNs) that the attacker can take over as a result of hijacking that account. In our analysis we consider only the domain registrars and the RIRs and their customers. Since there is no public database of customers of cloud providers and certificate authorities we exclude them from this analysis[9]. We list the correlation between the vulnerable resources and the vulnerable accounts in Table 6.

IP resources. We compute the fraction of the assigned AS Numbers (ASNs) as well as assigned IPv4 address space which could be taken over by hijacking the vulnerable accounts of customers of RIRs. For this purpose, we combine IP-to-ASN and ASN-to-LIR mappings with our customer vulnerability data, which allows us to evaluate vulnerabilities in 73% of the assigned IPv4 address space (for 27% we could not extract LIR account information). Our results show that using any of the attack methodology in Section 4 the adversary could take over 68% of the IPv4 address space. This constitutes 93% of the address space assigned to the accounts in our dataset. Even the weaker attack methodologies (FragDNS and SadDNS), which do not require controlling a BGP router, allow the adversaries to take over 59% of the address space. Similarly, 74% of the ASNs are associated with the accounts that can be hijacked via any of the DNS cache poisoning attacks in Section 4 and 30% with the SadDNS or FragDNS attack. The difference between the vulnerability volume for IP addresses and ASNs is due to the fact that large parts of the IPv4 address space is owned by a small number of ASes, e.g., 21% of the assigned IPv4 address space is attributed to the top 10 LIR accounts.

Domain resources. We use our domain-to-account mapping to determine user accounts at registrars. This includes 11% of the accounts for which we were able to extract customer account information. We believe however that the fraction of the vulnerable accounts is representative of all the 1M-top Alexa domains since the vulnerabilities only depend on the nameservers of the customers' domains. Our study shows that 65% of the domains could be hijacked via any of HijackDNS, SadDNS or FragDNS, while 35% could be hijacked via SadDNS or FragDNS.

[9]These can be collected via a dictionary attack against the provider: the adversary inputs usernames and checks for error messages. Such a study however creates a significant load on the infrastructure of the provider.

	HijackDNS	SadDNS	FragDNS	Any	SadDNS or FragDNS
IP addresses	81%	30%	51%	93%	59%
Domains	47%	10%	27%	65%	35%

Table 6: Vulnerable resources mapped to accounts in our dataset.

Countermeasure	Layer	Provider- / Customer-side	FragDNS	SaDDNS	HijackDNS
2-FA TAN with out-of-band notif.	Web portal	both[1]	✓	✓	✓
2-FA login	Web portal	provider	✓	✓	✓
IP-level account access restrictions	Web portal	both	✓	✓	✓
DNSSEC signing and validation	DNS	both	✓	✓	✓
Disable/Patch ICMP rate-limit	IP	provider	✗	✓	✗
Disable NS rate-limit	DNS	customer	✗	✓	✗
Disable PMTUD	IP	customer	✓	✗	✗
Blocking Fragments	IP	provider	✓	✗	✗
MTA-STS [RFC8461]	Email	both	✓	✓	✓
Hide public account details	General	both	✓	✓	✓
Request rate-limiting	Web portal	provider	✓	✓	✗
Captchas	Web portal	provider	✓	✓	✗
Separate systems	Web portal	provider	✓	✓	✗
Resolver hardening	DNS	provider	✓	✓	✗
Non-predictable IPID increment	IP	customer	✓	✗	✗
Out-of-band notifications	Web portal	provider	✓	✓	✓

Table 7: Countermeasures against different types of attackers. [1]: requires the user to verify the out-of-band delivered transaction details before entering the TAN.

8 Recommendations for Countermeasures

The fundamental problem that our attacks outline is the stealthiness and ease with which the adversaries can apply changes and manipulations over the Internet resources of providers of digital resources. Since Internet resources form the foundations for the stability and security of democratic societies, our work calls for a revision of the current practices of resource management and development of techniques that would secure the transactions over the Internet resources. For instance, selling Internet blocks should not happen immediately, and should require more than merely a scanned document over Email (which is easy to fake). In addition to the standard recommendations for hardening the DNS caches or blocking ICMP error messages, which we summarised in Table 7, we also provide recommendations for best practices for providers and customers.

Separate system for high privileged users. Currently, any user can create an account with most of the providers. The accounts can be used for managing Internet resources (high privileged) as well as for registering for events or mailing lists (low privileged). Low privileged accounts in the user management system have access to the same infrastructure (Email servers, DNS resolvers, etc) as the high privileged accounts, such as those of network operators. This enables adversaries to open low privileged accounts and use them to collect information about the infrastructure of the provider. The providers

should use separate user management systems and a separate set of servers for users which own digital resources vs. users that, e.g., are registered to mailing lists or events.

Two-factor authentication. Two factor authentication (2-FA) systems must be enabled by default. The two authentication factors must be independent of each other and an attacker should not be able to compromise both factors within a single attack. This for instance, rules out Email-based 2-FA for password recovery which is available at some of the providers we tested.

Deploy captchas. Our study shows that most providers do not use captchas, e.g., three out of five RIRs do not use captchas. Although captchas do not prevent the attack, they force the attacker to run manual tests making the attack more expensive to launch. Resolving the captchas is tedious and burdensome for the attacker (as well as for the researchers) to carry out in contrast to automated study of the victims. For instance, for studying vulnerabilities in DNS caches and for performing cache poisoning attacks we needed to run multiple password recovery procedures for triggering DNS requests to our domain. This study could not be automated in RIRs that use captchas.

Notifications of modifications. Changes performed over the resources of providers either do not generate any notifications or generate notifications to the Email configured in the compromised account. First, the Email notifications will be received by our adversary, since it hijacked the victim domains in the resolver of the provider, and second, the adversary can change the contact Email in the account, and even disable notifications. The accounts with providers should be associated with contact Email which cannot be changed through the account and which is different than the one used to access the account.

Email address masking. The Email addresses in the `whois` records of the domains should be masked. Some of the domain registrars are already following this practice.

Account level IP address access restriction. The registrars should restrict account access to only few static IP addresses belonging to the domain's owner.

Deploy DNSSEC. DNSSEC ([RFC4033] to [RFC4035]) would essentially make the attack methodologies in Section 4 practically impossible. Unfortunately, only 3.78% domains of customers of RIRs and 5.88% domains of customers of registrars are correctly signed. For instance, out of 1832 LIR domains under AFRINIC, only 58 are signed, and 27 of these domains are still vulnerable since the DNS resolvers cannot establish a chain of trust to them from the root anchor. Further, 12 use weak cryptographic keys (below 512 bits) and 12 use weak (vulnerable) hash functions. The remaining 95 domains out of those 1832 domains were not responsive. Unfortunately, even when the domain is signed and the resolver validates DNSSEC, as long as the human factor is in the loop, there is risk for vulnerabilities and misconfigurations, [12,42]. Hence we recommend that the providers and customers deploy additional measures that we list in this section to harden their infrastructure.

9 Conclusions

Each provider maintains a database that defines which customer owns which Internet resources and offers tools for the customers to manage their resources. We showed that these databases are poorly protected - the adversaries can take over the accounts for managing the Internet resources and can manipulate the databases, e.g., creating new or removing existing objects - stealthily and causing immediate changes to the customers' resources.

For our attacks we used different DNS cache poisoning methodologies and compared their applicability and effectiveness for taking over accounts. Our work shows that while challenging, our attacks are practical and can be applied against infrastructure of a large fraction of the resource providers to hijack accounts. Our results demonstrate feasibility even with weak off-path adversaries. Certainly, accounts associated with Internet resources are an attractive target also for stronger Man-in-the-Middle adversaries, such as cybercriminal groups or nation state attackers.

We described countermeasures for mitigating our off-path attacks for taking over the accounts of customers. Addressing the fundamental problem - easy manipulation of the Internet resources - requires creating policies and revising the Internet management infrastructure as well as techniques for securing the transactions applied over Internet resources.

Acknowledgements

This work has been co-funded by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) SFB 1119.

References

[1] ripe-720: Ipv4 address allocation and assignment policies for the ripe ncc service region. https://www.ripe.net/publications/docs/ripe-720. Accessed: 2019-6-13.

[2] BGPStream by CAIDA. https://bgpstream.caida.org/, July 2016.

[3] The CAIDA AS Relationships Dataset. http://www.caida.org/data/as-relationships/, January 2016.

[4] Louis Poinsignon. BGP leaks and cryptocurrencies, 2018.

[5] F. Alharbi, J. Chang, Y. Zhou, F. Qian, Z. Qian, and N. Abu-Ghazaleh. Collaborative client-side dns cache poisoning attack. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019.

[6] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. Hijacking bitcoin: Routing attacks on cryptocurrencies. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 375–392. IEEE, 2017.

[7] Dan J. Bernstein. DNS Forgery. http://cr.yp.to/djbdns/forgery.html, November 2002.

[8] Henry Birge-Lee, Yixin Sun, Anne Edmundson, Jennifer Rexford, and Prateek Mittal. Bamboozling certificate authorities with BGP. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[9] Henry Birge-Lee, Liang Wang, Daniel McCarney, Roland Shoemaker, Jennifer Rexford, and Prateek Mittal. Experiences deploying multi-vantage-point domain validation at let's encrypt. December 2020.

[10] Peter Boothe, James Hiebert, and Randy Bush. Short-lived prefix hijacking on the internet. *In Proc. of the NANOG*, 36, 2006.

[11] Markus Brandt, Tianxiang Dai, Amit Klein, Haya Shulman, and Michael Waidner. Domain Validation++ For MitM-Resilient PKI. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 2060–2076. ACM, 2018.

[12] Taejoong Chung, Roland van Rijswijk-Deij, Balakrishnan Chandrasekaran, David Choffnes, Dave Levin, Bruce M Maggs, Alan Mislove, and Christo Wilson. A longitudinal, end-to-end view of the dnssec ecosystem. In *USENIX Security*, 2017.

[13] Taejoong Chung, Roland van Rijswijk-Deij, David Choffnes, Dave Levin, Bruce M Maggs, Alan Mislove, and Christo Wilson. Understanding the role of registrars in dnssec deployment. In *Proceedings of the 2017 Internet Measurement Conference*, pages 369–383, 2017.

[14] Tianxiang Dai, Philipp Jeitner, Haya Shulman, and Michael Waidner. From IP to Transport and Beyond: Cross-Layer Attacks Against Applications. In *SIGCOMM '21: Proceedings of the 2021 Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication, Virtual Event, USA, August, 2021*. ACM, 2021.

[15] Chris C Demchak and Yuval Shavitt. China's maxim–leave no access point unexploited: The hidden story of china telecom's bgp hijacking. *Military Cyber Affairs*, 3(1):7, 2018.

[16] Doug Madory. Recent Routing Incidents: Using BGP to Hijack DNS and more, 2018.

[17] Amir Herzberg and Haya Shulman. Security of Patched DNS. In *Computer Security - ESORICS 2012 - 17th European Symposium on Research in Computer Security, Pisa, Italy, September 10-12, 2012. Proceedings*, pages 271–288, 2012.

[18] Amir Herzberg and Haya Shulman. Fragmentation Considered Poisonous: or one-domain-to-rule-them-all.org. In *IEEE CNS 2013. The Conference on Communications and Network Security, Washington, D.C., U.S.* IEEE, October 2013.

[19] Tomas Hlavacek, Italo Cunha, Yossi Gilad, Amir Herzberg, Ethan Katz-Bassett, Michael Schapira, and Haya Shulman. Disco: Sidestepping rpki's deployment barriers. In *Network and Distributed System Security Symposium (NDSS)*, 2020.

[20] Tomas Hlavacek, Amir Herzberg, Haya Shulman, and Michael Waidner. Practical experience: Methodologies for measuring route origin validation. In *48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2018, Luxembourg City, Luxembourg, June 25-28, 2018*, pages 634–641, 2018.

[21] P Hoffman and P McManus. Rfc 8484: Dns queries over https (doh), 2018.

[22] Z Hu, L Zhu, J Heidemann, A Mankin, D Wessels, and P Hoffman. Rfc 7858-specification for dns over transport layer security (tls), 2016.

[23] P. Jeitner, H. Shulman, and M. Waidner. the impact of dns insecurity on time. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*.

[24] Philipp Jeitner and Haya Shulman. Injection Attacks Reloaded: Tunnelling Malicious Payloads over DNS. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, August 2021.

[25] Josh Aas and Daniel McCarney and and Roland Shoemaker. Multi-Perspective Validation Improves Domain Validation Security, 2020.

[26] Dan Kaminsky. It's the End of the Cache As We Know It. Presentation at Blackhat Briefings, 2008.

[27] Josh Karlin, Stephanie Forrest, and Jennifer Rexford. Autonomous security for autonomous systems. *Computer Networks*, 52(15), 2008.

[28] Varun Khare, Qing Ju, and Beichuan Zhang. Concurrent prefix hijacks: Occurrence and impacts. In *Proceedings of the 2012 Internet Measurement Conference*, pages 29–36. ACM, 2012.

[29] Amit Klein. Bind 9 dns cache poisoning. *Report, Trusteer, Ltd*, 3, 2007.

[30] Amit Klein. Windows dns server cache poisoning,", 2007.

[31] Amit Klein. Cross layer attacks and how to use them (for dns cache poisoning, device tracking and more). *arXiv:2012.07432*, 2020.

[32] Aanchal Malhotra and Sharon Goldberg. Attacking NTP's Authenticated Broadcast Mode. *ACM SIGCOMM Computer Communication Review*, 46(1):12–17, May 2016.

[33] Keyu Man, Zhiyun Qian, Zhongjie Wang, Xiaofeng Zheng, Youjun Huang, and Haixin Duan. DNS Cache Poisoning Attack Reloaded: Revolutions with Side Channels. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020.

[34] Carolyn Duffy Marsan. Six worst internet routing attacks, 2009.

[35] Asya Mitseva, Andriy Panchenko, and Thomas Engel. The state of affairs in bgp security: A survey of attacks and defenses. *Computer Communications*, 124:45–60, 2018.

[36] Andreas Reuter, Randy Bush, Ítalo Cunha, Ethan Katz-Bassett, Thomas C. Schmidt, and Matthias Wählisch. Towards a rigorous methodology for measuring adoption of RPKI route validation and filtering. *CoRR*, abs/1706.04263, 2017.

[37] Jennifer Rexford, Jia Wang, Zhen Xiao, and Yin Zhang. Bgp routing stability of popular destinations. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 197–202. ACM, 2002.

[38] RIPE NCC. RIS Raw Data, 2021.

[39] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 199–212, 2009.

[40] Haya Shulman. Pretty bad privacy: Pitfalls of dns encryption. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, pages 191–200, 2014.

[41] Haya Shulman and Michael Waidner. Towards security of internet naming infrastructure. In *European Symposium on Research in Computer Security*, pages 3–22. Springer, 2015.

[42] Haya Shulman and Michael Waidner. One key to sign them all considered vulnerable: Evaluation of dnssec in the internet. In *NSDI*, 2017.

[43] Sandra Siby, Marc Juarez, Claudia Diaz, Narseo Vallina-Rodriguez, and Carmela Troncoso. Encrypted dns–> privacy? a traffic analysis perspective. *arXiv preprint arXiv:1906.09682*, 2019.

[44] University of Oregon. Route views project. http://bgplay.routeviews.org/, 2012.

[45] Paul Vixie. DNS and BIND security issues. In *Proceedings of the 5th Symposium on UNIX Security*, pages 209–216, Berkeley, CA, USA, jun 1995. USENIX Association.

[46] S Weiler and D Blacka. Rfc 6840: Clarifications and implementation notes for dns security (dnssec). *IETF Standard*, 2013.

[47] Xiaofeng Zheng, Chaoyi Lu, Jian Peng, Qiushi Yang, Dongjie Zhou, Baojun Liu, Keyu Man, Shuang Hao, Haixin Duan, and Zhiyun Qian. Poison over troubled forwarders: A cache poisoning attack targeting DNS forwarding devices. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 577–593, 2020.

2021 30th USENIX Security Symposium
Accepted version. https://www.usenix.org/conference/usenixsecurity21/presentation/dai

E. From IP to Transport and Beyond: Cross-Layer Attacks Against Applications

[5] T. Dai, P. Jeitner, H. Shulman, and M. Waidner, "From ip to transport and beyond: Cross-layer attacks against applications," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, ser. SIGCOMM '21, CORE A*, Virtual Event, USA: Association for Computing Machinery, 2021, pp. 836–849, ISBN: 9781450383837. DOI: 10.1145/3452296.3472933. [Online]. Available: https://doi.org/10.1145/3452296.3472933.

From IP to Transport and Beyond: Cross-Layer Attacks Against Applications

Tianxiang Dai
Fraunhofer SIT
Germany

Philipp Jeitner
Fraunhofer SIT
TU Darmstadt
Germany

Haya Shulman
Fraunhofer SIT
Germany

Michael Waidner
Fraunhofer SIT
TU Darmstadt
Germany

ABSTRACT

We perform the first analysis of methodologies for launching DNS cache poisoning: manipulation at the IP layer, hijack of the inter-domain routing and probing open ports via side channels. We evaluate these methodologies against DNS resolvers in the Internet and compare them with respect to effectiveness, applicability and stealth. Our study shows that DNS cache poisoning is a practical and pervasive threat.

We then demonstrate cross-layer attacks that leverage DNS cache poisoning for attacking popular systems, ranging from security mechanisms, such as RPKI, to applications, such as VoIP. In addition to more traditional adversarial goals, most notably impersonation and Denial of Service, we show for the first time that DNS cache poisoning can even enable adversaries to bypass cryptographic defences: we demonstrate how DNS cache poisoning can facilitate BGP prefix hijacking of networks protected with RPKI even when all the other networks apply route origin validation to filter invalid BGP announcements. Our study shows that DNS plays a much more central role in the Internet security than previously assumed.

We recommend mitigations for securing the applications and for preventing cache poisoning.

CCS CONCEPTS

• **Security and privacy** → **Network security**.

KEYWORDS

DNS Cache Poisoning, Fragmentation, BGP hijacking, Side Channels

ACM Reference Format:
Tianxiang Dai, Philipp Jeitner, Haya Shulman, and Michael Waidner. 2021. From IP to Transport and Beyond: Cross-Layer Attacks Against Applications. In *ACM SIGCOMM 2021 Conference (SIGCOMM '21), August 23–28, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3452296.3472933

1 INTRODUCTION

Domain Name System (DNS), [RFC1034, RFC1035] [59, 60], plays a central role in the Internet. Designed and standardised in the 80s to provide lookup services DNS has evolved into a complex infrastructure and is being increasingly used to support a wide variety of existing and future applications and security mechanisms. Given the large dependency of the Internet on DNS it also became a lucrative target for attacks.

DNS cache poisoning. In a cache poisoning attack an adversary injects malicious DNS records into the cache of a victim DNS resolver. Poisoning the cache enables the adversary to redirect the victims using that DNS resolver to malicious hosts instead of the genuine servers of the target domain. As a result, the adversary intercepts all the services in the target domain.

In this work we explore how practical off-path DNS cache poisoning attacks are and how such attacks can be exploited to launch cross-layer attacks against applications.

Taxonomy of cache poisoning methodologies. As we explain in Section 2, off-path DNS cache poisoning is challenging to launch in practice. Nevertheless, there are methodologies that, depending on different conditions, can result in practical attacks. In this work we evaluate such methodologies for launching cache poisoning attacks: (1) BGP prefix hijacking, (2) transport layer side channels and (3) injections into IP defragmentation cache. These methodologies were previously used for issuing fraudulent certificates, [21] or for hijacking bitcoins [16]. Attacks for issuing fraudulent certificates were also carried out by [23] using IP fragmentation; a method initially proposed in [38]. [57] combined ICMP error messages and rate limiting of nameservers to create a side channel for guessing the source port in DNS requests, but have not evaluated this attack against real Internet systems.

Which of the methods is more effective? Which has higher applicability? Which is stealthier and does not trigger alerts?

To answer these questions we perform the first comparative analysis of the methodologies for cache poisoning attacks. In addition, in order to gain a deeper understanding of the methodologies and their impact on the Internet applications, we also extend the evaluations in the previous work [21, 23, 57] for Internet scale measurements of applicability and effectiveness of these methodologies against multiple Internet networks.

Taxonomy of vulnerable applications. The implications of cache poisoning for the other Internet services and applications has not been explored. There is evidence of cache poisoning in the wild, mostly for redirecting victims to impersonating websites [64]. Cache poisoning was also demonstrated in research against the certificate authorities [21, 23]. But there is no comprehensive study of exploits of cache poisoning against Internet clients and services.

What applications are at risk due to cache poisoning? How can an attacker exploit cache poisoning to attack applications? What is

SIGCOMM '21, August 23–28, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8383-7/21/08…$15.00
https://doi.org/10.1145/3452296.3472933

SIGCOMM '21: Proceedings of the 2021 ACM SIGCOMM 2021 Conference, August 2021, Pages 836-849
Accepted version. https://doi.org/10.1145/3452296.3472933

the fraction of vulnerable applications in the Internet? What are the challenges and what cache poisoning methodologies are more suitable?

We answer these questions by evaluating the cache poisoning methodologies against a range of popular applications. We defined nine categories of applications, ranging from security mechanisms, to VoIP, email and intermediate devices; see Table 1. We provide the first systematic study of cache poisoning against a collection of popular applications and security mechanisms.

Poisoning is a threat to applications. Our results demonstrate that, *although challenging to launch, off-path DNS cache poisoning poses a realistic threat for many Internet applications.* Surprisingly, we show that DNS cache poisoning can be applied for downgrade attacks against security mechanisms causing the victims not to perform validation, e.g., RPKI or domain-based anti-spam validation. Taking RPKI as an example, we developed an attack that by redirecting the RPKI cache [RFC6810] [24] to a wrong repository via DNS cache poisoning, the attacker can cause the RPKI validation to result in status `unknown` (instead of `invalid`). As a result the RPKI cache will not validate correctness of the BGP announcements that it receives. Suppressing RPKI validation allows the adversary to perform BGP prefix hijacks even of ASes which have the corresponding RPKI material (Route Origin Authorization and resource certificates [54]) in the public repositories and hijack even the senders which enforce route origin validation [61].

Another example is malware distribution by causing the anti-spam validation to fail via cache poisoning.

This is the first demonstration of the devastating power of DNS cache poisoning, which shows that in addition to traditional threats, such as impersonation, DNS cache poisoning can facilitate much stronger attacks which were otherwise not possible. We also show that DNS cache poisoning can be used to inflict Denial of Service (DoS) on applications and their clients.

In our experimental evaluation against the applications we exploit DNS cache poisoning to subvert correctness and security of basic Internet functions, enabling the attackers to take over IP addresses, to hijack telephony, to de-synchronise local time, and even prevent victims from connecting to the correct VPN tunnel.

Off-path attacks. Our study is performed with off-path attackers. This is the weakest attacker model in the Internet, it can merely send packets from spoofed IP addresses, which is a realistic assumption since around 30% of the Internet networks do not enforce egress filtering [18–20, 55, 56, 58]. Essentially any adversary in the Internet has off-path capabilities and can select networks which allow it to send packets with spoofed source IP addresses. Stronger attackers, most notably the on-path Man-in-the-Middle (MitM), can do more devastating attacks. Nevertheless, MitM attackers are more rare and even such attackers have limitations: the strong government sponsored attackers can be on-path only to some of the Internet victims depending on the paths that they control but even they do not control all of the networks. Therefore, it is critical to understand the threat that an off-path attacker poses to applications.

Disclosure and ethics. Our attacks were tested against remote networks reliably, yet were ethically compliant. We measured and evaluated vulnerabilities in the DNS caches of the subjects of our study and measured which services use the caches but did not hijack their traffic nor Internet resources and neither did we place incorrect DNS records for Internet domains that are not under our control in the caches of our test subjects. Specifically, to avoid harming Internet customers and domains, we set up a victim AS and victim domains as well as adversarial AS and adversarial hosts on that AS, which were used by us for carrying out the attacks against the victims. Our measurement study for evaluating the vulnerabilities was performed using our victim domains, which ensured that the targets of our study would not use the spoofed records for any "real" purpose.

We believe that in addition to disclosing the vulnerabilities to the affected entities it is critical to raise awareness to the extent and the scope of the vulnerabilities.

Contributions. We present the first comprehensive study of the attack surface that off-path DNS cache poisoning introduces on the Internet ecosystem.

- We implement three methodologies for launching off-path DNS cache poisoning attacks: (1) BGP prefix hijacking, (2) side-channels and (3) fragmentation. We perform the first Internet-scale evaluation of these methodologies against DNS resolvers and compare them for applicability, stealthiness and success of cache poisoning.
- We apply these methodologies to launch *cross-layer attacks* against widely used applications and services (see taxonomy in Table 1). Our study shows that cache poisoning can be used to bypass security mechanisms, to cause DoS attacks, or for impersonation attacks.
- We provide recommendations for countermeasures for DNS caches against cache poisoning attacks and for applications against cross-layer attacks even when using poisoned caches.

Organisation. We review DNS cache poisoning and related work in Section 2. In Section 3 we present the DNS cache poisoning methodologies that we use throughout our work. In Section 4 we demonstrate cross-layer attacks against applications using DNS cache poisoning. We provide results of our measurements in Section 5 and recommend mitigations in Section 6. We conclude this work in Section 7.

2 DNS CACHE POISONING OVERVIEW

Domain Name System (DNS) [60] cache poisoning allows an attacker to redirect victims to attacker controlled hosts. Typically the attackers targets recursive DNS resolvers whose caches serve multiple clients. A single injection of a malicious DNS record propagates to all the hosts that use that resolver. The attacker can then intercept the traffic between the services (such as web, email, FTP) in the victim domain and the hosts that use the poisoned cache. DNS resolvers use defences to make launching successful cache poisoning attacks difficult.

2.1 Defences Against Poisoning

The DNS resolvers are required to randomise certain fields in DNS requests sent to the nameservers, [RFC5452] [43]. These include a random 16 bit UDP source port and the 16 bit DNS transaction identifier (TXID); additional defences include nameserver randomisation [43] and 0x20 encoding [28]. The nameservers copy these fields from the DNS request to the DNS response. DNS resolvers accept the first DNS response with the correctly echoed challenge values and ignore any responses with incorrect values.

SIGCOMM '21: Proceedings of the 2021 ACM SIGCOMM 2021 Conference, August 2021, Pages 836-849
Accepted version. https://doi.org/10.1145/3452296.3472933

To launch a successful cache poisoning attack, the attacker needs to guess the correct challenge values and make sure that his spoofed response arrives before the genuine response from the real nameserver. This is easy for an on-path (man-in-the-middle) attacker, which can simply copy the values from the request to the response. Cryptographic signatures with DNSSEC [RFC6840] [73] could prevent on-path attacks, however, DNSSEC is not widely deployed. Less than 1% of the second level domains (e.g., 1M-top Alexa) are signed, and most resolvers do not validate DNSSEC signatures, e.g., [25] found only 12% in 2017. Our measurements indicate that less than 5% of the domains we studied are signed. There is however an increase in the resolvers validating DNSSEC: we found 28.6% validating resolvers via our ad-network study. Deploying DNSSEC was shown to be cumbersome and error-prone [26]. Even when widely deployed DNSSEC may not always provide security: a few research projects identified vulnerabilities and misconfigurations in DNSSEC deployments in popular registrars [44, 67].

Recent proposals for encryption of DNS traffic, such as DNS over HTTPS [41] and DNS over TLS [42], although vulnerable to traffic analysis [65, 68], may also enhance resilience to cache poisoning. These mechanisms are not yet in use by the nameservers in the domains that we tested. Nevertheless, even if they become adopted, they were not designed to protect the entire resolution path, but only the link between the client and the recursive resolver, and hence will not prevent DNS cache poisoning attacks.

2.2 History of DNS Cache Poisoning

In 2007 Klein identified vulnerability in Bind9 DNS resolvers [50] and in Windows DNS resolvers [51] allowing off-path attackers to reduce the entropy introduced by the TXID randomisation. In 2008 Kaminsky [47] presented a practical cache poisoning attack even against truly randomised TXID. Vixie suggested to randomise the UDP source ports already in 1995 [72], subsequently in 2002 Bernstein warned that relying on randomising TXID alone is vulnerable [17]. Following Kaminsky attack DNS resolvers were patched against cache poisoning [43], and most randomised the UDP source ports in queries.

Nevertheless, shortly after new approaches were developed allowing cache poisoning attacks. In 2012 [37] showed that off-path attackers can use side-channels to infer the source ports in DNS requests. In 2015 [66] showed how to attack resolvers behind upstream forwarders. This work was subsequently extended by [74] with poisoning the forwarding devices. A followup work demonstrated such cache poisoning attacks also against stub resolvers [15]. [57] showed how to use ICMP errors to infer the UDP source ports selected by DNS resolvers. Recently [52] showed how to use side channels to predict the ports due to vulnerable PRNG in Linux kernel. In 2013 [38] provided the first feasibility result for launching cache poisoning by exploiting IPv4 fragmentation.

For the evaluations in this work we selected three generic cache poisoning methodologies developed in [21, 38, 57], which are not specific to implementation or setup and do not result due to bugs in randomness generation, such as [52]. We perform Internet-wide measurements of these methodologies testing experimentally DNS cache poisoning against DNS resolution platforms. We then exploit these poisoned caches to attack applications that use the poisoned records we injected.

2.3 DNS Cache Poisoning in the Wild

There is numerous evidence of DNS cache poisoning attempts in the wild, [7–13, 27, 64, 69], which were predominantly launched via short-lived BGP (Border Gateway Protocol) prefix hijacks or by compromising a registrar or a nameserver of the domain.

We consider only attacks done by network attackers by manipulating the protocols remotely but without compromising services or networks. Hence compromises of registrars or servers is not in our scope and in the review of works we focus only on BGP prefix hijacks, side channels and fragmentation attacks.

In 2017 [16] simulated the effects of BGP prefix hijacks on bitcoin without experimentally evaluating it in the wild. In 2018, [21] experimentally evaluated the impact of BGP prefix hijacks on domain validation and [23] evaluated the impact of DNS cache poisoning on domain validation. In 2020 a recent research project [70] evaluated BGP prefix hijacks for cross-layer attacks on Tor (the onion routing) [31] users, domain validation and bitcoin [34].

Except for fragmentation based DNS cache poisoning against domain validation [23] there were no studies of cache poisoning using different methodologies and their evaluation against applications. In this work we perform the first comprehensive study of DNS cache poisoning against different applications, and using different methodologies.

3 TAXONOMY OF POISONING METHODS

In our evaluations in subsequent sections we use three methodologies for poisoning DNS caches, which were shown to be practical in previous research: (1) intercepting DNS requests with BGP prefix hijacking [70], (2) guessing challenge values in DNS requests via side-channel [57] or (3) injecting content into IP defragmentation cache [38]. In this section we describe these attack methodologies, their unique properties and explain what attacker capabilities they assume. We compare effectiveness and stealthiness of each of these methods for carrying out cache poisoning attacks.

Setup. To test our attacks experimentally in the Internet we setup a victim AS and associate a /22 prefix with our AS. We register victim domains and setup nameservers and a DNS resolver.

3.1 Intercepting DNS with BGP Hijacking

A malicious Autonomous System (AS) can exploit vulnerabilities in BGP to hijack packets of some victim AS. A route hijack happens when an attacker announces an incorrect prefix belonging to a different AS. The attacker hijacks the prefix or a sub-prefix which has the IP address of a DNS nameserver or a resolver. If the hijack succeeds, the ASes that accepted the hijack will send all their traffic destined to the victim prefix instead to the attacker. The goal of the attacker is to intercept a single DNS packet, either a query sent by the resolver or a corresponding response of the nameserver. For simplicity in this discussion we focus on sub-prefix hijacks and assume that the attacker attempts to hijack the DNS query; see [21] for a taxonomy of BGP prefix hijack attacks. The attacker intercepts the DNS query and crafts a spoofed DNS response with malicious records and the correct challenge values, and sends it

SIGCOMM '21: Proceedings of the 2021 ACM SIGCOMM 2021 Conference, August 2021, Pages 836-849
Accepted version. https://doi.org/10.1145/3452296.3472933

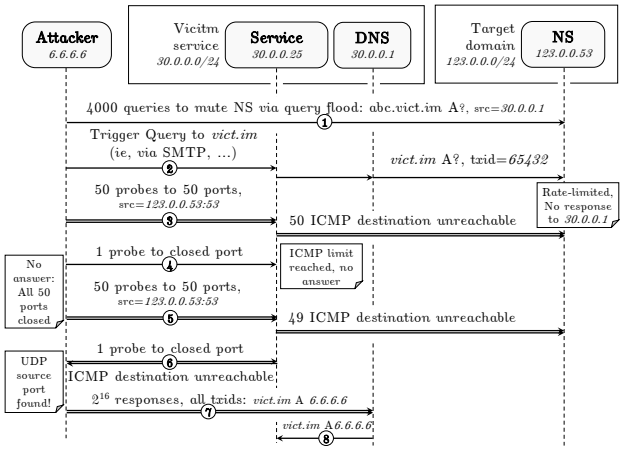


Figure 1: DNS Poisoning with side-channel.

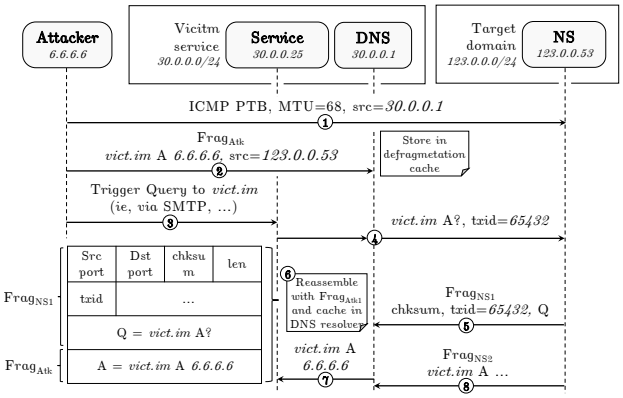


Figure 2: Fragmentation-based DNS poisoning.

to the victim DNS resolver. Additionally, to avoid detection due to blackholing, the attacker should relay all the traffic to the legitimate destination, except for the DNS query which it intercepted (to avoid race condition with the response from the genuine nameserver). We call this DNS cache poisoning attack method HijackDNS.

3.2 Guessing Challenges with Side-channel

The SadDNS off-path attack [57] uses an ICMP side channel to guess the UDP source port selected by the victim resolver in its query to the target nameserver. This is done via a side-channel present in most modern operating systems which allows the attacker to test if a given UDP port open or not. The operating systems have a constant, global limit of how many ICMP port unreachable messages they will return when packets are received at closed UDP ports (50 in the case of linux). The attacker splits the range of ports to sets of N ports and for every set performs the following: the attacker sends 50 probes with a spoofed source IP address of the nameserver to a range of UDP ports at the resolver. If the probes arrived at closed ports only, the returned ICMP error messages reach the global limit, and further messages will not be issued. The attacker sends a single probe from the IP address of the attacker to a known-closed port. If all of the previously probed 50 ports were closed the attacker will not receive an ICMP message in response to his own message. However, if one of the 50 probed ports was open, the limit was not reached, the attacker will receive a ICMP port unreachable message. The attacker repeats this process until a set containing an open port is found. Once a set with an open port is found, the attacker applies divide and conquer with the technique above dividing the ports until a single open port is isolated. This reduces the entropy of the challenge-response parameters unknown to the attacker from 32 bit (DNS TXID + UDP port number) to 16 bit.

Once the open port is identified the attacker sends multiple spoofed DNS responses from a spoofed IP address (of the nameserver) to that open UDP port of the resolver, for each possible TXID value, total of 2^{16} spoofed responses; e.g., [37, 45, 57]. A packet with the correct TXID is accepted by the DNS resolver. The attack is illustrated in Figure 1.

The attack applies to only about 18% of the domains with nameservers that use rate-limiting. The rate limiting allows the adversary to delay the response from the genuine nameserver and hence to win the race against it. Additionally, the attack applies only against resolvers with a global (un-patched) ICMP rate limit.

3.3 Injecting Records via IP Fragmentation

In this section we describe an attack which exploits IP fragmentation to inject spoofed fragments into the IP defragmentation cache on the victim system. The spoofed fragments contain malicious content, which when reassembled with the genuine fragments, manipulate the payload of the original IP packet without having to guess the values in the challenge-response parameters, [38].

We assume that the response from the nameserver is fragmented and arrives in at least two fragments. The fragment sent by the attacker is reassembled with the first fragment sent by the nameserver. The attacker replaces the second fragment of the nameserver with its malicious fragment, which overwrites part of the payload of the genuine DNS response from the nameserver, with malicious values. Since the challenge-response values (port, TXID) are in the first fragment, they remain unchanged. The illustration of the attack is in Figure 2.

To cause the nameserver to fragment a DNS response the attacker sends to the nameserver a *ICMP Destination Unreachable Fragmentation Needed* error message (type 3, code 4) with a DF bit set, signalling to the nameserver that the Maximum Transmission Unit (MTU) to the destination is smaller than the packet's length. The nameserver reduces the size of the packet accordingly by fragmenting the IP packet to smaller fragments.

4 EXPLOITING DNS POISONING FOR CROSS-LAYER ATTACKS

In this section we demonstrate how DNS cache poisoning can be used to launch cross-layer attacks against popular applications. In Section 4.1 we explain our methodology for selecting the applications. We list the categories according to which we selected the applications in Table 1. Our analysis of the applications is performed according to the key properties related to cache poisoning: (1) control over the query, (2) which records can be injected, (3)

Category	Protocol	Use Case	query name	known	query trigger method	Record Type	DNS used for loc.	fed.	auth.	Methodologies Hijack	SadDNS	Frag	Cache Poisoning impact
Authentication	Radius	Peer discovery	target	✓[1]	direct	NAPTR, SRV, A	✓	✓		✓	✓	✓	DoS: no network access
Online Chat	XMPP	Chat+VoIP	target	✓[1]	bounce	A, SRV	✓	✓		✓	✓	✓	Hijack: eavesdropping
Email	SMTP	Mail	target	✓[1]	direct/bounce	A, MX	✓	✓		✓	✓	✓	Hijack: eavesdropping
	SPF,DMARC	Anti-Spam	target	✓[1]	authentication	TXT			✓	✓	✓	✓	Downgrade: spoofing
	DKIM	Integrity Checking	target	✓[1]	direct/bounce	TXT			✓	✓	✓	✓	Downgrade: spoofing
Web	HTTP	Web sites	target	✓[1]	direct	A	✓			✓	✓	✓	Hijack: eavesdropping
	SMTP	Password recovery	target	✓[1]	direct	A, MX, TXT	✓			✓	✓	✓	Hijack: account hijack
Sync	NTP	Time synchronisation	known	✓	connection DoS	A	✓			✓	✗	✓[2]	Hijack: change time
Crypto-currency	Bitcoin	Peer discovery	known	✓	waiting	A	✓			✓	✗	✗	Hijack: fake blockchain
Tunnelling	OpenVPN	VPN	config	✗	connection DoS	A	✓			✓	✓[2]	✓[2]	DoS: no VPN aceess
	IKE	VPN	config	✗	connection DoS	A	✓			✓	✓[2]	✓[2]	DoS: no VPN aceess
	IKE	Opportunistic Enc.	target	✓[1]	bounce	IPSECKEY	✓		✓	✓	✓[2]	✓[2]	Hijack: eavesdropping
PKI	DV	Domain Validation	target	✓[1]	authentication	A, MX, TXT	✓		✓	✓	✗	✗	Hijack: fraud. certificate
	OCSP	Revocation checking	target	✓[1]	direct	A			✓	✓	✓	✓	Downgrade: no check
	RPKI	Repository sync.	known	✓	waiting	A	✓			✓	✗	✗	Downgrade: no ROV
Intermediate devices	–	Firewall filters	config	✗	waiting	A	✓			✓	✓[2]	✓[2]	Downgrade: no filters
	HTTP/...	Loadbalancers	config	✗	on-demand	A	✓			✓	✓[2]	✓[2]	Hijack: eavesdropping
	HTTP	CDN's	config	✗	on-demand	A	✓			✓	✗	✓[2]	Hijack: eavesdropping
	DNS	ANAME/ALIAS[33]	config	✗	on-demand	A	✓			✓	✓[2]	✓[2]	Hijack: eavesdropping
	HTTP/Socks	Proxies	target	✓[1]	direct	A	✓			✓	✓	✓	Hijack: eavesdropping

[1]: Depends on the attack scenario. [2]: Requires a third-party application to trigger queries.

Table 1: Evaluation of attacks against popular systems leveraging a poisoned DNS cache.

how the application uses the injected records, and (4) the outcome of the attack.

4.1 Methodology for Selecting Applications

We select the applications according to the following considerations: application category, usage of DNS by the application and the impact of DNS cache poisoning on the application.

4.1.1 Category. We categorise the applications to groups, covering most of the popular applications and security mechanisms in the Internet (left most column in Table 1). Within each category we selected a few representative protocols and systems for that category, see column 'Protocol' in Table 1.

4.1.2 Usage of DNS. One of the considerations for selecting the applications is how the application uses DNS: how the queries are sent by the application to the DNS resolver and how the results from the lookups are processed. The column 'Use-Case' in Table 1 describes the usage scenarios of the DNS by the application. We defined the following types:

Location (loc): DNS is used to locate a direct communication partner, typically in form of a hostname-to-ip (`A, AAAA`) mapping.

Federation (fed): DNS is used to locate a user's home-server based on the domain part of a user address of the form `user@domain`.

Authorisation (auth): DNS is used to authorise a certain action or host in the name of the domain's owner.

4.1.3 Queries. Applications differ in flexibility in allowing external entities to trigger queries. Our selection of applications aims to cover the variety of options for triggering queries. To initiate the attack, our adversary needs to cause the victim resolver to issue the target query or to predict when the query will have been issued.

Some applications enable the attacker to send arbitrary queries, e.g., in systems which use DNS for peer discovery in federated systems like Radius, XMPP and SMTP. This is because in these systems, the queried domains are part of the user's ID. This user ID can be controlled by the attacker to trigger a query to a domain of its choice. The same applies to all (sub-)systems used as part of web browsing, like HTTP, DANE and OCSP, since the attacker can establish direct connection from the victim client to arbitrary web servers which will trigger a DNS lookup that way. Setting the domain name is not always possible, e.g., in NTP the query is selected by the resolver based on the hostname that it receives from the local NTP server.

We evaluated popular appliances and systems for their query triggering behaviour. We list some selected systems in Table 2. As can be seen, some allow external adversaries to trigger queries (indicated with "on-demand" in column Trigger query). Other devices use timers for issuing queries. Hence the adversaries can often predict when the query is issued.

4.1.4 Impact of poisoning on applications. We select applications to demonstrate the impact that cache poisoning on applications can create: DoS (Denial of Service), downgrade of security or interception attacks.

4.2 Methodology for Attacking Applications

We developed cross-layer attacks that leverage DNS cache poisoning to attack applications listed in Table 1. The steps underlying all our cross-layer attacks against applications are:

(1) Use the application to send to the victim DNS resolver a query. In addition to the traditional ways of triggering queries, such as with a script or Email, we also developed new ways to trigger queries which were not known prior to our work. Some of these techniques are specific to appliances and platforms, see Table 2, while others are application-independent methodologies for triggering queries. We explain our methodologies for triggering queries in Section 4.3.

(2) Inject malicious records to poison the cache of the victim DNS resolver. We use the methodologies in Section 3 for injecting malicious records into the cache of the victim DNS resolver. In

Type	Provider	Trigger query	Caching time	Websites in Alexa 100K
Firewall	pfSense	timer	500s	-
	Sophos UTM	timer	240s	-
Load balancer	Kemp Technologies	timer	1h	-
	F5 Networks	timer	1h	-
CDN	Stackpath	on-demand	TTL	79
	Fastly	timer	TTL	1,143
	AWS	on-demand	TTL	11,057
	Cloudflare	on-demand	TTL	17,393
Managed DNS (ALIAS)	DNSimple	on-demand	TTL	248
	DNS Made Easy	timer	~35min	1,192
	Oracle Cloud	on-demand	TTL	1,382
	Cloudflare	on-demand	TTL	20,027

Table 2: Query triggering behaviour at middleboxes. Last column shows the number of websites in 100K-top Alexa which use that provider.

Table 1 we summarise the applicability of the cache poisoning methodologies for cross-layer attacks against each application, and explain this in Section 4.4.

(3) Exploit the poisoned records to cause a victim application to divert from the expected behaviour. The outcomes of our cross-layer attacks against applications range from downgrading security to denial of service attacks and to more traditional impersonation attacks, explained in Section 4.5.

4.3 Methodologies for Triggering Queries

4.3.1 Common ways for triggering queries. The most challenging aspect of cross layer attacks that use DNS cache poisoning is the ability to trigger or predict DNS requests. Typically an external adversary does not have access to internal services, such as the DNS resolver, and hence should not be able to cause the DNS resolver to issue arbitrary DNS requests. Adversaries can trigger queries via bounce. For instance, by sending an Email to a non existing recipient in the target domain the adversary will cause the Email server to return an error message with Delivery Status Notification. To send the error the Email server requires the IP address and hostname of the MX server in the domain that sent the Email message which triggered the error. This causes queries to the domain specified by the attacker.

The adversary can also set up a web server and lure clients to access it, this is a direct query triggering. The clients download the web objects from the adversary's domain, and send DNS requests to the DNS resolvers on their networks. When resolvers receive DNS requests from servers or clients on their networks they initiate DNS resolution. However, these approaches are limited. For instance, only about 18% of the Email servers trigger DNS requests when receiving Emails, [53]. The limitation with web clients is that the adversary must wait until the target client visits the malicious web page. Furthermore, web clients cannot be used to poison resolvers that are used only by servers, such as Email or NTP. In this section we develop new approaches for triggering queries.

4.3.2 Cross-applications DNS caches. The adversary may be able to use one application to trigger queries to inject a record that is meant to be used for cross-layer attack against a different application that uses the same DNS cache. For instance, when an adversary cannot trigger queries via an application that it wishes to attack, it may often be able to trigger queries via a different application, that uses the same DNS cache. The adversary may also choose to inject into such cross-applications caches an application agnostic records; for instance, a malicious NS record, mapping the nameserver of a domain of the target application to the attacker's IP address, is an example of an application agnostic record.

Such cross-applications DNS cache scenario is not uncommon. The DNS resolvers often serve multiple applications and the networks use the caching of the resolver to reduce traffic and latency for all the applications. We use open resolvers to check how common cross-applications DNS caches are. We perform our measurements against a list of open resolvers from censys [32] and probe their caches for the well-known domain(s) used by the applications on our list in Table 1, e.g., `pool.ntp.org` for NTP. For each application for which the records are in the cache we consider that the resolver is used by that application or by the clients of that application. We found that 69% of the open resolvers are shared between two or more of the applications on our list.

A recent study [45] analysed how an attacker can find third-party SMTP servers to trigger queries at typically closed forwarders used by web clients. By scanning the /24 network block of the resolver's outbound IP address, the study found that an adversary could find an SMTP server which allows triggering queries from the same resolver in 11.3% of the cases. Additionally 2.3% of the resolvers were open resolvers in the first place.

4.3.3 Triggering queries via forwarders. In this section we show how to trigger queries with resolvers when this is not possible from the target application.

DNS forwarders make up the majority of open resolvers in the internet. Finding an open forwarder which forwards to the resolver of choice whose cache the adversary wishes to poison is not difficult. We explore the prevalence of forwarders through which one can force a given recursive DNS resolver to trigger a query. We perform a two step measurement: we first collect the forwarders used by open DNS resolvers and then which of these forwarders are used by random clients in the Internet.

In our measurement we use the list of all open DNS resolvers from Censys [32] (which performs a full IPv4 scan for open resolvers). We query all the resolvers for a custom query with a randomised subdomain under a domain which we control. This allows us upon the arrival of the DNS requests to our nameservers to map the open resolver's IP address to the recursive forwarder that it uses. This forwarder is determined by the outbound IP address in the DNS query that arrives at our nameserver.

In the second step we run a web ad-based study against random clients in the Internet that download our object. We trigger DNS requests via those clients to our own domain. We use a random subdomain associated with each client. Per client, we then obtain the list of recursive resolvers' IP addresses that arrived at our nameserver. We search them in the list of recursive resolvers IP addresses from our dataset of open resolvers.

Our results are as follows: focusing only on the IP addresses of the recursive resolvers, we find 4146 addresses out of which 3275

(79%) addresses are in the open resolver database. Consequently, assuming that an adversary targets a DNS resolver used by a typical web client (represented by the ad-net clients in our study), there is a high probability (79%) that it can find an open forwarder which can be used to poison the cache of the target victim recursive resolver used by that web client.

4.4 Applicability to Applications

In this section we explore which cache poisoning methodology is applicable to which of the applications listed in Table 1.

For all methodologies, the attacker requires the knowledge of the domain which is queried. In cases where the domain is pre-configured in the applications configuration ("config" in Table 1), this information needs to be fetched out of band.

4.4.1 HijackDNS. The adversary can hijack a sub-prefix or same-prefix of the victim AS. We explain the success probability of cache poisoning through both methods.

Sub-prefix hijack. The attacker can advertise a sub-prefix of the victim. The routers prefer more specific IP prefixes over less specific ones, hence this announcement will redirect all traffic for that sub-prefix to the attacker.

Same-prefix hijack. Same-prefix hijack occurs when the attacker hijacks a route to an existing IP prefix of the victim. The attacker can advertise the same prefix as the victim AS and depending on the local preferences of the ASes will intercept traffic from all the ASes that have less hops (shorter AS-PATH) to the attacker than to the victim AS. The success of the hijack depends on the topological relationship between the attacking AS and the domain and the victim resolver.

4.4.2 SadDNS. The attack is probabilistic since it depends on the ability of the adversary to win the race, by correctly guessing the randomised TXID before the timeout event. A prerequisite to a successful attack is the ability to trigger a large volume of queries. Typically, this is the case when the query domain can be set by the attacker ("target" in Table 1, Column "query name") or when a third party application is used to trigger the queries (marked with ✓[2] in Table 1, see Section 4.3.3).

4.4.3 FragDNS. FragDNS is also a probabilistic attack since its success depends on correctly guessing the IP ID value in the spoofed IP fragment. This is easy when systems have large IP defragmentation buffers, such as old linux versions which allows the adversary to send multiple fragments with different IP ID values, or when systems use incremental IP ID counters which can be predicted. A successful poisoning with FragDNS typically requires more packets than with prefix hijacks but less than with SadDNS attack.

4.5 Exploiting Poisoned Caches for Attacks

Applications that use DNS resolvers with poisoned caches are exposed to a range of attacks. In this section we explain the possible outcomes of the attacks via DNS cache poisoning.

Downgrade attacks. In downgrade attacks the attacker makes the security mechanism not available, as a result, causing the processing of the data to be performed without the additional information provided by the security mechanism. For instance, by poisoning the responses to queries for SPF or DKIM records the attacker can trick the victim Email server into accepting phishing Emails or Emails with malicious attachments. Similarly, by causing the RPKI validation to fail, the adversary can make a network, that filters bogus BGP announcements with route origin validation, to accept hijacked prefixes as authentic. This is due to the fact that RPKI validation will result in status 'unknown' and hence will not be used.

The attacker can also trick a security mechanism via DNS cache poisoning. For instance, the attacker can bypass domain validation, by redirecting the validation to run against attacker's host [23], and hence can issue fraudulent certificates.

Hijack attacks. In hijack attacks the victims are redirected to attacker's host which impersonates a genuine service in the Internet. Network adversaries can hijack traffic to take over Internet resources, such as SSO accounts at public providers. For instance, the adversaries can take over the SSO accounts at Regional Internet Registries (RIRs), by exploiting a combination of DNS cache poisoning with password recovery [29]. The idea is to poison the cache of the RIR, and to inject a record that maps the victim LIR to the host of the attacker. Running a password recovery procedure causes the password for the victim SSO account to be sent to the attacker instead of the victim. As a result, the attacker can hijack the digital resources, such as IP addresses and domains, that belong to the victim LIR.

DoS attacks. The attacker can block connectivity, e.g., for radius clients or access to services, such as secure tunnels. The idea is that if the attacker cannot forge cryptographic material, such as a certificate to authenticate a radius client, it can redirect the client to the wrong host via cache poisoning, preventing the client from connecting to the genuine target service. The adversary will not be able to provide authenticated material which will result in a failure, and lack of service for the victim client.

5 INTERNET MEASUREMENTS

In this section, we analyse the fraction of the vulnerable resolvers and nameservers with respect to each DNS poisoning method. We evaluate properties which influence the success of the cross-layer attacks against applications. Our measurements in this section show that the vulnerabilities do not significantly differ for most of the application-specific datasets. The outliers can be summarised as follows:

• Vulnerabilities to BGP sub-prefix hijacking are exceptionally high for eduroam and low for RPKI domains. The cause may be inherent in networks' sizes (large in case of universities and small for RPKI repository operators) and accordingly use BGP announcements which are larger than /24 for large networks or equal to /24 for small networks.

• Vulnerabilities to fragmentation cache poisoning among open resolvers is low compared to other resolvers in our dataset. This may be due to the fact that the distribution of the open resolvers is skewed towards poorly configured devices which cannot handle fragmentation.

• Domains with MX, SRV, NAPTR (eduroam) records are more often vulnerable to fragmentation based cache poisoning than the

domains in the 1M-top Alexa dataset. One reason is that the responses to ANY queries result in much larger packets, which often exceed the minimum MTU limit.

5.1 Vulnerabilities in Resolvers

We test the DNS resolvers for vulnerabilities to the three cache poisoning methods (Section 3) for different applications. The results of our evaluations for all datasets and all poisoning methods are summarised in Table 3.

5.1.1 Dataset. For each application from Section 4, we gather datasets of resolvers used by the front-end systems (i.e., Web clients, Alexa MX records, etc.) of that application. To achieve this, we first look for an appropriate dataset of front-end systems and then trigger queries through those front-end systems. This allows us to discover and test the corresponding resolver.

For front-end systems, we use the following datasets, listed in Table 3: (1) Our local university eduroam service. (2) Password recovery of popular infrastructure service providers, consisting of: All 5 Regional Internet Registries, popular domain registrars used by Alexa Top 100K domains and popular cloud providers [1, 2, 4, 5]. (3) Domain validation of most popular Certificate authorities [3]. (4) Popular CDNs in Alexa Top 100K (by mapping A record to ASN). (5,6) SMTP and XMPP servers of Alexa Top 1M domains. (7) Web clients gathered via an Ad-network. (8) Open resolvers from Censys [32] and (9) a subset of those open resolvers who cache `pool.ntp.org`. This resulted in a dataset of 89,924 resolvers (back-end IP addresses) in 13,804 ASes associated with 33,418 prefixes.

We report the dataset size in terms of front-end systems (i.e., number of SMTP servers or number of open resolver front-end IP addresses) in column "Dataset size" of Table 3. For vulnerability, we report the percentage of vulnerable front-end systems which was measured as described in Section 5.1.2. When a front-end system uses multiple resolvers, we consider it vulnerable if any of the resolvers it uses is vulnerable.

5.1.2 Measuring cache poisoning vulnerabilities in resolvers. The results of our measurements and evaluations against resolvers for different poisoning methodology are summarised in Table 3. In the following sections we explain the measurements we carried out of each attack methodology against the resolvers in our dataset.

*Sub-prefix BGP hijacks (*HijackDNS*).* Since many networks filter BGP advertisements with prefixes more specific than /24, we consider an IP address hijackable if it lies inside a network block whose advertised size is larger than /24. We therefore map all the resolvers' IP addresses to network blocks and consider those vulnerable to sub-prefix hijacks whose advertisement is larger than /24, since an advertisement with a smaller prefix will always take precedence over a bigger one. For the remaining addresses, a BGP-hijack may still be possible using same-prefix hijacks. To infer the scope of DNS platforms potentially vulnerable to cache poisoning via BGP sub-prefix hijack attacks we perform Internet measurements checking for DNS platforms on prefixes less than /24. We collect information on the state of the global BGP table in the Internet with Routeview [71] and RIPE RIS [63] collectors. We analyse the BGP announcements seen in public collectors for identifying networks vulnerable to sub-prefix hijacks by studying the advertised prefixes sizes. The measurements of resolvers vulnerable to BGP sub-prefix hijacks are listed in Table 3 and plotted in Figure 3.

*Same-prefix BGP hijacks (*HijackDNS*).* We perform simulations of same-prefix BGP hijacks using a set of randomly selected attacker and victim AS pairs using a simulator developed in [39] and Internet AS level topology downloaded from CAIDA [6]. The simulator selects Gao-Rexford policy compliant paths [35], and considers prefix lengths and AS-relationship (provider, customer and peer) and sizes (stub, small, medium and tier one). The attackers are randomly selected from all the ASes whereby the victim ASes are selected from our dataset of DNS resolvers and 1M-top Alexa domains. For each (attacker, victim)-pair we perform a simulation of same-prefix hijack that the attacker AS launches against a victim AS. If the attacking AS is closer to the victim, the attack succeeds. The simulation shows that the attacking AS was capable of hijacking the traffic in 80% of the evaluations.

SadDNS. To test resolvers vulnerable to SadDNS, we test the resolvers back-end IP addresses for a global ICMP message limit which allows to use the side-channel identified by [57]. To limit our dataset to functional resolvers which are still reachable, we furthermore send an ICMP echo-response ('ping') packet to the resolver first. This is especially important for the open resolvers dataset, since this dataset tends to include resolvers operating from dynamic IP addresses, which may have changed since the dataset was collected.

For the open resolver dataset we measured a vulnerability rate of 12%, a notable reduction from the 35% vulnerability rate of the original paper [57]. This difference could be influenced by various factors including the fact that our dataset contained more resolvers than [57]. The crucial difference is likely that our study was conducted after the vulnerability which allowed the global rate limit to be exploited was patched in many systems. For example, all updated versions of Ubuntu should have been patched by the time we carried out our evaluations[1].

FragDNS. To test vulnerability to fragmentation-based DNS cache poisoning, we use a custom nameserver application which will always emit fragmented responses padded to a certain size to reach the tested fragment size limit. The nameserver is configured to only send CNAME responses in the first fragmented response. This means that if the resolver receives a fragmented response, it needs to re-query for the CNAME-alias. This allows us to verify that the answer arrived at the resolver and thus, that the resolver is vulnerable to this type of attack.

Using this setup, we test all resolvers by triggering queries to our nameservers and observe if the fragmented responses are accepted. In our bigger datasets, vulnerability rates range between 31% for Open resolvers and 91% for Ad-net resolvers. For the smaller datasets, we still observe many vulnerable services. However, all certificate authorities' resolvers in our dataset rejected our fragmented responses, maybe attributed to the fact that this attack method was already evaluated and disclosed to CAs previously [23]. We report results for all datasets and all poisoning methods in Table 3.

[1] https://ubuntu.com/security/CVE-2020-25705

SIGCOMM '21: Proceedings of the 2021 ACM SIGCOMM 2021 Conference, August 2021, Pages 836-849
Accepted version. https://doi.org/10.1145/3452296.3472933

Dataset	Protocol	Vulnerable against			Dataset size
		BGP hijack sub-prefix	Sad-DNS	Frag-ment	
(1) Local university	Radius	100%	0%	100%	1
(2) Popular services	PW-recovery	93%	16%	90%	29
(3) Popular CAs	DV	75%	0%	0%	5
(4) Popular CDNs	CDN	100%	0%	25%	4
(5) Alexa 1M SRV	XMPP	73%	1%	57%	476
(6) Alexa 1M MX	SMTP SPF DMARC DKIM	79%	9%	56%	61,036
(7) Ad-net study	HTTP DANE OCSP	70%	11%	91%	5,847
(8) Open resolvers	All	74%	12%	31%	1,583,045
(9) Cache test	NTP	79%	9%	32%	448,521

Table 3: Vulnerable resolvers.

5.2 Vulnerabilities in Domains

In this section we perform measurements of the vulnerabilities in domains to our cache poisoning methodologies for different applications. We collect lists of the domains associated with these applications and test all the nameservers serving each domain according to the properties required for each cache poisoning method (from Section 3). The results of our evaluations and measurements for all the tested datasets and poisoning methods are summarised in Table 4.

Dataset	Protocol	Vulnerable against				DNS SEC	Total
		BGP hijack sub-prefix	Sad-DNS	Fragment Any	Fragment Global		
(1) Eduroam list	Radius	96%	11%	44%	18%	10%	1,152
(2) Alexa 1M	HTTP DANE DV	53%	12%	4%	1%	2%	877,071
(3) Alexa 1M MX	SMTP SPF DKIM DMARC	44%	6%	7%	1%	3%	63,726
(4) Alexa 1M SRV	XMPP	44%	4%	29%	5%	7%	2,025
(5) RIR whois	PW-recovery	59%	9%	14%	4%	4%	58,742
(6) Registrar whois		51%	10%	23%	5%	6%	4,628
(7) Well-known	NTP	25%	0%	25%	25%	25%	9
(8) Well-known	Crypto-currency	28%	17%	21%	3%	21%	32
(9) Well-known	RPKI	14%	0%	0%	0%	67%	8
(10) Cert. Scan	IKE OpenVPN	51%	11%	5%	1%	7%	307

Table 4: Vulnerable domains.

5.2.1 Dataset. For each application in Section 4, we collect datasets of typical domains looked up by clients (or servers) of that application. We collect such domains from the following data sources, listed in Table 4:

(1) Eduroam institution lists from United Kingdom [46], Germany [30] and Austria [14]. (2) Alexa Top 1 Million domains, including subsets of domains which have (3) MX and (4) SRV (XMPP) records. Domains from account email addresses from whois databases of (5) RIRs and (6) Registrars. (7) Well-known NTP server domains. (8) Well-known cryptocurrency domains. (9) Well-known RPKI validator database domains. (10) Domains of IKE and OpenVPN servers' certificates. This resulted in 904,555 domains hosted on 200,086 nameservers in 24,353 ASes associated with 60,511 prefixes.

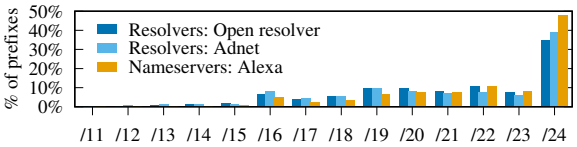


Figure 3: Announced prefixes.

5.2.2 Measuring cache poisoning vulnerabilities in nameservers.

HijackDNS. We perform a similar analysis as in Section 5.1.2, to check the vulnerabilities to BGP prefix hijacks. The results are plotted in Figure 3. The differences between the fractions of nameservers in 1M-top Alexa domains that can be sub-prefix hijacked are not significantly different than those of the resolvers.

The resilience of the DNS infrastructure to BGP hijack attacks is also a function of the distribution and the topological location of the nameservers in the Internet. We measured the characteristics of the nameservers from the Internet routing perspective. Our findings show that the nameservers are concentrated in just a few ASes. Our measurements show that 80% of the ASes host less than 10% of the nameservers, and the rest of the nameservers are concentrated on the remaining ASes. This concentration of the nameservers on a few ASes, typically CDNs, makes it easier to intercept traffic of multiple nameservers with a single prefix hijack.

SadDNS. For a nameserver to be vulnerable to side-channel attack (Section 3.2), the attacker must be able to 'mute' the nameserver to extend the time-window for the attack. This is achieved by abusing rate-limiting in nameservers. To find out if a nameserver supports rate-limiting, we use the following methodology: we send to the nameserver a burst of 4000 queries in one second, and see if this stops (or reduces) the subsequent responses received from this server. We consider a nameserver to be vulnerable if we can measure a reduction in responses after the burst.

Fragmentation. We evaluate the vulnerability to fragmentation-based poisoning in nameservers and domains by testing three properties required to create a sufficiently large fragment in order to inject malicious records into it: (1) support of IMCP fragmentation needed, (2) record types for optimising response size, (3) by bloating the queried domain and (4) fitting the response into the limitation of EDNS.

PMTUD. We first check for the support of path MTU discovery (PMTUD) with ICMP fragmentation needed: we send to the nameserver an ICMP fragmentation needed packet, which indicates that the nameserver should fragment packets sent to our test host. Then we send queries of different type to that domain. We consider a nameserver vulnerable if the responses return fragmented.

Record types. We evaluated fragmentation with three record types: `ANY`, `A` and `MX`. We use DNS requests of type `ANY` to increase

Implementation	Vulnerable	Note
BIND 9.14.0	yes	cached
Unbound 1.9.1	no	doesn't support ANY at all
PowerDNS Recursor 4.3.0	yes	cached
systemd resolved 245	yes	cached
dnsmasq-2.79	no	not cached

Table 5: ANY caching results of popular resolvers.

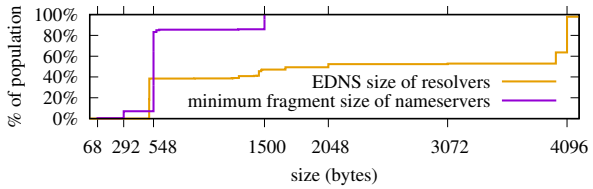


Figure 4: CDF of resolver EDNS UDP size vs. minimum fragment size emitted by nameservers.

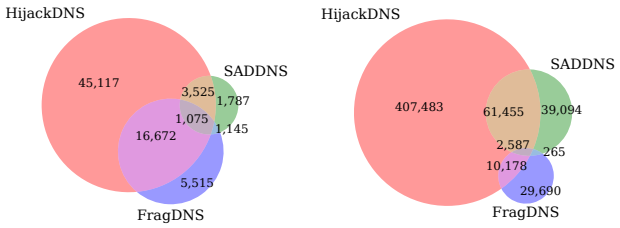
the response size above the fragmentation limit of the nameserver. We find that for 19.50% of domains in 1M-top Alexa there is at least one nameserver which emits fragmented DNS responses, which can be used for cache poisoning attacks via injection of IP fragments. We plot the minimum fragment size emitted by those nameservers in Figure 4, which shows that most affected nameservers (83.2%) fragment DNS responses down to a size of 548 bytes and 7.05% even down to 292 bytes. We tested ANY response caching in 5 of the most popular resolver implementations and found that 3 out of 5 use the contents of an ANY response, to answer subsequent A queries, without issuing further queries (See Table 5). Namely, the adversaries can often launch cache poisoning attacks by issuing queries for ANY record type in the domain.

However, only open resolvers (or forwarders) allow the attacker to trigger ANY queries. We repeat the same study using queries for A record type and then for MX record type, which are the query types typically triggered using the other query-triggering methods, such as via email or a script in a browser. We get vulnerability rates of 0.29% and 0.44% respectively due to the smaller response sizes which are often not sufficiently large to reach the nameserver's minimum fragment size. However, these numbers represent the lower bound.

Bloat query. The attacker can bloat the queries by concatenating multiple subdomains which increases the responses sizes. The maximum increase is up to 255 characters. The labels are limited to max 63 characters (+1 for the label delimiter) and the attacker can concatenate four subdomains: 4*64 (minus the parent domain). This increases the vulnerable resolvers to above 10%.

Fitting into response. Additionally to the requirement that the DNS response size must be big enough to trigger fragmentation on the nameserver side, it must also be small enough to fit in the resolvers maximum response size advertised in EDNS.

To evaluate this, we measure the EDNS UDP size of more than 1.5K open resolvers collected from Censys [32] IPv4 Internet scans. We query each resolver by triggering a query to our own nameserver and measure the EDNS UDP payload size advertised in the query. The results are shown in Figure 4. Approximately 40% of the resolvers support UDP payload sizes of up to 512 bytes, while 50% of the resolvers advertise a payload size equal or larger than 4000 bytes. The remaining 10% are between 1232 and 2048 bytes. Given the minimum MTU size measurement of the nameservers in 1M-top Alexa domains in Figure 4, this means that the resolver population is essentially portioned in two groups: one group (40%) which is vulnerable to poisoning attacks with 7% of all vulnerable domains and one group (50-60%) which is vulnerable to poisoning attack with all the vulnerable domains.



(a) Resolver **(b) Domain**

Figure 5: Venn diagram of all vulnerable resolvers (by number of back-end addresses) and domains.

5.3 Comparative Analysis

Our measurements show that the methodologies for DNS cache poisoning can often result in practical attacks, depending on the setup, network conditions and server configurations. In this section we compare the DNS cache poisoning methodologies with respect to stealthiness, effectiveness and applicability.

The main insights of the experimental measurements that we performed using each of the methods in Section 3 are summarised in Table 6. The columns in Table 6 correspond to the attacks we carried out against the domains and resolvers in our dataset (see Section 5).

	BGP Hijack		SadDNS	Fragmentation	
	sub-	same-		any IPID	global IPID
Applicability					
Vuln. resolvers	70%	80%	11%	91%	
Vuln. domains	or 53%	or 70%	and 12%	and 4%	and 1%
Effectiveness					
Hitrate	100%		0.2%	0.1%	20%
Queries needed	1		497	1024	5
Total traffic (pkts)	2		987K	65K	325
Stealthiness					
Visibility	very visible	visible	stealthy, but locally detectable (Packet flood)		very stealthy
Additional requirements					
Additional requirements	none		none	max(resolver EDNS size) < min(nameserver MTU)	

Table 6: Comparison of the cache poisoning methods.

5.3.1 Applicability. A method is applicable against a resolver for some domain if it results in a practical DNS cache poisoning attack. The applicability for each method for resolvers and domains is listed in Table 6.

To compare the applicability of the methodologies we use the results of our internet measurements (Tables 3 and 4) and take the numbers for the ad-Net resolvers and 1M-top Alexa domains datasets. We also show the absolute number of all vulnerable resolvers (according to a back-end address) and domains in all our datasets in Figure 5. This figure shows that the number of resolvers and domains vulnerable to HijackDNS is by far the highest, while SadDNS has more vulnerable domains and FragDNS has more vulnerable resolvers. The overlaps between the vulnerable domains and resolvers can be seen as expected for a distribution of unrelated properties, i.e., SadDNS and FragDNS have a significant overlap with HijackDNS, which is due to the fact that 53-70% of the systems we measured are vulnerable to HijackDNS, while SadDNS and FragDNS only have a small overlap compared to number of vulnerable systems in each category. Only 11% of the DNS resolvers and 12% of the domains are vulnerable to SadDNS attack. Many more resolvers are vulnerable to injection of content via IPv4 fragments, hence FragDNS attack is more applicable than SadDNS. In addition, due to its large size, the open resolver dataset dominates the results in our comparison.

5.3.2 Effectiveness. Attack effectiveness is demonstrated with the traffic volume needed for a successful attack, which is a function of the number of queries that should be triggered for a successful attack. The larger the attack volume, the less stealthy the attack is. We define hitrate as the probability to poison the target DNS cache with a single query and calculate the expected number of queries for each of the poisoning methods by inversion of the hitrate. We estimate the expected number of packets sent to the resolver by multiplying this with the traffic volume generated per query. For SadDNS where the amount of traffic during the attack is not stable, we analyse the experimental data for the amount of traffic needed.

HijackDNS. If an AS prefers a malicious BGP announcement of the adversary to the announcement of the victim AS, then the attack is effective, requiring only a single packet to send a malicious BGP announcement and then another packet to send a spoofed DNS response with malicious DNS records.

SadDNS. Using our implementation of SadDNS attack from Section 3.2 we find that the DNS cache poisoning with SadDNS succeeds after an average of 471 seconds (min: 39 seconds, max: 779 seconds). This is inline with the results in [57] which report an average of 504 seconds. To achieve a successful attack we needed to run 497 iterations on average. This is correlated with the attack duration since we do not trigger more than two queries per second. When more queries within one attack iteration are triggered, the resolvers respond with `servfail`. By inverting this number we get a hitrate of 0.2%. Notably however, since most of the queries do not result in attack windows of meaningful length, an attacker should be able to optimise the attack by analysing the exact back-off strategies used by the target resolver, and adjusting the queries according to this.

Using the results from our SadDNS experiment, we also obtain statistics for how may packets are sent to the target resolver. On average, our implementation sent 986,828 packets or 88MB of traffic, which is again, comparable to the original attack (69MB in [57]).

FragDNS. Only about 1% of the domains allow deterministic fragmentation-based cache poisoning attacks thanks to slowly incremental global IPID counter in nameservers. More than 4% of the domains are vulnerable to probabilistic attacks by attempting to hit an unpredictable IPID counter and to match the UDP checksum. When the IPID values are not predictable, the probability to hit the correct value is roughly 0.1%. To match the UDP checksum, the attacker needs to predict the partial UDP checksum of the second fragment of response sent by the nameserver. This means that the probability to match the UDP checksum is the inverse of the number of possible second fragments emitted by the nameserver (assuming equal distribution).

To calculate the per-nameserver hitrate of FragDNS attack for each domain we calculate the product of both probabilities, matching the IPID as well as matching the UPD checksum. We take the average of these per-nameserver hitrates to calculate a per domain hitrate. The results of our evaluation are: when the nameservers use a single global counter for IPID, depending on the rate at which queries arrive at the nameserver, the median hitrate over all vulnerable domains (for different rates of queries from other sources) is 20%. When the nameserver selects IPID values pseudorandomly, the median hitrate is 0.1% which is the probability to correctly guess the IPID, as most servers to not randomise the records in DNS responses.

FragDNS attack also requires large traffic volumes with 1024 packets median computed over vulnerable domains with 65K packets for an unpredictable IPID, and with only 325 packets on average against a predictable IPID against high load servers, such as the servers of top-level domains.

In the worst case, the attack requires 64 packets to fill the resolver IP-defragmentation buffer and another packet to trigger the query. Combined with a 0.1% success rate, this translates to an average of 65,000 packets.

5.3.3 Stealthiness. In BGP prefix hijacks malicious BGP announcements manipulate the control plane and a single BGP announcement suffices to change the forwarding information in the routers. BGP prefix hijacks generate lower traffic volume when performing the hijack but may be more visible in the Internet since the attack impact is more global. The more networks are affected as a result of the BGP hijack the higher the chance is that such attacks may be detected. Same-prefix hijack is more stealthy in control plane than sub-prefix hijack since it does not affect the global routing BGP table in the Internet, but causes manipulations only locally at the ASes that accept the malicious announcement. Furthermore, as we already mentioned, short-lived BGP hijacks typically are ignored and do not trigger alerts [22, 48, 49]. In contrast, guessing the source port with SadDNS method (Section 3.2) or injecting malicious payload via IPv4 fragmentation (Section 3.3) generate more traffic than BGP hijacks, but only locally on the network of the victim DNS resolver or the target nameserver. In contrast to BGP hijacks the attack is performed on the data plane, and is hence not visible in the global BGP routing table in the Internet.

SadDNS attack creates a large traffic volume and hence may be detected by the affected networks. FragDNS attacks against domains that uses a global sequentially incremental IPID counter are the stealthiest.

SIGCOMM '21: Proceedings of the 2021 ACM SIGCOMM 2021 Conference, August 2021, Pages 836-849
Accepted version. https://doi.org/10.1145/3452296.3472933

6 COUNTERMEASURES

Almost all Internet systems, applications and even security mechanisms use DNS. As we showed, a vulnerable DNS introduces not only threats to systems using it but also to security mechanisms, such as PKI. We provide recommendations to mitigate that threat.

We also set up a tool at https://crosslayerattacks.sit.fraunhofer.de to allow clients to check if their networks are operating DNS platforms vulnerable to the cache poisoning attacks evaluated in our work. In the rest of this section we separately explain our recommendations for DNS servers to prevent cache poisoning attacks and then for applications to prevent cross-layer attacks.

6.1 DNS servers

In addition to recommendations and best practices for patching DNS servers, such as those in [RFC5452] [43], we recommend a new countermeasure we call **security by obscurity**. Our experience of cache poisoning evaluation in the Internet showed that the less information the adversary has, the more hard it becomes to launch the attacks in practice. Security by obscurity proves effective not only against off-path but also against on-path MitM attacks. Although it is a known bad practice in cryptography it turns out useful in practice. Specifically, for launching the attacks the attackers need to collect intelligence about the target victims, such as which caching policies are used, which IP addresses are assigned to the resolver - randomising or blocking this information, will make a successful attack harder. The network administrators can deploy countermeasures to make such information difficult to leak, e.g., DNS resolvers should use multiple caches with different DNS software on each, resolvers should not send ICMP errors, nameservers should randomise records in responses.

Preventing queries. Server operators might choose to configure systems to do less (or no) DNS lookups, ie. in the case of email servers. This reduces the chance an attacker can trigger a query to start the poisoning.

Blocking fragmentation. Resolver operators can block fragmented responses in firewalls to reduce the applicability of FragDNS attacks. Some operators only implement filtering of small fragments (i.e., Google's 8.8.8.8) which can prevent the attack since the attacker might not be able to cause a nameserver response of the size needed to reach the filtering limit.

Randomise DNS responses. Randomising nameserver responses complicates the FragDNS attack as the attacker needs to predict the UDP checksum of the original nameserver's response.

0x20 Encoding. 0x20 Encoding adds entropy to the DNS query which must be matched by the response. This complicates the SadDNS attack to a point where it is no longer viable (ie. adding 0x20 Encoding to a domain with 16 alphanumeric characters adds 16 bits of entropy to the query). Since this randomness is only contained in the question section of the DNS packet, it cannot prevent the FragDNS attack as it will be in the first fragment along with the TXID.

Securing BGP. Full deployment of RPKI (together with BGPSec) would prevent the HijackDNS attack. However, because of several deployment barriers, most of the prefixes are not protected by RPKI and most ASes do not enforce Route Origin Validation (ROV) [36, 40, 62]. We refer to [39] for a comprehensive discussion of the deployment issues.

6.2 Applications

In the rest of this section we provide recommendations for preventing cross-layer attacks that use DNS cache poisoning.

Separate resolvers and caches. It is common in networks to use one DNS resolver for multiple services and servers. Our attacks exploit that. We recommend using different DNS resolvers (each with a distinct cache) for each system.

Third party authentication (TLS). Third party authentication, like TLS, can mitigate the attacks against all DNS use-cases which aim to locate a server (i.e.m federation and address lookup use-cases). However, even such mechanisms can only reduce the harm of DNS poisoning, but not completely mitigate it, e.g., adversaries can use DNS cache poisoning to subvert the security of DV during certificates issuance. Furthermore, an attacker can still use cache-poisoning for DoS attacks.

Two factor authentication. Should be enabled by default (and not optional as it is now). This would prevent the attacker from getting access to the account even if it has acquired the login credentials for the victim.

Secure fallback. Instead of allowing a transaction when no information about its authorisation state can be gathered (like done currently in SPF and RPKI) a security-mechanism could decide to not allow it. This however would mean that attacking the availability of DNS for a certain domain would allow DoS attacks instead, preventing a resolver from looking up a domain's SPF records would prevent that domain from sending any emails to the servers using this resolver.

7 CONCLUSIONS

We evaluated methodologies for launching practical DNS cache poisoning attacks and derived insights on the applicability, effectiveness and stealth of these attacks. We then applied the methodologies for a systematic evaluation of cross-layer attacks against popular applications.

Our work demonstrates the significant role that DNS plays in the Internet for ensuring security and stability of the applications and clients. If DNS is vulnerable, our work shows that in addition to traditional attacks, such as redirection to adversarial hosts, weak off-path adversaries can even downgrade protection of security mechanisms, such as RPKI or DV. We provide recommendations for mitigations and developed a public tool at `https://crosslayerattacks.sit.fraunhofer.de` to enable clients to identify vulnerabilities in DNS platforms on their networks.

ACKNOWLEDGEMENTS

We thank the anonymous referees for thoughtful feedback on our work. This work has been co-funded by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) SFB 1119.

SIGCOMM '21: Proceedings of the 2021 ACM SIGCOMM 2021 Conference, August 2021, Pages 836-849
Accepted version. https://doi.org/10.1145/3452296.3472933

REFERENCES

[1] [n.d.]. Best Infrastructure as a Service (IaaS) Providers. https://www.g2.com/categories/infrastructure-as-a-service-iaas. Accessed: 2020-10-09.

[2] [n.d.]. Market Share Analysis: IaaS and IUS, Worldwide, 2018. https://www.gartner.com/en/newsroom/press-releases/2019-07-29-gartner-says-worldwide-iaas-public-cloud-services-market-grew-31point3-percent-in-2018. Accessed: 2020-10-09.

[3] [n.d.]. Market share trends for SSL certificate authorities. https://w3techs.com/technologies/history_overview/ssl_certificate. Accessed: 2020-10-09.

[4] [n.d.]. Quarterly Cloud Spending Blows Past $30B; Incremental Growth Continues to Rise. https://www.srgresearch.com/articles/quarterly-cloud-spending-blows-past-30b-incremental-growth-continues-rise. Accessed: 2020-10-09.

[5] [n.d.]. Top IaaS Providers: 42 Leading Infrastructure-as-a-Service Providers to Streamline Your Operations. https://stackify.com/top-iaas-providers/. Accessed: 2020-10-09.

[6] 2011. The CAIDA AS Relationships Dataset, 2011. http://www.caida.org/data/active/as-relationships/.

[7] 2015. Hacked or Spoofed: Digging into the Malaysia Airlines Website Incident. https://www.trendmicro.com/vinfo/us/security/news/cyber-attacks/hacked-or-spoofed-digging-into-the-malaysia-airlines-website-compromise. Accessed: 2021-1-19.

[8] 2015. Webnic Registrar Blamed for Hijack of Lenovo, Google Domains. https://krebsonsecurity.com/2015/02/webnic-registrar-blamed-for-hijack-of-lenovo-google-domains/. Accessed: 2021-1-19.

[9] 2018. DNSpionage Campaign Targets Middle East. https://blog.talosintelligence.com/2018/11/dnspionage-campaign-targets-middle-east.html. Accessed: 2021-01-19.

[10] 2019. Global DNS Hijacking Campaign: DNS Record Manipulation at Scale. https://www.fireeye.com/blog/threat-research/2019/01/global-dns-hijacking-campaign-dns-record-manipulation-at-scale.html. Accessed: 2021-1-19.

[11] 2019. Sea Turtle keeps on swimming, finds new victims, DNS hijacking techniques. https://blog.talosintelligence.com/2019/07/sea-turtle-keeps-on-swimming.html. Accessed: 2021-01-19.

[12] 2019. 'Unprecedented' DNS Hijacking Attacks Linked to Iran. https://threatpost.com/unprecedented-dns-hijacking-attacks-linked-to-iran/140737/

[13] 2020. Security Incident on November 13, 2020. https://blog.liquid.com/security-incident-november-13-2020. Accessed: 2021-01-19.

[14] aconet. [n.d.]. eduroam-Teilnehmer in Österreich. https://www.aco.net/eduroam_teilnehmer.html. Accessed: 2020-12-02.

[15] F. Alharbi, J. Chang, Y. Zhou, F. Qian, Z. Qian, and N. Abu-Ghazaleh. 2019. Collaborative Client-Side DNS Cache Poisoning Attack. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. 1153–1161. https://doi.org/10.1109/INFOCOM.2019.8737514

[16] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. 2017. Hijacking bitcoin: Routing attacks on cryptocurrencies. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 375–392.

[17] Dan J. Bernstein. 2002. DNS Forgery. Internet publication at http://cr.yp.to/djbdns/forgery.html.

[18] Robert Beverly and Steven Bauer. 2005. The Spoofer project: Inferring the extent of source address filtering on the Internet. In *Usenix Sruti*, Vol. 5. 53–59.

[19] Robert Beverly, Arthur Berger, Young Hyun, and K Claffy. 2009. Understanding the efficacy of deployed internet source address validation filtering. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*. 356–369.

[20] Robert Beverly, Ryan Koga, and KC Claffy. 2013. Initial longitudinal analysis of IP source spoofing capability on the Internet. *Internet Society* (2013), 313.

[21] Henry Birge-Lee, Yixin Sun, Anne Edmundson, Jennifer Rexford, and Prateek Mittal. 2018. Bamboozling Certificate Authorities with BGP. In *27th USENIX Security Symposium (USENIX Security 18)*. 833–849.

[22] Peter Boothe, James Hiebert, and Randy Bush. 2006. Short-lived prefix hijacking on the Internet. *In Proc. of the NANOG* 36 (2006).

[23] Markus Brandt, Tianxiang Dai, Amit Klein, Haya Shulman, and Michael Waidner. 2018. Domain Validation++ For MitM-Resilient PKI. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2060–2076.

[24] R Bush and R Austein. 2013. RFC 6810: The Resource Public Key Infrastructure (RPKI) to Router Protocol.

[25] Taejoong Chung, Roland van Rijswijk-Deij, Balakrishnan Chandrasekaran, David Choffnes, Dave Levin, Bruce M Maggs, Alan Mislove, and Christo Wilson. 2017. A Longitudinal, End-to-End View of the DNSSEC Ecosystem. In *26th USENIX Security Symposium (USENIX Security 17)*. 1307–1322.

[26] Taejoong Chung, Roland van Rijswijk-Deij, David Choffnes, Dave Levin, Bruce M Maggs, Alan Mislove, and Christo Wilson. 2017. Understanding the role of registrars in DNSSEC deployment. In *Proceedings of the 2017 Internet Measurement Conference*. 369–383.

[27] D. Madory. 2018. Recent Routing Incidents: Using BGP to Hijack DNS and more. https://www.lacnic.net/innovaportal/file/3207/1/dougmadory_lacnic_30_rosario.pdf

[28] David Dagon, Manos Antonakakis, Paul Vixie, Tatuya Jinmei, and Wenke Lee. 2008. Increased DNS forgery resistance through 0x20-bit encoding: security via leet queries. In *ACM Conference on Computer and Communications Security*, Peng Ning, Paul F. Syverson, and Somesh Jha (Eds.). ACM, 211–222. http://doi.acm.org/10.1145/1455770.1455798

[29] Tianxiang Dai, Philipp Jeitner, Haya Shulman, and Michael Waidner. 2021. The Hijackers Guide To The Galaxy: Off-Path Taking Over Internet Resources. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association. https://www.usenix.org/conference/usenixsecurity21/presentation/dai

[30] DFN-Verein. [n.d.]. Karte der aktuellen eduroam Standorte in Deutschland. https://www.dfn.de/dienstleistungen/eduroam/. Accessed: 2020-12-02.

[31] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. *Tor: The second-generation onion router*. Technical Report. Naval Research Lab Washington DC.

[32] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J. Alex Halderman. 2015. A Search Engine Backed by Internet-Wide Scanning. In *22nd ACM Conference on Computer and Communications Security*.

[33] Tony Finch, Evan Hunt, Peter van Dijk, Anthony Eden, and Willem Mekking. 2019. *Address-specific DNS aliases (ANAME)*. Internet-Draft draft-ietf-dnsop-aname-03. IETF Secretariat. http://www.ietf.org/internet-drafts/draft-ietf-dnsop-aname-03.txt.

[34] Pedro Franco. 2014. *Understanding bitcoin*. Wiley Online Library.

[35] Lixin Gao and Jennifer Rexford. 2001. Stable Internet routing without global coordination. *IEEE/ACM Transactions on networking* 9, 6 (2001), 681–692.

[36] Yossi Gilad, Avichai Cohen, Amir Herzberg, Michael Schapira, and Haya Shulman. 2017. Are We There Yet? On RPKI's Deployment and Security.. In *NDSS*.

[37] Amir Herzberg and Haya Shulman. 2012. Security of Patched DNS. In *Computer Security - ESORICS 2012 - 17th European Symposium on Research in Computer Security, Pisa, Italy, September 10-12, 2012. Proceedings*. 271–288.

[38] Amir Herzberg and Haya Shulman. 2013. Fragmentation Considered Poisonous: or one-domain-to-rule-them-all.org. In *IEEE CNS 2013. The Conference on Communications and Network Security, Washington, D.C., U.S.* IEEE.

[39] Tomas Hlavacek, Italo Cunha, Yossi Gilad, Amir Herzberg, Ethan Katz-Bassett, Michael Schapira, and Haya Shulman. 2020. DISCO: Sidestepping RPKI's Deployment Barriers. In *Network and Distributed System Security Symposium (NDSS)*.

[40] Tomas Hlavacek, Amir Herzberg, Haya Shulman, and Michael Waidner. 2018. Practical Experience: Methodologies for Measuring Route Origin Validation. In *48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2018, Luxembourg City, Luxembourg, June 25-28, 2018*. 634–641. https://doi.org/10.1109/DSN.2018.00070

[41] P Hoffman and P McManus. 2018. RFC 8484: DNS Queries over HTTPS (DoH).

[42] Z Hu, L Zhu, J Heidemann, A Mankin, D Wessels, and P Hoffman. 2016. RFC 7858-Specification for DNS over Transport Layer Security (TLS).

[43] A. Hubert and R. van Mook. 2009. *Measures for Making DNS More Resilient against Forged Answers*. RFC 5452. RFC Editor.

[44] Philipp Jeitner and Haya Shulman. 2021. Injection Attacks Reloaded: Tunnelling Malicious Payloads over DNS. In *30th USENIX Security Symposium (USENIX Security 21)*.

[45] Philipp Jeitner, Haya Shulman, and Michael Waidner. 2020. The Impact of DNS Insecurity on Time. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 266–277.

[46] Jisc. [n.d.]. Organisations participating in eduroam in the UK. https://www.jisc.ac.uk/eduroam/participating-organisations. Accessed: 2020-12-02.

[47] Dan Kaminsky. 2008. It's the End of the Cache As We Know It. Presentation at Blackhat Briefings.

[48] Josh Karlin, Stephanie Forrest, and Jennifer Rexford. 2008. Autonomous security for autonomous systems. *Computer Networks* 52, 15 (2008), 2908–2923.

[49] Varun Khare, Qing Ju, and Beichuan Zhang. 2012. Concurrent prefix hijacks: Occurrence and impacts. In *Proceedings of the 2012 Internet Measurement Conference*. ACM, 29–36.

[50] Amit Klein. 2007. BIND 9 DNS cache poisoning. *Report, Trusteer, Ltd* 3 (2007).

[51] Amit Klein. 2007. Windows DNS Server Cache Poisoning,".

[52] Amit Klein. 2020. Cross Layer Attacks and How to Use Them (for DNS Cache Poisoning, Device Tracking and More). *arXiv preprint arXiv:2012.07432* (2020).

[53] Amit Klein, Haya Shulman, and Michael Waidner. 2017. Internet-wide study of DNS cache injections. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 1–9.

[54] Matt Lepinski, Stephen Kent, and Derrick Kong. 2012. A profile for route origin authorizations (ROAs). *IETF, RFC* 6482 (2012).

[55] Qasim Lone, Matthew Luckie, Maciej Korczyński, Hadi Asghari, Mobin Javed, and Michel van Eeten. 2018. Using Crowdsourcing Marketplaces for Network Measurements: The Case of Spoofer. In *2018 Network Traffic Measurement and Analysis Conference (TMA)*. IEEE, 1–8.

[56] Matthew Luckie, Robert Beverly, Ryan Koga, Ken Keys, Joshua A Kroll, and k claffy. 2019. Network Hygiene, Incentives, and Regulation: Deployment of Source Address Validation in the Internet. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 465–480.

SIGCOMM '21: Proceedings of the 2021 ACM SIGCOMM 2021 Conference, August 2021, Pages 836-849
Accepted version. https://doi.org/10.1145/3452296.3472933

[57] Keyu Man, Zhiyun Qian, Zhongjie Wang, Xiaofeng Zheng, Youjun Huang, and Haixin Duan. 2020. DNS Cache Poisoning Attack Reloaded: Revolutions with Side Channels. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, USA) *(CCS '20)*. Association for Computing Machinery, New York, NY, USA, 1337–1350. https://doi.org/10.1145/3372297.3417280

[58] Jared Mauch. 2013. Open resolver project. In *Presentation, DNS-OARC Spring 2013 Workshop (Dublin)*.

[59] P. Mockapetris. 1987. *Domain names - concepts and facilities*. STD 13. RFC Editor. http://www.rfc-editor.org/rfc/rfc1034.txt http://www.rfc-editor.org/rfc/rfc1034.txt.

[60] P. Mockapetris. 1987. *Domain names - implementation and specification*. STD 13. RFC Editor. http://www.rfc-editor.org/rfc/rfc1035.txt http://www.rfc-editor.org/rfc/rfc1035.txt.

[61] Pradosh Mohapatra, John Scudder, David Ward, Randy Bush, and Rob Austein. 2013. BGP prefix origin validation. In *IETF RFC 6811*.

[62] Andreas Reuter, Randy Bush, Ítalo Cunha, Ethan Katz-Bassett, Thomas C. Schmidt, and Matthias Wählisch. 2017. Towards a Rigorous Methodology for Measuring Adoption of RPKI Route Validation and Filtering. *CoRR* abs/1706.04263 (2017). arXiv:1706.04263 http://arxiv.org/abs/1706.04263

[63] RIPE NCC. 2021. RIS Raw Data. https://www.ripe.net/analyse/internet-measurements/routing-information-service-ris/ris-raw-data

[64] S. Goldberg. 2018. The myetherwallet.com hijack and why it's risky to hold cryptocurrency in a webapp. https://medium.com/@goldbe/the-myetherwallet-com-hijack-and-why-its-risky-to-hold-cryptocurrency-in-a-webapp-261131fad278

[65] Haya Shulman. 2014. Pretty bad privacy: Pitfalls of DNS encryption. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*. 191–200.

[66] Haya Shulman and Michael Waidner. 2015. Towards security of internet naming infrastructure. In *European Symposium on Research in Computer Security*. Springer, 3–22.

[67] Haya Shulman and Michael Waidner. 2017. One Key to Sign Them All Considered Vulnerable: Evaluation of DNSSEC in the Internet.. In *NSDI*. 131–144.

[68] Sandra Siby, Marc Juarez, Claudia Diaz, Narseo Vallina-Rodriguez, and Carmela Troncoso. 2019. Encrypted DNS–> Privacy? A Traffic Analysis Perspective. *arXiv preprint arXiv:1906.09682* (2019).

[69] Jonathan Spring. [n.d.]. Probable Cache Poisoning of Mail Handling Domains. https://insights.sei.cmu.edu/cert/2014/09/-probable-cache-poisoning-of-mail-handling-domains.html.

[70] Yixin Sun, Maria Apostolaki, Henry Birge-Lee, Laurent Vanbever, Jennifer Rexford, Mung Chiang, and Prateek Mittal. 2021. Securing internet applications from routing attacks. *Commun. ACM* 64, 6 (2021), 86–96.

[71] University of Oregon. 2012. Route Views Project. http://bgplay.routeviews.org/.

[72] Paul Vixie. 1995. DNS and BIND Security Issues. In *Proceedings of the 5th Symposium on UNIX Security*. USENIX Association, Berkeley, CA, USA, 209–216.

[73] S Weiler and D Blacka. 2013. RFC 6840: Clarifications and Implementation Notes for DNS Security (DNSSEC). *IETF Standard* (2013).

[74] Xiaofeng Zheng, Chaoyi Lu, Jian Peng, Qiushi Yang, Dongjie Zhou, Baojun Liu, Keyu Man, Shuang Hao, Haixin Duan, and Zhiyun Qian. 2020. Poison Over Troubled Forwarders: A Cache Poisoning Attack Targeting DNS Forwarding Devices. In *29th USENIX Security Symposium (USENIX Security 20)*. 577–593.

SIGCOMM '21: Proceedings of the 2021 ACM SIGCOMM 2021 Conference, August 2021, Pages 836-849
Accepted version. https://doi.org/10.1145/3452296.3472933

F. Injection Attacks Reloaded: Tunnelling Malicious Payloads over DNS

[6] P. Jeitner and H. Shulman, “Injection attacks reloaded: Tunnelling malicious payloads over DNS,” in *30th USENIX Security Symposium (USENIX Security 21)*, CORE A*, USENIX Association, Aug. 2021. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/jeitner.

Injection Attacks Reloaded: Tunnelling Malicious Payloads over DNS

Philipp Jeitner
TU Darmstadt

Haya Shulman
Fraunhofer SIT

Abstract

The traditional design principle for Internet protocols indicates: "Be strict when sending and tolerant when receiving" [RFC1958], and DNS is no exception to this. The transparency of DNS in handling the DNS records, also standardised specifically for DNS [RFC3597], is one of the key features that made it such a popular platform facilitating a constantly increasing number of new applications. An application simply creates a new DNS record and can instantly start distributing it over DNS without requiring any changes to the DNS servers and platforms. Our Internet wide study confirms that more than 1.3M (96% of tested) open DNS resolvers are standard compliant and treat DNS records transparently.

In this work we show that this 'transparency' introduces a severe vulnerability in the Internet: we demonstrate a new method to launch string injection attacks by encoding malicious payloads into DNS records. We show how to weaponise such DNS records to attack popular applications. For instance, we apply string injection to launch a new type of DNS cache poisoning attack, which we evaluated against a population of open resolvers and found 105K to be vulnerable. Such cache poisoning cannot be prevented with common setups of DNSSEC. Our attacks apply to internal as well as to public services, for instance, we reveal that all eduroam services are vulnerable to our injection attacks, allowing us to launch exploits ranging from unauthorised access to eduroam networks to resource starvation. Depending on the application, our attacks cause system crashes, data corruption and leakage, degradation of security, and can introduce remote code execution and arbitrary errors.

In our evaluation of the attacks in the Internet we find that all the standard compliant open DNS resolvers we tested allow our injection attacks against applications and users on their networks.

1 Introduction

Domain Name System (DNS) is a key component of the Internet. Originally designed to translate domain names to IP addresses, DNS has developed into a complex infrastructure providing platform to a constantly increasing number of applications. The applications that are built over DNS range from Internet specific services, such as location of hosts using GPOS record [RFC1712] [1] to security mechanisms, such as authentication with certificates using TLSA record [RFC6698] [2]. The core design feature that allows DNS to support new applications without involving any changes to its infrastructure is the requirement that the handling of the DNS records is done transparently [RFC3597, RFC1035] [3, 4]. Namely, DNS should not attempt to interpret nor understand the records that it is serving. Thanks to this feature new DNS records can be easily added to the DNS infrastructure without requiring any modifications, and novel applications can instantly run over DNS using the newly added records.

In this work we show that the transparency-feature of DNS, while critical for fast and smooth deployment of new technologies, introduces a gaping hole in Internet security.

Exploiting transparency to encode injections. We exploit the transparency of the DNS lookups to encode injection strings into the payloads of DNS records. The attacker places the malicious records in the zonefile of its domain. When provided by the attacker's nameserver the records appear to contain legitimate mappings under the domain controlled by the attacker. However, when the record is processed by the receiving victim application, a misinterpretation occurs - resulting in the injection attack. Our attacks exploit two key factors caused by the transparency of DNS: (1) the DNS resolvers do not alter the received records hence the malicious encoding is preserved intact and (2) the receiving applications do not sanitise the received records. We devise injection payloads to attack popular applications.

Applications do not sanitise DNS records. Classical injection attacks are well known and have been extensively studied: the attacker provides a malicious input through a web application to alter the structure of a command, hence subverting the logic of the application, e.g., [5, 6]. Such injection attacks are easy to mitigate in practice: the input of the user is validated and invalid characters are filtered before

2021 30th USENIX Security Symposium
Accepted version. `https://www.usenix.org/conference/usenixsecurity21/presentation/jeitner`

reaching the application. Due to the long history of injection vulnerabilities in web applications and the awareness to the potential risks, most applications validate user input [7].

We show that in contrast to user input, the inputs provided by the DNS resolvers are not validated. For instance, user credentials provided to LDAP via a web interface to authenticate the user and enable it to use services, are validated, while DNS values that are provided to LDAP to route the authentication request to an authentication server are not validated. We show how to construct malicious payloads to launch injection attacks, such as XSS and cache poisoning, against a variety of applications and services, including DNS caches, LDAP, eduroam.

Attacker model. The attacker causes the victim resolvers to issue queries for records that encode malicious payloads, e.g., by deploying an ad-network or by sending an Email from attacker's domain to the victim. The resolvers cache the records received in DNS responses and provide them to applications and users. We illustrate the attacker model and the setup with eduroam as example victim application, in Figure 1. Using our "weak" attacker we demonstrate a range of attacks against popular applications and services that use DNS lookups, including DNS cache poisoning, applications' crashes, downgrade of security mechanisms, remote code execution vulnerabilities, XSS.

Contributions

The core issue that we explore in this work is the balance between security and the requirement to enable easy deployment of new applications over DNS. Our contributions include:

• **Analysis of components in resolution chain.** We analyse the interaction between the applications and the components in DNS resolution chain. We find that the processing applied by the DNS resolvers over DNS records is compliant with the requirement in [RFC3597,RFC1035] and preserves the structure of the malicious inputs encoded by the attackers - this property is key to our attacks. We validate this also in the Internet against 3M open DNS resolvers. Our measurement study reveals that more than 96% of the open DNS resolvers do not modify the records that they receive from the nameservers, and serve them intact to the calling applications.

• **Study of DNS input validation.** We find that although DNS delivers untrusted data from potentially malicious Internet servers the applications trust the data returned by the DNS resolvers. Our study shows that the lack of input validation is systematic and prevalent and is not a bug mistakenly introduced by developers in some isolated cases – this includes custom functions in applications as well as standardised function calls of IEEE POSIX, e.g., `gethostbyname()`.

• **Implementation of injection attacks over DNS.** We demonstrate that the attackers can systematically and efficiently construct attack vectors and show how to integrate them into the zonefile of a malicious domain operated by the attacker. We then demonstrate injection attacks over DNS against popular applications using these malicious DNS records.

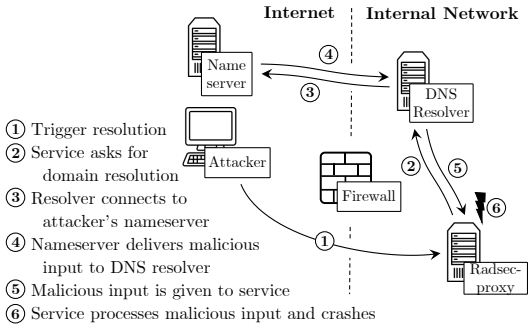


Figure 1: Attack and setup overview, with eduroam radsecproxy as example application.

• **Injections into DNS caches.** We show how to encode payloads for injecting malicious records into DNS caches. When cached by the victim DNS resolver, a misinterpretation occurs mapping a resource of some victim domain to an attacker's IP address. In contrast to classical cache poisoning, [8–10], which require a strong (on-path) attacker or assume specific network properties, such as side channels or fragmentation, our cache poisoning attacks do not make any requirements on attacker capabilities. We also do not need to spoof IP addresses in DNS responses - a requirement which is essential in prior cache poisoning attacks. We automated the evaluation of our poisoning attacks, which allowed us to launch them against a large set of 3M target DNS resolvers, performing successful poisoning against 105K resolvers. Implementation of previous cache poisoning attacks had to be manually tailored per each target - automating the attacks would result in a negligible success probability. Hence the previous attacks were carried out against at most a handful of targets, and the rest of the resolvers' population was merely checked for properties that make them potential targets. Furthermore, in contrast to previous poisoning attacks, ours cannot be prevented with common setups of DNSSEC [11–13].

• **Evaluation of injection attacks over DNS.** We evaluate our injection attacks against popular applications (listed in Table 1). Our analysis of the vulnerabilities in applications, where suitable, combines fuzzing, source code review and dynamic (black box) execution. We evaluate our injection attacks against a population of more than 3M open resolvers in the Internet. We provide additional information on our evaluations at `https://xdi-attack.net`.

Ethics and Disclosure

We have already taken preliminary steps to address these vulnerabilities by contacting the DNS software vendors as well the applications evaluated in this work. We experimentally evaluated the attacks reported in this work against servers that we set up as well as against open DNS resolvers in the Internet using domains that we control. This allowed us to

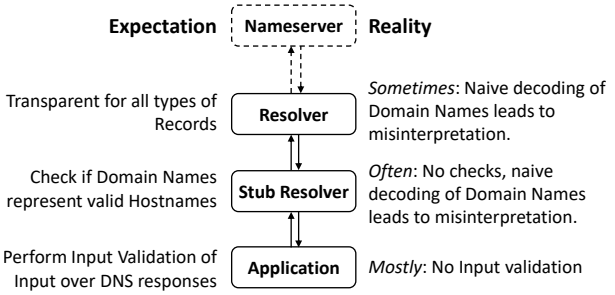


Figure 2: Expected vs. actual behaviour in DNS lookup.

validate the presence of the vulnerabilities without exploiting them against real victims and without causing damage to the networks nor services in the Internet. Our attacks similarly apply also to non-open DNS resolvers.

Prior to performing the validation of the vulnerabilities in the wild we received an approval from our research institution. In the next steps we will be coordinating countermeasures with the DNS and applications vendors, as well as the IETF community.

Organisation

In Section 2 we analyse the interaction between components in DNS resolution chain. In Section 3, we demonstrate injection attacks against popular applications. In Section 4 we evaluate our attacks against open resolvers in the Internet. We propose countermeasures in Section 5, review related work in Section 6 and conclude in Section 7.

2 Analysis of DNS Resolution Chain

In this section we analyse the interaction between the components relevant to processing DNS records in responses. In our analysis we use popular DNS resolvers and stub resolver implementations built into operating systems and programming languages, and experimentally test how they handle control characters in domain names and if they modify any of the maliciously crafted payloads needed to conduct the application-specific exploits that we evaluate in this work.

2.1 Components in DNS Lookup

We consider 3 different types of software components which fulfil different roles during a DNS lookup: (recursive) DNS resolvers, stub DNS resolvers and applications. In our setup the DNS namerservers are controlled by the attacker and provide maliciously-encoded DNS responses. The victim resolvers serve these records to the stub resolvers in applications. We provide an illustration of these software components together with the expectations on their behaviour and their actual behaviour discovered in this work in Figure 2. DNS lookups by system stub resolvers are implemented in various ways in applications. For standard A, AAAA and PTR queries, system C libraries include POSIX-standardised [14, 15] functionality in form of the `gethostbyname()`, `getaddrinfo()`, `gethostbyaddr()` and `getnameinfo()` functions. When considering full fledged resolution functionality, which also supports other query types, like MX, SRV or TXT, there is no standardised API so applications need to rely on third party libraries for constructing the DNS packets or for parsing the DNS responses they receive from the network (e.g., recursive resolvers). DNS software implementations in recursive resolvers and forwarders typically implement their own packet decoding logic since they do not interface with the applications directly and do not need to decode parts of the DNS records at all.

2.2 System Stub Resolvers

Stub resolvers provide the interface between the applications and the DNS resolvers. Applications typically do not issue DNS requests to the recursive resolvers directly, but instead use a POSIX standardised API [14, 15] to supply the hostname they want to resolve, and the system's standard C library translates this into a DNS request and parses the response from the recursive DNS resolver. The applications can perform hostname-to-address (A, AAAA) and reverse (PTR) lookups via the `gethostbyname()`, `gethostbyaddr()`, `getaddrinfo()` and `getnameinfo()` calls. This API is defined to return CNAME aliases and results of reverse lookups as null-terminated 'host names' [14, p. 320] in its returned `hostent` structure, defined [16, 17] to only contain Latin characters, digits and hyphens (`"A-Z","a-z","0-9","-"`).

According to this definition the expected behaviour of a system stub resolver is to check that any domain name returned by a call to any of the POSIX-standardised resolver functions must be checked before returning it to the application.

DNS Record Processing. We analyse the DNS record processing done by stub-resolvers in Section 3.6.1 in detail. We find that while one implementation (glibc) is fully conforming with our expectations, most stub-resolvers are not. Stub resolvers that do not validate the value of the hostname expose the applications to attacks; we show these in Section 3.6.2.

2.3 Recursive Resolvers and Forwarders

Recursive resolvers provide lookup services to system DNS resolvers: they locate nameservers and look up the requested records.

DNS resolvers and forwarders process DNS records in their line-format, i.e., they store and handle domain names in their encoded form and do not try to parse the records that they do not need to understand, like TXT or MX. This makes them transparent for any binary values inside domains or other record data as is required by the DNS standard [4, 18, 19]. For internal caching purposes, a DNS resolver may choose to decode a domain name for further processing or to store it inside a cache.

DNS Record Processing. DNS resolvers are expected to transparently handle known and unknown DNS record types [RFC3597] to ensure forward-compatibility and compatibility

with mechanisms such as DNSSEC, as any change in the encoded record would invalidate the DNSSEC signatures. Moreover, [RFC1034] states that software like DNS resolvers should not try to decode domain names into a string as stub-resolvers would do. Most resolvers we tested are [RFC1034] compliant: they handle any payload transparently or only change the case of letters to lowercase, which is allowed since domain names are defined to be case insensitive [18]. One tested resolver software, MaraDNS Deadwood 3.2.14, was unable to handle inject$_{\backslash 000}$.

We also identified an '[RFC1034] non-compliant' behaviour, which we describe next. We take as an example a systemd-resolved forwarding DNS resolver which when receiving a DNS record decodes and escapes all included domain names into zero-terminated strings like a system stub resolver would do. Then it caches the records as decoded strings. The resolver unescapes and re-encodes the records when sending them to a requesting client or application. This behaviour ensures that any misinterpretation during decoding (step 2 in Figure 7) will cause the domain name to be modified. This modification cannot be detected by downstream resolvers or applications because the misinterpreted record is re-encoded instead of just being passed as binary.

2.4 Applications

Applications are the source of DNS lookups and process the records in the response after it has traversed all the other components in the resolution chain. The records may contain unexpected characters which the application cannot process correctly. The impact of injection attacks on the applications depends on the use case of the DNS lookup. For example, applications which do service discovery or authentication lookups typically need to parse DNS packets by themselves, because system stub resolvers do not support queries for such record types. This can also be abused to cause misinterpretation of domain names or other data in DNS records.

DNS Record Processing. When applications perform DNS lookups of types other than A, AAAA or PTR, they implement DNS lookup functionality by themselves as there is no standardised API for this. While the standard libraries of some programming languages like java, go or nodejs include functions for query types like MX, SRV and TXT, there is no recommended behaviour for such functions and they do not perform any validation of the data which is passed to the application. This also applies to DNS lookup implementations done in applications directly. Domain names are typically not validated nor escaped when decoded into a string.

Theoretically, when applications use the system resolver to do DNS lookups, they could implement validation (step 3 in Figure 7) by themselves, to ensure no malicious input is processed. However, applications would still not be able to detect decoding errors in step 2 in Figure 7. For example, since the application does not see the binary DNS data, it cannot determine if the example domain in Figure 7 is `"a\.b.<>.com."` or `"a.b.<>.com."`.

3 Injection Attacks Against Applications

In this section we demonstrate injection attacks using malicious payloads tunnelled over DNS. We first explain our study methodology (Section 3.1) and then show attacks against selected popular applications (Sections 3.2 – 3.6) taking DNS software as the first example application.

3.1 Study Methodology

3.1.1 Attack overview

The attack is illustrated in Figure 1. The target victim application, e.g., radsecproxy of eduroam, is behind a firewall on the victim network. The attack is initiated by causing the target service to issue a request via its DNS resolver to the attacker's domain, e.g., by trying to authenticate at eduroam's wireless access point (steps ①, ② and ③ in Figure 1). In the zonefile of its domain, the attacker encodes malicious payloads into the DNS records. The records are then provided in responses to the queries of the DNS resolvers (step ④), and are subsequently relayed to the requesting services, in the example in Figure 1, to the radsecproxy server (step ⑤) which processes the attacker's authentication request. The payload then causes the application to divert from a standard behaviour (step ⑥), e.g., causing it to allow unauthenticated network access.

3.1.2 Selecting Target Applications

We evaluate injection attacks against popular services and applications. In this work we present attacks against some selected applications listed in Table 1. We select them based on the following considerations:

DNS Use-Case. We identify 4 different use-cases of DNS (address lookup, service discovery, reverse lookup and authentication). We select a few popular applications and services for each DNS use case.

Triggering query. The attacker must be able to trigger a DNS lookup, e.g., via a script in a browser, via an Email to a target Email Server. We summarise methods for triggering query and setting the query domain in Table 1, column 'Trigger/Set query'. We also prefer target applications which allow the attacker to trigger queries to attacker-selected domains, e.g., by sending an Email or triggering a query via javascript in browsers.

Attack surface. To find meaningful attacks, we focus on applications where input from DNS is used for some interesting action, e.g., for implementing a cache, creating a URL, etc. We do not analyse applications which only do standard address lookups (without caching), as such a scenario does not create a meaningful attack surface, even if no input validation is performed. We list the applications, along with how the DNS inputs are used by those applications, in Table 1, column 'Input use'.

Usage of vulnerable resolvers. For applications which use the system's libc resolver for DNS lookups, we prefer those

DNS Use-Case	Application	Trigger	Set	Uses libc	Validates	Input use	Attack found
		Query					
Address lookups (A, CNAME)	Chrome	js,html		yes	no	cache	no
	Firefox	js,html		yes	no	cache	no
	Opera	js,html		yes	no	cache	no
	Edge	js,html		yes	no	cache	no
	unscd	client app		yes	no	cache	no
	java	client app		both	no	cache	no
	ping(win32)	✗	✗	yes	no	display	yes
discovery (MX, SRV, NAPTR)	openjdk	login	✗	no	no	create URL	yes
	ldapsearch	login	✗	no	no	create URL	no
	radsecproxy	login		no	no	configure	yes
Reverse lookups (PTR)	ping(linux)	✗	✗	yes	no	display	yes
	trace(linux)	✗	✗	yes	no	display	yes
	OpenWRT	✗	ping	yes	no	display	yes
	openssh	login		yes	no	display,log	yes
Authentication (TXT, TLSA)	policyd-spf	SMTP		no	no	text protocol	no
	libspf2	SMTP		no	-	parse	yes
All	Resolvers	client app		no	some	cache	yes

Table 1: Analysed software and tools.

which are often used on systems with vulnerable libc implementations. For example, OpenWRT was chosen because it uses a vulnerable libc implementation with uclibc.

3.1.3 Vulnerabilities Analysis

After identifying a target application, we analyse its DNS usage and whether input from the DNS is validated, as follows: (1) source code review, (2) fuzzing and (3) by executing the application, feeding it with inputs and analysing the resulting behaviour and the outputs. We first test if an application does not validate DNS records received in input. For such applications we then check how the input is used by the application, and construct attack vectors accordingly, e.g., XSS injection. The results of this analysis are listed in Table 1, the found vulnerabilities in Table 2.

3.2 DNS Caches

The attacks exploit the fact that domains and hostnames are not restricted to characters, and implements misinterpretation of domain names due to presence of `"\."` and of `"\000"` characters. These characters cause the appearance of `"."` to be altered hence manipulating the subdomains of a given parent domain.

The attacker can trigger a DNS query directly when launching the attack against open resolver or can initiate the attack via an application which uses the target DNS resolver, e.g., a web browser or an Email server.

3.2.1 DNS Cache Poisoning Attacks

In this section we present two types of cache-injection attacks which are based on domain name misinterpretation and verify them against popular DNS resolvers' software as well as against 3M open DNS resolvers in the Internet. We also show how to extend our poisoning attacks against forwarders and provide an example of the poisoning attack we launched using the Verisign Public DNS[1] resolver.

[1] Verisign Public DNS was operated by Verisign at the time the research was conducted. Neustar acquired the IP addresses from Verisign last November to be incorporated into its own UltraDNS Public service [20].

• **Attack #1: Period injection.** To inject a malicious DNS record or to overwrite a cached DNS record with a new value (controlled by an attacker), we design the following record set inject$_{\backslash}$: `www\.target.com. A 6.6.6.6`.

This attack requires the attacker to control a specially-malformed domain `www\.target.com.` under the same parent domain (in this example `com.`) as the domain of its victim, say `www.target.com`. Since most client software does not allow triggering a query for a domain `www\.target.com` directly, to perform injection of a malicious record into the victim's cache, the attacker can set up a CNAME record with arbitrary subdomain, e.g., `injectdot.attacker.com`, as follows:

```
injectdot.attacker.com. CNAME www\.target.com.
www\.target.com. A 6.6.6.6
```

When decoding these records naively without escaping the period (`"\."`) it appears that `www.target.com` has IP address `6.6.6.6`. Caching this misinterpreted record after decoding leads to DNS cache injection.

• **Attack #2: Zero-byte injection.** We design the following record set inject$_{\backslash 000}$, which indicates end of data, for performing DNS cache poisoning.

```
injectzero.attacker.com CNAME
    www.target.com\000.attacker.com
www.target.com\000.attacker.com A 6.6.6.6
```

When naively decoded and fed into a victim cache this record enables an attacker to inject records for arbitrary domains into the cache. In this attack we also use a CNAME alias mapped to some secondary domain `injectzero.attacker.com`, since triggering a query to `www.target.com\000.attacker.com` without direct access to the resolver is not possible with most client software. When decoding this record set into a C-string without escaping the zero-byte after `www.target.com`, the `.attacker.com` is removed since it is after the end of data `\000` value, the DNS software misinterprets the record and caches a record mapping `www.target.com` to IP address `6.6.6.6`.

3.2.2 Evaluation of the Attacks

Every application-level DNS-cache running on a system which misinterprets the inject$_{\backslash}$ or inject$_{\backslash 000}$ payloads (See Table 5) is vulnerable to these attacks. In our Internet study (see Section 4) we found that 105,854 open DNS resolvers (or 8% of 1,3M) are vulnerable to our attacks. Our attack evaluation was automated hence did not include potentially vulnerable resolvers which could result in a successful attack when the evaluation was manually tailored per resolver. These cases include lost packets (we sent only one response to avoid loading the network), resolvers with multiple caches (the attack was tested once against each client-exposed-IP of the resolver). Adjusting our attack to these cases is straightforward, would however generate much more traffic to the tested systems.

Section - Category	DNS use-case	Software	Mis-interpretation is in	Record type	Attacker can choose domain	Possible outcome(s)
3.2 - DNS	Address-lookup	Verisign Public DNS(*)	Resolver	CNAME	yes	Cache injection
3.3 - Eduroam	Service discovery	radsecproxy	Application	NAPTR, SRV	yes	Strip TLS, hijack connection, Crash
3.4 - LDAP	Service discovery	openjdk	Application	SRV	no	Crash
3.5 - Email	Authorization	libspf2	Application	TXT (SPF)	yes	Crash, (potential code execution)
3.6 - Admin tools	Address-, Reverse-lookup	ping,openssh,trace	Stub resolver	CNAME	no	Terminal Escape Code injection
3.6 - Web-interface	Reverse-lookup	OpenWRT luci	Stub resolver	PTR	yes	XSS in Admin web-interface

(*) Recursive service operated by Verisign at the time the research was conducted.

Table 2: Applications' categories with vulnerabilities and attacks exploiting them.

3.2.3 No Countermeasures Against Cache Poisoning

Classic countermeasures against DNS cache poisoning do not mitigate our cache poisoning attacks. The situation is even more risky when the same host is configured as nameserver and resolver, [21], as a lack of validation by the DNS resolver can allow the attacker to also manipulate the zonefile which is hosted on the same machine.

Defences against off-path attackers. Defences against off-path attackers, such as [RFC5452] [22], are not effective against our attacks: we do not send the malicious DNS responses from spoofed IP addresses but respond from a nameserver that we control. Hence in our attacks the attacker does not need to guess the randomisation values, such as UDP source port and the TXID. The bailiwick check [23], which prevents the attackers from responding with values not under their domains is also ineffective against our attacks since the bailiwick check is applied over the records *before* the misinterpretation occurs.

Defences against on-path attackers. Cryptographic defences, most notably DNSSEC [RFC4033-RFC4035], can not prevent our cache poisoning attacks in common setups: in situations where upstream resolvers are used the misinterpreted records are not detected by the downstream DNS forwarders, since those typically do not perform DNSSEC validation[2]. DNSSEC validation is performed by the recursive resolvers over the DNS records in line-format, *before* the decoding and the misinterpretation occur. After the records successfully pass DNSSEC validation, they are cached in a "misinterpreted" form.

Cross-zone CNAME caching. Additionally to the misinterpretation, these attacks require resolvers to cache and process CNAME records across zone-boundaries, i.e., the resolvers must use the misinterpreted second record `www.target.com\000.attacker.com` from zone `attacker.com` to answer queries for `www.target.com`. While this is not typically the case for recursive resolvers, we validated such behaviour in dnsmasq, the most frequently used forwarder on our open resolver dataset: given the records `injectdot.attacker.com CNAME www.victim.com` and `www.victim.com A 6.6.6.6` in response to a query for `injectdot.attacker.com`, dnsmasq will answer queries for `www.victim.com` with `6.6.6.6`. We illustrate how this leads to a vulnerable configuration of forwarder and recursive resolver in the case dnsmasq is combined with a misinterpreting recursive resolver like Verisign Public DNS in Figure 3; our resolver evaluation is in Section 4.

[2]Neither dnsmasq, systemd-resolved nor OpenWRT or Fritz!Box SOHO routers perform DNSSEC validation by default.

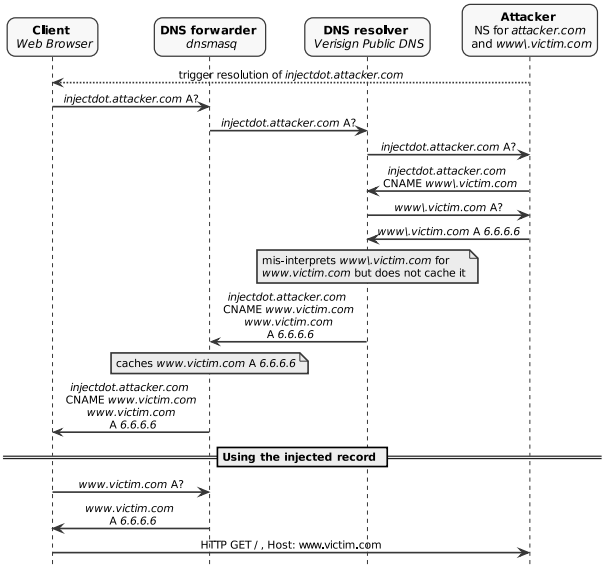


Figure 3: Downstream forwarder attack using dnsmasq with Verisign Public DNS misinterpretation of inject\. payload.

3.2.4 Required attacker capabilities

To launch the inject\000 attack, the adversary only needs to control a nameserver for an arbitrary domain in the internet. In contrast, in order to launch the inject\. attack the adversary has to control a specially crafted malicious sub-domain under the same parent-domain (e.g., `com.`) as his target. This means that conducting this attack requires registering a sub-domain like `www\.target` via a domain registry. Applicability of this attack depends on ability of the attacker to register such sub-domains. For instance, a registry.pw for .pw reported that registering domain `www\.asd.pw` was possible, while `www.asd.pw` or `asd.pw` was not (indicating that they are existing registered domains). Namely the attacker can register `www\.asd.pw` and use it to attack the existing victim domain `asd.pw`.

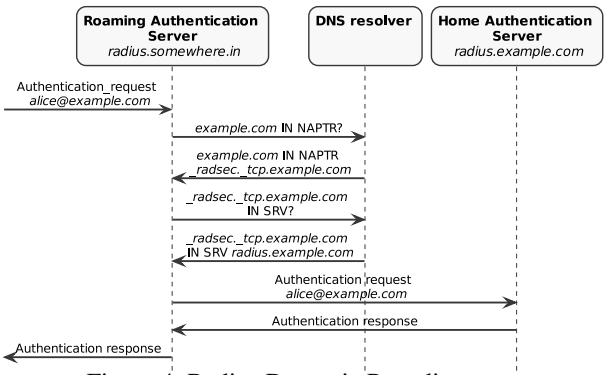


Figure 4: Radius Dynamic Peer discovery.

3.3 Eduroam Peer Discovery

Eduroam federation uses Remote Authentication Dial-In User Service (Radius) [24] for authentication of guest access. Radsecproxy is an application that implements Radius transport over TCP and TLS as well as dynamic peer discovery for servers which do not support these features themselves. Radsecproxy uses a shell-script-based method for dynamically updating the configuration to support the DNS lookups needed for Dynamic Peer discovery. The script is invoked with the domain component of the user's network access identifier (i.e., `example.com` in Figure 4) as its first argument by the radsecproxy server and outputs a new dynamic radsecproxy configuration for the user's realm. Example output of this script (called naptr-eduroam.sh) when invoked from shell is below:

```
$ ./naptr-eduroam.sh example.com
server dynamic_radsec.example.com {
        host radius1.example.com:2083
        host radius2.example.com:2083
        type TLS
}
```

3.3.1 Radius Dynamic Peer Discovery

In this section we provide a detailed explanation of the radius dynamic peer discovery process illustrated in Figure 4, as well as how an adversary can abuse the mechanism to trigger queries. First, a client (alice) connects to a wireless access point at the campus of `somewhere.in` (the domain of that university), providing authentication material including her network access identifier (NAI) `alice@example.com`. The access point then forwards the authentication request to the roaming authentication server at `somewhere.in`. From alice's NAI this server defers that alice's authentication request needs to be routed to the home authentication server of `example.com`. To find the home authentication server, it issues DNS queries for `example.com IN NAPTR?` followed by an `SRV` query of the domains listed in the `NAPTR` record. Finally, the roaming authentication server forwards the authentication request to the home authentication server at `radius.example.com`, which answers the request. The attacker sets up his own domain `attacker.com` and configures his nameserver to answer with one of the attack payloads from Table 3. The attacker provides a username `user@attacker.com` when connecting. This leads the roaming authentication server to send DNS requests via its resolver to the attacker's nameserver. Depending on the payload, a corresponding attack is launched against the roaming authentication server.

3.3.2 Attacks Against Radsecproxy

We found multiple security vulnerabilities in the script [25] for Dynamic Peer Discovery in eduroam, see vulnerabilities in Table 3. The vulnerabilities allow an attacker to control various variables inside the script as well as in the generated dynamic configuration. These vulnerabilities are caused by the lack of input validation of the resulting output of `dig` as well as the usage of `printf`, which negates the escaping of special characters done by `dig`.

To initiate an attack the attacker causes the target system to issue a query to its domain. To make radsecproxy query for a domain of attacker's choice, the attacker just needs to attempt to log-in at an eduroam access point with a username ending with the malicious domain. This triggers `NAPTR` and `SRV` queries to locate the correct authentication server, see messages exchange in Figure 4.

No validation of `dig` output: By changing the NAPTR record's replacement field (`$HOST`), the attacker can control one argument to `dig`, which is not checked for its format. We exploited this to launch two attacks:

- **Attack #1 in Table 3.** Make dig query an attacker-chosen DNS resolver, instead of the default resolver from `/etc/resolv.conf`. We used this attack to verify the vulnerability remotely without causing damage to the tested eduroam network.
- **Attack #2 in Table 3.** Make dig use *any file* on the radsecproxy-system as a "batch-file" (option `-f`), thereby querying all lines in this file as DNS queries. Attackers which are located on-path to the DNS resolver can apply the second attack to read arbitrary files from the system.

Vulnerable usage of `printf`: The (double) use of printf in `naptr-eduroam.sh` allows the attacker to inject arbitrary strings into the dynamically generated configuration file via a format-string attack. The reason is that printf removes the escaping done by dig over the user input, which is given in format specifier argument. This allows the attacker to access radsecproxy's configuration parser and subsequent `confserver_cb` function, which can be used to make the process read any file on the fileystem using `include /path/file`.

- **Attack #3 in Table 3.** We evaluated a 'resource starvation' attack using `/dev/zero` as an input, and caused an infinite 100% CPU usage loop in the configuration file parser.
- **Attack #4 in Table 3.** In this attack we demonstrate how the attacker can manipulate the generated dynamic configuration file, specifying a TLS certificate CommonName (CN) regular expression. This expression is passed to the libc's `regcomp()` function, which on many implementations of libc

#	Variable in script	Record type	Malicious record data (dig-escaped)	Induced behaviour	Outcome
1	$HOST	NAPTR	`\@6.6.6.6.`	change dig DNS resolver	verification of vulnerability
2	$HOST	NAPTR	`-f/some/file.`	pass `/some/file` as dig batch-file	disclose contents of `/some/file`
3	$SRVHOST	SRV	`asd\\n\\tinclude\\t/dev/zero\\n.`	read `/dev/zero` as config file	100% CPU utilisation
4	$SRVHOST	SRV	`as.d\\n\\tmatchcertificateattribute\\t CN:/\(.*++++++++++++++++++\(\\\\w+\)\) /im\\n\\ttype\\ttls\\n}\\n%%p.`	provide malicious regex to regcomp()	radsecproxy crash
5	$SRVHOST	SRV	`6.6.6.6\\n\\ttype\\tTCP\\n \\tsecret\\tsomething\\n}\\n%%p.`	provide own RADIUS server and disable TLS-authentication	unauthorised network access

Table 3: Radsecproxy exploits. The exploits were successfully verified in the lab and against large operators of Eduroam.

(e.g., glibc) has known, unfixed vulnerabilities[3] which can be used, e.g., to crash radsecproxy via stack consumption.

• **Attack #5 in Table 3.** When the attacker provides a functional server configuration it can also override parameters of the dynamically generated server entry, most importantly the `type` parameter. When changing the `type` parameter to TCP and providing a known `secret`, the attacker can make radsecproxy connect to his own radius server despite not having a trusted TLS certificate from the eduroam-PKI. Attack #5 can be used to allow or deny access or log any authentication attempt for users using the attacker's domain as a realm. This enables the attacker to use any eduroam network effectively unauthenticated. In our evaluations we exploited this attack to even successfully inject malicious authentication server of the attacker for third-party domains, which enables the attacker to log usernames and/or hashed credentials when the wireless clients fail to verify the TLS-certificate provided in the protected-EAP tunnel to the attacker's RADIUS server.

3.3.3 Evaluation of the Attacks

All listed exploits and outcomes were verified in the lab using the latest version of radsecproxy. We also validated real-world applicability of the attacks on different eduroam networks (of two research institutions and university) by exploiting the vulnerabilities listed in this section. We launched exploit #1 against large operators of eduroam infrastructure. This exploit causes no harm but demonstrates that the infrastructure is vulnerable and uses the `naptr-eduroam.sh` script.

3.4 LDAP Peer Discovery

To locate the appropriate LDAP server dynamically, an LDAP client supporting dynamic peer discovery extracts the domain components, re-creates the domain name (e.g., `example.com`) and queries the DNS SRV-record for `_ldap._tcp.example.com`. This query is triggered either at application startup or when a user tries to connect to the LDAP-using service. In addition to SRV lookups, LDAP also supports the URL-based description of search operations [27]. For example, a URL for a search operation for john's user account entry may look like `ldap://ldap.example.com:389/uid=john,gid=users,dc=example,dc=com`. This instructs the LDAP client to connect to the LDAP server at `ldap.example.com`, port 389 and look for an entry with Distinguished Names (DN) `uid=john,gid=users,dc=example,dc=com`. The attacker triggers queries by attempting to connect to the LDAP-using service.

[3] E.g., [26] can still be exploited on current Ubuntu and used in attack #4.

3.4.1 LDAP Injection Attacks

When the SRV lookup is used in combination with LDAP URLs, it opens an attack vector which is caused by the SRV lookup handling of LDAP client implementations: the LDAP URL is checked for a hostname, and if it is not present, the hostname is looked up using SRV requests and pasted into the existing LDAP URL. An attacker controlling the SRV record can inject arbitrary characters into the URL, changing the URL path component and thereby the requested resource's DN or filter expression.

Algorithm 1 shows the LDAP peer discovery process as implemented by OpenJDK. The function *ConnectURL* is called with an LDAP URL like `ldap:///uid=john,gid=users,dc=example,dc=com` and parsed into a URL. The URL is then tested whether it includes a hostname, and if not the domain component (`dc=`) parts of the LDAP distinguished name (DN) are used to construct the the query domain for dynamic peer discovery. In our case this is the domain `example.com`, so the process continues by requesting that domains LDAP `SRV` record at `_ldap._tcp.example.com`. The hostname and port included in this SRV record are now used to construct a new LDAP URL by concatenating the hostname and port with the part of the path of the old LDAP URL. Finally, the new URL is used to call *ConnectURL* again, this time taking the other path and connection to the LDAP server at the specified hostname.

We show how the user input concatenated to an LDAP query can change the meaning of the query by injecting control characters like braces, similar to SQL injections. Our LDAP injection uses the contents of the SRV record for dynamic peer discovery (instead of direct user input) and leads to information disclosure, or authentication as a different user.

• **Attack #1: Privileges escalation.** When executing Algorithm 1 with the following URL `ldap:///uid=john,gid=users,dc=example,dc=com` and the SRV record set to

```
_ldap._tcp.example.com IN SRV ldap.example.com/
uid=admin,gid=users,dc=example,dc=com????.
```

the resulting URL becomes `ldap://ldap.example.com/uid=admin,gid=users,dc=example,dc=com????./uid=john,gid=users,dc=example,dc=com`, which means the client will

```
Function ConnectURL(ldapurl: URL) is
    if ldapurl.host == None then
        domain = extractDC(ldapurl.path)
        hostname,port = lookupSRV(domain)
        ldapurl = new URL("ldap://" + hostname + ":" + port + "/" +
          ldapurl.path)
        ConnectURL(ldapurl)
    else
        ip = lookupA(ldapurl.host)
        // Proceed with connection ...
    end
end
```

Algorithm 1: LDAP SRV lookup.

search for user admin instead of john, enabling john to execute actions with admin privileges. This attack enables to circumvent security mechanisms like LDAP over TLS (`ldaps://`) because it changes the information in the URL before it is transmitted over the TLS secured channel.

• **Attack #2: Denial-of-Service via malformed records.** The LDAP SRV lookup function calls itself recursively after looking-up an SRV record, see Algorithm 1. We manipulate an SRV record so that the resulting URL does not contain a hostname-component, which then causes an infinite recursion and crashes the 'LDAP-using' application with a stack overflow.

3.4.2 Evaluation of the Attacks

We tested attack #1 against two LDAP library implementations (ldapsearch and opejdk's javax.naming). We find that both applications use a potentially vulnerable LDAP peer discovery algorithm, which just concatenates the SRV record with the rest of the URL and do not check the contents of the SRV record for sanity. However, in both implementations, the DN (e.g., `uid=john,gid=users`) from the LDAP URL is actually ignored and must be given in an additional function call or parameter in order to allow execution of multiple search queries after the connection to the server has been established. We verified attack #2 experimentally using the record we constructed, and evaluated it against openjdk's 11.0.6 javax.naming API:

```
_ldap._tcp.attacker.com. IN SRV /dc=attacker,dc=com.
```

Triggering a query. A query for the LDAP `SRV` record is triggered when a new connection to the LDAP server is created, i.e., when a user triggers an action which requires an LDAP-lookup such as logging into a web application which uses LDAP for user management. However, to execute the attack, the attacker must either be able to (1) control the full LDAP DN or (2) modify the SRV record on the network via a MitM position. In our evaluation we tested the implementation of LDAP middleware/libraries, which do not restrict how the LDAP DN is set. However, typical applications will restrict control over the LDAP DN to the components relevant for the user[4], such that control over the necessary `dc=` components is not available to the attacker.

[4] `https://docs.spring.io/spring-ldap/docs/current/reference/`

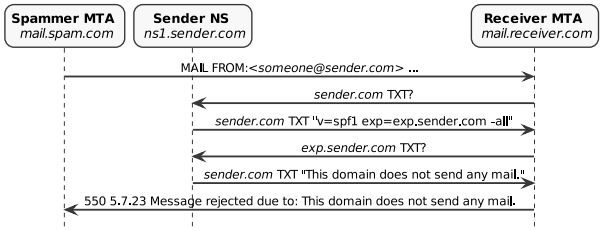


Figure 5: `SPF` resolution example.

3.5 Domain-Based Anti-Spam Validation

Sender Policy Framework (SPF) [28] is a domain-based mechanism to prevent forgery of SMTP envelope headers. To trigger an SPF DNS query, the attacker needs to send an Email to an SPF-supporting Email server.

We provide a detailed explanation of the Email `SPF` validation process shown in Figure 5 in the case where an incoming Email is rejected: first, an non-authorised Email transfer agent at `mail.spam.com` (Spammer MTA) connects to the mail server at the receiver domain (`mail.receiver.com`) and tries to send a Mail coming from `someone@sender.com` to a mailbox at `receiver.com` using SMTP. To check if the Spammer MTA is authorised to send mail from `sender.com`, the Receiver MTA will query the DNS for the SPF records for `sender.com`. In this case the record indicates that no one is authorised to send mail from that domain (option -all) and that a detailed explanation why the mail is rejected is stored at `exp.sender.com`. After the receiver MTA has received this record it will decide to reject the Email and fetch the explanation from `exp.sender.com` via DNS to include it together with the rejection message. Finally, is parses the rejection message, replaces any included macros and sends it back to the spammer MTA notifying it that the mail was rejected and why.

3.5.1 Attacks Against Checks of SPF Records

• **Attack #1: Injection against policyd-spf.** The default implementation, e.g., Ubuntu [29] used by postfix, for checking SPF records is based on a separately running daemon called policyd-spf. This daemon is listening at a unix socket for responses to determine whether an Email should be rejected or not according to the SPF records [30]. The interface to this daemon is line based, the client (postfix) provides properties of the received Email line-by-line and submits the request with an empty line, as shown below. In this example policyd-spf session, the client's request lines are marked with '>', deamon response lines with '<'.

```
> request=smtpd_access_policy
> protocol_state=RCPT
> client_address=192.168.234.20
> sender=someone@hardfail.example.com
> recipient=vagrant@postfix
...
> policy_context=
< action=550 5.7.23 Message rejected due to: SPF fail -
not authorized. Please see http://www.openspf.net/Why?s=helo;
id=hardfail.example.com;ip=192.168.234.20;r=<UNKNOWN>
```

```
attacker.com TXT "v=spf1 exp=exp.attacker.com"
exp.attacker.com TXT "AAAAAA..." ; (510 times)
```

Figure 6: libsfp2 exploit with malicious SPF record payload.

The server (policyd-spf) will answer with a single line providing information on how to proceed. We construct a malformed SPF record using the SPF `exp=` parameter. This parameter allows to include an explanation why an Email was rejected. We inject control characters into this client-server interface by including them in the rejection message, specified as a separate DNS record. The attacker can further include newline characters (`"\n"`) to create additional output lines in the line-based connection to policyd-spf. This is interpreted as the responses to requests asked in the future thereby changing the SPF result for the next Email.

• **Attack #2: Stack buffer overflow in libspf2.** Libspf2 is a library for checking SPF records for incoming Email messages for Mail Transfer Agents. Libspf2 is used by some versions of the command line utility `spfquery` and also directly from SMTP server source code. Because of the complexity of the library we used fuzz-testing with `afl-fuzz` against a custom-built application calling libspf2 functions. This allowed us to provide the SPF record as a file rather than via the network to test libspf2 against potential vulnerabilities exploitable via malformed SPF records. The evaluations showed that libspf2 is vulnerable to the malformed records attack, see malicious SPF record payload in Figure 6. We performed attacks against the `spfquery` command line utility using the vulnerable record set which resulted in 'stack-smashing detected' error and crashes, further allowing remote-code-execution. This attack exploits a stack-buffer overflow while parsing the SPF explanation macro.

3.5.2 Evaluation of the Attacks

We evaluated attack #1 against postfix using policyd-spf-perl, and were able to inject additional lines of output to the unix socket, showing that policyd-spf-perl does not verify the contents of the SPF explanation record. In contrast to other attacks in this work, attack #2 cannot be prevented by validating the DNS records since the malicious SPF record presents a theoretically valid SPF explanation message.

3.6 Administrative Tools

In the attacks that we presented until now, the applications implemented the DNS lookup themselves, not by using an API like `gethostbyname()`, where the behaviour is standardised and data validation is performed by the systems' stub resolvers. In this section we present vulnearbilities in applications which do not implement DNS lookups themselves but use the system stub resolver to do so. First, we analyse different stub resolver implementations in Section 3.6.1 and then show vulnerabilities in applications using these stub-resolvers in Section 3.6.2.

3.6.1 DNS Record Processing in stub resolvers.

Libc is the C standard library for C programming language, specified in ANSI C and is a subset of C library POSIX specification. There are different implementations of the standard C system library. We experimentally tested, as well as analysed the source code of, all the major implementations of the C system library and except two found them to be vulnerable, see Table 5. We explain our analysis of the DNS record processing on two implementations: glibc[5] and on uClibc[6]. We selected those implementations as examples because they represent two distinct methodologies that we observed in processing DNS records. The other tested implementations are similar to uClibc. We demonstrate the processing on the `gethostbyname()` library function as an example; the same applies to other calls.

After a DNS response has been received by glibc (resp uClibc) library, it is first checked against the length field, DNS transaction identifier and the return code (e.g., OK, NXDOMAIN). The libraries then go through the resource record sets in the answer section and process each record. For each domain name in a DNS record the following steps are done, as shown with our example domain name `036123...6d00` in Figure 7: (1) domain name compression is removed; (2) domain name is decoded from DNS line format into a (zero-terminated) string; (3) domain name is validated. We explain these steps next.

DNS decompression and decoding into a string. These two steps are typically done simultaneously in one function, e.g., `dn_expand` for glibc, or `__decode_dotted` for uClibc. When decoding a domain name into a string, the resolver must ensure that the characters which cannot be represented in an ASCII string, must be escaped appropriately [RFC4343] [18]. This also applies to zero-bytes (which would otherwise be interpreted as string terminators) and period characters (which would otherwise be interpreted as label-separators). Escaping values outside the range of 0x21 (`"!"`) to 0x7E (`"~"`) is required by [18].

In our example in Figure 7 this means that to avoid confusion with label separators the second byte (0x3e) of the first label must be expressed as `"\."` instead of `"."`. The final decoded domain name is then `"a\.b.<>.com."` if decoded correctly applying escaping to non-printable characters, or `"a.b.<>.com."` if decoded incorrectly when escaping is not applied.

Our analysis shows that glibc applies escaping to the decoded domain name, while uClibc does not. This means that any record which contains zero-bytes or dots inside labels will be misinterpreted during decoding when processed by uClibc and other non-printable characters will be included in the returned string unescaped. Such incorrect decoding logic can allow cache-injection attacks when the misinterpreted record for the domain `a.b.<>.com.` is cached and re-used in another context, as we show in Section 3.2.

Domain name validation. After the record has been decoded, it should be validated to check that it represents a

[5]The GNU's project implementation of the C standard Library
[6]The Linux standard library for mobile and embedded devices.

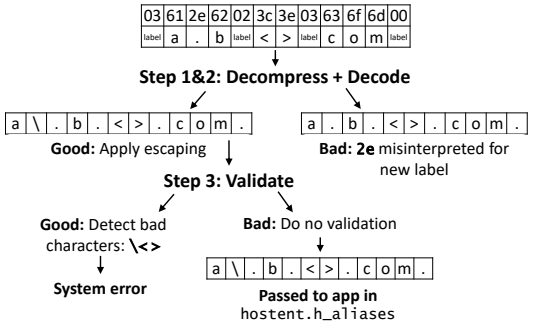


Figure 7: DNS record processing.

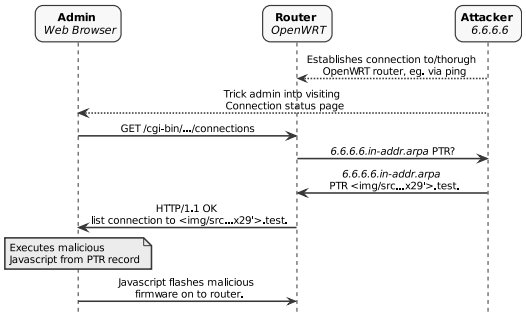


Figure 8: Attack flow against OpenWRT.

valid hostname. The POSIX standard defines `getnameinfo()` to return 'hostnames', not 'domain names'. Notice that domain names are defined in [RFC1034, RFC2181] [23, 31] as a list of binary labels that can contain any value. The only limitation over domain names is on the length of the name [RFC2181] [23]: 63 octets per component, 255 octets for a domain name. Values like, brackets (`[, ]`), colons, NULs (`\000`), newlines, backslashes, and so on are all legal. In contrast, hostnames are only allowed to contain alphanumeric characters (`"A-Z"`, `"a-z"`, `"0-9"`), hyphens (`"-"`) and dots (`"."`) to separate labels, as defined in [RFC952] [16], this specification is referenced by newer standards [RFC1123, RFC2181, RFC3492] [17, 23, 32]. As a result the system library should validate that all the returned values represent valid hostnames, not domains. Our evaluation shows that most stub resolvers use a naive domain name decoding logic which misinterprets `"\."` for `"."` and `"\000"` as a string delimiter; we show how to exploit this for cache poisoning attacks in Section 3.2.

In our example in Figure 7 step 2, we assume that the library has decoded the domain name correctly, hence the value `a\.b.<>.com.` is passed on to the validation step. Based on the logic that the POSIX standard defines the return value of `gethostbyname()` as a hostname the library should ensure that the hostname does not contain invalid characters, and if it finds any, it should signal an error to the application. If this is not done, the string `a\.b.<>.com.` is passed to the application unchanged and exposes to a range of vulnerabilities as shown in Sections 3.6. Again we review the functions in glibc and uClibc and find that only glibc implements this validation correctly, uClibc does not validate the decoded domain name at all.

Notice that the steps 1-3 are always needed to transform a domain name from its line format to a (zero-terminated) ASCII string. An implementation might choose to switch the order of steps 2 (decoding) and 3 (validation), or combine these steps into one, but this does not change the fact that both steps are needed to correctly implement parsing of a line-format domain names into a hostname string. Therefore, our analysis of glibc and uClibc can be extended to other resolver implementations. We validated this assumption experimentally and via code review in popular system resolver implementations, and summarise the results in Table 5. Our results indicate that most stub resolvers do not check that domain names constitute valid hostnames. The other libc implementations, such as *dietlibc* and *windows*, all result in the same incorrectly processed output. We show how to exploit lack of validation to attack applications which do not expect special characters inside hostnames, launching different injection attacks, such as XSS or ANSI terminal escape code injection.

3.6.2 Attacks Against Administrative Utilities

We demonstrate attacks against two vulnerable example applications from different contexts: Windows 10's *ping* as an example of an simple command-line utility and *OpenWRT LuCi* as an example of a integrated web-based administration interface.

• **Attack #1: Windows ping.** Ping shows the CNAME alias of the ping-ed host without checking the alias for disallowed characters. We use this to inject arbitrary bytes into the output of ping, which is then displayed by the terminal. By including ANSI terminal escape codes in the CNAME record, an attacker controlling the DNS response can manipulate the terminal output by moving the cursor, replacing already printed data or changing the Window title[7].

We tested a similar attack against linux ping, traceroute and openssh. Different than windows ping, these applications query the reverse PTR record of the remote address. We found that none of the applications perform input validation of the returned domain name and therefore allow the same kind of attack.

• **Attack #2: Cross-site scripting (XSS) attack in OpenWRT LuCi.** OpenWRT is an aftermarket operating system for residential-gateway routers based on linux. It uses uClibc as its standard C library. This means that any application included in OpenWRT which does not check CNAME or reverse-DNS responses itself is vulnerable to our attacks. We explore the web-based LuCi [35] administration interface which can be used to configure all of a router's settings via a web-browser.

Apart from letting the user change the router's settings, LuCi also contains a status page listing all currently ac-

[7] Terminal-escape injection vulnerabilities were found in applications like web-servers [33, 34], however they were exploitable via direct input over HTTP.

Use-Case	Apps	validates	attack prevented by
Address Lookup	Chrome, Firefox, Opera, Edge, unscd, java	no	Cache does not evaluate CNAME aliases for indexing
Service discovery	ldapsearch	no	Generated URL not used to specify LDAP query
Authentication	policyd-spf + postfix	no	postfix resets connection if second answer is injected

Table 4: Non-validating applications without meaningful exploits.

tive connections through the router at `/cgi-bin/luci/admin/status/realtime/connections`. When a user opens this page, LuCi performs reverse-DNS lookups for all IP addresses currently connected to or through the router. The attacker abuses this to trigger reverse queries for its own IP addresses in the reverse-DNS tree and delivers malicious DNS records from there. The attacker also maintains a connection to or through the OpenWRT router when the victim opens the 'Connections page', by continuously sending ICMP echo-request ('ping') messages to the router. Using the `6.6.6.6.in-addr.arpa` record shown in Figure 9 we were able to successfully launch this attack by placing javascript code inside the `PTR` record of the attacker's IP address which is not validated by the LuCi web interface. This record is queried when the user views the 'Connections page' and is placed in the page's HTML code which executes the malicious code. A successful attack allows different exploits, as an example we created a record which, when injected, loads a third party script from an external HTTP server. This script then issues various requests to the OpenWRT configuration interface in name of the user which finally results in replacing the OpenWRT firmware running on the device with a malicious attacker-provided firmware. We show an example of a full attack flow against OpenWRT using this vulnerability in Figure 8.

3.7 Impact of the Attacks

Our analysis of the attacks shows that none of the tested applications perform DNS input validation. Some attacks do not result in meaningful exploits, see examples in Section 3.7.1, the causes are not defences against potentially malicious inputs, but rather not security related implementation decisions. We also showed attacks with limited impact, such as the ping exploit, to demonstrate that the problem (lack of DNS-input validation) is systematic and prevalent, affects different systems, services and tools and is not limited to isolated cases or a specific application. These examples show that developers do not check DNS results, which contain untrusted data. Furthermore, using such tools in certain scenarios could expose to the exploitable attack. For instance, output of utilities like Ping, can also be saved in a format which allows command injection, like an HTML report, or SQL database, this allows for a much more severe attack.

3.7.1 Non-validating applications without exploits

In this section we provide examples (listed in Table 4) where DNS input validation vulnerability does not lead to meaningful attacks. We caution that the factors preventing meaningful attacks are not security related, but are due to different implementation considerations: (1) in browsers and app-caches, the cached records are only indexed by their query domain, not by additional domain aliases inside the response. (2) in ldapsearch, a potentially vulnerable peer discovery algorithm is used to generate an LDAP URL, but depending on the configuration this URL may not be used to specify the LDAP query, which would prevent the attacker from changing the query. (3) in policyd-spf if implementations are configured to check for additional data on unix socket, the result will be discarded to prevent desynchronisation with the policyd-spf daemon.

4 Internet Evaluation on Open Resolvers

In Section 3 we examined how the popular DNS resolvers and stub resolver implementations built into operating systems and programming languages handle control characters in domain names and if they modify any of the maliciously crafted payloads needed to conduct our application-specific exploits. In this section we extend our evaluation to open DNS resolvers in the Internet, and confirm the results of our in-lab study in the Internet. Specifically, we evaluate behaviour when processing a set of crafted DNS records designed to trigger classic input validation vulnerabilities like SQL injection, as well as DNS-specific special cases like handling of period characters inside DNS labels. We do not evaluate attacks against the applications using the vulnerable resolvers, since this would result in an attack against an application or service in the network which we do not own. We do not risk evaluating even the 'more benign' attacks, which when run against the servers that we setup do not cause critical damage. This is since our attacks can trigger unexpected outcome when evaluated against the servers in the Internet, e.g., due to differences in configurations. We observe this phenomena during the DNS cache poisoning evaluation of domains that we control against open resolvers in the Internet – some of the DNS software which was found secure when running the cache poisoning attack against our servers, resulted in cache poisoning vulnerabilities in the Internet.

4.1 Methodology

We run the same set of queries as for the in-lab evaluation, but employ additional logging at the nameserver-level to gain knowledge about the nature of the resolvers we test. As this study targets the same class of resolver as our in-lab evaluation, the expected behaviour is the same as described in Section 2.

Dataset. To conduct the study, we use a dataset from Censys [36] with 3M open resolvers. We include baseline tests for each record type (A, CNAME, SRV, TXT) to ensure that the

resolver supports the record type we use in our payloads and only consider resolvers who respond to our queries and return the correct result for all of these baseline tests. This results in 1,328,146 open resolvers from 228 different countries.

Tests are conducted by using custom test applications which send DNS queries to the resolvers over the network or by calling the respective stub resolvers using the appropriate API, by calling the POSIX `gethostbyname()` and `getnameinfo()` functions. To test handling of commonly used control characters like slash ("/"), at ("@"), zero-byte ("\000"), etc. we use the CNAME and PTR records with the injection payloads. We furthermore test all the resolvers against the applications'-specific records listed in Section 3 whenever applicable. All tested payloads are listed in Figure 9 in the order they appear in Table 5 and 6.

Forward-lookups. We evaluate hostname-to-address records for all 3 groups of resolvers (resolver, stub and open resolver) by triggering a query to the domain name of the payload (e.g., `cnameslash.example.com`) and observing the response from the resolver.

Injection payloads. For the DNS-specific injection payloads, the test takes place in two stages: First we trigger a query to the domain name of the injection payload (i.e., `injectzero.example.com`) twice and observe if the result was misinterpreted, ie. if the result CNAME is `www.target.com\000.example.com` (marked ✗) or just `www.target.com` (marked ✗[5]). We then trigger a query to the potentially misinterpreted domain name (i.e., `www.target.com` instead of `www.target.com\000.example.com`) and observe if the IP address was successfully injected into the resolvers' cache (marked ✓or given in % for open resolvers). We test each payload (inject\000 and inject\.) in both scenarios: via a CNAME-record which points to the malicious record (i.e., `injectzero.attacker.com`) and by triggering a query to the malicious domain (i.e., `victim.com\000.attacker.com`) directly.

Reverse-lookups. Reverse-DNS lookups (PTR) were tested against system stub resolvers only by setting the upstream DNS server to a custom controlled nameserver directly providing the records under `in-addr.arpa`-tree and triggering a reverse PTR lookup for the respective IP address, e.g., `1.1.1.1.in-addr.arpa` and observing the response.

Additional considerations. To enhance the robustness of our tests, we randomise all the queried domain names by prepending a random subdomain to ensure that the query is not cached before the test is conducted and processed by all the components of the DNS lookup chain. This also allows us to link the open resolver we sent the query to, to the final recursive resolver which connects back to our nameserver by matching these random prefixes. To prevent other users of the resolver to be negatively affected by our tests, we run these tests only against domains we own, which allows us to validate the full injection attack without performing an attack against any other domain.

```
cnamebase.example.com  CNAME works.cnameslash.example.com
cnameslash.example.com CNAME t/t.cnameslash.example.com
cnameat.example.com    CNAME t\@t.cnameat.example.com
cnamexss.example.com   CNAME <img/src=''/onerror='alert
                             &#x28&#x22xss&#x22&#x29'>.
                             cnamexss.example.com
cnamesql.example.com   CNAME 'OR''='''--.cnamesql.example.com
cnameansi.example.com  CNAME \027[31\;1\;4mHello\027[0m.
                             cnameansi.example.com

injectdot.example.com. CNAME www\.target.com.
www\.target.com.           A 6.6.6.6
injectzero.example.com CNAME www.target.com\000.example.com
www.target.com\000.example.com A 6.6.6.6

_ldap._tcp.example.com.  IN SRV /dc=example,dc=com.
_radsec._tcp.example.com IN SRV 6.6.6.6\\n\\ttype\\tTCP\\n\\t
                                secret\\tsomething\\n}\\n%
exp.example.com          IN TXT "AAAAA..." (510 times)

1.1.1.1.in-addr.arpa   PTR   works.test
2.2.2.2.in-addr.arpa   PTR   te/st.test
3.3.3.3.in-addr.arpa   PTR   te\@st.test
4.4.4.4.in-addr.arpa   PTR   t\000t.test
5.5.5.5.in-addr.arpa   PTR   t\.t.test
6.6.6.6.in-addr.arpa   PTR   <img/src=''/onerror='alert
                             &#x28&#x22xss&#x22&#x29'>.test
7.7.7.7.in-addr.arpa   PTR   'OR''='''--.test
8.8.8.8.in-addr.arpa   PTR   \027[31\;1\;4mHello\027[0m.test
```

Figure 9: Injection payloads based on CNAME and PTR records.

4.2 Evaluation Results

We present the results for forward-lookups in Table 5 and the results of reverse-lookups in Table 6. For each test, ticks (✓) mark that the resolver is vulnerable to this kind of payload and crosses (✗) mark that the resolver is not vulnerable. Note however that depending on the test, 'vulnerable' means that the resolver performs as expected and conforms to the DNS standard (in case of non-stub resolvers and special character tests like cnameslash) or that it misinterprets a domain name which allows for a cache-poisoning attack (in case of injection payloads, such as inject\000). For injection payloads, we only call a resolver vulnerable (✓) when the malicious IP address was cached, otherwise we use (✗[5]) to show that the misinterpretation occurs, but can only exploited in conjunction with a caching downstream resolver like dnsmasq (See Section 3.2.3). We list the percentage and absolute number of open resolvers vulnerable to each payload in the bottom of Table 5.

Transparent handling of DNS records. The results from Internet evaluation of open resolvers correspond to our in-lab evaluation: Around 96% of all tested open DNS resolvers are transparent for application-specific payloads in DNS records.

Comparing the CNAME-based cnameslash and SRV-based LDAP and Eduroam payloads, we can observe that there is some difference of resolver behaviour in handling these records, even thought they are based on the same property (including non-standard characters in a domain name field). This can be seen as a confirmation that many resolvers indeed ignore the contents of record type field that is not directly needed for the DNS resolution (like SRV), but do not do so for records which are important to finish the lookup (like A or CNAME).

Cache injection vulnerabilities. For the injection specific payloads inject\. and inject\000, the evaluation results also mostly match the expectation from the in-lab evaluation. Like all resolver implementations except Verisign Public DNS, most Open DNS resolvers which do handle the injection payloads transparently and are not vulnerable to our injection payloads.

Nevertheless, we found 1.3% to 4.6% of the open DNS resolvers to be vulnerable to a cache poisoning attack without any further requirements, which was verified by querying for the maliciously injected record. Overall, 8.0% (or 105,854) of the open resolvers are vulnerable to cache poisoning via *any* of the injection payloads. This result is alarming considering that the inject\000 attack does not require any attacker resources other than control over an arbitrary nameserver in the internet.

Misinterpretation analysis. In our Internet measurement we observed the following phenomenon: the resolvers respond with the non-misinterpreted value in return to the first query for a record with injection payload in our domain. However, subsequent queries for the same resource record (with the same domain name) which are responded from the cache (without issuing a query to our nameserver) can result in one of the two outcomes: (1) a misinterpreted value or (2) non-misinterpreted value. Nevertheless a subsequent validation of the cached record confirms that in both cases the injection was successful and the target cache stored a misinterpreted record.

Vulnerable DNS software in Internet. We use `version.bind` special query to infer the implementation and version of the open resolvers. While most servers do not respond to these queries (in our study 65%, see column 'our study' in Table 5), we find that, depending on the implementation and attack type, even those exact versions we have found not vulnerable during our lab evaluation were found vulnerable during our evaluation of open resolvers in the Internet, e.g., 19.2% of vulnerable resolvers were Bind resolvers. In these cases we find that the misinterpretation causing the vulnerability does not occur in the 'visible' software implementations, but rather in one of the upstream (forwarding) resolvers in the resolution chain, which we cannot identify.

In some cases upstream forwarders is a public DNS service, like Google, OpenDNS and Cloudflare. We infer this by mapping the IP addresses of the DNS queries received by our nameservers to the Autonomous System (AS) numbers of the resolvers sending the queries.

[RFC1034] non-compliant resolvers. 12% of the open resolvers in the Internet exhibit the same flaws as we found in stub resolver implementations. These are misinterpretations when decoding line-format domain names into zero-terminated string. Not only this allows for cache injection attacks, but it also indicates that those resolvers do not follow the recommendations in [RFC1034] [31], requiring that the DNS software should store domain names in their line format and not as decoded strings.

5 Root Causes, Insights and Mitigations

Missing Specifications

The study that we carried out in this work showed us that one of the root factors allowing our attacks is a lack of threat model in the standard RFCs as well as a lack of specifications on DNS and on its interactions with applications:

Threat modelling. There is a lack of threat modelling in the DNS infrastructure and in the interaction of DNS with the applications. The RFCs should provide a threat model discussing potential pitfalls. For instance, most applications typically expect hostnames, but in the RFCs, this is not considered.

DNS record parsing. There is a lack of detailed specification on how to parse DNS records. This critical functionality should be specified in the RFCs.

Validation of DNS records. There is a lack of standardised implementation for validation of DNS records. This is important esp. for non-address record types as these are not supported by the OS resolver (e.g., libc). Similarly to, say libraries for generating DNSSEC keys, there should be an implementation for validation of received DNS records.

Domain names vs hostnames. There is a discrepancy between definitions of domain names and hostnames, which leads to confusion in DNS software in how to parse the DNS records. To avoid pitfalls the same rules should apply to both.

Mitigations

Applications. Since DNS resolvers serve data from untrusted Internet sources, the applications should always treat data from DNS the same way they treat user input, hence validation of formatting and escaping should always be performed (ie. validate html special characters, etc.), regardless if the API used to receive the data indicates an already checked result (POSIX) or not.

System stub resolvers. Stub resolvers should be modified to check if domain names returned by POSIX calls like `gethostbyname()` are valid hostnames, [17]. If not, the domain name should not be given to the application, like it is implemented in glibc already. Merely escaping special characters as done by netbsd can still be vulnerable. For instance, we demonstrate this with the `6.6.6.6.in-addr.arpa` Cross-Site Scripting payload in Figure 9 which does not contain any character typically escaped in domain names and, for example, can be used to execute an attack against OpenWRT. Furthermore non-libc DNS libraries should follow the same rules (i.e., only allow hostnames as per [17]) even for non-standard lookup types like SRV to prevent confusion among developers who only used libc resolvers before. Guidance on how to implement such checks should ideally be given by standardisation bodies.

DNS resolvers. Filtering DNS responses on the DNS resolver or forwarder level is possible but is against the DNS

	DNS Payload shown in	Fig. 9						Section 3.2.1		Section 3.2.1		Sec. 3.4.2	Tab. 3	Fig. 6
	Test	Base	/	@	XSS	SQL	ANSI	inject$_{\backslash.}$ CNAME	Direct	inject$_{\backslash 000}$ CNAME	Direct	LDAP	Eduroam	libspf2
Recursive resolvers	BIND (9.14.0)	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓
	MaraDNS Deadwood (3.2.14)	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✗[2]
	Unbound (1.9.1)	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓
	PowerDNS Recursor (4.3.0)	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓
	Windows Server (2012 R2)	✓	✓	✓	✓	✓[1]	✓[1]	✗	✗	✗	✗	✓	✓	✓
	Windows Server (2016)	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓
	Windows Server (2019)	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓
Forwarders	pdnsd (1.2.9a)	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓
	dnsmasq (2.79)	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓
	NxFilter (4.3.3.9)	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓
	systemd resolved (237)	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓
Public resolvers	OpenDNS	✓	✓	✓	✓	✓[1]	✓[1]	✗	✗	✗	✗	✓	✓	✓
	Cloudflare Public DNS	✓	✓	✓	✓	✓[1]	✓[1]	✗	✗	✗	✗	✓	✓	✓
	Comodo Secure DNS	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓
	Google Public DNS	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓
	Hurricane Electric	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓
	Neustar UltraRecursive	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓
	Norton ConnectSafe	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓
	Oracle Dyn	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓
	SafeDNS	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓
	VeriSign Public DNS	✓	✓	✓	✓	✓	✓	(✗)[5]	✗	✗	✗	✓	✓	✓
	Yandex DNS	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓
Stub resolvers	glibc	✓	✗	✗	✗	✗	✗	✗	-	✗	-	n/a (no SRV/TXT support or does not apply)		
	musl	✓	✗	✗	✗	✗	✗	(✗)[5]	-	✗	-			
	dietlibc	✓	✓	✓	✓	✓	✓	(✗)[5]	-	(✗)[5]	-			
	uclibc	✓	✓	✓	✓	✓	✓	(✗)[5]	-	(✗)[5]	-			
	Windows	✓	✓	✓	✓	✓	✓	(✗)[5]	-	(✗)[5]	-			
	NetBSD	✓	✓	(✓)[3]	✓	✓	(✓)[3]	✗[3]	-	✗[3]	-			
	Mac OS X	✓	✓	✓	✓	✓	(✓)[3]	✗[3]	-	✗[3]	-			
	go*	✓	✓	✓	✓	✗	✓	(✗)[5]	-	✗	-			
	nodejs	✓	✓	✓	✓	✓	✓	✗[3]	-	(✗)[5]	-			
	openjdk8*	InetAddress.getCanonicalHostName() gives PTR instead of CNAME										✓		
Open resolvers in Internet		100% 1,328,146	96.6% 1,283,447	96.3% 1,279,573	96.4% 1,279,835	95.9% 1,273,710	96.3% 1,279,100	1.3% 45,596	2.7% 51,049	3.6% 82,589	4.6% 88,864	99.6% 1,322,529	99.6% 1,322,203	56.1% 745,599

✓: Vulnerable. [1]: record converted to lower case. [2]: NXDOMAIN/no response. [3]: output was escaped. [5]: Record is misinterpreted, injection is not cached.
∗: Uses system stub resolver by default but offers a builtin one.

Table 5: Forward-lookup test results for all groups of resolvers.

Test	**Base**	/	@	\.	\000	XSS	SQL	ANSI
Payload (Fig.9)	1.1.1.1	2.2.2.2	3.3.3.3	5.5.5.5	4.4.4.4	6.6.6.6	7.7.7.7	8.8.8.8
glibc	✓	✗	✗	✗	✗	✗	✗	✗
musl	✓	✓	✓	✓	✓	✓	✓	✓
dietlibc	✓	✓	✓	✓	✓	✓	✓	✓
uclibc	✓	✓	✓	✓	✓	✓	✓	✓
windows	✓	✓	✓	✓	✓	✓	✓	✓
netbsd	✓	✓	(✓)[2]	(✓)[2]	(✓)[2]	✓	✓	(✓)[2]
mac os x	✓	✓	✓	(✓)[2]	✓	✓	✓	✓
go*	✓	✓	✓	✓	(✓)[3]	✓	✓	✓
openjdk8*	✓	✗	✓	(✓)[2]	(✓)[3]	✓[4]	✓	✓
node	✓	✓	✓	(✓)[2]	✓	✓	✓	✓

✓: Vulnerable. [2]: output was escaped. [3]: Zero-byte did not stop output.
[4]: Alternative XSS payload with `" "` instead of `"/"`.
∗: Uses system stub resolver by default but offers a builtin-one.

Table 6: Reverse-lookup results for different stub resolvers.

standard [3, 4, 17]. Changing this requires a discussion in the corresponding working groups within the IETF, which we are initiated within our disclosure efforts. Nevertheless, performing checks on DNS records is challenging: some applications, like SRV service discovery [39], require domain names with characters that are not allowed in hostnames (e.g., underscore). Defining a list of allowed characters so that legitimate applications would still work but injection attacks would be blocked should be further investigated and is not straightforward. In particular, it is difficult to foresee what characters and formats will be needed by future applications, hence a 'too-restrictive' list of allowed characters would make DNS less transparent, possibly introducing obstacles in deployment of new applications, or when adding new versions or new features to existing applications. On the other hand, leaving this completely transparent may lead to confusion about what values a field can actually have, and can even introduce vulnerabilities – our work can be generalised to other Internet protocols. We show that the decision to enable easy future deployment of new applications by not restricting the domain names to alphanumeric characters exposes to attacks.

Nevertheless, as an immediate protection against our attacks, operators which are unable to implement changes on the application- or stub-resolver-level might use filtering proxies which implement those validation steps on the network-level instead[8].

Mitigations against cache-poisoning. Since the cache-poisoning attacks cannot be reliably detected by downstream forwarders, these attacks must be mitigated by patching the resolvers causing the misinterpretation. If the resolver is only misinterpreting malicious records (but not caching them, like Verisign Public DNS), switching to a DNS forwarder which does not cache cross-CNAME records can prevent the at-

[8] We provide a proof-of-concept implementation of such a proxy at `https://xdi-attack.net`.

tack. This however does not fix the root cause of the issue. When the resolver operator cannot not fix the vulnerability, switching to another DNS resolver is the best option.

6 Related Work

DNS cache poisoning. Kaminsky provided the first demonstration of DNS cache poisoning attack [40]. Since then DNS resolvers have been patched to support best practices [RFC5452] [22]: randomising fields in requests, such as source port and DNS TXID, and validating them in responses, and also to apply checks such as bailiwick [23]. This makes DNS resilient to off-path cache poisoning attacks. Nevertheless, recent works developed cache poisoning attacks when DNS responses are served over UDP [41]. The attacks use different side channels to predict the randomisation parameters, as well as other methodologies like fragmentation to bypass guessing the parameters altogether, [8–10,42–46]. Our attacks are not limited by the transport protocol and apply to DNS over TCP as well as DNS over UDP. In contrast to all existing DNS cache poisoning attacks which evaluate the cache poisoning on one victim DNS resolver and then check if some selected population of DNS resolvers have the properties that could potentially make them vulnerable, our attack is the first to have been fully automated and evaluated on a large set of target networks, 3M, and the first to have been successfully launched against 105K resolvers. Prior attacks cannot be automated since they need to be tailored per each target victim resolver [8–10, 45, 46]. For instance, the servers set the fragmentation offset slightly differently hence making fragmentation difficult to match, the servers randomise the records in responses making the UDP checksum extremely difficult to match, UDP ports need to be measured per each target separately, overwriting cached records with new values depends on already cached records and caching policies, and so on. Our attack is not restricted by these hurdles.

None of the proposed non-cryptographic defences prevent our cache poisoning attacks. Even the cryptographic protection with DNSSEC [RFC4033-RFC4035] [11–13], which blocks all previous DNS cache poisoning attacks, does not prevent our attacks in common settings. Furthermore, DNSSEC deployments were showed to often use weak cryptographic algorithms or vulnerable keys, [47–49]. Cipher-suite negotiation schemes were proposed to allow easy adoption of stronger cryptographic ciphers [50].

Recent proposals for encryption of DNS traffic, such as DNS over HTTPS [51] and DNS over TLS [52], although vulnerable to traffic analysis [53, 54], may also enhance resilience to cache poisoning but do not prevent our injection attack.

User input injections in web applications. Injection vulnerabilities [55] are the primary medium for performing remote exploits, including SQL injection attacks [55], Cross Site Scripting (XSS) [56], buffer overflow [57], XPath injections [58], LDAP injections [59], HTTP header injection [60], Email header injection [61], SMTP injection [62]. All these differ from our injection attacks. User injection attacks via the web interfaces are typically blocked as user input is sanitised prior to being accepted by applications. Our attacks apply even when user input is properly validated and also where the users cannot provide any meaningful input at all, since we deliver malicious payloads by encoding them into DNS records.

DNS rebinding attacks. DNS rebinding attack [63, 64] uses a script on the victim network and an external attacker to create a confusion in web browsers bypassing Same Origin Policy (SOP), say by mapping the external attacker to an internal IP address. This allows the attacker to impersonate internal hosts in order to bypass filtering that is applied on external packets, e.g., for spam or for Denial of Service (DoS) attacks. Our attacks target the internal services directly, without impersonation of internal devices. DNS rebinding are prevented, e.g., with filtering private IP addresses and blocking the resolution of external hostnames into internal IP addresses, or via DNS pinning [63] in web browsers - none of which prevent our attacks.

7 Conclusions

Our work shows that central transparency-related principles of development of Internet systems should be reconsidered:

Flexibility. Be strict when sending and permissive when receiving is a good principle in the Internet. Systems and protocols that are too rigid are much more difficult to use and require significant changes to the existing infrastructure for adoption of new technologies or mechanisms. The huge success of DNS in providing platform to new applications is thanks to its transparent handling of DNS records. If DNS is made less transparent, e.g., by requiring that the records are checked for invalid characters, it would make the roll out of new applications in the Internet much more challenging. For instance, if DNS parsed each record, new applications using records containing not yet supported characters, e.g., not just alphanumeric characters, like in SRV record type, would require changes to the DNS servers all over the Internet to enable support for new characters. Unupgraded servers would risk failures or even crashes when processing the new records. On the other hand, making systems too tolerant can expose to vulnerabilities. We showed that leaving the specification completely open exposes DNS and the applications using DNS to attacks. Hence, a balance should be found between the ease of deployment and security.

Layering. Although it is a known networking principle that each layer provides services to the layer above it, and the upper layer does not have to worry about the data provided by lower protocols, we show that when it comes to security this principle may result in vulnerabilities. We recommend that the validation of DNS data is integrated into applications directly not relying on the lower layers to do this for them. For instance, it may not always be possible for DNS to predict

all the applications of the data that it provides and scenarios where it will be used. Hence even if DNS is changed to apply checks over the data in DNS records, the applications should nevertheless do the validation also themselves.

Acknowledgements

We are grateful to Yuval Yarom and to the anonymous referees for their thoughtful feedback on our work.

This work has been co-funded by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) SFB 1119.

References

[1] C. Farrell, M. Schulze, S. Pleitner, and D. Baldoni, "DNS Encoding of Geographical Location," RFC 1712 (Experimental), Internet Engineering Task Force, Nov. 1994. [Online]. Available: http://www.ietf.org/rfc/rfc1712.txt

[2] P. Hoffman and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA," RFC 6698 (Proposed Standard), Internet Engineering Task Force, Aug. 2012, updated by RFCs 7218, 7671. [Online]. Available: http://www.ietf.org/rfc/rfc6698.txt

[3] A. Gustafsson, "Handling of Unknown DNS Resource Record (RR) Types," RFC 3597 (Proposed Standard), Internet Engineering Task Force, Sep. 2003, updated by RFCs 4033, 4034, 4035, 5395, 6195, 6895. [Online]. Available: http://www.ietf.org/rfc/rfc3597.txt

[4] P. Mockapetris, "Domain names - implementation and specification," RFC 1035 (INTERNET STANDARD), Internet Engineering Task Force, Nov. 1987, updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2673, 2845, 3425, 3658, 4033, 4034, 4035, 4343, 5936, 5966, 6604, 7766. [Online]. Available: http://www.ietf.org/rfc/rfc1035.txt

[5] W. G. Halfond, J. Viegas, A. Orso *et al.*, "A classification of SQL-injection attacks and countermeasures," in *Proceedings of the IEEE international symposium on secure software engineering*, vol. 1. IEEE, 2006, pp. 13–15.

[6] J. Grossman, S. Fogie, R. Hansen, A. Rager, and P. D. Petkov, *XSS attacks: cross site scripting exploits and defense*. Syngress, 2007.

[7] T. Pietraszek and C. V. Berghe, "Defending against injection attacks through context-sensitive string evaluation," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2005, pp. 124–145.

[8] A. Herzberg and H. Shulman, "Fragmentation Considered Poisonous: or one-domain-to-rule-them-all.org," in *IEEE CNS 2013. The Conference on Communications and Network Security, Washington, D.C., U.S.* IEEE, 2013.

[9] X. Zheng, C. Lu, J. Peng, Q. Yang, D. Zhou, B. Liu, K. Man, S. Hao, H. Duan, and Z. Qian, "Poison over troubled forwarders: A cache poisoning attack targeting DNS forwarding devices," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 577–593.

[10] K. Man, Z. Qian, Z. Wang, X. Zheng, Y. Huang, and H. Duan, "DNS Cache Poisoning Attack Reloaded: Revolutions with Side Channels," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security CCS*. ACM, 2020.

[11] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "DNS Security Introduction and Requirements," RFC 4033 (Proposed Standard), Internet Engineering Task Force, Mar. 2005, updated by RFCs 6014, 6840. [Online]. Available: http://www.ietf.org/rfc/rfc4033.txt

[12] ——, "Resource Records for the DNS Security Extensions," RFC 4034 (Proposed Standard), Internet Engineering Task Force, Mar. 2005, updated by RFCs 4470, 6014, 6840, 6944. [Online]. Available: http://www.ietf.org/rfc/rfc4034.txt

[13] ——, "Protocol Modifications for the DNS Security Extensions," RFC 4035 (Proposed Standard), Internet Engineering Task Force, Mar. 2005, updated by RFCs 4470, 6014, 6840. [Online]. Available: http://www.ietf.org/rfc/rfc4035.txt

[14] "Standard for Information Technology–Portable Operating System Interface (POSIX(R)) Base Specifications, Issue 7," *IEEE Std 1003.1, 2016 Edition (incorporates IEEE Std 1003.1-2008, IEEE Std 1003.1-2008/Cor 1-2013, and IEEE Std 1003.1-2008/Cor 2-2016)*, pp. 1–3957, Sep. 2016.

[15] R. Gilligan, S. Thomson, J. Bound, J. McCann, and W. Stevens, "Basic Socket Interface Extensions for IPv6," RFC 3493 (Informational), Internet Engineering Task Force, Feb. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3493.txt

[16] K. Harrenstien, M. Stahl, and E. Feinler, "DoD Internet host table specification," RFC 952, Internet Engineering Task Force, Oct. 1985, updated by RFC 1123. [Online]. Available: http://www.ietf.org/rfc/rfc952.txt

[17] R. Braden, "Requirements for Internet Hosts - Application and Support," RFC 1123 (INTERNET STANDARD), Internet Engineering Task Force, Oct. 1989, updated by RFCs 1349, 2181, 5321, 5966, 7766. [Online]. Available: http://www.ietf.org/rfc/rfc1123.txt

[18] D. E. 3rd, "Domain Name System (DNS) Case Insensitivity Clarification," RFC 4343 (Proposed Standard), Internet Engineering Task Force, Jan. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4343.txt

[19] ——, "Domain Name System (DNS) IANA Considerations," RFC 6895 (Best Current Practice), Internet Engineering Task Force, Apr. 2013. [Online]. Available: http://www.ietf.org/rfc/rfc6895.txt

[20] Neustar, "Neustar Announces Acquisition of Verisign's Public DNS Service," 2020. [Online]. Available: https://www.home.neustar/about-us/news-room/press-releases/2020/neustar-announces-acquisition-of-verisigns-public-dns-service

[21] H. Shulman and M. Waidner, "Towards security of internet naming infrastructure," in *European Symposium on Research in Computer Security*. Springer, 2015, pp. 3–22.

[22] A. Hubert and R. van Mook, "Measures for Making DNS More Resilient against Forged Answers," RFC 5452 (Proposed Standard), Internet Engineering Task Force, Jan. 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5452.txt

[23] R. Elz and R. Bush, "Clarifications to the DNS Specification," RFC 2181 (Proposed Standard), Internet Engineering Task Force, Jul. 1997, updated by RFCs 4035, 2535, 4343, 4033, 4034, 5452. [Online]. Available: http://www.ietf.org/rfc/rfc2181.txt

[24] C. Rigney, A. Rubens, W. Simpson, and S. Willens, "Remote Authentication Dial In User Service (RADIUS)," RFC 2058 (Proposed Standard), Internet Engineering Task Force, Jan. 1997, obsoleted by RFC 2138. [Online]. Available: http://www.ietf.org/rfc/rfc2058.txt

[25] radsecproxy, "naptr-eduroam.sh," 2012. [Online]. Available: https://github.com/radsecproxy/radsecproxy/blob/8d287300f510e0559f01a2e7a4dec90674215f25/tools/naptr-eduroam.sh

[26] MITRE, "CVE-2010-4052 in GNU C Library (glibc)," 2010. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-4052

[27] M. Smith and T. Howes, "Lightweight Directory Access Protocol (LDAP): Uniform Resource Locator," RFC 4516 (Proposed Standard), Internet Engineering Task Force, Jun. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4516.txt

[28] S. Kitterman, "Sender Policy Framework (SPF) for Authorizing Use of Domains in Email, Version 1," RFC 7208 (Proposed Standard), Internet Engineering Task Force, Apr. 2014, updated by RFC 7372. [Online]. Available: http://www.ietf.org/rfc/rfc7208.txt

[29] Ubuntu Community Help Wiki. (2013) Postfix/SPF. https://help.ubuntu.com/community/Postfix/SPF, accessed 2020-02-04.

[30] The Postfix Home Page. (2020) Postfix SMTP Access Policy Delegation. http://www.postfix.org/SMTPD_POLICY_README.html, accessed 2020-02-04.

[31] P. Mockapetris, "Domain names - concepts and facilities," RFC 1034 (INTERNET STANDARD), Internet Engineering Task Force, Nov. 1987, updated by RFCs 1101, 1183, 1348, 1876, 1982, 2065, 2181, 2308, 2535, 4033, 4034, 4035, 4343, 4035, 4592, 5936. [Online]. Available: http://www.ietf.org/rfc/rfc1034.txt

[32] A. Costello, "Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)," RFC 3492 (Proposed Standard), Internet Engineering Task Force, Mar. 2003, updated by RFC 5891. [Online]. Available: http://www.ietf.org/rfc/rfc3492.txt

[33] MITRE, "CVE-2009-4487 in nginx 0.7.64," 2009. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-4487

[34] ——, "CVE-2013-1862 in Apache HTTP Server," 2013. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-1862

[35] The OpenWrt Project, "LuCI - OpenWrt Configuration Interface," 2020. [Online]. Available: https://github.com/openwrt/luci

[36] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman, "A search engine backed by Internet-wide scanning," in *22nd ACM Conference on Computer and Communications Security*, Oct. 2015.

[37] M. Kührer, T. Hupperich, J. Bushart, C. Rossow, and T. Holz, "Going wild: Large-scale classification of open dns resolvers," in *Proceedings of the 2015 Internet Measurement Conference*, ser. IMC '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 355–368. [Online]. Available: https://doi.org/10.1145/2815675.2815683

[38] D. Tatang, C. Schneider, and T. Holz, "Large-scale analysis of infrastructure-leaking dns servers," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2019, pp. 353–373.

[39] A. Gulbrandsen, P. Vixie, and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)," RFC 2782 (Proposed Standard), Internet Engineering Task Force, Feb. 2000, updated by RFC 6335. [Online]. Available: http://www.ietf.org/rfc/rfc2782.txt

[40] D. Kaminsky, "It's the End of the Cache As We Know It," Presentation at Blackhat Briefings, 2008.

[41] Y. Gilad, A. Herzberg, and H. Shulman, "Off-path hacking: The illusion of challenge-response authentication," *IEEE Security & Privacy*, vol. 12, no. 5, pp. 68–77, 2013.

[42] A. Herzberg and H. Shulman, "Security of patched DNS," in *European Symposium on Research in Computer Security*. Springer, 2012, pp. 271–288.

[43] ——, "Socket overloading for fun and cache-poisoning," in *Proceedings of the 29th Annual Computer Security Applications Conference*, 2013, pp. 189–198.

[44] ——, "Vulnerable delegation of dns resolution," in *European Symposium on Research in Computer Security*. Springer, 2013, pp. 219–236.

[45] M. Brandt, T. Dai, A. Klein, H. Shulman, and M. Waidner, "Domain validation++ for mitm-resilient pki," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 2060–2076.

[46] F. Alharbi, J. Chang, Y. Zhou, F. Qian, Z. Qian, and N. Abu-Ghazaleh, "Collaborative Client-Side DNS Cache Poisoning Attack," in *INFOCOM*. IEEE, 2019.

[47] T. Dai, H. Shulman, and M. Waidner, "Dnssec misconfigurations in popular domains," in *International Conference on Cryptology and Network Security*. Springer, 2016, pp. 651–660.

[48] H. Shulman and M. Waidner, "One key to sign them all considered vulnerable: Evaluation of DNSSEC in the internet," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017, pp. 131–144.

[49] T. Chung, R. van Rijswijk-Deij, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson, "Understanding the role of registrars in dnssec deployment," in *Proceedings of the 2017 Internet Measurement Conference*, 2017, pp. 369–383.

[50] A. Herzberg, H. Shulman, and B. Crispo, "Less is more: cipher-suite negotiation for dnssec," in *Proceedings of the 30th Annual Computer Security Applications Conference*, 2014, pp. 346–355.

[51] P. Hoffman and P. McManus, "Rfc 8484: Dns queries over https (doh)," 2018.

[52] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. Hoffman, "Rfc 7858-specification for dns over transport layer security (tls)," 2016.

[53] H. Shulman, "Pretty bad privacy: Pitfalls of dns encryption," in *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, 2014, pp. 191–200.

[54] S. Siby, M. Juarez, C. Diaz, N. Vallina-Rodriguez, and C. Troncoso, "Encrypted dns–> privacy? a traffic analysis perspective," *arXiv preprint arXiv:1906.09682*, 2019.

[55] Z. Su and G. Wassermann, "The essence of command injection attacks in web applications," *Acm Sigplan Notices*, vol. 41, no. 1, pp. 372–382, 2006.

[56] Y. Nadji, P. Saxena, and D. Song, "Document structure integrity: A robust basis for cross-site scripting defense." in *NDSS*, vol. 20, 2009.

[57] M. Dalton, H. Kannan, and C. Kozyrakis, "Real-world buffer overflow protection for userspace and kernelspace." in *USENIX Security Symposium*, 2008, pp. 395–410.

[58] J. Blasco, "Introduction to xpath injection techniques," in *Hakin9, Conference on IT Underground, Czech Republic*, 2007, pp. 23–31.

[59] J. M. Alonso, R. Bordon, M. Beltran, and A. Guzman, "LDAP injection techniques," in *2008 11th IEEE Singapore International Conference on Communication Systems*, Nov. 2008, pp. 980–986, iSSN: null.

[60] M. Johns and J. Winter, "Requestrodeo: Client side protection against session riding," in *Proceedings of the OWASP Europe 2006 Conference*, 2006.

[61] S. P. Chandramouli, P.-M. Bajan, C. Kruegel, G. Vigna, Z. Zhao, A. Doupé, and G.-J. Ahn, "Measuring e-mail header injections on the world wide web," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, 2018, pp. 1647–1656.

[62] T. Terada, "Smtp injection via recipient email addresses," *MBSD White Paper (December 2015)*, 2015.

[63] C. Jackson, A. Barth, A. Bortz, W. Shao, and D. Boneh, "Protecting browsers from dns rebinding attacks," *ACM Transactions on the Web (TWEB)*, vol. 3, no. 1, pp. 1–26, 2009.

[64] G. Acar, D. Y. Huang, F. Li, A. Narayanan, and N. Feamster, "Web-based attacks to discover and control local iot devices," in *Proceedings of the 2018 Workshop on IoT Security and Privacy*, 2018, pp. 29–35.