Saarland University

Department of Computer Science

# Anatomy of DNS: Investigating Vulnerabilities and Countermeasures from Clients to Authoritative Servers

Dissertation
zur Erlangung des Grades
des Doktors der Ingenieurwissenschaften
der Fakultät für Mathematik und Informatik
der Universität des Saarlandes

von
Jonas Bushart

Saarbrücken, 2024

|  |  |
|---|---|
| Tag des Kolloquiums: | 4. Juni 2025 |
| Dekan: | Prof. Dr. Roland Speicher |

**Prüfungsausschuss:**

|  |  |
|---|---|
| Vorsitzender: | Prof. Dr. Karl Bringmann |
| Berichterstattende: | Prof. Dr. Christian Rossow |
|  | Dr. Nils Ole Tippenhauer |
| Akademischer Mitarbeiter: | Dr. Robert Künnemann |

# Zusammenfassung

Das Domain Name System (DNS) ist eine wichtige Komponente der Internet-Infrastruktur, die für die Übersetzung von menschenlesbaren Internet-Domains in IP-Adressen zuständig ist. Es wurde gezeigt, dass das DNS für verschiedene Angriffe anfällig ist, darunter die Analyse des Datenverkehrs, semantische Fehler und Denial-of-Service-Angriffe (DoS). In dieser Dissertation werden die Sicherheit und der Datenschutz von DNS erforscht, wobei der Schwerpunkt auf der Ermittlung von Schwachstellen und der Entwicklung wirksamer Gegenmaßnahmen liegt.

Unsere Forschung zeigt, dass verschlüsselte DNS-Protokolle trotz der Verwendung von „Padding“ zur Verschleierung der Nachrichtengröße immer noch anfällig für Angriffe zur Verkehrsanalyse sind. Wir demonstrieren eine neuartige Methode zur Analyse des Datenverkehrs, mit der Websitebesuche mit hoher Genauigkeit deanonymisiert werden können, was die Notwendigkeit effektiverer Schutzmaßnahmen aufzeigt. Darüber hinaus identifizieren wir Schwachstellen in rekursiven DNS-Resolvern mithilfe eines mutationsbasierten Fuzzers, ResolFuzz, und zeigen, dass differentielles Fuzzing ein effektiver Ansatz zur Aufdeckung von DNS-Schwachstellen sein kann.

Unsere Arbeit erforscht auch die Bedrohung durch (D)DoS-Angriffe gegen die DNS-Infrastruktur, einschließlich eines neuen DDoS-Angriffs auf der Anwendungsebene, DNS Unchained, der die Verstärkung nutzt, um autoritative Namensserver zu überlasten. Wir zeigen, dass dieser Angriff mit gepulsten Angriffen kombiniert werden kann, um einen leistungsfähigeren und schwieriger zu blockierenden Angriff zu schaffen. Abschließend bewerten wir die Widerstandsfähigkeit autoritativer DNS-Infrastrukturen gegen (D)DoS-Angriffe auf der Anwendungsebene und schlagen eine Abwehrmaßnahme zur Erkennung von Anomalien vor, die von vorgelagerten ISPs oder Internet Exchange Points eingesetzt werden kann, um solche Angriffe zu entschärfen. Insgesamt trägt diese Dissertation zum Verständnis der DNS-Sicherheit und des Datenschutzes bei und bietet Einblicke in die Entwicklung wirksamer Gegenmaßnahmen zum Schutz dieser kritischen Infrastruktur.

# Abstract

The Domain Name System (DNS) is a critical component of the Internet infrastructure, responsible for translating human-readable domain names into IP addresses. However, DNS has been shown to be vulnerable to various attacks, including traffic analysis, semantic bugs, and denial-of-service (DoS) attacks. This dissertation explores the security and privacy of DNS, with a focus on identifying vulnerabilities and developing effective countermeasures.

Our research reveals that encrypted DNS protocols are still susceptible to traffic analysis attacks, despite the use of padding to obscure message sizes. We demonstrate a novel traffic analysis method that can deanonymize website visits with high accuracy, highlighting the need for more effective mitigations. Furthermore, we identify vulnerabilities in recursive DNS resolvers using a mutation-based fuzzer, ResolFuzz, and show that differential fuzzing can be an effective approach to uncovering DNS vulnerabilities.

Our work also explores the threat of (D)DoS attacks against DNS infrastructure, including a new application-layer DDoS attack, DNS Unchained, which uses amplification to overload authoritative name servers. We demonstrate the potential for this attack to be combined with pulsing attacks to create a more powerful and harder-to-block attack. Finally, we assess the resilience of authoritative DNS infrastructures against application-layer (D)DoS attacks and propose an anomaly detection defense that can be deployed by upstream ISPs or Internet Exchange Points to mitigate such attacks. Overall, this dissertation contributes to the understanding of DNS security and privacy and provides insights into the development of effective countermeasures to protect this critical infrastructure.

# Background of this Dissertation

This dissertation is based on five peer-reviewed papers published at RAID 2018 [P1], WOOT 2018 [P2], FOCI 2020 [P3], IEEE Euro S&P 2023 [P4], and ESORICS 2023 [P5]. I contributed to all papers as the corresponding author. I contributed to all papers as the only author next to faculty-level co-authors and in the case of [P2] as a single author.

Artifacts produced while working on the individual papers are partially available online. The FOCI 2020 paper “Padding Ain’t Enough: Assessing the Privacy Guarantees of Encrypted DNS” [P3] has the source code[1] and datasets published. The dataset is split into the main dataset[2] and the extra sets of Section 3.2.2 for Firefox[3] and Chrome[4].

To ease reproducibility of the ESORICS 2023 paper “ResolFuzz: Differential Fuzzing of DNS Resolvers” [P5] the released data[5] includes (i) the full source code of ResolFuzz, (ii) the build scripts and configuration files for the resolvers, (iii) the raw data of our evaluation, (iv) a list of found differences, and (v) scripts for further analysis.

The Euro S&P 2023 paper “Anomaly-based Filtering of Application-Layer DDoS Against DNS Authoritatives” [P4] has the code and some datasets[6] available online The released code (i) ingest NetFlow data into a database, (ii) builds the respective allowlists, and (iii) evaluates the methodology based on given datasets. We have a prototype implementation for the allowlist creation, which converts a NetFlow file into an allowlist. The data sets used to model the Mirai, Sality and Open Resolver populates are included. Given the sensitivity of the DNS name lookups, the country-code TLD (ccTLD) asked for a non-disclosure agreement prior to our cooperation. According to national data protection laws, sharing any data from the country-code TLD (ccTLD) is impossible, as it would violate the well-defined reasons under which this data was collected in the first place.

[P1] **Bushart, J.** and Rossow, C. DNS Unchained: amplified application-layer DoS attacks against DNS authoritatives. In: *Proceedings of the 21th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*. Sept. 2018. DOI: 10.1007/978-3-030-00470-5_7 (cit. on pp. vii, 16, 78, 85, 90).

[P2] **Bushart, J.** Optimizing recurrent pulsing attacks using application-layer amplification of open DNS resolvers. In: *12th USENIX Workshop on Offensive Technologies (WOOT)*. Aug. 2018 (cit. on pp. vii, 16).

[P3] **Bushart, J.** and Rossow, C. Padding ain’t enough: assessing the privacy guarantees of encrypted DNS. In: *10th USENIX Workshop on Free and Open Communications on the Internet (FOCI)*. USENIX Association, Aug. 2020 (cit. on pp. vii, 15).

[1] https://github.com/jonasbb/padding-aint-enough
[2] https://zenodo.org/records/7319358
[3] https://zenodo.org/records/7319364
[4] https://zenodo.org/records/7319380
[5] https://github.com/dns-differential-fuzzing/dns-differential-fuzzing
[6] https://github.com/cispa/DNS-Applayer-DDoS-Protection/

[P4] **Bushart, J.** and Rossow, C. Anomaly-based filtering of application-layer DDoS against DNS authoritatives. In: *8th IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, July 2023. DOI: 10.1109/EUROSP57164.2023.00040 (cit. on pp. vii, 17).

[P5] **Bushart, J.** and Rossow, C. ResolFuzz: differential fuzzing of DNS resolvers. In: *28th European Symposium on Research in Computer Security (ESORICS)*. Springer Nature, Sept. 2023. DOI: 10.1007/978-3-031-51476-0_4 (cit. on pp. vii, 16).

## Further Contributions of the Author

I was granted the privilege of contributing to the IMC 2015 paper [S1] as a student research assistant.

[S1] Kührer, M., Hupperich, T., **Bushart, J.**, Rossow, C., and Holz, T. Going Wild: large-scale classification of open DNS resolvers. In: *Proceedings of the 2015 ACM Internet Measurement Conference (IMC)*. Oct. 2015. DOI: 10.1145/2815675.2815683 (cit. on pp. viii, 64, 67, 69, 78).

# Acknowledgments

First and foremost, I would like to thank my advisor Christian Rossow. Ever since my Bachelor studies, he has guided me throughout my academic career. He has offered me exciting topics to work on, as well as given me the freedom to explore my own ideas. Throughout this whole time, he has guided me and provided the necessary motivation during the difficult times.

I would also like to thank all of my colleagues from the System Security group: Ahmad, Benedikt, Fabian, Giancarlo, Giorgi, Ilya, Johannes, Leon, and Michael. A special thanks to Benedikt and Markus with whom I've shared an office. Thank you for all your fruitful discussions and late-night paper deadlines. Doing a PhD was mostly a solitary adventure for me, but sharing an office with you made it much more enjoyable.

Special thanks also go to all collaborators for their knowledge exchange and support.

A special mention goes to the CTF team saarsec and especially Ben for his work in organizing it.

Finally, I would like to thank my family for their unconditional support. Most importantly, thank you Simina, for always believing in me.

# Contents

# List of Figures

# List of Tables

# 1 Introduction

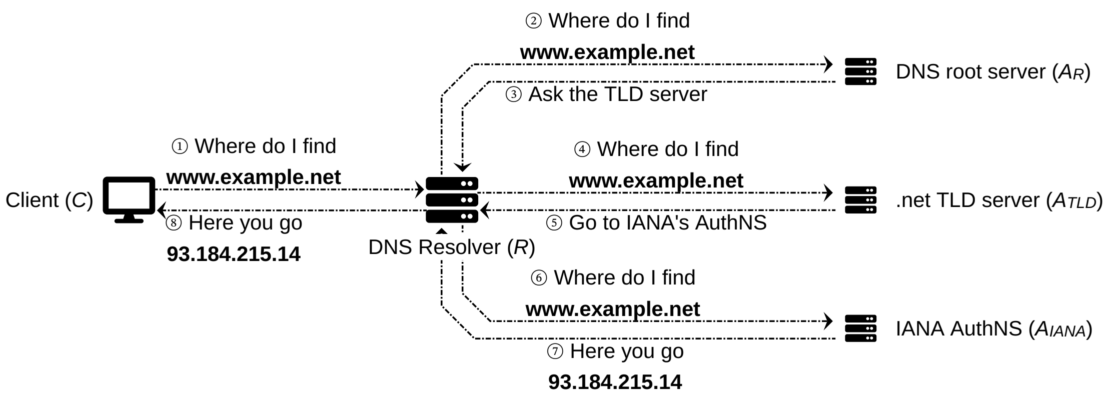


**Figure 1.1:** This simple example shows the high-level messages that are exchanged when fetching the IP address of a domain. The client ($C$) requests the IP address information from the resolver ($R$). The resolver does not have the information cached and now needs to ask the Authoritative Name Servers (AuthNSes) for the information. The resolver starts at a root server $A_R$ and gets redirected at each level to a more specific AuthNS ($A_{TLD}$ and $A_{IANA}$).

The Domain Name System (DNS) is a critical part of the Internet infrastructure. Most users are unaware of the DNS system and its importance. They interact with it through their web browser, where the DNS translates a domain name (`www.example.net`) into an IP address (`93.184.215.14`). The IP address is then used to establish a connection to the web server. Because of this, DNS is often described as the telephone book of the Internet. DNS is used for much more than web browsing and is entrenched in most interactions on the Internet, by providing connectivity information, storing cryptographic keys for secure communication, and helping with email delivery and spam prevention.

A simple illustrative example in Fig. 1.1 shows the basic workings of a DNS query. The client ($C$) is interested in the IP address (type `A`) of domain `www.example.net` and sends the query (①) to a recursive resolver ($R$). The resolver has a cache and can directly return available data. Here, the cache is empty, so the resolver forwards the query (②) to a root server ($A_R$). The root does not have the final answer but can get one step closer, by responding (③) with a delegation (`NS` answer), that instructs the resolver to retry the query at the Authoritative Name Servers (AuthNSes) for `net` ($A_{TLD}$). The same process repeats at the AuthNS for `net`, which has no final answer either but can delegate to the AuthNSes for `example.net` (steps (④) and (⑤)). This AuthNS ($A_{IANA}$) has the final answer and returns (⑦) the desired IP address for `www.example.net`. The IP address gets cached at the resolver and is returned (⑧) to the client. A more realistic depiction of how DNS operates is in Fig. 2.1.

DNS is not only involved when opening a website. It plays a role in issuing TLS certificates via `CAA` records, determining the mail servers when sending email, and storing anti-spam information (e.g., DKIM [108], SPF [99], or IP address blocklists [48]). DNS aids in service connectivity with `SRV` and `SVCB` records and stores cryptographic keys in `SSHFP`, `OPENPGPKEY`, and `TLSA` records. As DNS is used for so many interactions on the Internet, it plays a central role in the Internet infrastructure, making it a target for censorship and surveillance and a point of failure with widespread consequences.

In many countries, DNS filtering is used to block access to certain websites. Prominent examples are the Great Firewall of China [80] or the blocking of Twitter in Turkey [69]. DNS interference and filtering can be mandated by a court, where they force Internet Service Providers (ISPs) to block access to certain websites. These cases occur for content that is illegal in a country, such as gambling or porn, or for copyright violation. DNS blocking is often used because the operators of the websites are located in countries out of reach from law enforcement and DNS blocking is relatively simple. For example, the website `kinox.to` was blocked in Germany by court order [172] for distributing copyrighted materials. DNS operators are targeted too, like in the case of Sony Music vs Quad9 [64, 155, 156]. Another kind of DNS manipulation is NXDOMAIN hijacking, where ISPs redirect DNS queries for non-existing domains to a search page [13, 55]. This is done to generate extra revenue through advertising on the search page. This practice can be disruptive for users and can even break some applications. Further, it has privacy implications, for example, if the domain name reveals sensitive information like a medical condition, and it leaks browser cookies to the search page, allowing user tracking or account hijacking.

This centralization in the DNS, which makes filtering and manipulations easy and effective, also makes DNS a point of failure with widespread consequences if it fails. Outages are quite common and even make it into the news. Distributed Denial-of-Service (DDoS) attacks use many devices, often compromised by hackers, to send a large amount of traffic to a target, making it unreachable. They were used on November 30, 2015, to attack the DNS root servers [139], but with limited effect, due to high redundancy, use of anycast, and caching in the DNS network. A year later, on October 21, 2016, the DNS provider Dyn was attacked [79], causing many website failures, especially in the US and US East Coast. It is not only attacks but configuration errors that can cause outages. On July 22, 2021, the DNS provider Akamai had a configuration error that broke DNS for two hours [9, 175, 192], causing many websites to be unreachable. Some DNS features are complicated to get right, like DNSSEC, which attaches signatures to DNS records to ensure the authenticity of DNS responses. Hundreds of outages are reported to be caused by DNSSEC misconfigurations, bugs, or expired signatures [61]. On September 30, 2021, Slack had a 24 hour outage due to a DNSSEC-related bug [59, 60]. Their DNS provider Amazon Route53 created signed wildcard records with wrong information, which caused validating DNS resolvers to think that the Slack domain does not exist, thus making Slack unreachable. Outages are not always the fault of DNS, but DNS can play a large part in the harm caused by the outage. On October 4, 2021, Facebook had a configuration error that took their data centers offline [94, 114, 127]. Their DNS servers have a health check that checks for reachability to the data centers, and if it fails, the DNS locations withdraw their Border Gateway Protocol (BGP) announcements, making them unreachable from the Internet. Even though the DNS servers were fine this behavior caused the whole DNS of Facebook to fail. Without DNS, many internal tools stopped working, communication with their employees broke down, and even door locks were no longer functional. These examples show the need for a robust and functional DNS ecosystem, which is why this dissertation focuses on understanding DNS and closing weaknesses in DNS setups.

## 1.1 Research Questions

The cases mentioned earlier highlight some challenges within the DNS. For the most part, DNS works for the everyday user in large part due to the network engineers working for network operators and content providers. Yet, the incidents show that DNS has gaps in protecting users and providing resilient operations. This leads us to *three research questions* we want to answer in this dissertation.

### 1.1.1 (RQ1) How can we protect users' privacy and limit surveillance possibilities?

Improving privacy for users is an ongoing effort in the Internet Engineering Task Force (IETF), especially after the Snowden revelations. The IETF has adopted a policy to improve privacy and consider monitoring an attack on the Internet [31, 65]. For DNS the work is done in the DPRIVE working group, which has published several standards to improve privacy. This takes many different forms, from encrypting the DNS traffic (DNS over TLS (DoT) [87], DNS over HTTPS (DoH) [82], DNS over QUIC (DoQ) [88]) to extending the protocol with padding to hide size information (RFC 7830 [128]). DoT, DoH, and DoQ are protocols to encrypt DNS traffic between the client and the resolver, protecting privacy by hiding data from prying eyes. Other working groups contribute as well, such as dnsop with QNAME minimization [14] or the dedicated doh group for DoH [82]. QNAME minimization avoids sending unnecessary information to root and Top-Level Domain (TLD) servers, covering the communication between the resolver and AuthNSes. It increases privacy by not leaking private information in the first place. In the simple DNS overview Fig. 1.1 it replaces the queries to the AuthNSes with shorter domains `net` (②) and `example.net` (④), thus stopping all but one AuthNS from learning the full domain.

In our example Fig. 1.1, the most critical link is the initial one (②/⑧), between the client ($C$) and the resolver ($R$). Any information leaked here contains the identity of the user (IP address) and the website they are visiting (domain name). In the resolver the information for multiple users is mixed, making it harder to identify a single user on the links between the resolver and AuthNSes.

The privacy weaknesses of bad or insufficient encryption schemes are well-known, with a vast amount of research on the topic of traffic analysis. Traffic analysis is the process of extracting information from encrypted traffic by looking at the size, timing, and direction of the traffic. In Chapter 3 "Padding Ain't Enough: Assessing the Privacy Guarantees of Encrypted DNS" we investigate the privacy protection of the existing encryption schemes against an attacker using traffic analysis to extract information about the websites a user visits (website fingerprinting). We perform the analysis using the existing padding schemes suggested in RFC 8467 [129], which further hide size information. To this end, we study different attacks, with varying hardware configurations and attacker models.

We implement and evaluate mitigations that further hide or mask timing side channels in the DNS data stream. These mitigations are client-side only, making them easy to deploy and compatible with the existing DNS infrastructure. Even simple schemes can provide a large improvement in privacy, but they are not perfect and still allow some information leakage.

### 1.1.2 (RQ2) How can we improve correctness and reduce bugs in DNS software?

The best system is useless if it is not implemented correctly. The DNS ecosystem is complex with hundreds of RFCs [18, 29, 113, 161] defining the standard and many different implementations [19, 200] that all need to interoperate. This means a correct implementation is hard to achieve since mistakes are easy. Mistakes can be exploited by attackers to crash the software, infiltrate systems, or manipulate the data.

Correctness is important for all parts of the DNS system, but especially for the recursive resolvers ($R$). They are the most complex part, as they query many "untrusted" AuthNSes ($A_R$ to $A_{IANA}$) and combine their results. Clients ($C$) usually blindly trust the recursive resolvers, such that any data modifications here are challenging to detect.

Another complicating factor is the lack of good test suites or operational semantics that allow testing of DNS implementations. Kakarla et al. tackled this challenge for AuthNSes, by first developing a formal model [96] and then use the model to guide a fuzzer [97]. This works well for the simpler model of AuthNSes, but is not directly applicable to recursive resolvers, since the formal model does not cover the them.

Instead, in Chapter 4 "ResolFuzz: Differential Fuzzing of DNS Resolvers" we develop a differential fuzzer for recursive resolvers. We use the fuzzer to find bugs in the most popular recursive resolvers, including BIND 9 [11], Unbound [205], and Knot Resolver [103]. By using differential fuzzing we circumvent the lack of a ground truth, such as a formal model or test suite. Instead, given that many implementations exist and are compatible with each other, we only need to detect the outliers to spot potential bugs. This reduces the problem to a simpler decision problem: deciding if two implementations behave the same or not.

### 1.1.3 (RQ3) How can we protect the DNS infrastructure from attacks?

Besides the aforementioned privacy and correctness issues with DNS, availability is another important aspect. The best system is useless if it is unusable. As such, ensuring availability throughout the DNS system is an important role for the protocol designers and operators.

Besides bugs that can crash a server, the biggest threats are Denial-of-Service (DoS) and Distributed Denial-of-Service (DDoS) attacks. Here one or multiple parties send large amounts of traffic to a server, such that it is unable to respond to legitimate requests. The traffic can be sent from other DNS components, such as recursive resolvers, or from other systems, such as botnets. In both cases, the simplest kind of attacks are floods, which consist of thousands or millions of requests. The attacks differ in the content, the way the requests are created, how simple they are to filter, and the threat they pose.

There are two different aspects to consider. First are problems in the DNS specification or the implementations of DNS servers, which can help facilitate attacks. A simple vulnerability is an amplification attack, where a small request can cause a large response or numerous responses, such as with the ANY pseudo query type. The second aspect is the infrastructure itself. We must consider the options available to operators for mitigating attacks, providing capacity, and filtering out malicious traffic while still providing a good service to their users.

We tackle a protocol-inherent problem in Chapter 5 “DNS Unchained: Amplified Application-Layer DoS Attacks Against DNS Authoritatives”. While analyzing the DNS specification and recursive resolver implementations, we find an inherent amplification vulnerability using the CNAME and DNAME records. These records are meant to point to a canonical name in the DNS, basically acting like an alias. While discouraged, chaining of these records is allowed, such that a CNAME record can point to another CNAME record. A recursive resolver will follow these chains until it reaches a non-CNAME record or runs into a timeout. Many websites make use of chained CNAME records, like Content Delivery Networks (CDNs), making it impossible to simply remove chaining support without breaking the Internet. We measure the impact of this vulnerability and find that it can be used to amplify traffic by a factor of 8.51 and the impact it has on both the AuthNSes and the recursive resolvers.

With a slight modification to the previous attack, we can extend its potential impact to DoS links close to an AuthNS. This is a more powerful attack as now victims can be any service running in the same data center as an AuthNS. In Chapter 6 we show how a smart attacker can coordinate multiple recursive resolvers to send their traffic synchronized. This leads to traffic spikes that can cause packet loss and harm the throughput of TCP connections.

In the last chapter “Anomaly-based Filtering of Application-Layer DDoS Against DNS Authoritatives” (Chapter 7) we inspect the current state of the art for anycast DNS infrastructures and their exposure to different kinds of application-layer attacks. We investigate the setup of a large European TLD operator and notice gaps in their infrastructure that make flooding attacks feasible. For different attackers we estimate the potential in attack traffic, modeling a botnet and an attacker using recursive resolvers as intermediaries. For these threats and other flooding attacks, we develop an anomaly-based filtering scheme, that makes use of existing traffic flow information and allows upstreaming the filtering to ISPs or IXPs, thus filtering the traffic closer to the source.

## 1.2 Outline

The remainder of this dissertation is structured as follows: The next chapter contains the technical background explaining the DNS fundamentals. The next five chapters cover the main content and address the research questions. The chapters are ordered to match the research questions and each has “motivation” and problem description sections to introduce them. Lastly, we summarize our results and check how we addressed the research questions.

# 2 Technical Background

This chapter provides the background on DNS, its actors, and their interactions. First, we give a brief overview of the DNS functionality (Section 2.1) from an end-user perspective, such as daily interactions and other systems that rely on DNS. Then, we describe how DNS works internally when providing information, the different actors, and their interactions (Section 2.2). Finally, we discuss the security of DNS (Section 2.3) by describing an ideal DNS system and pointing out existing weaknesses. We provide a comprehensive analysis of our work and demonstrate how we enhance the DNS security.

## 2.1 DNS Functionality

Every interaction on the Internet starts with a DNS query. Most users are unaware of the DNS system and its importance. The most common interaction with DNS is typing a domain name into a web browser. Here the DNS system translates the domain name (`www.example.net`) into an IP address (`93.184.215.14`), which is then used to establish a connection to the web server, similar to finding a phone number in a telephone book.

This is the most common use case of DNS, but fundamentally, DNS is a distributed key-value database system. The keys are domain names, and the values are Resource Records (RRs). These RRs can contain arbitrary data, commonly IP addresses, but also other information such as mail server addresses, cryptographic keys for secure communication, or text records for domain verification, spam prevention, and other purposes. This information makes DNS entrenched in many interactions on the Internet, by providing connectivity information, offering a single global namespace, and being fundamental to TLS certificates[1].

For the most part, DNS is invisible to the end user and acts in the background. A client interacts with a DNS resolver and queries it for the desired RRs. A DNS server is usually provided by the network operator or ISP and configured on the user's device via Dynamic Host Configuration Protocol (DHCP) or Point-to-Point Protocol (PPP). This default can manually be overridden. There are many reasons for doing so, such as performance, increased privacy or security, preferring DNS-based content filtering of malicious domains and adult content, or circumventing filtering and censorship. For example, Turkish protesters used Google's DNS service to circumvent the DNS-based censorship of Twitter and YouTube in 2014 [188].

### 2.1.1 DNS Message and Resource Record Format

A DNS message contains a header and the four sections questions, answers, authority, and additional. Each section is a list of Resource Records (RRs) which we describe later on. The size of the header is too tiny for modern features, such that a special `OPT` RR in the additional section is used to carry extra data. This extension mechanism for DNS (EDNS) is defined in RFC 6891 [184].

For the purposes of this work, we care primarily about DNS queries and replies, indicated with an opcode of 0 and the QR flag of 0 for queries and 1 for replies. The

[1]DNS appears for TLS certificates as DNS names in the Subject Alternative Name extension and as domain validated (DV) certificates.

DNS queries and replies are the forms used for fetching IP addresses of domains. Other opcodes exist for more specialized operations mainly around operating an AuthNS. A DNS query has a single RR in the questions section and optionally the `OPT` RR in the additional section. While the format allows for multiple RRs in the question section, this has no defined semantics and will be rejected by most software. A DNS reply mirrors the question section of the query and some header values to make it possible to match the replies to the queries. The reply can have zero or more RRs in any of the three sections answers, authority, and additional. For DNS clients, like browsers, the answers section is the most important. The other two sections are mainly used to carry extra information from AuthNSes to DNS resolver.

A DNS Resource Record (RR) is a type-length-value encoded piece of data, with an owner name, like `www.example.net`, a type, like `A` to IPv4 addresses, a class, always `IN` for Internet, and a variable sized Record Data (RDATA) depending on the type. Other classes beside `IN` exist for now defunct alternative networks, but do not play a role besides DNS servers diagnostics. The format is simpler in the questions section, only consisting of name, type, and class, since the RDATA cannot be known at question time. All RRs with the same name, type, and class form a single Resource Record Set (RRset). The records in the RRset are unordered. While possible to have different Time-to-Live (TTL) values in a RRset, this is not permitted [62].

DNS resolvers and AuthNSes can handle any RR type and do not need to understand them. The exceptions are all RRs needed for the operation of a DNS resolver. Some records can be supported optionally, like all DNSSEC [8, 110, 169] related ones. Among the record types that are needed for operation are types defining the DNS hierarchy (`SOA`, `NS`), address records to reach DNS servers (`A`, `AAAA`, for IP addresses), and special operating instructions (`CNAME`, `DNAME`, both defining canonical aliases). DNSKEY records contain the keys (`DNSKEY`), delegations (`DS`), and signatures (`RRSIG`, `NSEC`, `NSEC3`). The `NS`, `A`, and `AAAA` records are needed for the DNS resolvers to learn the correct AuthNSes to connect to and the IP addresses to use. The `CNAME` and `DNAME` records are followed by the resolvers. They “point” to a different canonical name and cause the resolver to restart its resolving with the new name but same type. The final answer will contain the original name and all intermediate `CNAME` and `DNAME` records and the data record.

## 2.2 DNS Network and Ecosystem

The user interacts with a small part of the DNS network and most of it remains invisible. Four different kinds of actors exist in the DNS network: (i) DNS clients as the consumers of the data, i.e., browsers, (ii) stub resolvers in operating systems (OS) or customer-premises equipment (CPE) (iii) recursive DNS resolvers, and (iv) Authoritative Name Servers (AuthNSes). The DNS clients are the consumers of the data and initiators of a chain of DNS queries, that traverse the network and reach the AuthNS.

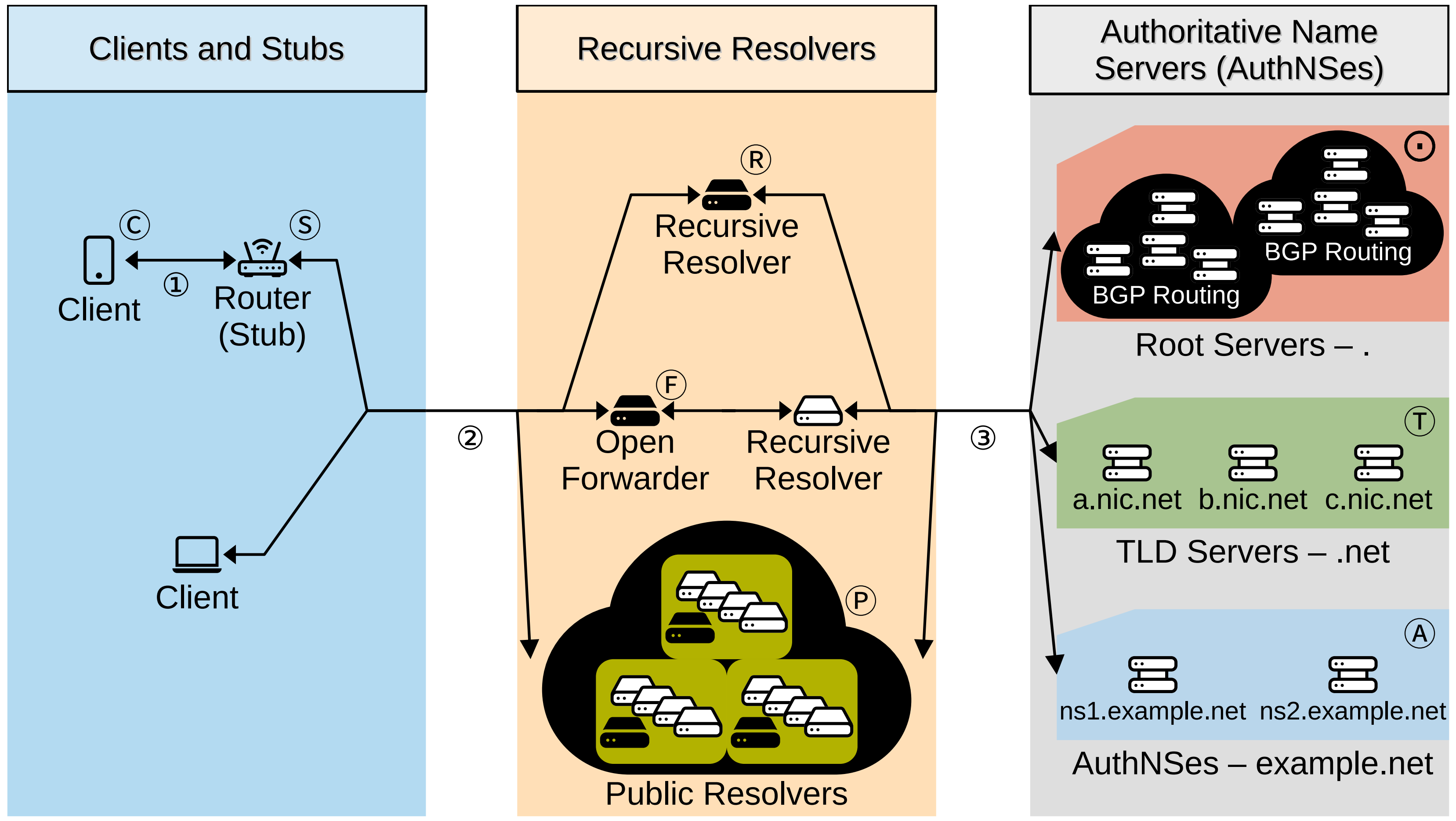


**Figure 2.1:** The picture shows the DNS network and its actors. On the left side are different user devices, like a smartphone or a router, which are issuing DNS queries. They are processed by the resolvers in the middle. Different kinds of resolvers are depicted, which are commonly operated by ISPs or special organizations. On the right side are the AuthNSes which are operated by the domain owners or their web hosters and CDN providers and they host the data needed by the clients.

The DNS clients are the initiators of queries. They send the query using the operating system to a stub or recursive DNS resolver. The stub resolver often runs on the same system or in the same local network and caches answers to reduce load and improve latency. Unless the corresponding data is already cached the queries are forwarded to a recursive DNS resolver. Some stubs offer additional features like filtering or secure protocols (DoT/DoH) but these do not change the behavior. Recursive DNS resolvers are the backbone of the DNS network as they find where the requested data is stored and retrieve it. They are often operated by ISPs or the network operator. Their job is to navigate the DNS hierarchy, starting at a (hard-coded) root, and find the correct AuthNSes for the requested domain and intermediate steps. Then they retrieve the requested data, validate it, cache it, and return it to the client. Finally, AuthNSes are the source of the data and are operated by domain owners or their web hosters and CDN providers. Their task is comparatively simple, as they provide only data for a limited number of domains (or zones) and respond with negative answers if they do not have the requested data.

A domain name looks like `www.example.net` and describes a path in the DNS hierarchy. The DNS hierarchy is a tree structure, where the root is at the top and each label in the domain name is a node in the tree. Our domain `www.example.net` has three labels, `www`, `example`, and `net`. These labels are usually written separated by dots, with `net` being the highest in the hierarchy. Implicitly there is another dot at the end `www.example.net.`, which represents the root of DNS. Written like this the domain name is called a Fully Qualified Domain Name (FQDN), but the trailing dot is often omitted. We call `.` the root, `net.` a TLD, and `example.net.` a Second-Level Domain (SLD). Sometimes a further distinction is made between the parts of a domain that can be registered by the public and the part that acts like a TLD. For example, `bbc.co.uk` has a user registrable part `bbc` and a effective Top-Level Domain (eTLD) `co.uk`. There is no clear definition for eTLD, as this is not encoded in the DNS system itself. Instead, the *Public Suffix List* [153] is used to determine the eTLD of a domain.

We show with an example how the DNS messages flow through the network in Fig. 2.1. All the caches are empty here. If data is already present in the cache, the stub or recursive resolver can skip some steps. A user wants to visit `www.example.net` in their browser. The browser Ⓒ wants to know the IP address, which is stored in the `A` (IPv4) and `AAAA` (IPv6) RRs. This query "`www.example.net. A`" is sent ① to the stub resolver Ⓢ. If the stub has the answer cached, it returns it to the browser. Otherwise, it forwards ② the query to the recursive resolver Ⓡ. The recursive resolver has hard-coded information about the root servers ⊙ of the DNS. Having no other information, it forwards ③ the "`www.example.net. A`" query to one of the root servers. These do not know the answer either, but they know the TLD servers for `de.` Ⓣ, so they return the address of one of them. This is called a referral and tells the recursive resolver the next step in the hierarchy, by providing it the names and addresses of the AuthNSes. The same process repeats for the TLD server, which returns ③ the address of the AuthNS for `example.net.` Ⓐ. Finally, the recursive resolver can ask ③ the AuthNS for `example.net.` for the `A` record of `www.example.net.` and receives the desired answer. The answer now flows back through the stub to the browser and the user can visit the website.

This only covers the simple case while reducing all the real-world complexities that exist in DNS deployments. A recursive resolver Ⓡ can be a single computer. Some DNS servers on the Internet do not perform the full recursive resolution but forward the query to another server. We call them forwarders Ⓕ and they can forward the traffic to a single machine or to multiple like a load balancer. They are similar to the stub resolvers Ⓢ but can employ different caching rules and be shared among more users. Another common setup are public resolvers Ⓟ, like Google DNS [76], Cloudflare DNS [27], or Quad9 [154]. They are intentionally open to the public Internet, which is often avoided for other recursive resolvers, as this can open up the system to misuse and attacks. This openness attracts many users requiring more complex setups to be redundant and performant. Symbolized in the graphic is a setup with multiple machines, since the recursive lookup can be costly. Many operators choose anycast deployments, where machines are running in many physical locations but are reachable with the same IP address. This provides redundancy and better performance, due to lower Round-Trip Time (RTT) when sending traffic to them. We symbolize anycast deployments here with the cloud symbol.

## 2.3 DNS Security

The DNS protocol was designed in the 1980s and lacks many security features we expect from modern protocols. Some features have been added as optional extensions like data authentication with DNSSEC [81, 167, 168, 169, 214] and data encryption (DoT [87]/DoH [82]). This has two consequences: Security features are not widely deployed since they are optional. The security is designed as an afterthought and does not cover all weaknesses of the protocol.

There are five properties we expect from a secure DNS system.

**Privacy:** Only the relevant parties should see the queried domains and the responses. This necessarily includes all stub and recursive resolvers along the path and the final AuthNS.

DNS is unencrypted by default, meaning any party on the path can see the queries and responses. This is especially problematic for the early parts between clients, stubs, and recursive resolvers, as here the queries and the IP address of the querier are visible. This allows the correlation of people with their web browsing behavior, which can be very sensitive if it reveals sexual preferences or medical information, like the name of a specialized doctor. Data encryption between client and recursive resolver is possible using DoT [87] or DoH [82], but not all clients and resolvers support it, and some networks block it. No standardized solution exists for encrypting the communication between recursive resolvers and AuthNSes, but some resolvers use DoT for this purpose.

In Chapter 3 “Padding Ain’t Enough: Assessing the Privacy Guarantees of Encrypted DNS” [P3], we analyze the privacy guarantees of encrypted DNS protocols, like DoT and DoH. We show that even with these protocols, the privacy guarantees are not as strong as one might expect. Visiting a website requires multiple DNS queries, which can be correlated by an attacker, even if the queries are encrypted and the message sizes are obfuscated with padding.

**Authenticity:** A client should be able to verify that responses are authentic, meaning they are unmodified, ensuring data integrity, and are created by the expected source, ensuring data origin authentication.

Any communication is unencrypted by default, and not even protected by signatures. This means that any party on the path can modify the messages, and censorship systems perform these modifications to block access to certain domains. DNSSEC [81, 167, 168, 169, 214] provides data authentication between resolvers and AuthNSes, but it is not widely deployed.

**Correctness:** A client should be able to verify that responses are correct, the data stems from the expected source, and no mistakes were made.

There are many ways to attack the correctness of DNS responses. With cache poisoning, the attacker tries to inject wrong data into the cache of a resolver, redirecting users and systems to a malicious server. This is possible because DNS does not verify the identity of the communication parties or the integrity of the messages. Old information of the AuthNS domain or IP addresses can cause broken referrals. In the best case they cause no harm and only slow down the resolution, but in the worst case they allow somebody to take over a domain, by operating a malicious AuthNS. Implementation bugs are another source of incorrect responses.

In Chapter 4 "ResolFuzz: Differential Fuzzing of DNS Resolvers" [P5] we show how to find bugs in DNS resolvers using differential fuzzing. Differential fuzzing as a tool is well suited for finding correctness bugs, as it highlights any discrepancies between two implementations.

**Misuse Resistance:** A user must be unable to cause harm to the system or others by misusing the system.

DNS defaults to using the UDP protocol for communication and has no required identification step. This makes it easy for an attacker to spoof the source address of a DNS message and redirect the response to a victim, flooding them with unwanted traffic. This is called a reflection attack and is possible because neither UDP nor DNS check the identity of the communication parties. The TCP 3-way handshake provides a weak form of IP address authentication, and DNS over TCP support is required for all parties, but not the default. The reflection can be combined with an amplification attack, where the DNS responses are significantly larger than the queries. This can be achieved by using RR types with large payloads, like `TXT` or `DNSKEY` records, or by using a special query type `ANY`, which instructs the DNS server to provide all available records. Configuration options to limit response sizes over UDP and peer identity check using DNS cookies [56, 193] are available, but as optional extensions, they cannot block all attacks.

In Chapter 5 "DNS Unchained: Amplified Reflections, Traffic Analysis, and Circuit Switching Attacks" [P2, P1] we show how a fundamental design in the DNS protocol can be misused to amplify traffic. This allows attacks that target recursive resolvers and AuthNSes and show how a synchronized attack can cause larger issues. We discuss countermeasures applicable to recursive resolvers and AuthNSes that help mitigate this and similar attacks. The issue arises from RRs that redirect the resolver and cause it to send more queries than it received. We show this for `CNAME` and `DNAME` records, but similar observations have been made for `NS` records [1].

Chapter 6 "Optimizing Recurrent Pulsing Attacks using Application-Layer Amplification of Open DNS Resolvers" provides a twist on the DNS Unchained attack by intelligently combining the attack of the reflectors to create traffic pulses that can harm a wider range of targets, like TCP connections that operate via a shared network link.

**Availability:** The system should be resilient against attacks, both on a technical level (e.g., DoS attacks) and on a legal level (e.g., censorship). This is a special case of misuse resistance since it only covers the DNS itself. Many systems rely on a functional DNS to work, causing DoS attacks to have a large impact.

The DNS protocols provide the necessary tools to offer resilience, by allowing multiple AuthNSes for a domain, but it depends on the AuthNS operator to make good use of these tools and further harden their systems with DoS mitigation techniques, like traffic filtering, overprovisioning, and fast interventions. Managed badly, DNS remains weak against attacks.

In Chapter 7 "Anomaly-based Filtering of Application-Layer DDoS Against DNS Authoritatives" [P4] we analyze the resilience of AuthNSes against DoS attacks. We discuss how modern DNS setups attempt to provide resilience against different attack vectors. We identify gaps in the handling of application-layer attacks and propose a new approach to detect and mitigate these attacks.

# 3
# Padding Ain't Enough: Assessing the Privacy Guarantees of Encrypted DNS

## 3.1 Motivation

In our journey through the DNS landscape, we first look at the clients and stub resolvers. This chapter covers the network traffic on the links ① and ② from the DNS overview in Fig. 2.1. The defenses discussed in this chapter affect primarily the clients Ⓒ and stub resolvers Ⓢ, with optional changes in the DNS resolvers Ⓡ/Ⓕ/Ⓟ for better effectiveness. One weakness here is the plaintext nature of DNS messages, which allows passive adversaries to eavesdrop on DNS traffic. The DoT RFC 7858 [87] and DoH RFC 8484 [82] protocols provide encryption and in this chapter, we analyze the privacy guarantees of these protocols, by testing whether Website Fingerprinting (WF) attacks are still possible.

DoT and DoH encrypt DNS to guard user privacy by hiding DNS resolutions from passive adversaries. Yet, past attacks have shown that encrypted DNS is still sensitive to traffic analysis. As a consequence, RFC 8467 [129] proposes to pad messages before encryption, which heavily reduces the characteristics of encrypted traffic. In this chapter, we show that padding alone is insufficient to counter DNS traffic analysis. We propose a novel traffic analysis method that combines size and timing information to infer the websites a user visits purely based on encrypted *and padded* DNS traces. To this end, we model *DNS Sequences* that capture the complexity of websites that usually trigger dozens of DNS resolutions instead of just a single DNS transaction. A closed-world evaluation based on the Tranco top-10k websites reveals that attackers can deanonymize test traces for 86.1 % of all websites, and even correctly label all traces for 65.9 % of the websites. Our findings undermine the privacy goals of state-of-the-art message padding strategies in DoT/DoH. We conclude by showing that successful mitigations to such attacks have to remove the entropy of inter-arrival timings between query responses.

## 3.2 Problem Description

DNS is one of the most fundamental protocols on the Internet because it is the starting point of nearly all Internet traffic. It translates memorable names into cryptic IP addresses. One critical path for privacy is between the client and the resolver where several entities, such as Wi-Fi access points or ISPs can eavesdrop. They can use this information for advertisement or to create browsing profiles of the victims. Upon visiting a website by the user, the client sends one or multiple DNS queries to the resolver, which can either answer them from the cache or perform the iterative lookup to query the AuthNS. This means that DNS leaks almost all user behavior to any eavesdropper.

Users are understandably concerned about privacy, yet for years DNS has had no means of protecting the messages integrity and the messages confidentiality. Partial solutions like DNSSEC [49, 50] exist, but do not provide confidentiality. DNS over TLS (DoT) [87] and DNS over HTTPS (DoH) [82] close the gap and finally provide confidential DNS. These solutions are supported in some resolvers [45], browsers [17, 130, 198] and operating systems [95, 100].

Yet, similar to other privacy-preserving communication systems that leverage encryption (e.g., HTTPS [177, 217] or Tor [145, 185, 210]), DoT and DoH are susceptible to traffic analysis. Gillmor's empirical measurements [72] show that passive adversaries can leverage the mere size of a single encrypted DNS transaction to narrow down the queried domain. RFC 8467 [129] then follows Gillmor's suggestions to pad DNS queries and responses to multiples of 128 B / 468 B.

## 3.3 Contribution

In this chapter, we study the privacy guarantees of this widely deployed DoT/DoH padding strategy. We assume a passive adversary (e.g., ISP) who aims to deanonymize the DNS resolutions of a Web client. Modern Web services are intertwined, such that visiting a website, creates many DNS queries. While this padding strategy destroys the size entropy of messages, we assess to what extent DNS resolution *sequences* (e.g., due to third-party content) allows adversaries to reveal the Web target. Such sequences utilize DNS *message sizes and timing information* between DNS transactions.

We then leverage a $k$-Nearest Neighbors classifier to search for the most similar DNS transaction sequences in a previously trained model. Our closed-world evaluation shows that an attacker can deanonymize 86.1 % of all test traces and we show how DNS provides a better basis for performing subpage-agnostic domain classification. These findings undermine the privacy goals of state-of-the-art message padding strategies in DoT/DoH, which is highly critical.

Two previous papers [86, 182] also address DoT/DoH. [86] addresses DoT by using statistical features of sizes and timing. In their most similar setting, they report 83 % correct classifications for a dataset of only 98 websites. [182] attacks DoH without utilizing timing, which we found to be significant. For a comparable dataset (1500 domains) they report 94.0 % precision, but they also report that DoH is easier to classify. Compared to both papers we use the largest dataset (10k domains) and address *subpage-agnostic classification* and evaluate *countermeasures*.

Summarizing, we provide the following contributions: (i) We illustrate a traffic analysis attack that leverages DNS transaction *sequences* to reveal the website a client visits. (ii) We provide an extensive analysis of the privacy guarantees offered by DNS message padding (RFC 8467 [129]) against our attack in a Web browsing context. We demonstrate severe privacy losses even against passive adversaries that sniff on encrypted and padded DNS traffic and *extend* the analysis beyond just the index page. (iii) We are the first to evaluate alternative padding strategies and constant-rate communication systems against our proposed attack. We reveal that even perfect padding cannot mitigate traffic analysis, and show that any promising countermeasure needs to obfuscate timing.

## 3.4 Traffic Analysis and DNS Padding

We now present how an adversary can use traffic analysis to infer the browsing target purely based on encrypted DNS traffic. Encrypted DNS only leaves three characteristics, which can be used to infer the communication content, namely (i) counts, such as packets, (ii) sizes, such as overall transmitted bytes, and (iii) time. These three dimensions provide valuable hints about the communication content. Two standards describe how to add padding to reduce size information: RFC 7830 [128] describes how to add padding to

DNS messages and RFC 8467 [129] recommends a padding scheme of padding all queries to a multiple of 128 B, while all responses are padded to a multiple of 468 B.

Encrypted DNS only protects against eavesdroppers between the client and the resolver, but assumes that the resolver is trusted. This threat model does not cover upstream communication of the resolver towards the AuthNSes. The resolver sees all communication in plaintext and has to be trusted to uphold privacy. We follow this threat model and assume a passive attacker who can observe the communication between the client and the resolver, yet cannot delay, alter, or inject data into the traffic. The attacker is allowed to initiate their own network connections from the same network-topological location as the victim.

### 3.4.1 DNS Sequences

The padding recommendations are based on single query/response pairs. We show that attackers can leverage a *sequence* of DNS query/response pairs, to increase the uniqueness compared to individual DNS transactions. Web browsing causes the sequences of DNS queries for many reasons, such as redirects, loading of third-party resources, or resources from subdomains. As we will show, these sequences characterize a user's website visit quite well.

Consider a visit to `wikiquote.org` which triggers four DNS requests/responses upon visit. It fetches the IP address for the domain, then after 287 ms for the `www` subdomain, and after another 211 ms for both the `meta` and `upload` subdomains simultaneously. The resulting DNS Sequence (which ignores requests) looks like `Msg(1), Gap(8), Msg(1), Gap(7), Msg(1), Msg(1)`. The DNS Sequence encodes the four DNS responses (each 468 B long, i.e., one padding block). If there is a time gap between two responses, we note this gap and its magnitude using a millisecond log scale (e.g., $\lfloor log_2(287) \rfloor = 8$). This makes it less susceptible to timing variations, e.g., due to network jitter. We remove all time gaps with a numerical value $\leq 0$ (i.e., those shorter than 1 ms), such as between the last two Msgs.

The DNS Sequence only includes DNS responses, since we found the DNS queries to have almost no entropy and their timing is highly correlated with the replies. We chose to represent message sizes and timestamps abstractly as this provides higher flexibility and generalizes over different implementations and events outside of our control (e.g., network performance, jitter). This simplifies our design while keeping most features.

**DNS Sequence Extraction:** We derive the DNS Sequences from encrypted and padded traffic. We identify a DNS carrying connection using ports (853 for DoT), IP addresses (e.g., `9.9.9.9`), the TLS handshake [89, 90], or DNS-like characteristics (like packet sizes). Then we reassemble these TCP streams and extract "application data" TLS records. After some data cleanup, like merging consecutive records and ignoring overhead such as certificates, we only keep the message sizes and inter-arrival times of the DNS messages.

### 3.4.2 DNS Sequence Classifier

Our classifier uses $k$-Nearest Neighbors ($k$-NN) to assign labels to DNS Sequences based on the labels of the nearest neighbors, i.e., the DNS Sequences most like the unlabeled one. This assumes that similar DNS Sequences belong to the same website. The $k$ parameter specifies how many neighbors should be searched and the plurality of the $k$ found labels determines the output classification.

We base the distance function upon the Damerau-Levenshtein/edit distance [40, 119]. It counts the edit operations required to turn one sequence into another using the four operations insertion, deletion, substitution, and transposition. We assign each operation a different cost based on the importance of each operation. For example, changes to the volatile timing information have a lower cost than to the rather stable size information. We optimize these constants using a hyperparameter search over a subset of our closed-world dataset.

## 3.5 Evaluation

We will now evaluate the efficacy of our proposed methodology to classify DNS Sequences that we obtain from traffic captures. We shortly describe our dataset generation setup and then introduce two evaluation scenarios. The closed-world scenario shows the baseline classification performance, while the subpage-agnostic domain classification extends the classification beyond the index page.

### 3.5.1 Measurement Setup

We create our dataset by visiting the top websites in the Tranco list [203] and recording the DNS traffic. We use a server with a 10 GB/s network interface running Debian 9.11. For the closed-world scenario, we also collect data using a Raspberry Pi 3 running Raspbian 10. The low-power Pi allows us to measure the effect of hardware performance on dataset collection and classifier.

For each website, we spawn a Docker container running Firefox 72 and Unbound 1.9.4 as the DNS stub resolver. We configure Unbound to forward all DNS queries using DoT to Cloudflare's resolvers at `1.0.0.1` and `1.1.1.1`. Unbound's DNS cache is preloaded with the `NS` entries for all TLDs since we assume a user will have these records cached due to past resolutions. We control Firefox using Selenium. Measurements are repeated up to two times if we detect errors, such as missing DNS traffic or HTTP errors. Lastly, we convert the network traffic into a DNS Sequence as described in Section 3.4.1 and label it based on the website. We group identical websites under a single label, such as websites using different TLDs for their localized versions.

### 3.5.2 Evaluation Results

We evaluate our classifier in two settings. First, the *closed-world*, which highlights the best case for an attacker, and provides a baseline for the performance and second, we perform *subpage-agnostic domain classification*, where we classify more than just the index page.

#### 3.5.2.1 Closed-World Scenario

The *closed-world* scenario is the easiest for the attacker since all websites a client can visit are known in advance. The attacker just has to decide which website was visited.

Our dataset consists of the top 10 000 websites from the Tranco list [151, 203] (2019-08-27) for which we collect ten samples each. For 9235 websites we could collect this data, the others repeatedly caused errors, and we could not collect ten DNS Sequences. A second set of DNS Sequences is collected on the Raspberry Pi, for which we collect four traces per website. Data for 7699 websites was collected successfully.

**Table 3.1:** Percentage of correctly classified DNS Sequences in the closed-world scenario with different classifiers and datasets.

| Classifier | Server | Raspberry Pi |
|---|---|---|
| k-NN ($k = 1$) | 86.1 % | 80.9 % |
| k-NN ($k = 3$) | 85.6 % | 79.0 % |
| NN | 81.4 % | 63.5 % |

This fully labeled dataset allows us to use cross-validation to measure our classifier. We use 10-fold cross-validation between all the DNS Sequences since we have exactly ten traces per website and measure the percentage of correctly classified DNS Sequences, and show the results in Table 3.1. Our $k$-NN classifier achieves up to 86.1 % on the server dataset and 80.9 % for the Raspberry Pi dataset, both with $k = 1$. We notice that higher $k$'s slightly reduce the performance of the classifier, but not significantly, so even with $k = 9$ the performance only falls to 82.9 % for the server dataset.

**Table 3.2:** Per website results for fixed $k = 1$ in the closed-world scenario. The table shows the percentage of websites we can re-identify in how many of the 10 data points/traces.

| $n/10$ traces | $1/10$ | $2/10$ | $3/10$ | $4/10$ | $5/10$ | $6/10$ | $7/10$ | $8/10$ | $9/10$ | $10/10$ |
|---|---|---|---|---|---|---|---|---|---|---|
| Precision | 92.2 | 91.8 | 91.3 | 90.8 | 89.9 | 88.5 | 86.9 | 84.4 | 79.6 | 65.9 |

The per-website results in Table 3.2 show how well we can classify websites instead of DNS Sequences. We achieve a perfect classification (i.e., $10/10$) for about two-thirds of all websites. If we relax the requirements to 90 % correct classification per website, we can classify about 80 % correctly—purely based on *encrypted and padded* DNS traffic.

We build a second type of classifier using Neural Networks (NNs), also shown in Table 3.1. It performs in the same ballpark for the server dataset (81.4 %), but falls to only 63.5 % for the Raspberry Pi data. Since the results are worse than the $k$-NN classifier, we do not consider it further.

#### 3.5.2.2 Subpage-Agnostic Domain Classification

One challenging aspect of Website Fingerprinting (WF) is subpage-agnostic domain classification, which is also a more realistic scenario. Instead of performing classification only on the index page, the attacker is faced with an arbitrary subpage of the target domain. The challenge in this setup is the high variability of the subpages, for example, due to different embedded images or different amounts of texts, and the much larger space, as usually a domain has many subpages.

Our encrypted DNS-based classifier has a distinct advantage here, as DNS requests are more stable than HTTP(S)-based WF. Intuitively, DNS requests mainly depend on third-party domains (e.g., for JS libraries or fonts). These are commonly defined in templates and thus are identical for many subpages. HTTP(S) traffic is more noisy, as the amount and concrete images often change from subpage to subpage.

We compare our method to Panchenko et al. [145, Fig. 11 (b)] by calculating our subpage confusion matrix. The test consists of a closed-world evaluation with 20 domains and 51 subpages each. For the same set of 20 domains as Panchenko et al. we pick 51 random URIs from the Common Crawl [28] dataset from October 2019. Only domains that resolve to a 200 status code are eligible.

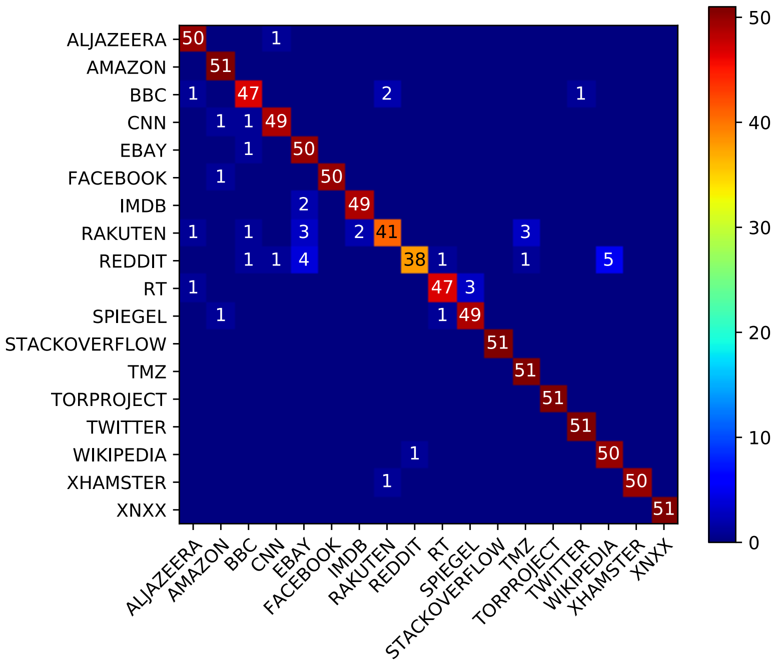


**Figure 3.1:** Subpage confusion matrix between different domains and their 51 distinct subpages.

Compared to Panchenko et al. [145, Fig. 11 (b)] we have a much lower classification error in Fig. 3.1. Out of the total of 918 subpages only 43 (4.7 %) are wrongly classified, whereas Panchenko et al. wrongly classify 175 (19.1 %) subpages. Overall, our methodology performs better on 15 of the 18 domains. The most challenging domains are "Rakuten" and "Reddit" in which our classifier performs worse. One reason why we perform better is the lower variance of DNS traffic compared to HTTP(S).

## 3.6 Countermeasures

We showed that attacks based solely on encrypted DNS traffic are feasible and can be successful. We now analyze in which directions countermeasures need to be developed by understanding the impact of the two major feature types (size and timing) of our classifier. Based on these insights, we then describe potential mitigations.

### 3.6.1 Evaluating Perfect Mitigations

Our classifier uses two feature types: packet sizes and timing information. To build a better countermeasure, we first need to understand which feature contributes more entropy because mitigating that will have the largest effect. To this end, we perform a thought experiment in which we assume perfect mitigations, i.e., a perfect padding scheme and a perfect timing defense. We simulate these by removing the Msg elements, for perfect padding, or the Gap elements from the DNS Sequences.

**Table 3.3:** Comparison between our classifier, a simulated perfect padding scheme, and a simulated perfect timing defense.

| | $k = 1$ | $k = 3$ | $k = 5$ | $k = 7$ | $k = 9$ |
|---|---|---|---|---|---|
| Baseline | 86.1 % | 85.6 % | 84.9 % | 83.9 % | 82.9 % |
| Perfect Padding | 84.5 % | 84.2 % | 83.9 % | 83.2 % | 82.4 % |
| Perfect Timing | 4.2 % | 4.0 % | 4.0 % | 3.9 % | 3.8 % |

We re-run the closed-world classification with these modifications and compare the results in Table 3.3. We see only a minuscule difference for a perfect padding defense, however, we see a large decline in performance for the timing defense. This indicates that the existing padding mechanism is already close to optimal. In contrast, a perfect timing-based defense destroys the classification results. The inter-arrival timings of the DNS responses in the sequences carry significant entropy that we use to classify perfectly padded traffic.

From these observations, we can derive important novel insights. First, the currently proposed padding strategy [72, 129] is indeed a good compromise between overhead and the maximum privacy guarantees that an optimal padding could guarantee. Yet, second, even an optimal padding strategy does not decrease the trace's entropy and does not suffice to preserve the user's privacy. Third, countermeasures should also consider timing information, as timing has proven to contribute significant entropy in DNS Sequences.

### 3.6.2 Evaluating Practical Mitigations

Based on these observations, we now implement two practical mitigations and measure their efficacy and efficiency.

**Constant-Rate (CR) schemes** [54] send a packet on a fixed schedule every $x$ ms. The packet is filled with payload, if some is waiting, otherwise with padding data. CR entirely removes timing information as everything is constant. However, CR has a significant bandwidth and latency overhead, since packets *must* be sent, even if no payload is waiting, and packets have to wait until the next scheduled transmission time thus increasing latency. A larger $x$ reduces the bandwidth overhead but creates a larger latency overhead. Finally, CR requires a termination condition to avoid infinite transmissions. We define a probability $p$, which specifies the likelihood that a dummy packet is sent after the end of the stream.

**Adaptive Padding (AP)** [180] mitigates timing side-channels by masking the statistical timing features on the client side. AP sends dummy traffic with indistinguishable timing from real traffic by switching between creating bursts and waiting for the next burst. The burst sizes, inter-burst, and intra-burst timings must be from realistic distributions, so we extract them from our closed-world dataset.

**Comparing AP with CR:** To compare them, we create an experimental setup to measure them in varying configurations. We implement a DoT proxy that we place between Unbound and the DoT server, providing the mitigations on the client side. Technically, we simulate the effects on DNS Sequences, which allows for quicker testing without regenerating traces.

We measure the mitigation effects with our 10-fold cross-validation setup for varying packet rates $x$ ms (for CR) and the probabilities $p$ (for CR and AP). We test four packet rates from 12 ms to 100 ms and six probabilities $p$ from 0.4 to 0.9. Likewise, we measure the bandwidth overhead in additional DNS messages, the increase in resolution time, ignoring dummy packets, and the impact on the classification results.

Figure 3.2 shows the classification results. It is color-coded with AP being blue (left-most column of circles) and CR in the remaining colors. The x-axis shows the time overhead as a factor compared to the baseline of no modification. Similarly, the y-axis measures the overhead in the number of DNS messages with 1 being the baseline. The size of the colored slice of each circle represents the classification results. A full circle is equal to a 10 % classification success in the $^5/_{10}$ traces correctly classified setup.

All variations are successful in mitigating our classifier. For a similar defense strength AP has a higher bandwidth overhead than low-rate CR, but lower than fast-rate CR. For interactive use cases AP is likely better since it has no timing overhead, yet when bandwidth is of concern, CR with a slow sending rate is probably preferred over AP.

## 3.7 Discussion

Our work helps to understand the privacy threats that DNS users face and how they can be protected. Analyzing new side-channel attacks becomes more important, as there is a general trend for more encryption that aims to mitigate (obvious) privacy breaches.

This leaves DNS as an important target for snooping on the privacy of users since it is the first step in connecting to the Internet. Given that we constrained ourselves to *encrypted and padded* DNS traffic, we find the provided classification results quite alarming. We can partially deanonymize 92.2 % of websites and correctly classify 86.1 % of DNS Sequences. The foremost goal of our study, assessing the privacy guarantees of encrypted DNS, was thus successful, as we have shown drastic privacy problems. The classification accuracy can be further boosted by combining our approach with existing WF methodologies. Having said this, there are some limitations to our proposed methodology, which we will describe next.

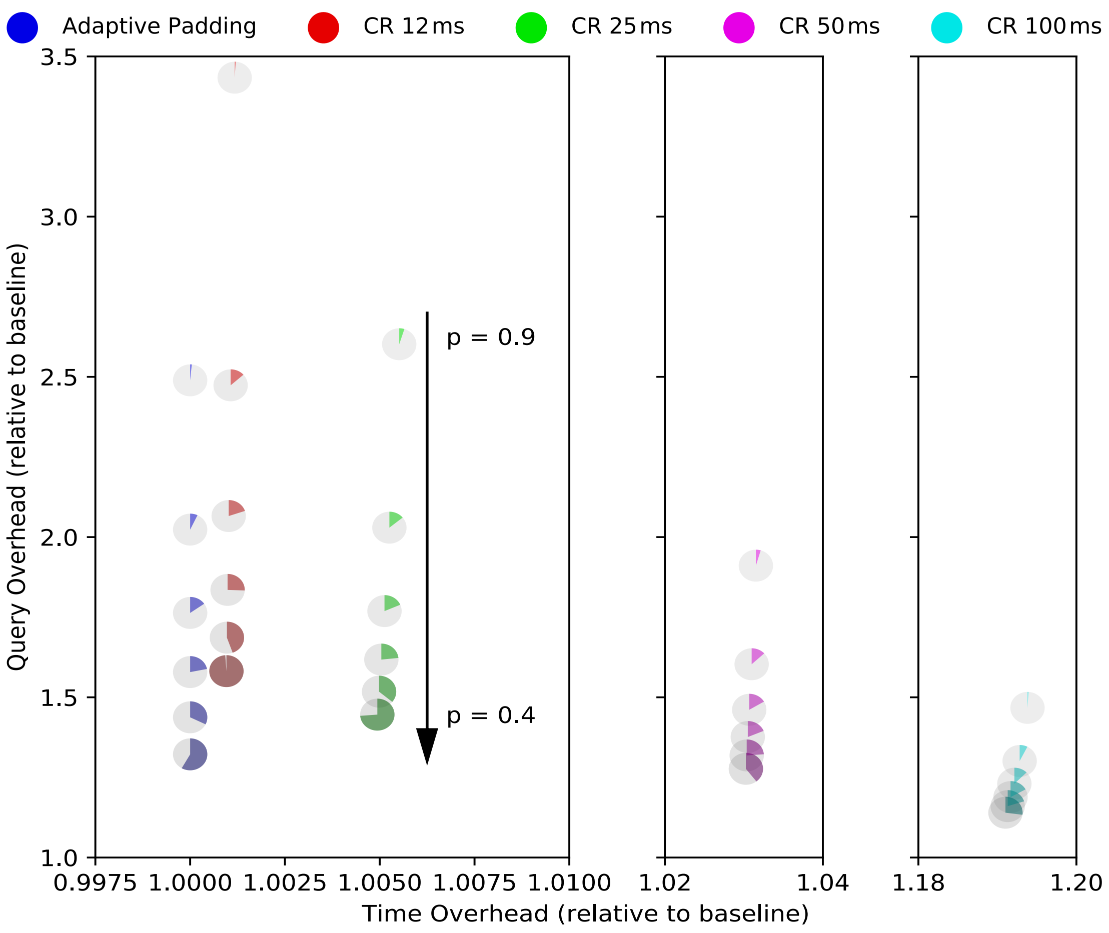


**Figure 3.2:** Comparison of time overhead (x-axis) and packet overhead (y-axis) between AP and CR. A full circle represents $10\%$ correctly labeled domains ($\geq 5/10$ traces correct).

**Choice of Classifiers/Features:** We implement two classifiers, $k$-NN, a conservative choice, easy to understand and reason about. The NN is based on DNS Sequence abstraction and follows a similar model to sentiment analysis, which helps us understand its behavior. $k$-NN performs better in our case and requires no learning phase. However, it might be slow on large datasets and requires specifying a manually engineered distance function. The NN performs close to the $k$-NN yet not as well. Its main benefit is that it requires no manual feature engineering except for the DNS Sequence to vector conversion, yet it requires a long learning phase when new data is added.

**Datasets:** We use a rather extensive dataset with 9235 websites / 92 350 DNS Sequences in the closed-world dataset. Related WF attack papers regularly use a much smaller number of websites in their datasets, in the range of tens to hundreds [78, 145, 185]. A larger dataset causes precision and recall to decline [145, Fig. 10], which means our results would shine better on datasets of identical size to these papers.

**DNS Traffic Identification and Extraction:** The approach to extract a DNS Sequence from network captures assumes DNS servers are identifiable as such. DoT helps, by using a dedicated TLS port, and most DoH servers have dedicated IP addresses.

Similarly, we assumed that we can extract message sizes from the TLS stream, which is possible for the current implementations. They try to use small TLS records to transmit the data, by either using a single TLS record for a DNS reply. In principle, DNS messages could be transmitted in many tiny or equally sized TLS records, hampering attempts of exact message size extraction, but increasing the cost due to more processing and network overhead.

**User Modeling:** For our experiments, we modeled a certain user behavior, which, however, might deviate in practice. First, we assumed the client waits until the website has fully loaded without any background DNS traffic. Second, the evaluations were performed with an empty browser cache and DNS cache, which only included the eTLDs. In practice, users may have different states for their DNS cache and browser cache, which can result in fewer DNS requests sent. However, all major browsers implement strategies to partition the browser cache based on first-party domain [24, 66, 213]. This prevents the browser from re-using common resources across domains and is implemented to prevent user tracking from websites. We think that simulating an empty cache is, therefore, a fair approximation for many websites, given that our attack can only work on the first page visit of a stay on a website. While partial caching may decrease the classifier's accuracy, one could argue that a stateful adversary can use our attack to model the DNS cache state of a user. The adversaries can adapt the training datasets by retraining on the traces that are expected with a certain cache state. We plan to perform such analyses and adapt our classifier accordingly in the future.

**Lack of Entropy:** We noticed that DNS-based classification of structurally less complex websites fails due to a lack of entropy. This affects websites without third-party resources, but we also noticed similar problems for some CDNs. While this is an inherent problem of DNS-based classification, we argue that the continuing rise of the complexity of websites will mitigate this limitation over time.

This restriction also limits the ability to transfer this attack to other domains. Web browsing is unique in how it triggers diverse DNS request chains. Other uses, such as email or web APIs, often cause fewer and more independent requests, which makes it difficult to build DNS Sequences with enough entropy.

## 3.8 Related Work

DoT [87] and DoH [82] were created to provide confidential DNS transactions. The standardized padding policies [129], required to combat traffic analysis, are based on Gillmor's work analyzing individual query/response pairs [72]. We assess this general threat model to encrypted DNS in a Web setting, in which we can leverage dependencies between multiple DNS requests. This argument is agnostic to the specific proposal and applies to future schemes.

Parallel to our work two papers from Houser et al. [86] and Siby et al. [182] were published. Their work is closely related to ours, and we will highlight the key differences.

Houser et al. [86] analyze DoT by using a Machine Learning (ML) classifier on statistical features of a time series of DNS messages. Their threat model is identical to ours. The feature set is built from a time series of DNS messages, containing the timestamp, length of the encrypted message, and the traffic direction. From this, they extract higher-level features, such as the query or response length, total number of packets, time intervals, or queries per second. For each higher-level feature, they calculate a range of statistics (minimum, maximum, median, mean, deciles, and count) which they feed into the ML classifier. They use two self-learning classifiers, random forests, and Adaboost. For domain category inference (i.e., the domain is dating or gambling related) they report 93.65 % to 96.12 % correct classification for unpadded data. This drops to only up to 78.7 % when using padded data. Identifying individual websites with padding, like our work, achieves only 83 % correct classifications. These results are for a dataset with only 98 sensitive websites.

Siby et al. [182] analyzed the privacy of DoH, focusing on unpadded traffic but evaluating the padding impact. The main feature is a sequence of bytes, with the sign indicating the traffic direction and the value either representing the TLS record lengths or the combined size of a burst. This sequence is then converted into bi-grams using a sliding window and fed into a random forests classifier. Importantly, they do not use any timing information, however, in Table 3.3 we found timing to be very important. The dataset ranges from 700 to 1500 domains for the closed-world and 5000 for the open-world. They report a precision of 94.0 % when evaluating on a closed-world dataset comparable to ours. By their own report DoT is much harder to classify than DoH, having a 0.3 reduction in the F1 score (see [182, Table VII]). The paper's strength is the diverse evaluation, by testing with different clients, resolvers, and padding configurations.

**Dataset:** Compared to both papers, we use more domains for our dataset with 9235 domains in the closed-world. Smaller datasets benefit the classification results, as the possibility for wrong classifications and the diversity in the dataset is lower. Even with the larger dataset, we beat Houser et al.'s performance when classifying individual websites with 86.1 % compared to their 83 %. The data collection pipeline is similar in both papers and this chapter.

**Feature Set:** Houser et al. use the most extensive feature set, by including time, sizes, and directionality, while Siby et al. use sizes and directionality. We use sizes and timing, but ignore directionality, as we only use the downstream traffic, since we found the upstream to contain almost no entropy.

Houser et al.'s use of inspectable self-learning classifiers allows them to list the most important features. In the case of individual website classification with padding, they are different timing features, like our results in Section 3.6.1. Interestingly, when performing domain category inference timing plays almost no role, even with padding. Siby et al. only use low-level byte counts and no timing, and therefore the feature importance does not apply to their classifier.

**Unique Contributions:** Most importantly, we are the only ones to implement and test countermeasures specific to encrypted DNS fingerprinting (see Section 3.6). Both papers measure the impact of padding on their classification results, whereas we only evaluate the impact of our classifier on padded data with the standardized 128 B/468 B block padding. This padding is also the most aggressive one tested in the related papers. Siby et al. test two additional configurations, a perfect padding in which all sizes are indistinguishable analog to our work in Section 3.6.1 and the effect of tunneling DNS over Tor. Houser et al. only discuss the high-level ideas of defense, namely hiding exact size and timing information.

We are the only ones measuring subpage-agnostic classification (see Section 3.5.2.2) in which we visit webpages beyond the index page, an inherently more complicated task due to the variety of pages.

## 3.9 Conclusions

Our work underlines the importance of carefully studying the possibility of traffic analysis against encrypted protocols, even if message sizes are padded. While there is a plethora of literature on Website Fingerprinting based on HTTPS and Tor traffic, we turned to encrypted DNS—an inherently more complex context, given the low entropy due to short sequences and small resources. We show that passive adversaries can inspect sequences instead of just single DNS transactions to break the widely deployed best practice of DNS message padding. We hope that our observations will foster more powerful defenses in the DNS setting that can withstand even more advanced traffic analysis attacks like ours.

# 4 ResolFuzz: Differential Fuzzing of DNS Resolvers

## 4.1 Motivation

We continue the DNS landscape journey by next focusing on the software of recursive DNS resolvers Ⓡ from the DNS overview in Fig. 2.1. For fuzzing the resolvers, the network traffic on the links ② and ③ is of interest. The clients Ⓒ and AuthNS Ⓐ are not the focus of this chapter. They are only relevant since they can be attacker-controlled and can send crafted DNS queries to the resolver, but are not the target of the fuzzer.

This chapter identifies and analyzes vulnerabilities in the DNS infrastructure, with a particular focus on recursive DNS resolvers. We aim to identify semantic bugs that could lead to incorrect resolver responses, introducing risks to the Internet's critical infrastructure. To achieve this, we introduce ResolFuzz, a mutation-based fuzzer to search for semantic differences across DNS resolver implementations. ResolFuzz combines differential analysis with a rule-based mechanism to distinguish between benign differences and potential threats. We evaluate our prototype on seven resolvers and uncover multiple security vulnerabilities, including inaccuracies in resolver responses and possible amplification issues in PowerDNS Recursor's handling of `DNAME` RRs. Moreover, we demonstrate the potential for self-sustaining DoS attacks in resolved and trust-dns, further underlining the necessity of comprehensive DNS security. Through these contributions, our research underscores the potential of differential fuzzing in uncovering DNS vulnerabilities.

## 4.2 Problem Description

The DNS is often explained as the Internet's phone book since it turns human-readable names like `www.example.net` into an IP address. This analogy greatly simplifies the central role DNS plays on the Internet besides just delivering IP addresses. In fact, DNS defines the singular namespace of the Internet, provides cryptographic material for secure communications, and acts as a backbone for other services such as anti-spam measures or secure routing.

This makes DNS part of the critical infrastructure. Thus, risks and vulnerabilities in DNS are of the highest concern. Recursive resolvers are the centerpiece of DNS, as they resolve domains for clients, iteratively getting answers from AuthNSes. By design, resolvers are public or at least semi-public, exposing them to various threats from malicious clients. Likewise, they interact with potentially malicious AuthNSes. This complex setting also complicates the task of testing DNS resolvers, as it requires modeling both, clients and AuthNSes. Combined with their central role, this means DNS servers are a highly prized target for malicious actors. Infiltration and manipulations of DNS allow far-reaching exploits like preparing for DoS attacks, intercepting communication, or forging TLS certificates. Further security protocols like DNSSEC or full TLS encryption of services like email are not widespread enough to catch DNS manipulations.

## 4.3 Contribution

In this chapter, our primary goal is to identify and analyze semantic bugs and gaps in the DNS resolver implementations. Such bugs could allow an attacker to get a resolver to return wrong answers. With this work, we want to support developers by highlighting problems earlier in the development process, as they hint at bugs or problems with the specification. To this end, we developed ResolFuzz, a fuzzer specifically designed for DNS and implementing differential testing mechanisms, allowing us to test thousands of scenarios. We build a rule-based mechanism to identify common and benign differences, such as random values, underspecified behavior, or feature differences.

In the course of our research, we have uncovered a multitude of critical issues. These range from cases where the DNS resolver returned incorrect values, to more complex problems like traffic looping bugs and potential amplification issues. These findings underscore the importance of our differential fuzzing approach in identifying and addressing vulnerabilities in the DNS infrastructure.

In summary, our chapter presents the following contributions: (i) We create a fuzzer for recursive DNS resolvers to uncover vulnerabilities. (ii) We build a differential analysis framework for investigating DNS outputs. This includes rules to separate common benign differences from other sources. (iii) We have discovered multiple new bugs in popular open-source resolvers.

## 4.4 Methodology

In this section, we describe our methodology for finding semantic differences between DNS resolvers. Before we can dig into the technical details, we first need to provide a bit of background on the DNS protocol and the functionality of a DNS resolver. With that in mind, we can lay out our goals and the challenges we face. Lastly, we provide the technical details of ResolFuzz, the infrastructure choices we made, and describe the input generation and output analysis.

### 4.4.1 Threat Model

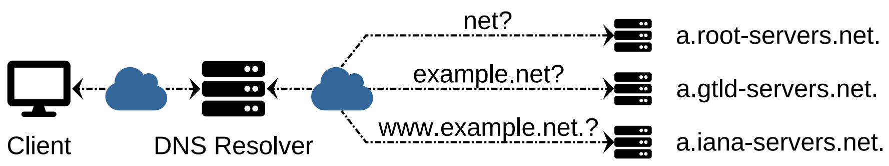


**Figure 4.1:** Basic DNS resolution process. The client sends a query to the resolver, which then recursively resolves the query by interacting with multiple AuthNSes.

Recursive resolvers sit at a very precarious place on the Internet, as shown in Fig. 4.1. They receive queries from clients and answer them from their cache. If the information is missing, they traverse the DNS hierarchy to find the answer from an AuthNS (right). Many resolvers are exposed to the whole Internet and have to talk to many untrusted AuthNSes, indicated by the blue clouds. This opens them up to attacks from clients, AuthNSes, or combined attacks.

For testing the behavior of resolvers, we need to assume that all network communication is potentially malicious. We assume an attacker can send arbitrary queries to the resolver and has control over one or more AuthNSes. While this assumption is trivially fulfilled for public resolvers, it even holds for private resolvers. Server-side requests in HTTP servers, like fetching link previews, or checking email authentication information in SMTP servers allow users to send queries to specific domains, although with less control over the query.

Creating different resolver states is trivial if the attacker is allowed to send different data to the resolvers. For a fair analysis, we must restrict the scenario such that all resolvers receive the same logical data. Some variation must be allowed because DNS messages are not fully deterministic, since they contain random values and do not have canonical encoding.

### 4.4.2 Goals

Our goal is to develop a semi-automated approach for finding and evaluating semantic differences between DNS resolvers. We envision two use cases: (i) Finding semantic bugs in DNS resolvers, which lead to differences, such that two DNS clients no longer agree on the same answer. In the worst case, this is a cache poisoning vulnerability, which can lead to a Certificate Authority (CA) that incorrectly issues a domain-validated certificate. (ii) Our system can be used during the development of DNS resolvers, either to ensure a new resolver is compatible with the existing ecosystem or to verify how a new RFC is implemented.

The system should be scalable to many resolvers and be easy to use, thus it should work with minimal domain-specific knowledge from the human user. This allows it to be run by developers and protocol designers to check their implementations. Integrating a resolver should be free of code modifications or complex adoption necessary by some fuzzing systems.

We do not aim for bugs in the network packet parsing code. Fuzzing on that level will not yield many interesting results in the DNS behavior, since most packets will be dropped early. Bugs in the parsers can be found better with fuzz harnesses for the parsers. Therefore, we only create syntactically valid DNS packets, i.e., following the size and allowed values of the fields.

### 4.4.3 Challenges

DNS is a complex protocol. Over 300 IETF RFCs [18, 161] specify different aspects of it, and over 100 of them are relevant to resolver implementations [29, 113]. This provides a lot of potential for implementation differences and bugs. Automatically exploring such semantic differences is challenging for several reasons.

**State:** A DNS resolver is inherently stateful. Resolvers cache answers, metadata about answers, and information about AuthNSes. Over time the same query can result in different answers. Even at the same time, the same query from different clients can result in different answers, since the client IP address can be part of the cache key with Extended DNS (EDNS) Client Subnet (RFC 7871 [30]).

**Multiple Clients and Servers:** The DNS resolvers sit in the middle between many clients and AuthNSes. Answering a single query can involve multiple servers, pointing the resolver to the next place to ask. The resolver only handles untrusted data. Clients can send queries for any domain name, even for attacker-controlled ones. Our system

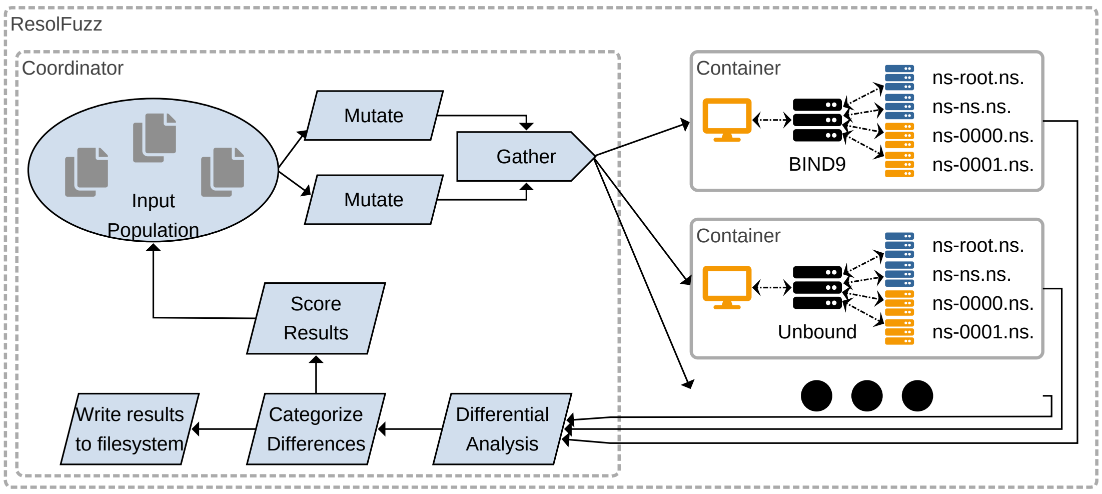


**Figure 4.2:** Overview of our ResolFuzz fuzzer. The boxes indicate different components, where the main component is the Coordinator. The Helper (blue/yellow) runs inside each container and acts as client and as multiple servers. The arrows indicate the data flow. From a pool of fuzzing inputs, some are selected and mutated. They are gathered into a larger set and sent to the resolvers. The yellow parts are controlled by the fuzzing input. The results are collected and compared. Inputs resulting in differences are written to the file system for later analysis. All new inputs are scored and added to the pool.

must be able to represent enough of this complex interaction to provide a large enough coverage but also to find bugs.

**Feedback:** A DNS resolver is for the most part a black box. It receives a query, does some processing, potentially triggering network requests, and returns an answer. On the protocol level, we only have input-output-based interaction, with multiple inputs and outputs. However this lacks information about what is happening inside the resolver, including its state (e.g., cache).

**Output Similarity:** The human language specification used in RFCs is often vague and leaves room for interpretation. Later RFCs might clarify or change the meaning of previous ones. This makes it hard to determine if a difference is a bug or a valid interpretation of the specification. For example, the wording around glue records in RFC 1034 [135] has been interpreted differently by different implementations, requiring further clarification [5].

## 4.4.4 Addressing the Challenges

We use differential fuzzing to tackle the aforementioned challenges. In differential fuzzing, the same fuzzing input is given to multiple implementations, here resolvers. The fuzzer then compares outputs to report on any differences, which may indicate bugs. This approach still leaves us with the challenge of classifying the differences into critical or benign ones, due to the *output similarity* problem. Unfortunately, there is no reference implementation of a perfect DNS resolver that we could use as an oracle. Thus, to reason about the identified differences, we use a combination of heuristics and manual inspection. First, we define rules that describe classes of benign differences, e.g., deviations in the

DNS ID of the header. We group the remaining differences based on fingerprints. They consist of the DNS headers of both outputs and a list of all fields that differ between the two. This cuts down the number of cases that need to be inspected manually.

We restrict the *state* by only using one client-side query during fuzzing, with arbitrarily many AuthNS side responses. This brings the state down to a manageable level but can miss some bugs. For example, the first query caches the wrong AuthNS for a domain and the answer of the second query is based on that value. We do inspect the cache of the resolver, after the fuzzing. For this, we send cache probing queries, i.e., queries with the recursion desired (RD) bit set to 0. This forces the resolver to answer only from the cache. This cache snooping also allows us to obtain *feedback* for fuzzing. But only relying on cache state and DNS messages as output would be too coarse-grained for a fuzzer, which performs best if learning about incremental progress. We thus extend the definition of output and include edge coverage between basic blocks [199].

The fuzzer runs in a fully separated DNS environment with separate DNS root servers, as shown in the containers in Fig. 4.2. For the *multiple clients and servers* challenge we simulate all AuthNSes. The DNS environment is separated using a custom Root Hint [91] configuration for all resolvers and only pointing to AuthNSes on localhost, which is simulated by ResolFuzz.

### 4.4.5 Fuzzing Infrastructure

This section describes the architecture of ResolFuzz. A flowchart of our data flow is shown in Fig. 4.2. ResolFuzz consists of multiple components. This is necessary to achieve scalability in the number of tested resolvers and to keep modifications on the resolvers minimal. The main component *Coordinator* is responsible for input generation, mutation, and output evaluation. A helper component *Helper* runs alongside each resolver and provides a simulated DNS ecosystem for the resolver and communicates with the Coordinator to get fuzzing inputs and deliver the outputs back. We describe how the Helper interacts with the resolver and afterward explain the interaction between the Coordinator and multiple Helpers.

**Resolver Isolation:** One of our goals is that adding a new resolver should require minimal adaptions. One way we achieve this is by running the resolver unmodified, with a normal network stack. However, this requires that we separate different resolvers from each other since all resolvers listen on the same port. For this, we use Linux namespaces in the form of containers. The Helper simulates a custom DNS ecosystem with separate roots and runs in each container.

The Helper also runs the fuzzing inputs against the resolver. Multiple fuzzing queries can be run sequentially, with separate AuthNSes on different IPs per query. This helps with the separation of state between multiple fuzzing inputs.

Lastly, the Helper gathers coverage information from the resolvers and communicates it to the Coordinator. Each resolver is instrumented with LLVM's Sanitizer Coverage [199] pass to gather edge coverage information and only include the edges between sending a query and receiving a response. All edges involved in startup and background tasks are excluded.

**Container Startup:** Spawning a resolver including the Helper in such a fashion is relatively expensive compared to individual DNS queries. For fast fuzzing iterations it is therefore necessary to limit the amount of time we wait for the resolvers to start. Generic network fuzzers like AFLNet [150] come with a forkserver to speed up the spawning of new processes, but that requires modifications to the program and is incompatible with threads. We pre-spawn the containers such that they are ready to receive fuzzing inputs and execute multiple fuzzing inputs sequentially, thus sharing the startup cost. Unfortunately, we cannot run them in parallel, as this would interfere with the coverage information gathering since coverage information is a global property of the resolver, and concurrent queries would interfere with each other.

Second, most of the resolver startup is independent of the concrete fuzzing input. We warm the resolver cache with unrelated DNS queries, such that all name servers will be cached before fuzzing. Testing the fuzzing inputs now only requires the Helper to read the inputs, configure the dynamic DNS server, and start sending the client queries.

**Coordinator and Helper Interaction:** The Coordinator is responsible for generating and mutating the fuzzing inputs and output evaluation, as well as, managing the containers with the Helper. We explain the steps of input generation and evaluations in more detail in Sections 4.4.6 and 4.4.7, respectively. For now, it is sufficient to know, that input generation generates a batch of fuzzing inputs, each consisting of a DNS client query and a set of DNS responses.

After the batch is completed for all resolvers, the output results are collected. The output contains (i) edge coverage information between basic blocks in the form of a hitmap, (ii) all DNS queries sent to AuthNSes by the resolver, (iii) the DNS responses provided to these queries, (iv) the DNS response sent to the client, and (v) information about the resolver cache state. From these outputs, we determine which inputs are “interesting”, i.e., cover new code paths or uncover behavioral differences. More details are in Sections 4.4.7 and 4.4.8.

### 4.4.6 Input Generation and Mutation

ResolFuzz uses mutation for input generation. By picking specific mutations, we can ensure that the mutated inputs stay syntactically valid DNS messages. Having a valid DNS message is important, such that we are fuzzing semantic bugs and not bugs in the DNS message parsers.

We have a population of inputs that we mutate and the ability to generate new inputs. In the beginning, our population is empty, and we start with only newly generated inputs. They are a single DNS query and response pair with randomized query names, labels, types, classes, and header flags. We always include some new random entries in a batch to ensure diversity in the population.

The top $n$ inputs in our population with the highest score are mutated by adding, removing, or modifying the different fields and values of the DNS messages. The score is assigned before adding the input to the population and is decremented each time the input is used as a base for mutations. It includes information about the increase in coverage, the number of known differences, and the number of unknown differences found the last time the input was used.

Mutation consists of modifying existing values and where possible adding or removing values. The typed in-memory representations of DNS messages allow us to walk over the structure and pick a mutation for each field. The DNS header has a fixed length, so the only mutations are changing the existing values to new ones. The different sections can have resource records added or removed. Each resource record has a domain, type, class, TTL, and data. These can be modified, but we always ensure that the data is still valid for the type.

Modifying domain names requires more care than random modifications to ensure enough "collisions" are created to trigger proper behavior in the resolvers. For example, the query name used in the DNS client query should also appear in one of the DNS responses, as otherwise the resolver likely ignores responses and we fail to test the core logic of a resolver. We use a small set of labels from which a domain name can be generated. The inputs use a fixed domain `test.fuzz.` during mutation, but the Helper will later replace the labels with unique ones to separate the inputs. We have two C-string specific mutations, by adding a zero-byte at the end of the last label, i.e., `test.fuzz\0.`, and adding the zero-byte but also duplicating the domain name, i.e., `test.fuzz\0.test.fuzz.`. Two labels can be merged into a new dot-containing label, i.e., `foo` and `bar` can be merged into `foo\.bar`. These mutations verify that the resolver treats domain names as a sequence of labels and not as a string.

Only a small set of RR type and RDATA is generated. While the RR type is a 16-bit value, most of the values are not assigned, and hence the creation of valid RDATA for them is impossible. Most RR types are for data storage, without interacting with the resolver, and only a few should be interpreted by the resolver. These include `A`/`AAAA` for the IP addresses of AuthNSes and `NS` for delegations between AuthNSes, `SOA` for zone information and caching of negative results, `CNAME`/`DNAME` for aliases/canonical names, and DNSSEC records when supported (`RRSIG`, `DNSKEY`, `DS`, `NSEC`, `NSEC3`, `NSEC3PARAM`). We only generate `A`, `AAAA`, `TXT`, `CNAME`, `NS`, `SRV`, `SOA` and the special query type (QTYPE) `*` often called `ANY`.

### 4.4.7 Fuzzing Output and Processing

We now explain the collected data for each input and how we use it to determine if an input is interesting. The coverage information we gather is edge coverage given from an instrumented resolver. We compile the resolver using LLVMs Sanitizer Coverage Instrumentation [199] and count how often each edge gets executed during the resolution for a single DNS client query.

For each input we capture the outgoing queries from the resolver, the responses sent to the DNS client, the cache state afterwards, how the dynamic DNS servers answered, and the coverage information.

We use all available data to determine if an input is interesting, i.e., produces new or different behavior, but the most important information is the client's response and the coverage information. The client responses reveal whether two resolvers behave significantly differently. If different responses are returned, two clients might behave differently. The coverage information is important to allow for partial progress during fuzzing, by giving a means of identifying new behavior in a resolver, even if the client response remains unchanged. The cache state is important too, but harder to interpret, since a difference here does not necessarily mean a semantic difference. Lastly, the outgoing queries to the dynamic DNS server have a low value, since comparing them is hard and has many downfalls. For example, there are many ways in which a resolver can implement Query Name (QNAME) minimization, such as which query type is used and which labels are removed. For a conforming DNS server, this is not relevant and in the end, the resolver will come up with the same answer in all cases. But these differences are a problem for automatic analysis since we have many semantically equivalent queries with different representations.

The collected data is cleaned and checked for known differences, like random DNS ID values, to only leave the unknown differences. DNS records in messages are unordered, but we normalize the representations by sorting them. We also have further known differences, for example, some resolvers already support extended DNS errors which will be added to the client response in some cases. We create rules for identifying these known differences and track them separately, see Section 4.5.2. The remaining differences are unknown and of the highest interest.

Unfortunately, the fuzzing is not fully deterministic, so we require a validation run for any new differences found. Randomness and unwanted interactions can come from many sources, such as choices the resolver makes, how we batch multiple inputs together, or the kernel via the network and scheduler. We detect non-deterministic differences by fuzzing the same input multiple times and only accepting those results that show the same class of differences each time.

Many differences have an identical root cause and we group them using a fingerprint to better model that. Our fingerprint consists of all the DNS header fields in the DNS client response and all the field names that differ between the two resolvers. For example, we describe the difference in the answer section of the client response using identifiers like `.fuzz_result.fuzzee_response.answers.0` which has the subfields `.name_labels`, `.dns_class`, `.rr_type`, `.ttl`, and `.rdata`. The `.0` refers to the first resource record in the answer section. We do not use the values here, only the field names, since the values often contain randomized data and thus would always lead to different fingerprints. The same problem does not exist for the header values, since these are mostly booleans.

### 4.4.8 Finding Bugs

Our main idea for finding semantic bugs is using differential fuzzing between many resolvers. DNS is a complex protocol with a lot of variability and edge cases. Defining valid behavior in the DNS is difficult as it requires an in-depth understanding of the standards and a lot of domain-specific knowledge. Instead, using differential testing, we can automatically leverage this knowledge as the resolvers are implemented by independent expert groups. Instead of deciding if every response we see is valid, we can focus on a much smaller set of cases where multiple resolvers disagree. Each difference is a case for further manual analysis, as it may indicate implementation or semantic bugs.

During the manual analysis, we also built further rules to describe known differences. For example, BIND 9 has a strict DNS response validation and discards the whole message if a single value is invalid, while Unbound will only discard the single invalid resource record. This causes BIND 9 to produce a ServFail answer while Unbound responds with NoError or NoData depending on the situation. We go into more detail about this in Table 4.2.

The effort for manual analysis and writing rules ranges from a couple of minutes to a few hours. Simple cases, like DNS ID randomization or optional features like extended DNS error, are easy to spot and describe and have no follow-on impacts. Other cases, like the BIND 9 strict validation, are more complex and require more time to understand and describe, as the root cause can lead to non-obvious and large differences. Concretely, BIND 9 might send more upstream queries than other resolvers.

ResolFuzz is guided by the score each input receives. The score is composed of the coverage results per resolver and the differences for each resolver pair, after validation. New differences score the highest, followed by differences we deem “interesting”; coverage is only a small contributing factor. New mutations partially inherit the score of their parent inputs. Finally, if the input population already contains many samples producing the same fingerprint, we apply a penalty to all samples in the population. This is to discourage further mutations based on those samples. This ensures that the input population is diverse and ResolFuzz will not get “stuck” mutating a group of similar inputs with high scores and thus drowning out other interesting inputs.

## 4.5 Evaluation

We evaluate our fuzzer on seven resolvers, as listed in Table 4.1. Where possible we disable features during compile time, to reduce the size of the binary and simplify fuzzing. These features do not affect our evaluation, as these relate to system integration, such as systemd, enhanced security with chroot and capabilities, extra logging like dnstap, or HTTPS support. Most of these features have no impact at all on the DNS protocol and HTTPS for DoH is not used by ResolFuzz as UDP is more efficient for us. Each resolver is compiled from scratch using LLVM with coverage instrumentation [199], which is available in clang and rustc, as both are LLVM-based compilers. The containers are a Fedora 37 image.

**Table 4.1:** Software versions used for the evaluation.

| Software | Version | Language |
|---|---|---|
| BIND 9 | v9.18.0, v9.11.0 | C |
| Deadwood | 3.5.0032 | C |
| Knot Resolver | 5.5.3 | C, Lua |
| PowerDNS Recursor | 4.7.3 | C++ |
| resolved | `463644c` | Rust |
| trust-dns | `0b6fefe` | Rust |
| Unbound | 1.15.0 | C |

### 4.5.1 Case Studies

In this section, we highlight our findings in three case studies.

**resolved fails with query and missing `CNAME`:** resolved misbehaves for `CNAME` queries if no `CNAME` record exists. Instead of returning the expected NoError response code, resolved returns ServFail. The `CNAME` record type redirects queries for a domain to another canonical domain. This means a resolver must follow the redirection, except here, because the original query is for the `CNAME` type. This means the first answer is the final one.

**PowerDNS Recursor self-loop:** We discovered a bug in handling `DNAME` RRs, similar to a known issue for `CNAME`s [36]. The problem occurs if the result after `DNAME` expansion matches the same `DNAME` again. `DNAME` and `CNAME` records both provide a way to specify a new canonical place where information is stored. In case of a `CNAME` RR like `this.old.domain CNAME new.canonical.name`, any query for `this.old .domain` should be redirected to `new.canonical.name` and the data from the new place should be used. A `DNAME` redirection is similar to `CNAME`. A `CNAME` only applies to a single specific domain, but a `DNAME` applies to a whole subtree. For example the `DNAME` RR `old.domain DNAME new.name` means that any query for *`this.`*`old.domain` redirects to *`this.`*`new.name`. The *italic* subdomain part is arbitrary and is preserved in the rewrite. The `DNAME` can point to itself, by having the re-written part match the pattern, for example, `old.domain DNAME` *`extra.`*`old.domain`, which will put *`extra`* many times in the domain. When PowerDNS Recursor encounters such a `DNAME` record, it applies the rewriting repeatedly. Consequently, in our setting, the resulting DNS answer will contain the same `DNAME` record 16 times. Further, each time the `DNAME` rewriting rule is applied, a synthetic `CNAME` record is created. Ultimately, the resolver gives up resolving this infinite loop and returns a ServFail with 32 records. The effect is that PowerDNS Recursor spends time following the loop and creating ever-increasing answers, thus wasting resources. The answer becomes large, making it a target for reflective DDoS attacks [171].

**Self-Sustained DoS in resolved and trust-dns:** We found a vulnerability in resolved and trust-dns that enables a self-sustaining DoS attack. The vulnerability is caused by these resolvers answering DNS responses (QR bit=1). When receiving responses on their listening port (UDP/53), both resolved and trust-dns return a FormErr error to indicate that the request was malformed.

Blindly responding to responses, even with error messages, can be abused to create a traffic loop. If both resolvers are vulnerable, attackers can inject IP-spoofed responses to provoke the two resolvers to send responses to each other in an endless loop, causing a self-sustaining DoS attack. The traffic loop will not stop except when packet loss in the network causes the traffic to stop. Enough looping traffic will overwhelm the resolver or network and cause a DoS.

### 4.5.2 Result Statistics

The described case studies were found over multiple runs. We now describe how a *single* run of the fuzzer behaves and the kind of differences it finds.

The results of this section refer to a six-hour run of the fuzzer using all eight configurations listed in Table 4.1. During that time, 7140 inputs were generated or roughly 0.33/s. In total, the fuzzer made 198 240 comparisons, as each input takes part in 28 comparisons. This resulted in 1903 distinct fingerprints. The fingerprint count is larger than the number of root causes. For example, to trigger the resolved and trust-dns bug the only prerequisite is a query with QR=1 bit set, but a fingerprint includes all header fields.

**Table 4.2:** Higher level categories of the common differences we found between resolver outputs. Each category lists the number of rules that fall into it and gives an example.

| Category | Size | Description | Example |
|---|---|---|---|
| Configuration | 6 | Configurable behavior | Limiting the maximal TTL values, EDNS buffer sizes |
| Error Handling | 9 | Behavioral differences in error messages (ServFail, NotImp, FormErr, Refused) | On receiving a query with TC bit set, ServFail and FormErr responses exist. |
| Incomparable | 3 | Values that always differ | Random DNS ID. |
| Metadata | 3 | Metadata about the captured results | Resolver Name, redundant Length Values |
| Missing Features | 3 | Optional missing features | EDNS Error Codes |
| Resolver Specific | 10 | Other uncategorized, but resolver specific behavior | EDNS buffer size (512B in PowerDNS vs. 1232B in others). |
| Upstream Queries | 4 | Behavior of the upstream queries | QNAME Minimization, re-transmissions |

We created 38 rules to describe benign differences, as shown in Table 4.2. The rules are grouped into seven categories. The nine *Error Handling* rules capture the variability of error messages (e.g., differences in error types). *Resolver-specific* behavior is similarly sized with ten rules, but the behavior covered here is more diverse. Two categories of differences will be found for any comparison of two outputs, *Metadata* and *Incomparable*. There are some fixed values about the captured results that are always different, such as the resolver names. In each comparison, the DNS ID and edge coverage information are incomparable.

The 38 rules cover between 2017 to 6555 (28.2 % to 91.8 %) of the differences identified by the fuzzer, fluctuating based on the compared resolvers. Most similar is the pair of BIND 9 (v9.18) and PowerDNS Recursor. Both are mature implementations and adhere to the DNS standard well. The worst pairing is Knot Resolver with trust-dns. The trust-dns recursor is quite new and not yet fully developed. As such it still has many missing features and bugs, all resulting in differences. These numbers translate directly into differences not covered by our rules. The pairing of Knot Resolver and trust-dns leads with 5067 (71.0 %) uncovered differences, while BIND 9 and PowerDNS Recursor have only 490 (6.9 %). The gap towards the total of 7140 is caused by inputs where the comparison was non-deterministic. We observed the highest rate here between both BIND 9 configurations with 143 (2.0 %) non-deterministic comparisons.

**Fuzzing progress:** We furthermore evaluate how much coverage our fuzzer achieves in the resolvers' code. This way we can understand when fuzzing saturates, i.e., no longer reveals new results. The coverage percentage increases sharply at the beginning, but then slows down but never stops. No resolver reaches a high edge coverage, with the highest being around 16 %. This can be explained by the fact that we only measure the coverage between sending a query and receiving a response. Any startup, background, or shutdown code is not included in the coverage. While we aimed to remove as many untested features as possible during compilation time, many features are still enabled but never activated by our fuzzer, such as DNSSEC, DNS forwarding or authoritative mode, and even complex features like Response Policy Zones (RPZs).

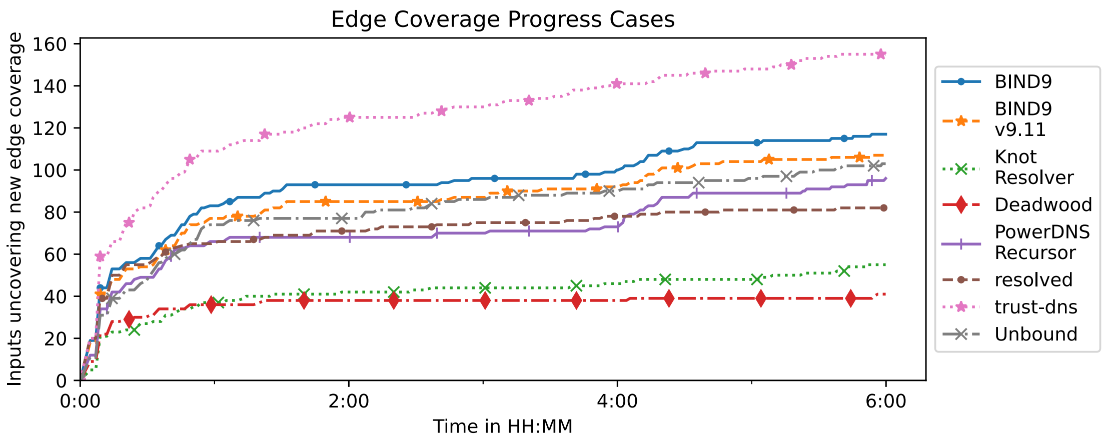


**Figure 4.3:** The number of times new coverage edges are found over time.

Figure 4.3 shows how often new edges are found over time. It shows more clearly, that during the entire period, progress is made. The main part falls into the first two hours, which is longer than the previous picture suggests. The continuous progress is a good indicator showing that even after a longer time ResolFuzz still makes progress and we have not yet reached the limits of it. Similarly, Fig. 4.4 shows the found differences grouped by category and shown for each resolver pair.

### 4.5.3 Bug and Vulnerability Disclosure

We reported all findings to the respective projects, except for one, which had no contact information. All projects acknowledged our findings. Most quickly fixed the issues, even releasing a security advisory RUSTSEC-2023-0041 [173], except for PowerDNS Recursor which deemed the risk as acceptable.

## 4.6 Limitations

We now discuss limitations that arise out of the design decisions we took to address the complex challenges of fuzzing DNS resolvers.

Foremost, our fuzzer has no notion of time. Indeed, faking time jumps requires knowledge of the resolver internals, as these determine when time jumps can be inserted

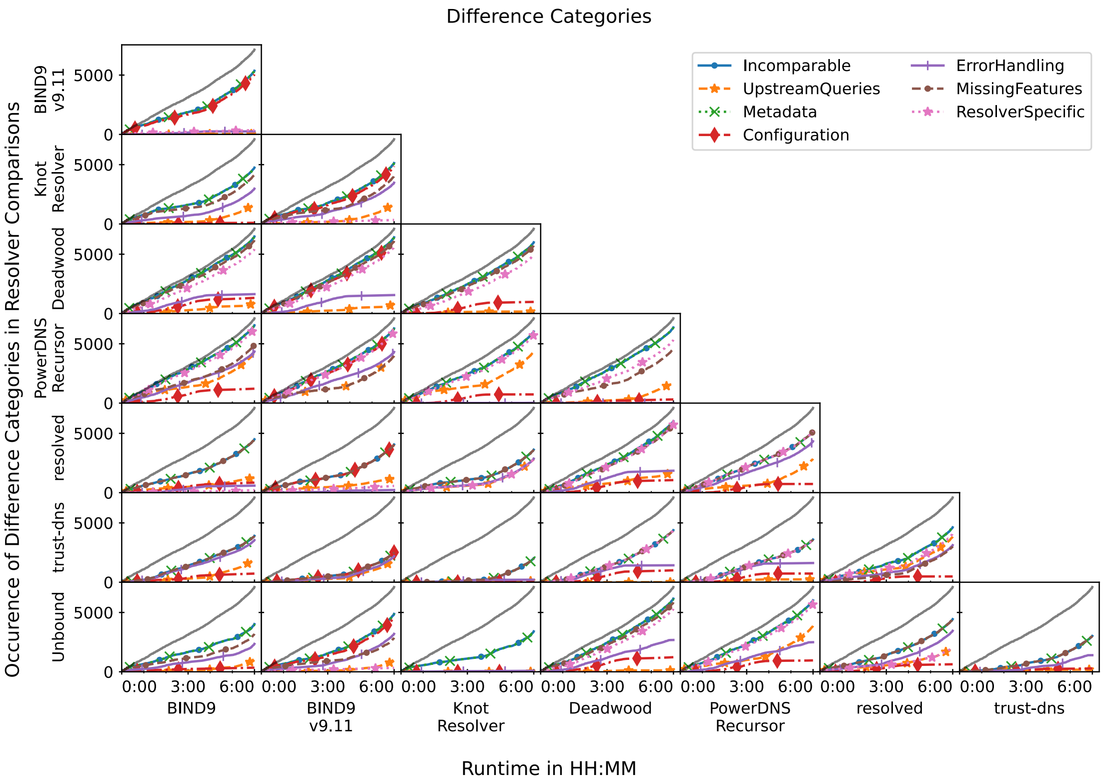


**Figure 4.4:** The number of explainable differences between the resolver outputs. Only explainable differences are shown. The gray line shows the total number of fuzzing inputs tested. The gap between the gray line and the colored lines shows the number of unexplainable differences. The categories are explained in Table 4.2.

and which parts they affect—internals we wanted to abstract from. The Deckard [42] testing framework managed to solve some of the problems and could be an inspiration for future work.

We use cache snooping as a generic mechanism to learn about the resolver's cache state. In some cases this fails, e.g., for specific query classes or resolvers, such that we skip the records during diffing. Resolver-specific ways to retrieve the cache status (e.g., via CLI tools or log files) would require resolver adjustments.

Not all DNS features are implemented, such as DNSSEC [81, 167, 168, 169, 214]. Cryptographic operations are challenging to fuzz, but represent interesting attack targets for malicious actors. Some part-way solutions are possible, like using a single validity bit and then dynamically signing the records, but we leave this for future work. Likewise, we ignored RPZ that can be used to block certain domains or IP addresses. They work by sending further queries and then changing or blocking the original request.

We use a minimal configuration for each resolver, leaving most options as their upstream default. Configurations we changed include the IP address to listen, the root server hints, and the reduction of timeouts to speed up fuzzing. We deem this configuration realistic and modification necessary for fuzzing. We publish all our source code including configurations.

Recursive DNS resolution is the most complex part between clients, stub resolvers, and AuthNS. Stub resolvers could be tested similarly to the recursive resolvers, except they only forward the query to the recursive resolver.

## 4.7 Related Work

Automatic testing of network services has been subject to several other related works [6, 47, 85, 97, 120, 133, 150, 159, 174, 194, 209, 221]. However, we are the first to create an advanced fuzzer for DNS resolvers. Indeed, DNS is a particularly challenging setting, as the resolver (i.e., the network service) acts as server and client at the same time—but most existing fuzzing frameworks test only single connections. Other projects solve DNS resolver fuzzing only partially. Related work can be grouped into two main categories: DNS testing and evaluation tools, and network fuzzers. We separately cover formal models for AuthNSes.

**Table 4.3:** Comparison of existing DNS testing tools, with a focus on fuzzing. The *Client* and *Server* columns indicate whether the project ships with a client or server component. *Search Strategy* indicates the strategy used for generating queries. Randomized means that all modifications are chosen by a random number generator, without any feedback. *Targets* indicates what the tool is looking for.

| Project | Cli. | Srv. | Search Strategy | Targets |
|---|---|---|---|---|
| ResolFuzz | yes | yes | Evolutionary mutations guided by code coverage and differential testing | Crashes and differential testing |
| Deckard [42] | yes | yes | None. Scripted communication. | Configurable checking. Comparison to fixed values. |
| dns-fuzz-server [174] | yes | yes | Randomized | None |
| dns-fuzzer [133] | yes | no | Randomized | Crashes |
| honggfuzz [194] | yes | no | Evolutionary, Code coverage feedback | Crashes |
| IBDNS [209] | no | yes | Fixed, based on query | None. Not a full tool. |
| nmap dns-fuzz [47] | yes | no | Randomized | Crashes |
| SCALE/Ferret [97] | yes | no | Statespace guided input generation | Diff. comparison between servers and formal model. |

**DNS testing:** Due to the complexities in DNS and the difficulty of covering a large input space, only few projects exist for testing or fuzzing DNS. Table 4.3 provides an overview of existing projects and their capabilities. Existing tools often only target crashes in the DNS resolver [47, 133, 194]. This makes them unsuitable for finding more complex failure conditions, which we can identify with the differential testing approach. Some projects use basic randomization for input generation. dns-fuzz-server [174] only creates random queries and responses, but has no target conditions, like crashes, it is looking for. ResolFuzz uses a more advanced evolutionary algorithm, which is better suited for complex inputs.

The Deckard [42] project is notable in that it is a full testing framework for DNS resolvers. It has extensive customization options, for query generations, mocking AuthNSes, and checking the responses. This extensiveness is great for writing detailed tests, but they often rely on the resolver implementation and are not portable between different resolvers. Relying on scripted tests means unknown behavior cannot be uncovered.

The Intentionally Broken DNS Server (IBDNS) [209] is a new project by Afnic for testing DNS resolvers or DNS tools. It applies known defects to existing zone files and serves them to clients. Its goal is to test DNS tools and DNS resolvers. IBDNS is not public so we cannot describe it in detail.

Other DNS test frameworks are either old and unmaintained [190], only test against a reference implementation [63], or are commercial with no public information [196]. Testing against a reference output is useful for limited features or regression testing, but is not viable for covering larger input spaces, due to missing reference implementations.

**Fuzzing AuthNSes:** Most relevant to ResolFuzz are the projects by Kakarla et al. [96, 97]. They tackle the challenge of fuzzing AuthNSes in the two papers GRoot [96] and SCALE [97]. In GRoot they lay the foundations by creating a formal model for the semantics of authoritative DNS. The model creates equivalence classes (EC) for a given DNS zone file. Each EC captures distinct behavior, like the difference between two different existing labels, and combines variants like queries that match the same wildcard record. With this model, they can symbolically execute these ECs and find bugs in the zones, like lame delegations, if the sub-zone has no reachable nameserver, or rewriting loops when `CNAME`/`DNAME` records form a loop leading to unresolvable names. SCALE builds an executable version of the GRoot model, with the ability to create zone files and matching queries. Using symbolic execution they find a wide variety of behaviors and create matching test cases using a constraint solver. That is an expensive process, so the created zone files are limited to four records. The test cases are fed into various AuthNSes and the responses are checked for compliance with the RFCs. The AuthNSes only agree in 35 % on the same answer. The rest is grouped by fingerprints, to make investigations easier. Using SCALE they could identify 30 new bugs.

**Network service fuzzing:** Apart from DNS with its special requirements, there are other network fuzzing approaches, which come in many variants.

AFLnet [150] is a greybox fuzzer. It has a corpus of network exchanges, which it mutates and sends to the target. AFLnet is guided by code coverage and learns a state machine for the target server. Fuzzing uses a forkserver, which allows for fast restarts and parallelization. SnapFuzz [6] is an iteration of AFLnet with increased performance, achieved with new binary rewriting. The rewriting replaces file system accesses with a custom in-memory implementation, replaces the TCP and UDP socket calls with UNIX domain sockets, and optimizes the forkserver. Both AFLnet and SnapFuzz are designed to fuzz single client-server connections, which makes them unsuitable for DNS resolvers.

Lin et al. [120] use GANs to infer the protocol of a black box network service. The GAN learns to generate attack packets for a protocol, not only the protocol syntax. It is only tested on stateless protocols, which makes it unsuitable for DNS.

Hoque et al. [85] use a model checker to find temporal semantic bugs in protocols. From a protocol implementation, they extract a finite state machine describing it. The temporal properties are protocol-specific and require expert knowledge, unlike ResolFuzz.

Differential fuzzing has been used successfully in contexts other than DNS. TCP-Fuzz [221] by Zou et al. uses differential checking as one bug detection method. They develop an input creation strategy that keeps track of dependencies between packets and system calls. The coverage metric is based on the state transitions of the TCP stack. DPIFuzz [159] by Reen and Rossow uses differential fuzzing for QUIC. It generates QUIC frames and mutates them with shuffling, duplication, and deleting data. While both works are related due to their differential checking approach, they are not directly comparable. TCP and QUIC work point-to-point and on a well-defined sequence of packets. A DNS resolver has many point-to-point connections and there is no clear sequence of packets, as each point-to-point connection runs independently.

## 4.8 Conclusion

Our analyses encourage more research on securing DNS resolvers, a critical part of the Internet infrastructure. With ResolFuzz, our differential fuzzer, we found multiple security-relevant bugs in both well-established and new resolver implementations. DNS' lack of a formal model makes semantic analysis hard, but we showed that differential fuzzing with specialized matching rules can be a powerful tool for finding bugs. We publish ResolFuzz as open-source in the hope that it will help to improve the security of DNS resolvers. This work provides a starting point for further research into the security of DNS resolvers, and we hope to uncover more bugs through improved insights into resolver decisions and covering more complex deployment scenarios.

# 5 DNS Unchained: Amplified Application-Layer DoS Attacks Against DNS Authoritatives

## 5.1 Motivation

The next step in our DNS landscape journey covers the interaction between recursive DNS resolvers Ⓡ/Ⓕ/Ⓟ and AuthNSes Ⓐ from the DNS overview in Fig. 2.1 and their network communication ③. Of special interest is the implementation of recursive resolvers Ⓡ and how this enables traffic amplification attacks. We inspect hosts on the Internet and measure the behavior of many forwarders Ⓕ and public resolvers Ⓟ.

We present *DNS Unchained*, a new application-layer DoS attack against core DNS infrastructure that for the first time uses amplification. To achieve an attack amplification of 8.51, we carefully chain `CNAME` records and force resolvers to perform deep name resolutions—effectively overloading a target AuthNS with valid requests. We identify 178 508 potential amplifiers, of which 74.3 % can be abused in such an attack due to the way they cache records with low Time-to-Live values. In essence, this allows a single modern consumer uplink to downgrade the availability of large DNS setups. To tackle this new threat, we conclude with an overview of countermeasures and suggestions for DNS servers to limit the impact of DNS chaining attacks.

## 5.2 Problem Description

The DNS is at the core of today's Internet and is inevitable for networked applications nowadays. It is the primary solution for mapping and translating domain names to IP addresses. Moreover, several other applications heavily rely on it, such as load balancing (e.g., for Content Delivery Networks), anti-spam methods (e.g., DKIM [108], SPF [99], or IP address blocklists [48]) and TLS certificate pinning [52, 83]. We rely on the availability of these services for everyday communication. Yet recent incidents have demonstrated how vulnerable DNS is to DoS attacks, even for hosters that massively invest in over-provisioning and deploy highly reliable anycast networks. For example, in October 2016, attacks against the DNS hoster Dyn have knocked Twitter, Netflix, PayPal, and Spotify offline for several hours [79]—simply because the AuthNSes for these services were hosted by Dyn and became unresponsive due to a successful DDoS attack against Dyn.

Up to now, DDoS attempts against the DNS infrastructure have focused mostly on volumetric attacks, where attackers aim to exhaust the bandwidth that is available to DNS hosters. In a successful attack, benign DNS queries are dropped such that normal users no longer see responses from the DNS hosters. A popular and powerful example of volumetric attacks are so-called amplification attacks [105, 171], where miscreants abuse the fact that open services (such as NTP servers) reflect answers to IP-spoofed requests. Yet, any of these rather simple volumetric attacks can be filtered with the help of data scrubbing services such as Arbor, Cloudflare, or Incapsula.

## 5.3 Contribution

In this chapter, we explore *application-layer* attacks against core DNS infrastructures, namely AuthNSes. Compared to volumetric DoS attacks, application-layer attacks are more appealing to adversaries. In particular, they (i) are significantly harder to distinguish from benign traffic, (ii) not only target bandwidth, but also computational resources, and (iii) do not rely on IP address spoofing and can be launched even though providers deploy egress filtering [178]. This makes them attractive for botnets.

We start by describing existing forms of application-layer attack against DNS that overload a target AuthNS with valid DNS requests. In the simplest form, a single attack source can send queries to domains hosted by this name server. Yet in practice, attackers have distributed the attack and use resolvers as intermediaries in so-called *random prefix attacks* [32, 197]. They are a form of flooding DNS attacks and get their name from the characteristic prefixes used to circumvent resolver caching. Such attacks can be launched from malware-infected devices [7] or even JavaScript and already have the potential to put large DNS hosters offline (e.g., Dyn in 2016).

We then describe a novel form of application-layer attacks that floods the victim with an order of magnitude more queries per second than random prefix attacks. We dub this attack DNS Unchained, as it abuses the chaining behavior of `CNAME` and `DNAME` resource records in DNS. The core idea of our attack borrows from random prefix attacks. However, instead of blindly sending out queries to random domains hosted by the target AuthNS, the attacker carefully crafts long chains of DNS records (`a.target.com`→`b.other.com`, `b.other.com`→`c.target.com`, `...`) that involve the target AuthNS in every other step. This has the effect that resolvers query the target AuthNS not just once, but *several* times—until the end of the chain is reached. To the best of our knowledge, this is the first DoS attack that combines amplification with application-layer attacks. We find that the vast majority of resolvers support chain lengths of 9–27 (and more) elements, resulting in tenfold amplification due to the number of times a target AuthNS is queried per request the attacker sends.

We complete this chapter with an extensive discussion of how such attacks can be remedied. We foresee countermeasures that can be deployed by AuthNS, such as detecting malicious DNS chains or enforcing lower bounds, ensuring more caching, for TTL values.

Our contributions can be summarized as follows:

- We present an application-layer attack against DNS that creates an order of magnitude more queries per second than existing attacks. For this attack, we revisit how DNS chains can be abused to amplify traffic, and are the first to combine application-layer attacks with amplification.
- We analyze the real-world impact by performing Internet-wide measurements of the resolver landscape and test for achievable amplification.
- We present and discuss the efficacy of countermeasures against application-layer DoS attacks. This discussion helps to defend against DNS Unchained and DNS application-layer attacks in general.

## 5.4 Threat Model

We now define the threat model and describe the attacker's capabilities, required resources, and our assumptions about the victim. The adversary in our model aims to degrade the availability of an AuthNS. AuthNSes answer DNS queries for a particular zone, as queried by any DNS resolver. Next to mapping domain names to IP addresses, AuthNSes provide several other services, e.g., anti-spam methods (e.g., DKIM [108], SPF [99], or IP address blocklists [48]) and TLS certificate pinning [52, 83], making them fundamental on the Internet.

In the highly redundant DNS setting, resolvers choose between all AuthNSes of a particular zone [140, 220]. Yet, even a single unresponsive AuthNS will cause decreased performance for the whole domain within the zone. In a redundant setup with multiple anycast sites, the loss of one anycast site will still affect the network's routing to this site, therefore the responsiveness of every single AuthNS matters.

In our model, the attacker targets a specific AuthNS, e.g., to render domains hosted by this AuthNS unreachable. We assume that the attacker can host at least one attacker-controlled zone on the target AuthNS. This involves that the attacker can create arbitrary DNS records that are within their zone, i.e., subdomains for a given second-level domain. We believe that this assumption is easily fulfilled. For example, if domains are hosted by web hosters such as GoDaddy or Rackspace, an attacker can set up a domain at the same hoster as the victim's website. Another possibility is that the victim's domain is hosted using one of the DNS hosters like NS1, Amazon Route 53, Dyn, or Google Cloud DNS.

Creating an account may be a problem for an attacker who wants to stay anonymous. We note that in such cases, the attacker could use fake or stolen IDs to register accounts.

The only other requirement for the attacker is the ability to send DNS queries to open DNS resolvers ("resolvers" hereafter). Attackers can find such resolvers by scanning the Internet UDP port 53 in less than an hour [53]. Internet scans are not a limiting factor, since lists of resolvers are available for download [22]. In contrast to amplification DDoS attacks [171], the attacker in our model does not need to spoof the source IP address of attack traffic. This allows an attacker to operate from a single source, or to increase anonymity and bandwidth by leveraging DDoS botnets to launch attacks.

## 5.5 Application-Layer DDoS Against DNS

Application-layer DoS attacks abuse a higher-level protocol—in our context DNS—and tie resources of other participants of the same protocol. This distinguishes application-level attacks from other forms of DoS attacks, e.g., volumetric attacks, which are agnostic to protocol and application, but relatively easy to filter and defend against. Since application-layer attacks can target more different resources, like CPU time and upstream bandwidth, they are more interesting than volumetric attacks, which can only consume downstream bandwidth. In this section, we will first introduce DNS water torture attacks, an emerging application-layer DoS technique that has already severely threatened the DNS infrastructure. We will then show that a smart attacker can craft delicate chains of DNS records to leverage resolvers for even more powerful attacks than those possible with DNS water torture.

### 5.5.1 DNS Water Torture

DNS water torture attacks—also known as random prefix attacks—flood the victim's DNS servers with requests such that the server runs out of resources to respond to benign queries. Such attacks typically target the AuthNS hosting the victim's domain, such that domains hosted at the target server become unreachable. Resolvers would typically cache the responses of the queried domains, and therefore mitigate naive floods in that they refrain from identical follow-up queries. To this end, attackers evade caching by using unique domain names for each query, forcing resolvers to forward all queries to the target AuthNS. A common way is prepending a unique sequence to the domain—the random prefix. In practice, attackers either use monotonically increasing counters, hash this counter, or use a dictionary to create prefixes. As the DNS infrastructure, on the other hand, heavily relies on caching on multiple layers in the DNS hierarchy, AuthNS are typically not provisioned to withstand many unique and thus non-cached requests—leaving AuthNS vulnerable to water torture attacks.

Water torture attacks were observed for the first time in early 2014 [32, 164, 216] and have since been launched repeatedly. The main ingredient for this attack is sufficient attack bandwidth, which overloads the target AuthNS with "too many" requests. As this does not require IP spoofing, attackers can easily facilitate botnets to maximize their attack bandwidth. Moreover, several large DDoS botnets (e.g., Mirai [7] or Elknot [121]) support DNS water torture.

While water torture attacks have been fairly effective, their naive concept has noticeable limitations:

(i) Water torture attacks can usually be easily detected because the attack traffic shows exceptionally high failure rates for particular domains, as none of the requested (random-looking) domain names actually exist. NXDOMAIN responses are normally caused by configuration errors and therefore often monitored. (ii) Water torture attacks provide no amplification, as every query by the attacker eventually results in only a single query to the target AuthNS—unless queries are resent in case of packet loss. The victim-facing attack traffic is thus bound by the number of queries that the attacker can send. This is in stark contrast to volumetric attacks that offer more than tenfold amplification [171].

### 5.5.2 Chaining-based DNS DoS Attack

We now propose a novel type of DNS application-layer attacks that abuse chains in DNS to overcome the aforementioned limitations of water torture, yet stay in a similar threat model (Section 5.4). The main intuition of our attack is that an attacker can utilize request chains that amplify the attack volume towards a target AuthNS. This is achieved via aliases, i.e., a popular feature defined in the DNS specification and frequently used in practice.

#### 5.5.2.1 CNAME Records

DNS request chains exist due to the functionality of creating aliases in DNS, e.g., using standard CNAME Resource Record (RR) [135, 136]. A CNAME RR, short for *canonical* *name*, works similarly to pointers in programming languages. Instead of providing the desired data for a resolver, `CNAME` specifies a different DNS location from where to request the RR. One common use is to share the same RRs for a domain and the `www` subdomain. In this case, a `CNAME` entry for `www.example.com.` points to `example.com.`. When a client asks the resolver for the RRs of a certain type and domain, the resolver recursively queries the AuthNS for the RRs, resulting in three cases to consider:

**Domain does not exist or no data** The domain does not exist (NXDOMAIN status) or no matching resource record (including `CNAME` records) was found (NODATA status). The AuthNS returns this status.

**Resource records exists** The desired resource record's data is immediately returned by the AuthNS. The DNS specification enforces that *either* data, *or* an alias (i.e., `CNAME`) may exist for a domain, but never both—i.e., there was no `CNAME` record for the requested domain.

**Domain exists and contains `CNAME` response** The resolver must follow the `CNAME` regardless of the requested record type. This may cause the resolver to send new queries, potentially even to different AuthNSes.

The last case allows the chaining of several requests. In the case of `CNAME` records, resolvers have to perform multiple lookups to load the data (unless the records are cached). `CNAME` records can also be chained, meaning the target of a `CNAME` record points to another `CNAME` record. This increases the number of lookups per initial query. There is no strict limit to the length of chains. However, resolvers typically enforce a limit to prevent loops of `CNAME` records. After reaching this limit, resolvers either provide a partial answer or respond with an error message.

`CNAME` records provide delegation between arbitrary domains, i.e., also to domains in unrelated zones. If all the `CNAME` records are hosted in the same zone, the AuthNS can provide multiple `CNAME`s in one answer, by already providing the next records in the chain. By chaining `CNAME` records between two different AuthNSes, i.e., by alternating between them, an AuthNS can only know the next `CNAME` entry in the chain.

#### 5.5.2.2 DNS Chaining Attack

The possibility to chain DNS queries via `CNAME` RRs opens a new form of application-layer DoS attack. Let an attacker set up two domains on different AuthNSes. The first domain will be hosted by the target AuthNS, and the second (or optionally further) domain(s) by some *intermediary* AuthNS(s). The zones are configured to contain long `CNAME` chains alternating between both domains. An example can be found in Listing 5.1, where a chain ping-pongs between the target and an intermediary AuthNS, until the record with prefix $i$. If an attacker now sends a single name lookup to query for the record at the start of the chain, the resolver has to follow all chain elements to retrieve the final RR. A large final RR, such as the `TXT`, additionally targets the AuthNS's upstream bandwidth. Figure 5.1 shows the queries sent between the attacker $A$, a resolver $R$, and both AuthNSes. The dashed arrows represent the `CNAME` pointers between the different

```
a.target-authns.com.   IN CNAME b.intermediary.org.
c.target-authns.com.   IN CNAME d.intermediary.org.
e.target-authns.com.   IN CNAME f.intermediary.org.
g.target-authns.com.   IN CNAME h.intermediary.org.
i.target-authns.com.   IN TXT "Huge record at the end."

b.intermediary.org. IN CNAME c.target-authns.com.
d.intermediary.org. IN CNAME e.target-authns.com.
f.intermediary.org. IN CNAME h.target-authns.com.
h.intermediary.org. IN CNAME i.target-authns.com.
```

**Listing 5.1:** Two zones `target-authns.com.` and `intermediary.org.`, which contain a `CNAME` that ends at the $i$th element in `TXT` records.

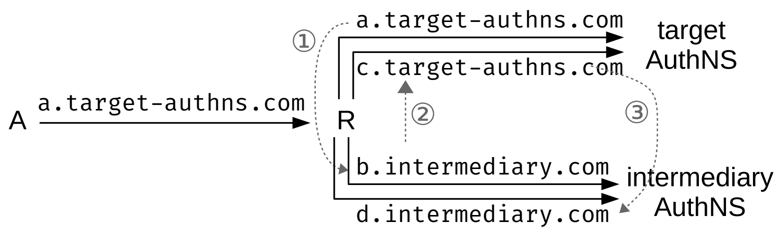


**Figure 5.1:** Attacker $A$ uses resolver $R$ to attack the target AuthNS. The dashed arrows represent the `CNAME` pointers between the domain. ①–③ show the order of `CNAME` records in the chain. The setup is according to Listing 5.1.

domains, while the circled numbers (①—③) represent the order in which they are resolved. The attacker queries the first chain element and forces the resolver to query the target AuthNS repeatedly.

This provides severe amplification, as a single request by the attacker results in several requests towards the target AuthNS. For each query by the attacker $N$ queries are sent by the resolver, where $N$ is equal to the minimum of the chain length and a resolver-dependent limit. The chain length is controllable by the attacker and effectively unlimited, but resolver implementations limit the maximum recursion depth (see Section 5.6.2). The *amplification*, as observed by the target AuthNS, is $\lceil N/2 \rceil$, as every second chain record is served by the target AuthNS.

For illustrative purposes, Fig. 5.1 just shows a single resolver. In practice, an attacker would likely aim to spread the attack requests to thousands of resolvers, that is, not to overload a single resolver—recall that in our threat model, the AuthNS is the victim (not the resolver). Furthermore, given two domains, an attacker can easily create multiple chains, e.g., by using distinct subdomains for each chain. The number of chains is bound (if at all) only by the number of subdomains supported by the target AuthNS.

There are no strict requirements for the intermediary AuthNS. In general, intermediary AuthNSes can be hosted by a hosting provider, self-hosted by the attacker, or even distributed between multiple hosters. The only exception is that the intermediary and target AuthNS should not be the same server. Some AuthNSes will follow `CNAME` chains if the AuthNS is authoritative for all domains in the chain. Requiring at least one dedicated intermediary AuthNSes ensures that only one answer can be returned. If the AuthNS is configured to only return one `CNAME` record, the same AuthNS can be used, doubling the amplification achieved with this attack. On the other extreme, it is perfectly possible to use *multiple* intermediate AuthNS, as long as every second element in the chain still points to the target AuthNS. Distributing the intermediary AuthNS will increase the reliability and reduce the load for each intermediary AuthNS, and raise the complexity of preventing the attack.

While the requirements for this attack may seem high, we note that attackers are already known to use complex setups for their operations. One example regarding DNS is *fast-flux networks* [84] which provide resilience against law-enforcement take-downs and work similarly to CDNs. Attackers use fast-changing DNS entries to distribute traffic across sometimes hundreds of machines.

### 5.5.3 Leveraging DNS Caching

DNS resolvers rely on record caching, such that queries for the same domain do not require additional recursive resolution if the resolver has those records cached. Technically, each resource record contains a TTL value, which specifies how long it may be cached by a resolver, i.e., be answered without querying the AuthNS. Caching has a large influence on the DNS chaining attack, as it determines how frequently resolvers will query target and intermediary AuthNSes.

An attacker would aim for two compatible goals. On the one hand, given an attack timespan, the target AuthNS should receive as many queries as possible. This means that caching for those records that are delivered by the target AuthNS should be ideally avoided. On the other hand, an attacker wants to minimize the number of queries sent to the intermediary AuthNS, as they would otherwise slow down the overall attack. We discuss both parts individually in the following.

**Avoiding Caching at Target AuthNS:** Determining the overall impact on AuthNS requires an understanding of how often each resolver can be used by the attacker during an attack. That is, if all records of a chain are cached, the resolver would not query the target AuthNS. To solve this problem, attackers can disable caching for records hosted by the target AuthNS. Specifying a TTL value of zero indicates that the resource should never be cached [136, Sect. 3.2.1]. We assume that resolvers honor a TTL of zero, i.e., do not cache such entries. We evaluate this assumption in Section 5.6.1.

However, we have observed that resolvers implement additional *micro-caching* strategies to further reduce the number of outgoing queries. A strategy we have typically observed is that resolvers coalesce multiple identical incoming or outgoing requests. If a resolver detects that a given RR is not in the cache, it starts requesting the data from the AuthNS. Queries by other clients for the same RR may arrive simultaneously. A micro-caching resolver can answer all outstanding client queries at once when the authoritative answer arrives, even if the RR would not normally be cached (i.e., TTL=0). In our context, such micro-caching might occur if the resolver receives a query for a `CNAME` record of which the target is not cached, but another query for the same target is already outstanding. Coalescing identical queries thus results in fewer outgoing queries to the AuthNSes because a single authoritative reply is used to answer multiple client queries. This reduces the amplification caused by the resolver. Micro-caching is a defense mechanism against cache poisoning attacks that make use of the "birthday attack", such as the Kaminsky attack [35, 98].

We thus define the *per-resolver query frequency* as the maximum number of queries per second an attacker can send to a given resolver *without* any query being answered by caching or micro-caching. It corresponds to the optimal attack speed: fewer queries would not use the resolver's full amplification potential, more queries would waste some of the attackers available bandwidth.

**Leveraging Caching at Intermediary AuthNSes:** Recall that every other chain element points to a record hosted by an intermediary AuthNS. In principle, this would require resolvers to query the intermediary AuthNS for every second step in the chain, which significantly reduces the frequency in which the target AuthNS receives queries. However, those records do not change, so we can leverage caching to increase this frequency. By setting a non-zero TTL for the records hosted by the intermediary AuthNSes, the resolvers only have to fetch the records on the first query of the chain. After the caches are "warmed up", the resolvers will only fetch the records from the target AuthNS. The frequency of attack queries is thus largely determined by the RTT between resolver and target AuthNS. In contrast, the RTT between resolver and intermediary AuthNS is irrelevant.

### 5.5.4 Attack Variant with `DNAME` Resource Records

One drawback of the `CNAME`-based attack is that it requires definitions of records *per chain*. If an attacker aims to abuse multiple chains in parallel (e.g., to increase the *per-resolver query frequency*), they have to define dozens of `CNAME` records. One slight variation of the `CNAME`-based attack thus uses `DNAME` records. Using `DNAME` resource records [33, 170] allows arbitrary many subdomains for the chain with only a single entry. Conceptually, `DNAME`s are similar to `CNAME`s and are created like `CNAME` records, e.g., "`www.target-ans.com.` IN DNAME `intermediary.org.`". The difference is that `DNAME` records allow the AuthNS to replace the occurrence of the owner (left-hand side) by the target (right-hand side) for all queries to a subdomain of the owner. For example, a query to `a.www.target-ans.com.` would be rewritten to `a.intermediary.org.` with the given rule.

Technically, the answer for a `DNAME` resource record does not only contain the `DNAME` resource records. For backward compatibility, AuthNSes will create a synthetic `CNAME` resource record for the exact query domain. Resolvers can also directly support `DNAME` resource records, providing a better user experience. However, resolvers that lack support for `DNAME` records fall back to using the `CNAME` records. An attacker can abuse those resolvers to query chains defined with `DNAME` entries, for simulating an arbitrary number of chains and avoid caching. Those resolvers have to use the synthetic `CNAME` records to follow the chain. Because the records are synthetically created for the exact query domain, they are indistinguishable from "normal" `CNAME` records in a zone. This forces the resolver to query the AuthNS for each newly observed subdomain.

Resolvers that support `DNAME`s can use a cached entry to directly answer queries for all subdomains, even if the exact subdomain has never been observed. This improves the resolver's performance, as only one cache entry has to be stored (compared to many CNAMEs) and authoritative queries only need to be issued if the DNAME entry expires (compared to once for each new subdomain). This effectively limits the number of simulated chains to one, which falls back to the same properties as the classic CNAME-based chain. Resolvers without DNAME support can be queried as often as permitted by the resolver's resources, without regard to any macro- or micro-caching. Moreover, handling DNAME queries consumes more resources at the target AuthNS, as resolvers usually create and send synthetic CNAME records in addition to DNAME records.

## 5.6 Evaluation

In the following, we analyze the behavior of resolvers, with Internet-wide measurements, and analyze four selected implementations in more detail. We will use those measurements to determine the per-resolver query frequency, possible amplification factor, and overall impact, focusing only on the CNAME variant.

Our manual analysis covers the four resolvers BIND 9.10.5 [11], Unbound 1.6.3 [205], PowerDNS Recursor 4.0.6 [152], and Knot Resolver 1.3.2 [103] because they are popular, open source, actively maintained, and backed by DNS operators. All tests were performed in the default configuration, as provided by Fedora 25. For the measurements, we set up two Virtual Machines (VMs). The first VM hosts the four resolvers, while the second VM hosts an AuthNS. We configured the resolvers to use the AuthNS for all queries, by setting corresponding root hints and configuring the AuthNS accordingly. Note that in this minimal setup, the second VM hosts both the target and intermediary AuthNS. We thus changed BIND 9's configuration such that it does not follow CNAME chains[1], to simulate two independent AuthNSes.

We scanned the Internet via Zmap [53] and a custom DNS module, following their recommended scanning guidelines. Networks could opt out of our scans. We encoded the IP address of the scan target into each DNS query, which allows us to correlate the scanned IP address with the traffic captured at our AuthNS. We used PowerDNS with a custom back-end as the AuthNS for the domains we scanned for. PowerDNS will never follow CNAME chains and only return a single CNAME record, simulating the setup with two zones.

### 5.6.1 Caching

So far, we assumed that resolvers honor non-cacheable DNS resource records (i.e., TTL=0). We evaluate this assumption and study the micro-caching strategies by different DNS resolver implementations.

First, we want to get a general understanding of how the different implementations handle non-cacheable responses. We configured our AuthNS to serve a short CNAME chain alternating between two zones. All RRs in the chain are served with TTL=0. We repeatedly issued the same query to the resolver and observed the responses. BIND 9,

[1]Config option `additional-from-auth` with two zones.

Unbound, and PowerDNS do not cache the response and serve it with a TTL of zero. Knot serves the record with a TTL of five but also does not cache the response.

To test the micro-caching behavior, we sent multiple queries to the resolvers for the same domain with slight delays between them and observed how frequently resolvers queried the AuthNS. The delay was chosen such that the resolver has forwarded the previous query to the AuthNS, but has not yet received the response. This happens if queries arrive faster than the RTT between resolver and target ($RTT_{RT}$). We delayed DNS responses from the authoritative VM to the resolvers, to simulate the effect of different values for $RTT_{RT}$. We observed micro-caching for identical incoming or outgoing queries for all tested resolvers. Effectively, this limits an attacker to start a chain once per $RTT_{RT}$. The RTT is measured between resolver and target AuthNS because resource records of the intermediary AuthNS can be cached by the resolver and thus do not limit the lookup speed.

Next, we observe the behavior for longer delays. We delayed the second query until the resolver processed the response of the first query (and hence started to resolve the second chain element). This simulates queries that arrive $RTT_{RT}$ after the previous query arrived. PowerDNS and Knot fully honor the no-caching TTL and perform a full lookup for all queries. BIND 9 performs one full lookup per second, then only issues one query to the first element of the chain per additional client query. Similarly, Unbound only performs one full lookup per second, but then issues one query to the *last* element of the chain, which is not a `CNAME` RR.

Summarizing the result, the per-resolver query frequency for PowerDNS and Knot is $\frac{\#\text{chains}}{RTT_{RT}}$. As each chain has distinct domains, micro-caching is irrelevant across chains. BIND 9 and Unbound can be queried at most for $\frac{\#\text{chains}}{1\,\text{s}+RTT_{RT}}$. Realistically, the attacker does not know when the resolver's internal clock ticks over to the next second. Before starting the next query, the attacker has to ensure a full second passes after the record was cached, which happens $RTT_{RT}$ after the query is received by the resolver. Thus, at $1\,\text{s} + RTT_{RT}$ the record is guaranteed to have expired. Querying more frequently reduces amplification.

We analyzed the code of BIND 9 and Unbound to understand why they only issue one query per second. Both use a time value, which is rounded to seconds for all cache operations, explaining the observed cache invalidation once per second. BIND 9 special cases the first `CNAME` RR in a query and always perform the authoritative lookup, even when it was fetched from the cache. Unbound's cache inserts referral resource records, which `CNAME`s are one variant of, regardless of the TTL, but not the last chain element.

### 5.6.1.1 Internet Measurements

While all locally tested resolvers honor non-cacheable RRs, resolvers deployed on the Internet may behave differently. To assess this, we performed a full Internet scan querying for a wildcard `A` RR with a TTL=0 hosted by our AuthNS. The queried domain encodes the scan target's IP address, which allows us to (i) ensure that all records are fetched from our AuthNS and are not cached and (ii) match the scan targets with the queries observed at the AuthNS. All responses are recorded and filtered to remove domains that do not belong to our test. Figure 5.2 shows a (simplified) diagram of the connections

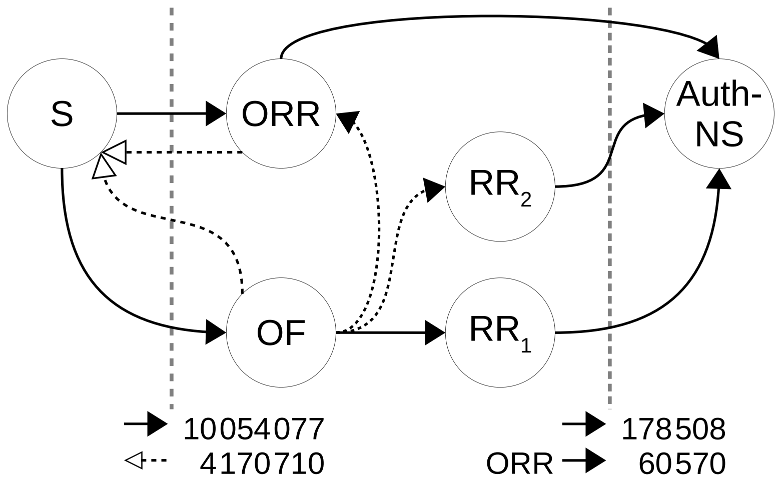


**Figure 5.2:** Connections between different resolver types. Our scanner (S) finds an open forwarder (OF) and open recursive resolver (ORR). A forwarder forwards the query to one or multiple recursive resolvers (RR). Recursive resolver (RR and ORR) query the Authoritative Name Server (AuthNS). Dashed arrows mark optional connections, like querying multiple recursive resolvers or sending a response to our scanner. An empty arrowhead marks a query response.

between our scanner, resolvers, and our AuthNS. The dashed gray lines mark the point of our packet capturing. Below them are the number of IP addresses we found.

4 170 710 resolvers responded to our scan query, of which 3 097 203 answers had a TTL of zero. For the same day (2017-08-02), Shadowserver's DNS scan [179] reports 4 198 025 resolvers found, i.e., a deviation of just 0.7 %. The scan shows that 74.3 % of all resolvers honor TTL=0, and they could be used for attacks.

Of those resolvers that enforce a minimal non-zero TTL in the response, most enforce large TTLs, making them unsuitable for DNS Unchained attacks. The ten most common TTL values we found are multiples of ten or 60. In decreasing order of occurrence they are 300, 600, 3600, 1, 30, 900, 60, 150, 14 400, and 20, which taken together account for 24.8 % (1 033 419) of all responses.

### 5.6.2 Amplification

After seeing that the vast majority of resolvers do not cache TTL=0 RRs, we now measure how much amplification in-the-wild resolvers would enable. The amplification factor is determined by the maximum number of elements of the chain that will be requested by each resolver. We thus configured a chain of 100 RRs and requested the first element from each resolver. The last chain element is an A record, and all RRs carry TTL=0.

BIND 9 follows the chain 17 times, whereas PowerDNS and Unbound only perform 12 and 9 lookup steps, respectively. Knot Resolver performs 33 lookups. BIND 9 is the only implementation that consistently responds with a "no error" status code. The other three reply with a ServFail status code if the end of the chain could not be reached.

Via our scans, we discovered 10 054 077 open resolvers and 178 508 recursive resolvers.

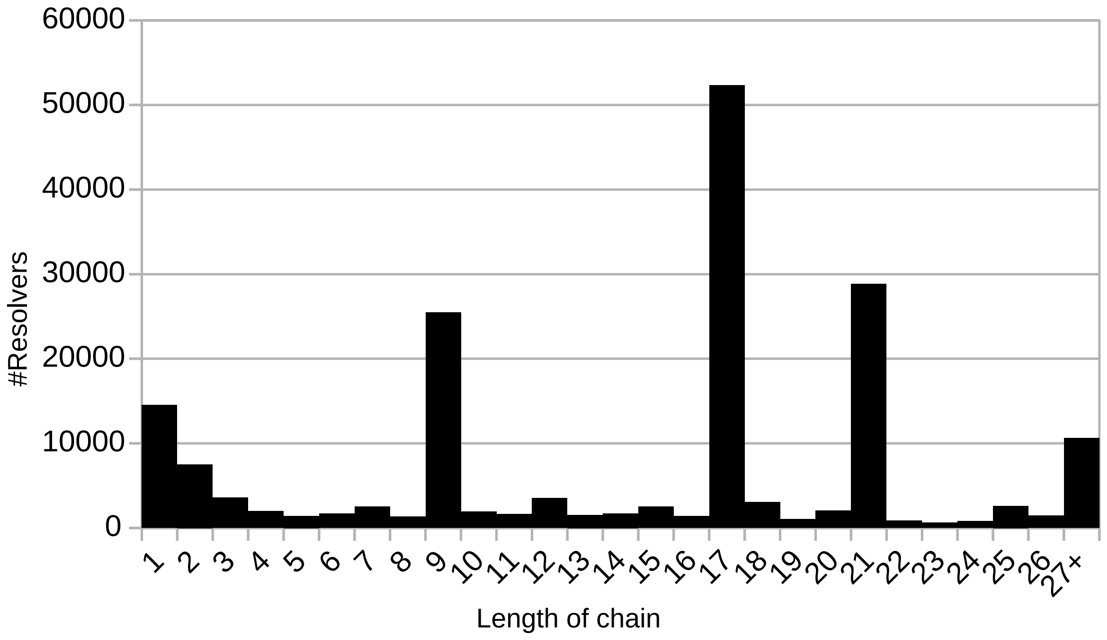


**Figure 5.3:** Supported chain length configurations for 178 508 recursive resolvers discovered with a full Internet scan. The spike at nine corresponds to Unbound and Microsoft DNS version; BIND 9 shows up as the spike at 17. The cause of the 21-spike is unknown to us.

Figure 5.2 gives an overview of the connections between scanner and resolvers. Open resolvers are open to the Internet and can be used by anyone. They can be recursive resolvers or simple forwarders, which forward the query to a recursive resolver. *Recursive resolvers* perform the recursive lookup procedure which we can detect at our AuthNS. We can count and distinguish the two types of resolvers based on the traffic captured at our AuthNS. If the encoded IP address of the scan target and the source IP address of the resolver querying our AuthNS are identical, then the resolver is an open recursive resolver. Otherwise, the encoded IP address belongs to an open forwarding resolver. We expect a much higher number of open resolvers than recursive resolvers, because as Kührer et al. [S1] found, most open resolvers are routers or other embedded devices. There is little reason for them to host a recursive resolver, as they require more resources.

Figure 5.3 shows how many resolvers support a given chain length. There are clear spikes for common values like nine, used by Unbound and Microsoft DNS, or 17 as used by BIND 9. Another spike is at length 21, yet we are not aware which software causes it. The quick drop-off at the beginning is caused by resolvers, which query the same domain from different IP addresses, often in the same subnet. In these cases, only one of the resolvers performs the full recursion, the others stop early leading to the drop. This could be caused by open resolvers querying multiple recursive resolvers in a short amount of time. Alternatively, it might result from an attempt to pre-fetch data for multiple resolvers as soon as one recursive resolver in the pool sees a new domain name.

From the data, we can conclude that resolvers do offer a considerable amplification potential. Intuitively, the amplification factor is the number of queries seen by the target

AuthNS in relation to the number of queries sent by the attacker. Factors larger than one mean the impact on the target is larger than the attacker's resources used for the attack. We can calculate the expected amplification ratio for all recursive resolvers by

$$\frac{\sum_{i=1}^{\infty}(\lceil \frac{i}{2} \rceil \times n_i)}{\sum_{i=1}^{\infty} n_i}$$

where $n_i$ is the number of resolvers that support chains of length $i$. The formula assumes that the first element in the chain is hosted on the target AuthNS, which is the more beneficial setup for an attacker.

All resolvers together (178 508) provide an amplification factor of 7.59. Focusing only on resolvers, which provide an amplification factor >1 (chains of length three or longer) results in an amplification factor of 8.51 with 156 481 available resolvers. These numbers are lower bounds because the early drop-off in Fig. 5.3 is caused by resolvers that query the same domain name from different IP addresses (which we then conservatively count as individual resolvers).

We already mentioned in Section 5.5.2.2 that some AuthNSes do not follow CNAME chains, even if they are authoritative for all domains. This is a performance optimization that reduces the work required to answer a query. For AuthNSes, which do not follow chains, all elements of the chain can be hosted on the target AuthNS thus an amplification factor of 14.34 can be achieved resulting in 89 % stronger attacks. Effectively, this removes the intermediary AuthNSes from the chain, and all resource records need to have a zero TTL value.

### 5.6.3 Overall Impact

Based on these observations, we conclude that CNAME chains enable attacks that are an order of magnitude larger (measured in queries per second) than naive water torture attacks. In practice, AuthNSes can handle 400 000 qps to 2 500 000 qps (queries per second) [10, 101, 143, 165]. An attacker only needs a fraction—determined by the amplification factor—of queries compared to that number. Often even lower query rates are sufficient to overload the AuthNS because the AuthNS also receives (and has to process) benign queries.

A single chain, which is resolved by all non-caching resolvers, causes more than a million queries to the target AuthNS ($7.59 \times 178\,508 \times 75\,\% \approxeq 1\,016\,157$). Each resolver can be queried roughly every second per chain (assuming a low $RTT_{RT}$). Using as few as two or three chains is enough to overload all commonly deployed AuthNS. For three chains, the attacker has to send 535 524 pps (packets per second). A DNS query packet with a 20 character long domain name requires 104 B (including Ethernet preamble and inter-packet gap) for transmission over the wire. The attacker needs 445.6 Mbit/s to overload even the fastest AuthNS.

In case the target AuthNS does not follow the CNAME chain, the stronger attack can be used where all elements are hosted on the target AuthNS. A single chain causes over 1.9 million queries ($14.34 \times 178\,508 \times 75\,\% \approxeq 1\,919\,854$) reducing the required bandwidth for the attacker accordingly.

## 5.7 Countermeasures

We will now discuss countermeasures to reduce the impact of DNS application-level attacks. First, we cover the authoritative view, how zones could be managed, and the effect of response rate limiting. Then we look at the behavior of recursive resolvers and how they could reduce the impact on AuthNSes.

### 5.7.1 Identification and Remedy by AuthNSes

A hard requirement for the proposed attack is that the attacker can create `CNAME` RRs on the target AuthNS. This gives the target AuthNS the power to inspect and deny problematic or malicious configurations, or completely remove zones from the AuthNS.

#### 5.7.1.1 Detection of `CNAME` Chains

Zone files for the DNS Unchained require several `CNAME` records pointing to external domains. If the attacker chooses random or pseudo-random domain names, AuthNSes can use this as an indicator for an attack. The target AuthNS operator could additionally check the target of `CNAME`s and discourage (or even forbid) `CNAME`s that point to `CNAME` RRs in other domains (which is already discouraged according to the specification). Exceptions are likely required for content delivery networks and cloud providers. Especially `CNAME` chains, i.e., several entries that eventually lead back to the same zone are not useful because both records are controlled by the same entity.

The AuthNS operator needs to implement periodic checks of all zones with `CNAME` entries. Only checking RRs during creation is insufficient, as the attacker can build the chain such that no `CNAME` points to another `CNAME` during creation. Given the same domains as in Listing 5.1, the attacker would first create `a.target-ans.com.` while the target domain (`b.intermediary.org.`) either does not exist or only contains other types, e.g., of type `A`. Checking the RR for `a.target-ans.com.` will not show any suspicious behavior. Now the same steps are repeated with `b.intermediary.o rg.`. This forces a non-trivial amount of work on the AuthNS. A too-long periodicity in the checking would allow the attacker to use the time between checks for the attack. Thus, the checks have to be relatively frequent.

#### 5.7.1.2 Lower Limit for Time-to-Live Values

In contrast to water torture attacks, chaining attacks fall apart if the chain's RRs are cached. Using a random prefix to circumvent caching is only possible for the specific combination of using `DNAME` RRs and abusing only those resolvers that do not support `DNAME`. Thus, forcing a minimal TTL of only a few seconds will have considerable impact, as it limits the per-resolver query frequency to $\frac{\#\text{chains}}{\text{TTL}+RTT_{RT}}$ compared to $\frac{\#\text{chains}}{1\,\text{s}+RTT_{RT}}$. Thus, a 10 s TTL will reduce the impact by roughly a factor of ten. However, an attacker can use more chains if a minimal TTL is enforced, which makes the setup more complicated. On the one hand, `CNAME` RRs with short (or zero) TTLs are used also for benign reasons, e.g., to implement DNS-based failover. On the other hand, in light of chaining attacks, we consider serving `A` and `AAAA` records with short TTLs as the better solution, which also closely resembles the desired semantics. Note, that a `CNAME` RR offers an additional canonical name for an already existing record, which is a relationship that rarely changes (and thus allows for non-zero TTLs).

### 5.7.2 Response Rate Limiting

Response Rate Limiting (RRL) is an effective technique to counter standard DNS-based amplification attacks. If DNS servers are abused for reflective amplification attacks [S1, 171], the attacker sets the request's source to the IP address of the victim. In turn, resolvers unknowingly flood the victim with DNS responses. To prevent such abuse, resolvers can implement IP address-based access control, which effectively turns them into closed resolvers.

Yet, this is not an option for intended open resolvers (Google DNS [76], Cloudflare DNS [27], Quad9 [154], etc.) and especially not for AuthNSes, as they *have* to be reachable by the entire Internet. Here, RRL plays an important role. RRL limits the frequency of how fast a client IP address can receive responses. The benefit for reflection attacks is clear, where a single source (the victim) *seemingly* requests millions of requests and now only faces a fraction of the actual responses due to RRL.

In principle, RRL also seems to mitigate chaining attacks. Yet, enabling RRL has its downsides, especially if resolvers hit a rate limit configured at an AuthNS. Resolvers will then retry queries, lacking an answer, which again increases the load on the AuthNS. Filtering all resolver traffic can even increase the incoming traffic tenfold, as observed by Verisign during a water torture attack [216, p. 24].

Additionally, RRL is implemented with a slip rate, which specifies how often the AuthNS will answer with a truncated response instead of dropping the packet. For example, a slip rate of two results in a truncated answer for every second query, the other times the query is dropped. Truncated responses then cause the resolver to retry the connection using TCP instead of UDP, which *drastically* increases the overall processing overhead for the AuthNS.

An ideal RRL configuration would thus never limit resolvers, as this may increase the required resources for the AuthNS in case of application-layer attacks. Filtering or rate limiting needs to be performed closer to the source. Naively, one could deploy RRL at resolvers to rate limit the initial attack requests ("chain starts") sent to them. However, then again the per-resolver request frequency is as low as one request per second, which would only be blocked by an overly aggressive RRL configuration. Even worse, if attacks are carried out via botnets, even those RRL configurations would not slow down the attacks.

### 5.7.3 Back-off Strategies

In case of packet loss at the target AuthNS, resolvers resend queries and thereby cause additional attack traffic. It is thus important to rate limit outgoing queries of resolvers and to implement suitable back-off strategies to give overloaded AuthNSes the chance to recover.

To assess how resolvers act in such situations, we have measured how four resolver implementations behave when querying a zone with two AuthNSes of which both are not reachable. BIND 9 sends a total of five packets with a delay of 800 ms in between packets. The AuthNS is chosen at random. After the third failed packet, BIND 9 has an exponential back-off with factor two. PowerDNS only sends out two packets in total, with a delay of 1500 ms in between. Unbound sends in total the most queries with 27 to 30. Worse, Unbound always sends two queries as a pair, which might go to the same or a different AuthNS. There is a delay of 375 ms between the pairs, which is doubled every two to four pairs. Knot has the most complicated retry strategy. Knot starts with sending UDP queries alternating to both servers, with a delay of 250 ms in between. After a total of two seconds, two TCP queries are sent to the first AuthNS, with a delay of 1000 ms between them. Six seconds after the start, the same pattern of UDP and TCP queries is sent to the second AuthNS.

BIND 9's and PowerDNS's behavior are not problematic, as the number of retries is small, and the retry delay is high. Especially problematic for Unbound is that it sends two identical queries to the same AuthNS without a delay between. Delaying retries is a good balance between providing fast answers (in case of packet loss) and not sending duplicate queries (in case of high round trip times). A delay of 250 ms between retries will cause unnecessary retries for many users. With our Internet scan, we found that 9045 recursive resolvers (14.9 %) have RTTs larger than 250 ms to both our AuthNSes and additional 19 773 resolvers (32.6 %) have such a high RTT to one of our AuthNSes.

An additional strategy is serving stale cache records [115]. Stale cache records are records in the resolver's cache of which the TTL has expired. A resolver can use them assuming that normally records contain working data, even if the TTL has expired (e.g., IP addresses change less often than the TTL of records expires). This technique is not new and already implemented in BIND 9.12 [131] and used by OpenDNS [144] and Akamai [117]. The usability improves as the client will receive an answer, which likely is usable, instead of receiving an error and failing to connect.

### 5.7.4 Recursion Depth Limit

Finally, also resolvers can more strictly limit the length of CNAME chains. Section 5.6.2 has shown that the resolvers do not agree on the maximum chain length. Limiting the length too strictly is harmful, as chains also exist for legitimate reasons—such as CDNs and DDoS protection services. The domain owner can often configure their DNS to point to a subdomain of the CDN and the CDN uses itself one or multiple CNAME RRs.

Legitimate use cases for CNAME chains must ensure the length is supported by all DNS resolvers if they want to support all users. We inspected the Active DNS [104] data set to identify benign chains. We extracted the CNAME entries from 2017-10-05 to reconstruct the longest benign chains, which consist of eight elements (seven CNAMEs and one final RR). This fits the shortest recursion limit of nine elements, which we observed for Unbound. Others [149] report nine elements as the longest legitimate chain they found, and certificate authorities are also only required to support chains with nine elements while fetching CAA RRs [77]. Based on those observations, a smaller recursion limit can be advised. We recommend supporting nine elements in a chain, which is the shortest value of all tested resolvers and covers benign chains. Such a recursion depth limit would limit the amplification of chaining attacks to factor five.

## 5.8 Related Work

**Application-Layer DDoS in DNS:** Several application-layer DDoS defenses have been proposed in the past [71, 134, 157, 218]. Many defenses are not immediately applicable to DNS. Protocol changes, such as client puzzles, would need widespread support, which is unrealistic to achieve in a short to medium time frame. Countermeasures that introduce more latency are especially problematic, as DNS is tuned for high efficiency. Filtering techniques, such as egress or ingress filtering, do not apply to DNS Unchained because it works without IP address spoofing. Blocking DNS traffic can even lead to more inbound traffic [216] and always risks blocking legitimate users.

The closest work to us is research on DNS water torture attacks. They were first presented in Feb 2014 [32] and are a known phenomenon for DNS operators [92, 206, 212, 216]. The DNS operator community focused on implementing mitigations, mainly to stabilize recursive resolver. Takeuchi et al. [197] propose a system to detect DNS water torture attacks based on lexical and structural features of domain names. They train a naive Bayes classifier and test it on captured traffic of their university networks. Our attack is related to DNS water torture as both are flooding attacks using resolvers, but water torture has several limitations, including the fact that it is easy to detect.

**Reflection and Amplification Attacks:** DNS also played a role in recent amplification attacks. The general risk of reflection attacks was identified by Paxson [147] and its full amplification potential was presented by Rossow [171]. Different proposals to detect and defend amplification attacks [106, 171, 178, 211] were made. They cover approaches to combat bandwidth exhaustion, like client puzzles, or prevent source address spoofing. In the context of DNS, Kührer et al. [S1, 109] analyzed the amplification potential of DNS resolvers. They found millions of open DNS resolvers on embedded devices or routers, meaning the openness of the resolver is likely a configuration issue. DNSSEC's potential to increase the amplification of DNS resolvers has also been documented [163]. While some attacks in fact abuse DNS, still, in contrast to our work, they do not represent application-layer attacks and are easy to filter.

**`CNAME` Chaining:** The possibility of chaining `CNAME` RRs is well-known and documented. For example, Shue and Kalafut [181] use differences in recursion strategies to fingerprint resolver implementations. Dagon et al. [38] use `CNAME` chains to amplify the number of queries from each resolver. They need multiple queries by the same resolver to analyze them for source port randomization of the resolver. Pfeifer et al. [149] measures the overhead for resolvers while looking up `CNAME` chains and recommends that AuthNSes should refuse `CNAME` chains before loading the zone files. Furthermore, they recommend that AuthNSes should also query destinations of `CNAME` RRs, similar to our recommendation in Section 5.7.1. In contrast to prior work, our attack focuses on the AuthNSes instead of the resolvers.

## 5.9 Conclusion

We have presented a new DDoS attack against AuthNSes that leverages amplification on the application-layer. DNS Unchained achieves an amplification of 8.51 using standard DNS protocol features, by chaining alias records (e.g., `CNAME`) and forcing resolvers to repeatedly query the same AuthNS. We performed full Internet scans and found 10 054 077 open DNS resolvers and 178 508 recursive resolvers. We determined that 74.3 % of those resolvers support non-cacheable DNS responses, creating a large pool of amplifiers that can be abused for chaining attacks.

We also discussed countermeasures to the new threat of DNS chaining attacks. This includes measures applicable to DNS operators to find and limit problematic DNS zones as well as enforcing minimal Time-to-Live values allowing caching. DNS resolvers can also be changed to have less aggressive re-transmission on unavailable name servers and limit chains to nine elements. A wide deployment of these techniques would severely degrade the performance of the proposed attacks, and we hope that our work raises awareness of the importance of these measures.

# 6 Optimizing Recurrent Pulsing Attacks using Application-Layer Amplification of Open DNS Resolvers

## 6.1 Motivation

This chapter is an extension of the previous chapter and covers the same interactions between resolvers Ⓡ/Ⓕ/Ⓟ and AuthNSes Ⓐ from the DNS overview in Fig. 2.1. We modify the attack to use temporal lensing, such that the traffic on link ③ can DoS servers and disrupt links, even unrelated to DNS infrastructure.

*Shrew attacks* or pulsing attacks are low-bandwidth network-level/layer-3 denial-of-service attacks. They target TCP connections by selectively inducing packet loss to affect latency and throughput. We combine the presented DNS CNAME-chaining attack from Chapter 5 with temporal lensing [158], a variant of pulsing attacks, to create a new, harder-to-block attack. For an attack, thousands of DNS resolvers have to be coordinated. We devise an optimization problem to find the perfect attack and solve it by using a genetic algorithm. The results show pulses created with our attack are 14 times higher than the attacker's average bandwidth. Finally, we present countermeasures applicable to pulsing and CNAME-chaining, which also apply to this attack.

## 6.2 Problem Description

Network-layer or volume-based DoS attacks are a common attack type in which the attacker tries to overwhelm the network connection of the victim, causing high delay, poor performance, or even total unreachability of the victim. One such attack is *DNS Unchained* which was presented in the previous chapter. It uses CNAME RRs to force recursive resolvers into sending many packets to the victim. CNAME records work similarly to pointers in programming languages in that they allow for arbitrary redirects and have to be resolved one by one to retrieve the desired RRs.

Another type of volume-based DoS attacks are pulsing ones, which have a much lower average bandwidth [111, 112, 124, 125], which makes them much harder to detect and defend against. In pulsing attacks the traffic consists of many short but high-bandwidth traffic spikes or pulses. However, for TCP connections traversing the same path as the attack traffic, this can be detrimental. The pulses cause packet loss in the TCP connections, making them adapt their congestion window size and reducing their throughput or even stalling it completely.

Attackers benefit from combining pulsing attacks with amplification to create stronger pulses. As shown by Rasti et al. [158], using reflectors allows for a technique called temporal lensing, which uses the difference in latencies for different reflectors to create fewer but much stronger pulses. They showed how recursive DNS resolvers are a good candidate for reflectors and performed measurements with 1201 resolvers. The DNS Unchained attack already has periodicity in its traffic pattern, making it suitable for repeated pulsing attacks.

## 6.3 Contribution

In this chapter, we draw on the two attacks described above and explore how they can be combined. We develop a methodology to describe a measure of success for pulsing attacks. We measure the path latencies for 60 570 open recursive resolvers, a much larger set compared to Rasti et al. [158], and devise an optimization problem to create an efficient and successful attack. The optimization problem is of high dimensionality, and we built an evolutionary algorithm that allows us to solve it. With that, we show the effectiveness of our attack. Reoccurring pulses, created with this attack, are 14 times larger than the attacker's average bandwidth. Lastly, we will cover countermeasures related to all three parts of the attack, namely pulsing, temporal lensing, and CNAME-chaining.

The remainder of this chapter is organized as follows. We start by introducing the background information for pulsing, temporal lensing, and CNAME-chaining in Section 6.4. We continue with the attack setup and threat model in Section 6.5. Section 6.6 describes our model for optimal pulsing and explains how we use evolutionary algorithms to solve the optimal pulsing problem. The evaluation is contained in Section 6.7 and followed by a discussion (Section 6.8) about the limitations of our current implementation. Lastly, we provide an overview of countermeasures applicable to pulsing, temporal lensing, and CNAME-chaining in Section 6.9.

## 6.4 Related Work

This section describes some previous work on pulsing Denial-of-Service attacks and CNAME-chaining attacks, both of which are building blocks for our attack scheme. We explain the basics of each of them, as far as it is relevant to this chapter.

### 6.4.1 Pulsing Denial-of-Service Attacks

DoS or Reduction-of-Quality attacks on TCP connections can be performed with short traffic pulses. These pulsing attacks, or *shrew attacks*, were first presented in 2003 by Kuzmanovic and Knightly [111, 112] and further analyzed by Luo and Chang [124, 125].

The attack creates short traffic pulses, which fill up the buffer of a router and cause packet loss in concurrent connections passing through this router. Packet loss causes two problems for TCP connections. First, TCP exhibits head-of-line blocking as in-order delivery of data is guaranteed. This increases the end-to-end latency because the receiver has to wait for a retransmission of the lost packet. Even with a fast retransmission strategy, it takes at least one round-trip until the retransmitted packet is received. In the case of flows with few packets in total, such as loading of small resources using HTTP, and long RTTs, this can be a considerable factor for the overall performance. Similarly, if multiple independent resources are multiplexed over a single TCP connection, for example in HTTP/2, a lost packet stalls the independent resources.

The second problem is a bandwidth reduction due to reduced congestion windows. Common TCP algorithms use packet loss as a congestion signal and are very sensitive to packet loss events, especially on high-speed and long-distance links [67, 146]. The common TCP algorithms include TCP Reno [12] and Cubic [162], as used in Linux, and Compound TCP [189] used in Microsoft Windows. Inducing packet loss via pulsing attacks, even if there is no permanent link congestion, will lead TCP to reduce the congestion window size and reduce the bandwidth. Scheduling repeated pulsing attacks, before the congestion window recovers, will lead to continuous reduction until it reaches a low steady state.

The effectiveness depends on the time between packet loss (the pulsing interval) and the amount of packet loss. Shorter pulsing intervals leave less time for the congestion window to recover. TCP algorithms can have different responses depending on the amount of packet loss. For small amounts, the congestion window is only reduced, while for larger amounts, the congestion window is fully reset to the initial size.

#### 6.4.1.1 Pulsing with DNS

Pulsing requires a high bandwidth from the attacker because the attacker's and the victim's bandwidth are directly correlated. To overcome this limitation, Rasti et al. [158] presented in 2015 *temporal lensing*, which trades off long low-bandwidth for a short high-bandwidth pulse. The attacker will pick a reflector to use as a relay to send a packet to the target. A reflector is any host, which can be brought to send packets to a third party. In the simple case, these can be servers that respond to spoofed UDP packets, like DNS or NTP [23, 171]. Each reflector has different path lengths, such that two packets sent at the same time will arrive at different times at the target. The attacker can measure this difference and adapt their timing while sending. They send the packets at different times, such that they arrive at the same time. The attacker uses reflectors with a wide range of path latencies to increase the lensing factor, which is the maximal bandwidth at the target compared to the attacker's.

Rasti et al. [158] used DNS resolvers as reflectors. They are a good fit due to their abundance, providing a wide range of different path latencies, and are easy to use as reflectors. In their paper, they cover single-pulse and multi-pulse attacks. They observe how the retransmission behavior of DNS resolvers leads to multiple pulses.

In this chapter, we explore how DNS Unchained can be combined with pulsing without higher bandwidth requirements for the attacker. The `CNAME`-chains generate a periodic request pattern at the AuthNSes, which we leverage to build recurrent pulsing.

## 6.5 Threat Model and Attack Setup

We introduced the primitive components used for our attack. We will now describe an attack scenario which will be used for the rest of this chapter. Based on this scenario we explain our attacker and threat model as well as countermeasures in Section 6.9.

Our attacker wants to attack a cloud-hosted service that uses many long-running TCP connections. The attacker knows or can find out where the servers are physically located. Additionally, they can control a DNS zone file on a server located "close" to the target servers and a DNS zone file on an arbitrary second server. We will refer to them as primary AuthNS and secondary AuthNS. The goal is to cause enough packet loss to severely reduce the throughput of, and increase the latency for, the target service.

The whole setup is shown in Fig. 6.1. It shows the placement of the primary AuthNS in relation to the target service. The bottleneck connection will be the congested link which causes the packet loss. As the traffic of the target service has to traverse this link, its TCP flows will be affected. The placement of the secondary AuthNS is unconstrained. Millions of open resolvers are on the Internet and can easily be found by an attacker.

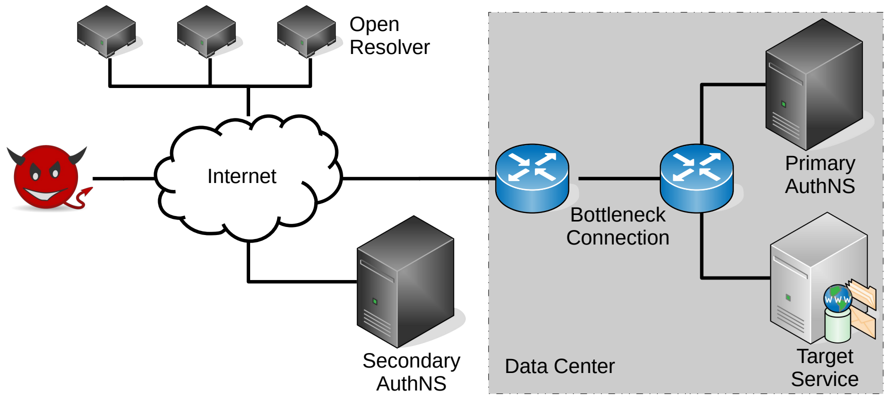


**Figure 6.1:** The figure shows how all parts are connected with each other. Important is the placement of the primary AuthNS, such that it shares the bottleneck connection with the target service. The bottleneck connection will be the congested link causing the packet loss during the pulsing. The placement of the secondary AuthNS is unconstrained. Open resolvers can be found by scanning the Internet or downloading lists of resolver IP addresses. The icons by VRT Systems are licensed under CC BY-SA 3.0 (34).

The attacker has ways to find the location of a server. Often the information can be gained through DNS, e.g., a `CNAME` to a domain associated with a cloud provider. The IP address of the target service is another good way to find the hosting provider.

For the second requirement, the primary AuthNS, the attacker could rent a VM from the same hoster. In other cases, buying DNS or web hosting from a provider using the same data center is possible. The important part is the "close" location to the target service, which for this case means that the primary AuthNS shares part of the path to the Internet with the target service. Sharing a part is important because this is where the packet loss can be triggered to affect the target.

The secondary AuthNS can be located almost arbitrarily, except being the same machine as the primary AuthNS. It will receive considerably fewer queries than the primary AuthNS as most of them will be cached by the resolver. The AuthNSes will be configured according to the setup used in our earlier Chapter 5 by creating two zones with `CNAME` entries pointing to the zone of the other AuthNS. This ensures that the attack works on all AuthNSes by preventing optimizations which would speed up the chain lookup for the resolver.

Besides the AuthNS setup, a list of reflectors and accurate timing between them and the AuthNSes is required. The reflectors will be open DNS resolvers or simply *resolvers*. They process queries from any client on the Internet and perform DNS resolution, forcing them to send multiple requests to each AuthNS. This results in a variable timing between the point a client sends the query and the point when the AuthNS receives it.

Scanning the Internet on UDP port 53 reveals many open resolvers and can be performed in under an hour [53]. Lists of resolvers can be downloaded [22] if scanning is not viable. Our attacker only needs the ability to send UDP packets. This is in contrast to volume-based reflective distributed DoS attacks [171], which require spoofed source IP addresses for the attack traffic. In our setup, the attacker can operate from a single source or even use botnets to increase anonymity and total bandwidth.

### 6.5.1 Accurate Timing

For achieving the temporal lensing effect, accurate timing information between all parties, i.e., the attacker, resolvers, and both AuthNSes is required. They are easy to measure by sending DNS queries to the resolvers and observing the response times. We name the RTT between attacker and resolver $RTT_{A/R}$ and analogously, the RTT between resolver and AuthNS $RTT_{R/NS}$.

Measuring $RTT_{A/R}$ is accomplished by requesting a cached resource record and measuring the response time. The time between resolver and AuthNS, denoted as $RTT_{R/NS}$, is measured by sending queries to non-cached resource records, such that the resolver has to fetch the records from the AuthNS. By subtracting $RTT_{A/R}$ from the measured RTT we gain $RTT_{R/NS}$. The important time is the one between resolver and primary AuthNS, as these RRs will not be cached and thus will cause traffic. All RRs of the secondary AuthNS will be cached by the resolver after the first lookup, thus they resolve instantaneously and do not affect the timing calculations.

Halving the RTTs gives the delay from attacker to AuthNS. This assumes a symmetric delay for the return path, which might not always hold true, but is a good enough approximation [158]. The attacker can now perform a pulse by picking an initial delay $d_R$ depending on the resolvers, such that $\forall R.d_R + (RTT_{A/R} + RTT_{R/NS})/2$ is constant. Sending a packet to resolver $R$ after $d_R$ will result in pulse where all packets arrive at the same time.

## 6.6 Pulsing

This section is constructed as follows. We assume the attacker has completed the attack setup which includes measuring the round-trip-times (RTTs) as described in the last section. We then define how an optimal recurrent pulsing attack will look like and describe an optimization problem to create them. Based on this optimization problem of coordinating thousands of reflectors, we develop an evolutionary algorithm which can solve this high-dimensional problem efficiently.

### 6.6.1 Optimal Recurrent Pulsing for Chains

Section 6.5.1 describes how the timing for a single pulse can be calculated. Chaining attacks requires a more complex strategy. Lookup chains started at different resolvers will get out of sync after a few chain elements, due to varying $RTT_{R/NS}$ values. To create several pulses, one can carefully align the chains such that the various request periods add up and exceed the victim's resources. This results in a trade-off between creating pulses (which require delaying requests) and the average attack bandwidth (which suffers from delays).

We define the goal of recurrent pulsing as exceeding the target's estimated maximum resources as long as possible. The intuition behind this goal is that at all times when the attack traffic exceeds the available resources, packet loss will occur and harm the victim. We model this in an optimization strategy that—given a set of initial per-resolver delays to start the first chain—optimizes the overall time the target's bandwidth is over a certain threshold. More formally, we define an optimization function $f$

$$f_T(\mathcal{D}) = \int s_T(t, \mathcal{D})dt \tag{6.1}$$

with

$$s_T(t, \mathcal{D}) = \begin{cases} 1 & \text{if } b(t, \mathcal{D}) \geq T \\ 0 & \text{else} \end{cases},$$

where $b(t, \mathcal{D})$ is the attack bandwidth reaching the victim at time $t$ and $T$ is the target's available bandwidth. For each resolver $r \in \mathcal{R}$, the attacker can choose an arbitrary delay $d_R$ with $\mathcal{D} = \{d_R | R \in \mathcal{R}\}$ to maximize $f(\mathcal{D})$. They can then estimate the attack bandwidth $b(t, \mathcal{D})$ by computing how many queries would arrive at the victim—given the per-resolver RTTs $RTT_{A/R}$ and $RTT_{R/NS}$.

BIND 9 is one of the most commonly deployed DNS resolvers [P1, S1], therefore to simplify matters we assume all resolvers behave like BIND 9. It follows `CNAME`-chains for 17 elements in total of which nine will be sent towards the victim. We refer to this as the *chain length limit*. RRs can be marked as non-cacheable, by setting the TTL value to zero. BIND 9 caches such RRs until the internal timestamp counter ticks to the next seconds. After the record has expired, fetching it again takes $RTT_{R/NS}$, so chain restarts can happen every $1\,\mathrm{s} + RTT_{R/NS}$ to ensure that they are never cached by the resolver.

Other resolvers have different chain lengths limit or truly never cache a RR. In practice, an attacker could perfectly fingerprint each resolver software and adapt the maximum chain length and request frequency per resolver. There are resolvers with lower chain length limits, which would result in an overall lower amplification, while the ones with higher limits increase it. Resolvers, which never cache a RR, are beneficial because they can be scheduled more freely, as chain restarts can happen every $RTT_{R/NS}$ for them.

### 6.6.2 Evolutionary Model

While the aforementioned method is optimal, solving the optimization is impractical. Choosing optimal values for $d_R$ quickly becomes intractable, as the number of resolvers increases. There is one delay value to choose per resolver, resulting in an $|\mathcal{R}|$-dimensional optimization problem.

To tackle this complexity, we decided to explore evolutionary algorithms for finding a good (yet not optimal) solution to this optimization problem. Evolutionary algorithms borrow ideas from evolution to build algorithms without much expert knowledge. Genetic algorithms [58], a form of evolutionary algorithms, have demonstrated quick convergence to a good solution for many optimization problems [126]. Technically, genetic algorithms choose an input population and then use mutation, crossover, and re-selection to optimize towards a given fitness (i.e., optimization) function. While *mutation* functions take a genome representation and mutate its values, *crossover* takes two parent genomes and combines them to create a new one.

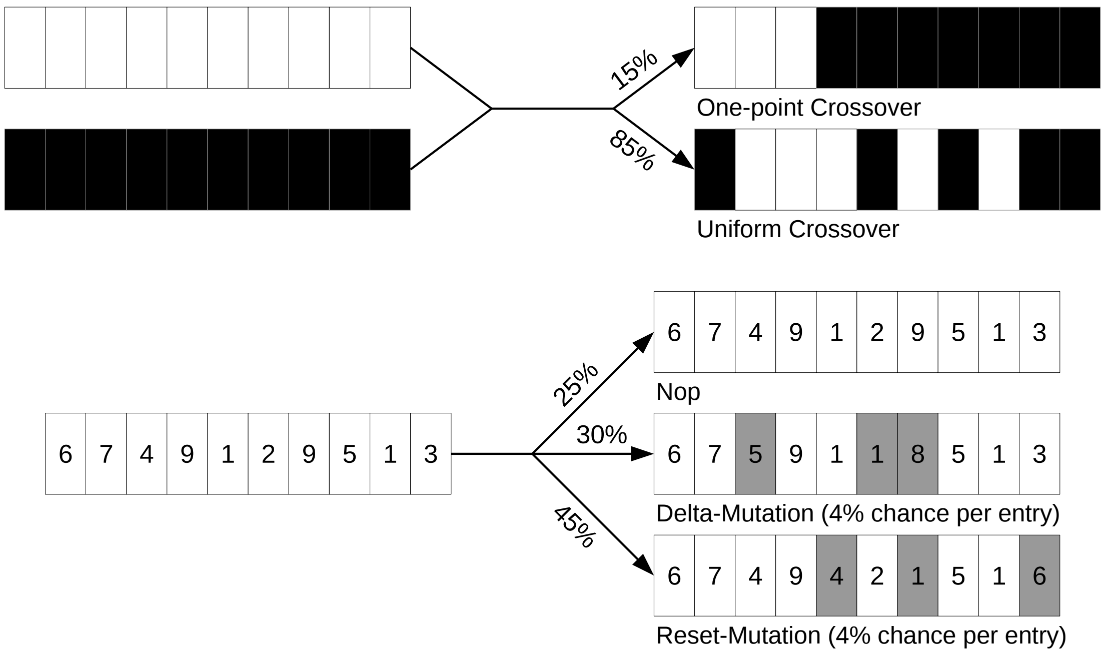


**Figure 6.2:** Graphical representation of implemented crossover and mutation strategies. The delta- and reset-mutation have a $4\,\%$ chance of mutation per entry, instead of the displayed $30\,\%$.

#### 6.6.2.1 Description

Before describing our implementation, please note that it is common practice to fine-tune genetic algorithms before applying them to a large data set. Different population sizes, selection strategies, or mutation probabilities can have large impacts on quality of the results and the speed in which the algorithm converges. In the following, we thus present configuration parameters that we obtained after a grid search to optimize for a low convergence time.

Figure 6.2 provides an overview of our genetic algorithm. An *individual* is the sequence of all resolvers' delay values $d_R$ and is represented as an array of length $|\mathcal{R}|$. We represent the delay values as integer multiples of 1 ms. Each delay is in the range of 0 ms to 1500 ms. Larger values have little influence on the performance of the algorithm. This is caused by the way the chain restarts are handled, because they occur in regular intervals of $1\,\mathrm{s}+RTT_{R/NS}$. Offsets which differ by a multiple of $1\,\mathrm{s}+RTT_{R/NS}$ only differ at the initial start but result in the same alignment. $94.2\,\%$ of all measured $RTT_{R/NS}$ values are below 500 ms. For these the maximal delay span of 1500 ms is enough to represent all possible alignments.

The *population* is the collection of all individuals which exist at any point in time. The initial population is seeded by randomly choosing each delay value from a uniform distribution (0 to 1500). Each individual can be mutated or mixed with other individuals in the population during crossover. A larger population provides more diversity in the genomes and allows more combinations in crossover. Our population has a constant size of 1250 individuals.

In each iteration of the genetic algorithm, 50 % of the population is selected for mating, creating 25 % (312) new individuals overall. The selection is based on a maximizing selector, selecting the fittest individuals. After applying crossover and mutation, some old individuals are replaced with new ones. All individuals in the population are equally likely to be replaced with new individuals.

Figure 6.2 gives an overview of our crossover and mutation strategies. We implemented a delta-mutation and a reset-mutation. The delta-mutation occurs with a 30 % chance and alters delays by a random value chosen from a normal distribution with $\mathcal{N}(0, 30^2)$. The reset-mutation occurs with a 45 % chance and replaces delays with a new value chosen uniformly from the range of 0 to 1500. Each individual delay is mutated with a 4 % probability. For the remaining 25 % of cases no mutation is performed.

Crossover is implemented using one-point-crossover and uniform crossover. For one-point-crossover, performed at 15 % chance, a crossover point $p$ in the range $p \in [0; |\mathcal{R}|]$ (the length of the genome) is chosen uniformly. The new genome will consist of the first $p$ delays of the first parent and the remaining $\mathcal{R} - p$ delays from the second parent. The uniform crossover is chosen 85 % of all times and selects each delay with equal probability from either parent.

#### 6.6.2.2 Optimization Goal

We adapt the optimization function defined in Eq. (6.1) to make it suitable for a fitness function in genetic algorithms. First, we use a discrete time model, thus we measure the target's bandwidth in intervals of 16 ms each. Aggregating the bandwidth in intervals is a performance optimization. Second, the fitness function should incrementally improve for bandwidth values which are *close* to the chosen threshold (i.e., the target's available bandwidth) to favor individuals that work towards the final goal. We define the fitness function as

$$f_T(\mathcal{D}) = \sum_{t=0}^{t_{end}} s_T(t, \mathcal{D}) \tag{6.2}$$

with

$$s_T(t, \mathcal{D}) = min(\frac{b(t, \mathcal{D})^p}{T^p}, 1),$$

where $b(t, \mathcal{D})$ is again the target's bandwidth, but per 16 ms interval, and $T$ is the target's available bandwidth. $\mathcal{D}$ is the genome which is optimized by the genetic algorithm to maximize $f_T(\mathcal{D})$. This fitness function offers more incentives the closer the bandwidth is to the threshold. We set $t_{end}$ to 1875, which corresponds to a 30 s interval. The power $p$ allows specifying how much the genetic algorithm should favor smaller spikes (the lower $p$, the more it favors also smaller spikes).

We consider two different chain restart strategies, which we term *greedy* and *self-pulsing*. The *greedy* strategy re-starts a chain as soon as possible, which is the aforementioned $1\,\mathrm{s} + RTT_{R/NS}$ delay between restarts. The *self-pulsing* strategy aligns all chain restarts of the same resolver with itself, such that all packets from the same resolver will arrive at the target at the same time. Restarts happen every $\lceil (1\,\mathrm{s} + RTT_{R/NS})/RTT_{R/NS} \rceil * RTT_{R/NS}$, which is the smallest multiple of $RTT_{R/NS}$

larger or equal to $1\,\mathrm{s} + RTT_{R/NS}$. The idea behind this strategy is that due to the self-pulsing of the chain, the alignment between resolvers becomes easier.

## 6.7 Evaluation

To evaluate how effective such pulsing might become, we created the following measurement setup. We measured the RTT between resolvers and two AuthNSes as described in the beginning of Section 6.6. Our primary AuthNS was located in Germany, while the less important secondary AuthNS was located on the US east coast. We primed the resolvers' caches to ensure reliable RTT measurements to the resolvers and set the *recursion desired bit* to false. The recursion desired bit instructs a resolver to only return an answer if one is available in the cache, and the resolver should not fetch it from the AuthNS. Randomizing the queries to measure the RTT between resolver and AuthNS ensures resolvers need to forward the query to the AuthNS. The measurements were repeated five times to gain reliable measurements and to calculate the standard deviation.

For more stable timing measurements, we only considered resolvers that are not forwarders. Forwarders have an additional layer of DNS resolvers to the AuthNS, making the measured RTTs unreliable. To gather such a list, we conducted full IPv4 scans, fingerprinting each resolver during the scan and observing which resolvers contact the AuthNS. This way we can differentiate between a resolver directly contacting our AuthNS or via some other resolver. We followed the scanning best practices as outlined by Durumeric et al. [53]. Not all resolvers provided usable data, e.g., because the resolver was not reachable during the whole measurement period. Other resolvers disabled queries with recursion-desired bit set to false and thus our methodology could not determine the RTT from us to the resolver. Excluding those resolvers, we overall gathered reliable RTT data for 60 570 resolvers.

We can evaluate this approach by checking how well the evolutionary algorithm performs while finding a solution. Completing the genetic algorithm is expensive, due to the high number of resolvers. The genetic model parallelizes well and can use all available CPUs. This allows calculating the model using 24 cores of an Xeon E5-2667 server in two hours. Our genetic algorithm stops after 5000 iterations, which we found to be sufficient to converge to a suitable solution.

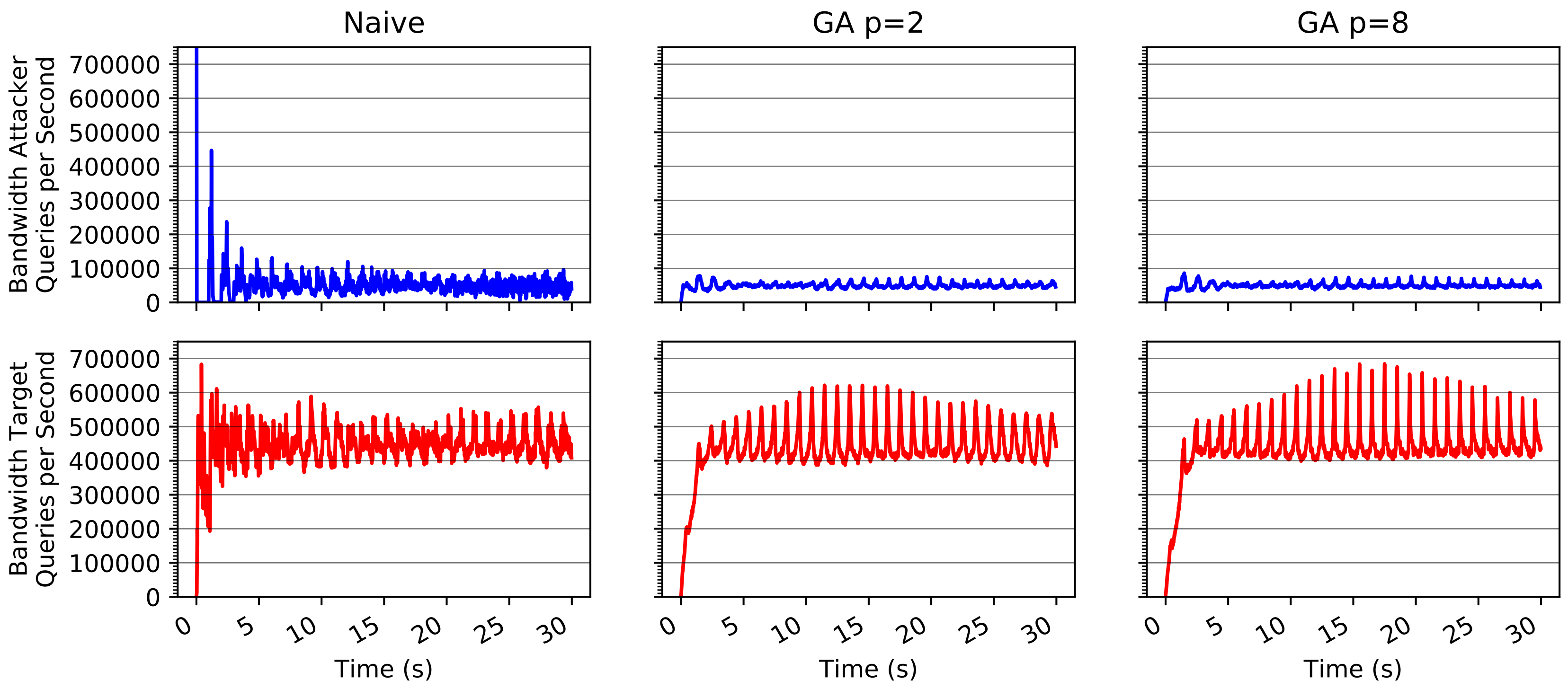


**Figure 6.3:** Bandwidth comparison of the genetic algorithm's solutions compared to a naive one, which starts all chains as soon as possible. The upper graphs are the outgoing bandwidth of the attacker, while the lower graphs are the incoming bandwidth of the target. The time values between attacker- and target-graphs are independent and start at zero on the first packet sent or received.

After the genetic algorithm terminated, we relate the performance of its output (i.e., the best individual) with the naive attack in which all chains start immediately (all $d_R$ are 0). Figure 6.3 compares the packet rates the attacker sends (upper graphs) and the victim receives (lower graphs) for the naive approach (left), the genetic algorithm with $p = 2$ (middle) and $p = 8$ (right).

Estimating absolute bandwidth numbers, such as kbitps, is very hard, as it depends on the multiple factors we do not control or are left undefined in our setup. Different factors are the length of the domain name, EDNS support and options used by the resolvers, and the chosen transport protocol (UDP or TCP). We therefore specify the bandwidth in queries per second (qps). A short DNS query (no EDNS, using UDP) requires roughly 100 B to transmit, including the inter-packet gap of Ethernet.

While the naive approach also shows some pulsing behavior (due to resolvers sharing similar $RTT_{R/NS}$), (i) the pulses decay after a few seconds, and (ii) the naive solution requires the attacker to start all chains virtually simultaneously—which is unrealistic in practice. In contrast, the solutions found by the genetic algorithm are practical, i.e., require a reasonably constant sending rate from the attacker. This is a positive side effect of this approach, even though the fitness function does not favor a uniform attacker bandwidth. Furthermore, the genetic algorithm creates slightly larger and significantly more steady spikes than the naive approach. This effectively means that an attacker can create reoccurring pulses that are 14 times higher than its average sending rate.

The original temporal lensing experiment and results are quite different. We showed that temporal lensing is viable with a high attacker bandwidth and very low per-resolver bandwidths. Rasti et al. [158] only achieved high ($> 10$) amplification rates if both (i) the attacker's maximum bandwidth is severely restricted ($< 10\,000$ qps) and (ii) the maximum bandwidth to any resolver is at least 100 times larger compared to our setup. Given an identical per-resolver bandwidth of 1 qps in both experiments, we achieve a factor of 14 while Rasti et al. report no amplification. As such, our experiment improves results given many more resolvers and lower per-resolver bandwidths.

Finally, we compared the performance of the greedy restart to self-pulsing. Self-pulsing has longer pauses between restarts, reducing the overall average bandwidth by 5.4 %. We noticed little influence on the genetic algorithm between those two strategies. The naive approach profited slightly from this change, as the naturally occurring pulses are amplified, due to the consistent alignment between chains of different resolvers. In the greedy approach, the alignment between chains of different resolvers becomes distorted over time and flattens the occurring pulses.

## 6.8 Discussion

We discuss the limitations and how our model could be extended to improve the shortcomings. Our first simplification concerns the time model we use. We assumed all measured times are constant, and based on that, chose a discrete time model. Network packets experience jitter resulting in varying RTT values. This can be modeled by using a probabilistic model. Instead of assigning a single time to each packet, the packet is represented as a probability distribution of when it arrives.

A second limitation regards redundancy in DNS. DNS requires at least two AuthNSes configured for a zone, and resolvers are free to choose between all available AuthNSes. If the primary AuthNS is under full control, both configured AuthNSes could use the same IP, thus the same machine, and reduce this uncertainty.

If the primary AuthNS is a hosted service, it will have multiple physical machines configured. Measurements of the server selection algorithm [140, 220] have shown that most resolvers prefer the most responsive (i.e., lowest RTT) AuthNS but most will use even the slower AuthNSes for some fraction of all queries. Resolvers use it to re-check the RTT of all available AuthNSes. This makes it impossible to predict exactly when a packet arrives at the target AuthNS, as the resolver might use a different AuthNS.

A general solution to combat the uncertainty of timing is to have re-synchronization points. At these points, the attacker stops sending queries to the resolvers and waits. This ensures that in-flight queries of resolvers will be received, and cache entries can time out. After some time, the attacker can restart the attack, which will be identical to the initial attack. Additionally, this allows an attacker to re-use the computed delay values, as the timings simply repeat.

Lastly, our genetic algorithm implementation so far has only optimized the initial delays. We have discussed greedy and self-pulsing strategies to restart the chains. Ideally, the timing between restarts of the chain could be optimized as well. Other values might yield better results, especially if the wait time is calculated for every restart. However, optimizing the restart times would multiply the dimensions in our search space and would require more advanced strategies to scale.

## 6.9 Countermeasures

Our attack consists of three parts (i) pulsing (ii) temporal lensing and (iii) `CNAME`-chains. There are different countermeasures applicable for each of the parts.

### 6.9.1 Pulsing and Packet Loss

Common TCP variants, like TCP Reno [12] and Cubic [162], use packet loss as the main congestion signal. Previous work [57, 111, 219] proposed randomizing the retransmission timeout (RTO) and extending buffer sizes [176].

Pulsing is most harmful when the attacker is synchronized with the victim. Causing packet loss on every retransmission effectively stalls the connection. Randomizing the RTO prohibits the attacker from being synchronized, thus reducing the effects.

Larger buffers will cause less packet loss, all else being equal. Extending buffers requires hardware changes to the network routers which makes it costly. The additional buffer space might be used by the existing flows, thus negating the effect. It can even increase the RTT for flows, thus having a negative effect under normal operations.

There are TCP algorithms, that use other feedback mechanisms as their congestion signal; an older variant of it being TCP Vegas [15]. However, TCP Vegas is being suppressed by algorithms with loss-based congestion control. Recently, TCP BBR [20, 21], short for bottleneck bandwidth and RTT, was presented, which uses RTT information as congestion signal. TCP BBR ignores packet loss up to a threshold, thus being more resilient to slight packet loss.

The second problem of TCP is the head-of-line blocking it creates after packet loss. This does not necessarily affect the throughput by much, but the end-to-end latency. Where applicable, for example with HTTP requests, connections with true multiplexing help. One candidate for this is QUIC [51, 93]. It can use UDP and multiplexes requests and responses through the UDP connection. Packet loss here will only affect the data where a packet was lost.

### 6.9.2 Temporal Lensing

Temporal lensing requires precise timing information between all hosts. Any disturbance, such as network jitter, will make the attack less effective. Jitter could be introduced by the network routers. Care must be taken so that packets of the same flow do not overtake each other, as such, having a constant jitter per flow 5-tuple would work [158].

Another approach would be to introduce the jitter on the DNS servers. This would not be a generic solution, as other kinds of reflectors can be used, but it would not have the problem of packets overtaking each other.

The problem with artificial jitter is, that it unnecessarily slows down all applications on the Internet. It forces higher memory and CPU consumption as packets have to be stored and re-scheduled for later processing.

### 6.9.3 DNS and `CNAME`-Chains

Countermeasures against `CNAME`-chains as presented in Chapter 5 also apply here. These can be summarized into preventative measures and mitigation ones. AuthNSes could prevent loading zones with too long chains. Since this is not forbidden by the DNS specification, this is more of a solution for managed DNS hosting providers.

Resolvers should not follow `CNAME`-chains too long. This does not prevent the amplification, but at least reduces its impact. Benign chains are only up to nine elements, which is the shortest supported length of common resolvers [P1, 149]. In contrast, some resolvers support lengths of up to 33 elements. These resolvers provide a huge amplification potential that can be reduced without impacting deployed DNS setups. Resolvers might also consider minimal TTL for DNS records so that caching can take effect for reducing the number of queries. However, there are legitimate reasons to use low TTL values, like load balancing, so this is also not a general-purpose solution.

## 6.10 Conclusions

Recurrent pulsing provides a new way of performing distributed denial-of-service attacks against TCP connections. Leveraging a DNS application-layer amplification attack allows for traffic pulses 14 times higher than the average attacker bandwidth, while also creating thousands of low bandwidth and hard-to-block flows.

We developed a genetic algorithm to solve the challenge of coordinating an attack between tens of thousands of reflectors, while still being efficiently computable. Our implementation shows the feasibility of launching this attack and the high amplification ratio achievable.

We present a wide range of potential countermeasures ranging from changes to networking equipment, like larger buffer sizes, over protocol design, such as different TCP algorithms, to DNS-specific countermeasures.

In the future, we want to extend our implementation by building a more realistic network model, which can factor in jitter. By giving the attacker more control over when chain restarts happen, we likely can improve the pulsing factor.

# 7 Anomaly-based Filtering of Application-Layer DDoS Against DNS Authoritatives

## 7.1 Motivation

For the last stop on our journey through the DNS landscape, we examine the defenses for AuthNSes Ⓐ/Ⓣ from the DNS overview in Fig. 2.1. Root servers ⊙ can benefit from the same defense we discuss, but are special in many ways and are managed well. The resolvers Ⓡ/Ⓕ/Ⓟ are of interest only in terms of how they shape the traffic ③ that reaches the AuthNSes.

Authoritative DNS infrastructures are at the core of the Internet ecosystem. But how resilient are typical AuthNSes against application-layer DoS attacks? In this chapter, we assess the expected attack load and DoS countermeasures, with the help of a large country-code TLD (ccTLD) operator. We find that standard botnets or even single-homed attackers can overload the computational resources of AuthNSes—even if redundancy such as *anycast* is in place. To prevent the resulting devastating DNS outages, we assess how effective upstream filters can be as a last resort. We propose an anomaly detection defense that allows both, well-behaving high-volume DNS resolvers and low-volume clients to continue name lookups—while blocking most of the attack traffic. Upstream ISPs or Internet Exchange Points (IXPs) can deploy our scheme and drop attack traffic to reasonable query loads at or below 100k queries per second at a false positive rate of 1.2 % to 5.7 % (median 2.4 %).

## 7.2 Problem Description

The DNS is at the core of Internet operations. Nearly all online services rely on DNS. Consequently, any DNS outage has broad consequences. For example, in July 2021, major online services, including Airbnb, FedEx, UPS, and large gaming networks (Steam, PlayStation Network), were unavailable for 30–60 minutes due to a DNS configuration issue at Akamai. In October 2016, a DDoS attack against DNS provider Dyn resulted in an hour-long downtime of Twitter, Amazon, Netflix, Spotify, and others. DNS outages become particularly critical higher up in the hierarchy. If TLD authoritatives are unresponsive, it interrupts the entire ecosystems underneath. The `.com` zone alone delegates more than 150 million domains [207] to name servers lower down in the hierarchy. TLD outages have severe consequences for all services relying on this TLD. Any name lookups of these domains (or subdomains) strictly depend on the availability of the TLD name servers. While caching at DNS resolvers may delay the attack effect in some cases, caching is only effective for recently resolved domains and clearly does not help against DNS outages in the long run.

Therefore, TLD operators aim to maintain high availability and responsiveness. They design the DNS infrastructure to be less susceptible to temporal traffic spikes. Redundancy and anycast infrastructures are typically at the core of these efforts. Most TLD zones are operated in one or more anycast “clouds”, i.e., multiple geographically scattered name servers announce the same name server IP address. DNS resolvers will transparently contact the best (typically, “closest”) anycast location for name resolution. Anycast is a significant pillar against DoS attacks. Single-sourced DoS attacks can only target a single anycast location, not the entire cloud at once. Likewise, DDoS attacks lose power, as they are automatically scattered among multiple anycast locations. Furthermore, when under attack, DNS operators can strategically update their BGP announcements to shift traffic from overloaded sites to those that still have resources. Rizvi et al. recently proposed so-called network playbooks that systematically automate this process [166]. Furthermore, DNS operators can use so-called *scrubbing services* to filter volumetric DDoS attacks that would otherwise overload the overall aggregated capacity of all anycast locations. For example, amplification attacks [171] allow adversaries to multiply their bandwidth by orders of magnitude and scatter the attack traffic across multiple sources—resulting in distributed target bandwidths that reach Tbit/s. Having said this, amplification attack traffic does not follow normal usage patterns of requests-facing AuthNSes [105, 106] and can be trivially filtered upstream.

But can existing defenses cope with targeted DDoS attacks against DNS infrastructures? In contrast to volumetric attacks, DNS application-level attacks blend into normal DNS traffic that follows valid semantics. Consequently, such attacks evade content-based filters. Moreover, as we will show, powerful application-level DoS attacks can easily overload DNS anycast infrastructures. For example, botnets may send massive volumes of benign-looking DNS queries that exceed the total (bandwidth or computational) capacity of DNS anycast setups. Even strategic traffic rerouting via BGP cannot mitigate this threat if the overall provisioning capacity falls short of the attack resources. The recent past has already documented application-level incidents in the form of random-subdomain attacks [32, 197, 206] (which force authoritatives to respond to non-existing query domains), and other DNS-specific attacks are possible [P1, 105, 137]. When existing defenses such as content-based filters and anycast fail, what remains? If DNS operators lack resources to mitigate massive attacks on their own, upstream filtering remains the ultimate resort to prevent severe outages.

## 7.3 Contribution

In this chapter, we explore whether upstream providers can leverage past resolver behavior to shield downstream DNS infrastructures. Given that attackers can evade content-based filters, we explore anomaly detection as *ultima ratio*. Our goal is to enable upstreams to reliably filter malicious traffic *before* it reaches the AuthNSes. To this end, we leverage past DNS lookup profiles derived from NetFlow statistics [3, 25, 26] to build behavioral profiles of DNS resolvers. TLD operators can learn such models before the attack and send them upstream—e.g., to scrubbing services or upstream providers using BGP Flowspec [122]—to filter attack traffic on demand. In principle, upstream filters can shield DNS infrastructures from abusive queries, mitigating application-level attacks that flood authoritatives with semantically valid lookups.

When designing DDoS defenses for DNS, we have to overcome several obstacles. First, the set and behavior of DNS resolvers are dynamic due to network churn and temporal activities. Thus, anomaly detection cannot assume a static set of resolvers in an attempt to identify benign clients. Second, to retain net neutrality, we would like to avoid giving preference to large resolver operators (Google DNS [76], Cloudflare DNS [27], Quad9 [154], etc.). That is, a defense should not favor “heavy hitters” (e.g., due to their sheer query volume), as not to discriminate against smaller decentralized DNS resolvers. And finally, if attackers can abuse resolvers that also serve benign users, DNS operators cannot clearly distinguish between “good” and “bad” resolvers.

In collaboration with a large European TLD operator, we propose a two-layered defense applied upstream to address these challenges. A low-pass filter allows DNS queries to pass if a resolver is below a particular query volume. We augment this low-pass filter with an allowlist of well-behaving, high-profile resolvers extracted from past behavioral profiles. We test our methodology on a real-world dataset of a large ccTLD operator that hosts one of the ten most popular TLDs worldwide[1]. To this end, we model two application-level DDoS attackers: (i) leveraging DDoS-capable botnets and (ii) abusing open DNS resolvers. We then align attack data following these models with query behaviors recorded at the global anycast network of the TLD zone. Data fusion grants insights into the effectiveness and accuracy of the proposed countermeasure. We show that by training just one hour before the attacks, our last resort defense can reduce the attack traffic to reasonable levels at or below 100 kpps (packets per second) while dropping only 1.2 % to 5.7 % (median 2.4 %) of queries from benign clients.

To summarize, we present the following contributions:

- We provide insights into the daily operations of a large ccTLD provider and demonstrate the typical client population of their anycast infrastructure.
- We model two powerful DNS application-layer (D)DoS attacks and evaluate their impacts on the distributed TLD infrastructure.
- We introduce a low-pass filter based on anomaly detection that can be readily deployed upstream to mitigate application-layer DDoS attacks against AuthNSes.

## 7.4 Background

This section outlines how modern authoritative DNS infrastructures are designed and how they achieve performance and resilience.

### 7.4.1 Modern AuthNS Infrastructures

*Redundancy* is key to ensuring availability, resilience, and load balancing. Instead of only providing a single large DNS server, the service is distributed across multiple servers. This limits the impact of a broken server and allows for better performance by placing these close to the clients, thus reducing network latency.

Modern DNS infrastructures leverage two core mechanisms to obtain redundancy, both are visualized in Fig. 7.1. First, DNS operators announce multiple IP addresses for each zone, such that clients can choose a “good” server. A zone lists the AuthNSes in the `NS` records—at least two, but often more. Each `NS` record can point to one or multiple IP addresses via the `A` (IPv4) and `AAAA` (IPv6) resource records. A DNS client sends queries to any such IP address. Usually, the client would pick a low-latency name server and leverage the fallback options in case the preferred server is unresponsive.

Second, DNS operators use anycast and load balancers to place multiple servers behind the same IP address. BGP-based anycast allows announcing the same IP address

[1]Source: Statista [191]; due to an NDA we cannot name the operator.

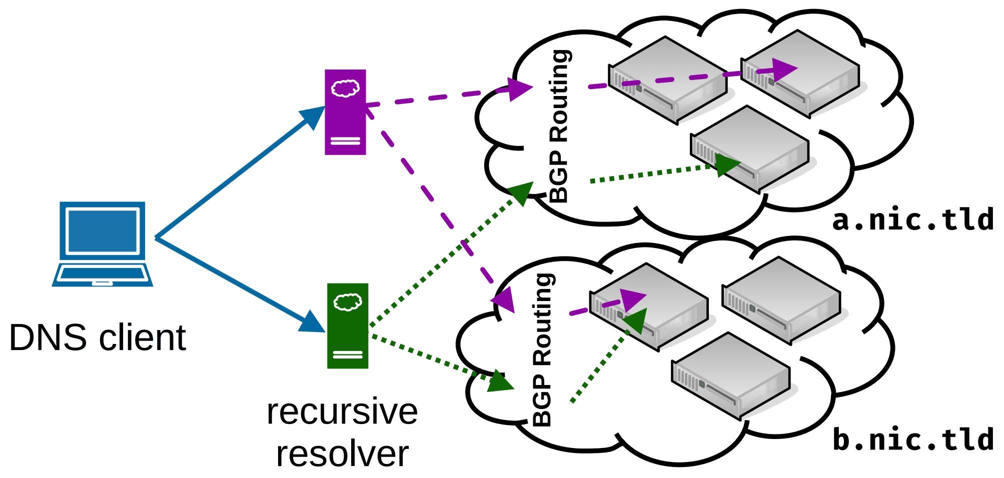


**Figure 7.1:** Example AuthNS setup with two `NS` records for two anycast clouds (`a.nic.tld`). Each cloud has multiple sites (blue servers). A resolver sends traffic to a specific cloud; BGP routing determines the site within a cloud.

space from multiple physical locations. Depending on the client's network origin, one of these anycast locations will receive and respond to the DNS request. The set of clients each AuthNS serves is its *catchment*. The catchment depends on the BGP announcement, the upstream provider connections, and the total number of locations announcing the same prefix. BGP allows operators to withdraw route announcements for faulty locations or add new locations to distribute the load more evenly. The convergence time is lower than the TTL of `NS` records, providing faster reactions. By using anycast, DNS infrastructures can place servers in the physical proximity of their clients, reducing latency, especially if it avoids transcontinental traffic.

Not surprisingly, most popularly used DNS infrastructures, therefore, use such anycast setups, as described by Sommese at al. [186]. They measured DNS infrastructures between 2017 and 2021. They find that 76.2 % of the TLDs consistently use anycast, and only 3.4 % are solely using unicast—the rest has a mixed setup. Furthermore, even the majority of *second*-level domains (SLDs) are now using anycast, with an ever higher rate among new SLDs. The adoption of anycast for SLDs is largely driven by big operators such as GoDaddy or Cloudflare. Given that these operators host millions of domains, their choice of anycast has a large impact.

### 7.4.2 DDoS Mitigation at the AuthNS

Regardless of the exact setup, DNS deployments face two main performance limitations: (i) compute power and (ii) network bandwidth [101]. Compute power is especially important for large zones with millions of registered domains, such as multiple European ones [2, 43, 142, 183, 207]. Large zones require more memory to store, and lookups become more expensive due to cache misses and necessary comparisons. Upload bandwidth can become scarce for DNSSEC-signed zones [202]. Cryptographic keys can be multiple kilobytes in size, and signatures take up hundreds of bytes. In the worst case, responses are up to 4 KiB large, where fewer than 500 kqps can exhaust a 10 Gbit/s connection.

An AuthNS can respond to DNS queries in the range of 100 kqps to 10 000 kqps [46, 101, 102], depending, among other things, on the zone file, software, configuration, and network connection.

To cope with traffic spikes, DNS operators leverage *over-provisioning* and provide more bandwidth and compute power than necessary for daily operations. Many DNS operators over-provision about 10× to 100× more than the daily peak [37, 46, 101, 195]. This is enough to mitigate many small DoS attacks, but is powerless against targeted DDoS attacks. Furthermore, over-provisioning is expensive, as it entails buying services and keeping them unused for most of the time. Being economically more attractive, on-demand scaling allows adding new resources to meet rising demands. It can cope with demands far larger than solvable with over-provisioning without permanently investing in unused infrastructure. Having said this, scaling needs to be set up and tested before an attack. Further, operational and policy constraints may apply and making this mitigation difficult or unfeasible to deploy.

*Scrubbing services* turn the concept of over-provisioning into a service. Here, the defense capability is shared and amortized among many customers. A scrubbing service announces the IP address space of the attack target to "steal" its traffic, then filters the traffic, and finally, forwards the cleaned traffic via a tunnel to the customer. Scrubbing services are effective against volumetric attacks, like amplification attacks, as they carry characteristic patterns. In such attacks, an AuthNS should never receive DNS responses, only DNS queries. However, they fail on application-layer attacks if the semantically valid attack traffic does not *per se* allow separating benign from malicious intent. This is further hampered if the scrubbing service is enabled only on demand, since that prevents the service from learning the typical traffic profile and identifying anomalies.

Finally, there are mitigations if attacks only overload a single anycast location but not the entire anycast network capacity. Operators can then use BGP to steer traffic from overloaded anycasted servers to less-utilized ones [166]. This shifts the traffic such that no single location is overwhelmed. The efficiency of this mitigation depends on the BGP capabilities at each location. Generally available techniques, such as BGP path prepending, are imprecise and only remove traffic from the prepended locations. Either way, BGP rerouting only works for smaller attacks, but as we will show, it cannot protect against large DDoS incidents that overload the entire anycast infrastructure.

## 7.5 Problem Statement

Attacks against the DNS infrastructure have devastating effects [39, 79, 139, 187]. In this work, we focus on powerful application-layer attacks in particular, as they—surprisingly—represent an attack class for which DNS operators currently lack solid defenses. This section clarifies our threat model (Section 7.5.1) and then shows that existing defenses do not cope well with application-layer attacks (Section 7.5.2).

### 7.5.1 Threat Model

In this work, we consider a powerful application-layer attacker that aims to overwhelm AuthNSes with valid DNS queries. We focus on this attack category, as such attacks are harder to defend at the network layer (e.g., by scrubbing services). Unlike semantics-ignoring floods, defenses against application-layer attacks require application context. That is, we assume attacks that flood AuthNSes with syntactically and semantically valid DNS queries, either directly or via intermediaries (recursives). Requests can be for random non-existing names, similar to random subdomain attacks [32, 197], or to the millions of existing domains [2, 43, 142, 183, 207]. For the remainder of this work, the actual query content is irrelevant. In fact, we assume that a defense cannot filter for benign requests based on their content.

We envision attackers that use real systems to attack directly (e.g., by a botnet) or indirectly (by using closed and open resolvers as intermediaries). Using intermediaries allows attackers to hide the true originating source IP address and abuse resolvers that benign users might also use. This makes detection a bit trickier, as we may falsely filter traffic of benign users ("false positives").

We assume that attack traffic does *not* use IP address spoofing. Spoofed traffic can be filtered upstream, e.g., by enforcing that clients complete a handshake. This can be achieved with TCP, which is an implementation requirement [44] for any DNS software, or by using DNS Cookies [56, 193]. Both the TCP handshake and DNS Cookies require the client to echo back a value sent by the server, thus enforcing the validity of the source IP address.

### 7.5.2 DNS-Specific Non-solutions

Although various DNS features may mitigate DDoS attacks, they are no match for them in our threat model. A significant limitation is that many either require support inside the resolvers or run fully resolver-side. As such, an AuthNS cannot expect these techniques to function since (i) many resolvers lack support and/or (ii) direct attacks do not need to adhere to these features. For completeness, we iterate these defenses below.

**Negative caching:** Aggressive negative caching [68] enables a resolver to answer new queries without any additional requests. This only works for DNSSEC-signed zones and only if the `NSEC3` opt-out mode is not used. Dynamic signing [70, 215] can also prohibit the usage, and DNSSEC deployment is still quite low, such that not many zones can benefit. Many TLDs are signed, but often using the opt-out mode since it requires fewer signed resource records, such that these TLDs cannot benefit. Using DNSSEC induces extra load on the AuthNSes since more records need to be stored, and upload bandwidth can become a bottleneck due to the large size of public keys and signatures. Finally, negative caching does not apply to *existing* records—as TLD zones that usually have tens (or hundreds) of millions of records, attackers can abuse those despite caching.

**RRL:** RRL [73, 208] limits the number of responses a DNS server sends per client. RRL thereby effectively limits the impact of name servers during amplification attacks. However, filtering is done only *after* generating a response. RRL thus only reduces upload bandwidth but does not save inbound bandwidth or processing time. In fact, processing requirements may even increase due to additional bookkeeping. To be effective in our scenario, RRL would have to be applied at the DNS client side. Indeed, some recursives even implement limits on outgoing queries to an AuthNS [74, 75], reducing overall traffic once the AuthNS is overloaded. Having said this, attackers are not bound to adhere to this pattern and can avoid such resolvers or even implement their own iterative lookup logic.

**Stale Records:** Lastly, recursives may mitigate the impact of an AuthNS's outage and serve records past their TTL time, so-called *stale records* [116]. This is effective in reducing queries while the AuthNS is unresponsive, but requires resolver support and deprives the AuthNS of its power of updating the records' data. Stale records can thus only be seen as a fallback "solution" to a situation (outage) that should really be prevented more proactively.

## 7.6 Methodology

We introduce an anomaly filtering methodology to mitigate non-spoofed application-layer attacks against AuthNSes. Before doing so, we explain the design goals of our envisioned defense in Section 7.6.1, then present our proposed defense methodology in Section 7.6.2, and discuss its deployment in Section 7.6.3.

### 7.6.1 Design Goals

In consultation with a partnering TLD operator (cf. Section 7.7.1), we derive eight design goals for a DNS application-layer DDoS defense:

**G1: Offer DDoS Protection, even for Large TLDs.** First and foremost, the defense should mitigate DDoS attacks such that only a manageable traffic level reaches the AuthNS infrastructure. It is an explicit non-goal to eliminate *all* attack traffic; it is sufficient if any remaining malicious queries do not overload any of the AuthNS's resources. Typical AuthNS can easily cope with up to 500 kpps, but struggle with loads higher than 10 Mpps [46, 101, 102]. Any defense should keep the attack volume below the former query rate, even if the AuthNSes hosts a reasonably large zone (e.g., a TLD).
**G2: Independence from Client Support.** The DNS ecosystem consists of various DNS resolver implementations, versions, and configurations [11, 103, 152, 200, 205]. Thus, any DDoS defense must not rely on AuthNS clients implementing certain optional DNS features that theoretically could mitigate an attack.
**G3: Upstream Filtering and Learning.** If attack traffic arrives at the AuthNSes, it is "too late". Therefore, we study defenses that can be applied upstream by providers or scrubbing services. The filter needs to integrate into the current environment of servers and anti-DoS solutions. They are the only locations with the capacity to remove attack traffic before it overloads the servers or data centers' network connection. The simplest filter with wide support is an allowlist/denylist of IP prefixes (in our setting, networks of recursives). The lists are easy to share, well understood, and have support for upstreaming, e.g., using BGP FlowSpec [122, 123]. An upstream provider should even be able to derive the filtering policies.
**G4: Content-Agnostic Filtering.** Defenses should not rely on DNS request content. First, we want to minimize privacy leaks and do not want to require an upstream to constantly inspect privacy-sensitive DNS communication content to filter attacks. Second, content filters are unreliable—attackers can easily adapt their tactics by querying arbitrary (semantically valid) DNS records.

**G5: High Evasion Resilience.** Acknowledging that any defense can be evaded by attackers with an unlimited budget, we aim to resilience against evasion to render evasion attempts intractable in practice.
**G6: Low False Positive Rate.** The ultimate dilemma any attack prevention mechanism faces is that it has to distinguish between benign and malicious intent. In our setting, we believe that a defense does not need to filter *all* malicious or anomalous traffic. The ccTLD operator has sufficient over-provisioning for everyday events, and together with caching in DNS resolvers, this is a solid baseline defense. In fact, overzealous blocking causes collateral damage. Filtered benign requests harm clients such as critical infrastructures, given DNS's central role in many networks. Such false positives even risk amplifying the attack due to the retry behavior of clients.
**G7: Ease of Use and Low Costs.** We believe DDoS defenses have to be easy to use, especially in pressing and hectic times, such as during a DDoS attack. Defenses should incur minimal costs before deployment and instead rely on *existing* network infrastructures and features. For example, the proactive collection of traffic statistics for IP prefixes can be done inexpensively on an AuthNS's router (or its upstream) in the form of NetFlow [3, 25, 26] data or a sampled stream of IP packets.
**G8: Resource Neutrality.** Defenses should not rely on adding resources at the DNS operator. While more resources may indeed provide a mid-term solution to higher loads, the defense should be able to handle temporal DDoS attacks that last 1–2 days max.

### 7.6.2 Anomaly Filtering

With these design goals in mind, we propose a defense based on anomaly detection specifically tailored for DNS operators. The key idea is to learn an expected traffic level for each source and use this to identify sources that send unexpectedly high amounts of traffic. This idea follows the observation that a DNS operator's traffic distribution matches a long-tail distribution, in which only a few sources are responsible for the vast amount of traffic. Consequently, we aim to filter traffic of DDoS attacks that abuse previously unobserved high-volume clients, independent of the content of queries (G4).

We approach this by classifying resolvers into three different traffic behaviors. (i) *Low-volume clients* have a relatively low traffic volume and never show traffic spikes. Low-volume clients can be active throughout an entire measurement period or only for a short time. (ii) In contrast, *steady high-volume clients* send a continuous, large stream of DNS queries. (iii) Finally, and most challenging, we observe *sporadic high-volume clients* which emit high but sporadic query spikes, e.g., outlier spikes that only occur temporarily in a longer measurement interval.

To judge which category a client falls in, we monitor content-agnostic (G4) traffic statistics at the AuthNSes. Such data is available to both the DNS operator itself and its upstream providers (G3). We split a training dataset into time intervals and traverse this dataset using a sliding window of length $W_{train}$ over time intervals of one hour. Per window, we count how many queries each source IP network (/24 for IPv4 and /48 for IPv6) sends. We regard a network as a steady high-volume client if it is active for at least $STEADY$ time intervals and shows an average query rate of at least $HEAVY$ packets per hour. All other networks either contain low-volume clients or sporadic

| window: | w1 | w2 | w3 | w4 | w5 | w6 | w7 | w8 | w9 | w10 | w11 | w12 | w13 | w14 | w15 | w16 | w17 | w18 | Active | Max $T_i$ | Parameter | Value |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Resolver A (allowlist) | 19 | 28 | 21 | 18 | 15 | 18 | 29 | 29 | 30 | 35 | 33 | 31 | 26 | 22 | 20 | 17 | 22 | 18 | 12 | 35 | HEAVY | 10 |
| Resolver B (allowlist) | 11 | 14 | 4 | 7 | 19 | 15 | 5 | 4 | 8 | 19 | 9 | 10 | 88 | 74 | 99 | 86 | 70 | 77 | 12 | 19 | LPF | 20 |
| Resolver C (LPF) | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 5 | 1 | 0 | 0 | 2 | 3 | STEADY | 5 |
| Resolver D (LPF) | 0 | 0 | 2 | 0 | 4 | 0 | 0 | 0 | 5 | 12 | 0 | 0 | 5 | 30 | 10 | 23 | 0 | 2 | 4 | 12 | TOL | 2 |

**Figure 7.2:** Application of our algorithm on an attack in windows w13 to w18 with four resolvers. The "active" and "max $T_i$" columns show if the value is large enough for allowlist inclusion. Resolvers A and B were added to the allowlist, and resolvers C and D were too low volume during the training phase. Resolver A behaves normally during training and attack, such that all its traffic passes the filter. An attacker abuses resolver B to launch an attack, such that we filter B's traffic (orange/red color), causing FPs in the benign fraction of B's traffic. Resolver C maintains a low query volume ($\leq$ *LPF*) and does not experience FPs. Finally, resolver D's temporal volume spikes cause a few FPs.

high-volume clients only. Figure 7.2 shows one example run of this algorithm, where the first twelve windows are used for training, and the next six show the attack. The color during the attacks shows if traffic is unrestricted (green) or if some traffic is filtered (orange–red). The "active" and "max $T_i$" columns show if the value is large enough for allowlist inclusion.

**Allowlist Creation:** We now leverage this classification to build a defense that allows continuous operation of the benign clients while blocking abnormal query profiles. To this end, we build an allowlist that includes steady high-volume clients. For each client network $i$, we record its maximum past traffic level $T_i$, measured in packets per hour. We use this past traffic level to detect if an IP network sends unexpected amounts of traffic during the attack phase. To cope with situations in which a benign client increases its query volume, we introduce a tolerance factor $TOL$. During the filtering period, for an allowlisted client $i$, we only block all $i$'s queries that exceed $T_i \times TOL$. This combination provides flexibility in how strict the filtering is as it compensates for traffic variability.

The allowlist does not capture low-volume and sporadic high-volume clients. We exclude low-volume clients to keep the allowlist reasonably small and manageable (G3). Ignoring sporadic high-volume clients makes it harder for an attacker to poison the training phase (G5). Yet, combined, these two query profiles likely still contribute a significant amount of the overall traffic. Just blocking all low-volume clients would create too many false positives, violating G6. Therefore, we permit a certain low traffic volume (*LPF*) for any traffic source not on the allowlist. Each non-allowlisted origin network can send as many queries as this low pass filter allows. Traffic exceeding this limit will be blocked. This policy allows low-volume sources to always reach the AuthNS—irrespective of whether the low-volume client sends only sporadic or continuous traffic. The *LPF* also allows queries of new low-volume clients, i.e., those not part of the training phase. The *LPF* only (but unavoidably) penalizes high sporadic traffic spikes of benign clients.

**Site-specific Allowlists:** Modern DNS infrastructures operate multiple logical AuthNS, separated by clouds (i.e., anycast IP addresses) and topological locations. We call each such logical AuthNS a *site*. Each combination of physical location and anycast IP address has a different catchment, meaning that IP networks visible on one anycast IP address for a location may not appear for the other anycast IPs. As client query behaviors may differ substantially between sites, and to minimize allowlists to ease deployment (G7), we train site-specific allowlists instead of a global one.

**Filtering Phase:** The filtering phase immediately succeeds the training phase and is assumed to contain attack traffic. During this phase, any source network may send up to *LPF* queries and allowlisted networks even more according to their past query volumes and the tolerance factor. Traffic that exceeds these limits will be blocked. We do not assume a certain length on the attack; we show in Section 7.7.4.4 the impact of attack lengths on detection accuracy. Our current prototype does not leverage retraining when under attack. That is, the allowlist is fixed as long as filtering is active, such that it ages and will slowly drift away from the current resolver population—an effect that we evaluate in Section 7.7.5. Having said this, we envision continuous retraining until a noticeable attack starts. This way, defenders have a recently trained detection model at hand when they most urgently need it.

### 7.6.3 Upstream Filtering

After having described *how* we filter traffic with our anomaly detection, we now discuss *where* filtering occurs. Filtering attacks on the AuthNS infrastructure happens too late. The downstream connection to the server or data center might already experience packet loss, or the server cannot handle filtering fast enough. As such, it is beneficial to offload filtering to upstream parties with sufficient bandwidth and processing power (G3). Commercial services such as *scrubbing services* or upstream ISPs/IXPs are perfect candidates. Such upstream services can filter and train the allowlist—all under the assumption that they have constant access to the network traffic to learn its characteristics. Indeed, the curated aggregated traffic statistics required for training can be captured/provided by NetFlow-like summaries that retain user privacy.

In some situations, upstream training is impossible or undesired, though. If scrubbing services are enabled "on demand"—only to mitigate an ongoing attack—they lack past behavior to build accurate defense models. In fact, AuthNS operators may deliberately choose to enable scrubbing services only when necessary (i) to save costs, (ii) to preserve user privacy, and (iii) to improve latency. In addition, upstream providers may lack the incentive to train behavioral models for their customers. In these cases, the DNS operator can train the allowlist themselves and push its filter decisions upstream, e.g., using standards such as BGP FlowSpec [122, 123] or proprietary APIs provided by upstream providers. The allowlist enforcement can then be implemented in software, such as using eBPF/XDP, or even closer to the hardware in P4-programmable switches.

**Costs:** Finally, we analyze the costs (G7) for the envisioned deployment. We discuss the allowlist creation and filter enforcement separately. The allowlist maintenance does not incur extra costs. TLD operators can easily (or already do) record statistical traffic information such as NetFlow data. For moderate training intervals of a few days, storage costs are negligible; so are the computational costs for creating the allowlists, which mostly require inexpensive statistical analyses. The costs for filtering, i.e., the actual enforcement, are a bit harder to estimate. But assuming that larger TLD operators have subscriptions with scrubbing services anyway, they need agreements for allowlists paired with rate limiting—standard features of most scrubbing services. In the worst case, filtering requires a new appliance or P4-programmable switch at a one-time cost of

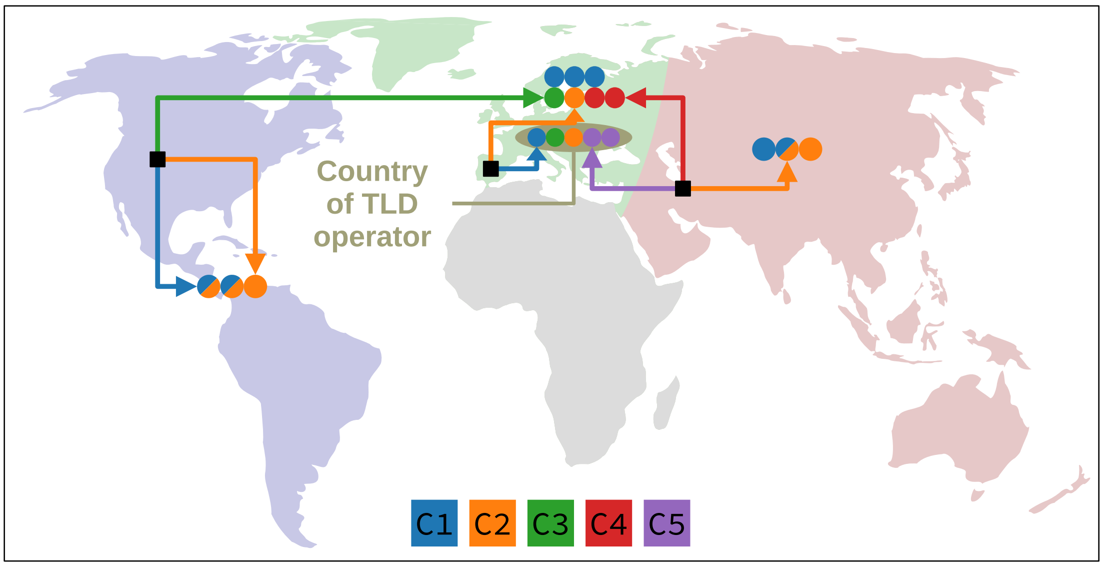


**Figure 7.3:** Approximate locations of AuthNS instances (circles colored by cloud), their anycast cloud assignment, and hypothetical DNS clients (■) using different anycast clouds (`C1–C5`). The country of the ccTLD, shown as the oval in Europe, contains five locations, remaining Europe (pale green) further seven locations, while South/North America (pale blue) and Asia/Pacific (pale red) have three locations each. The exemplified clients (■) have arrows in the colors of the anycast clouds, indicating how BGP could route them.

about $2000 per physical AuthNS location plus hosting fees. Overall, these worst-case costs are significantly lower than the economic damage of DNS outages at a TLD.

## 7.7 Evaluation

We now evaluate how effective an anomaly detection defense, as described in the previous section, can be against large-scale DDoS attacks. To this end, we acquire an attack-free DNS query dataset of a large TLD provider (Section 7.7.1). We then study the typical query behavior of benign clients in this dataset (Section 7.7.2). To model attacks, we obtain populations from two popular botnets and open resolvers (Section 7.7.3). We then train a detection model based on the benign dataset, optimize parameters, and evaluate the optimized model against the modeled attacks (Section 7.7.4). Finally, we assess to what extent concept drift (Section 7.7.5) and evasion attempts (Section 7.7.6) impede the proposed idea.

### 7.7.1 Dataset Description

We cooperate with one of the largest European (and global) ccTLD operators[2] to obtain a realistic evaluation dataset. The operator is authoritative for over 17 million domains, all of which are configured with a Time-to-Live (TTL) (caching) time of 24

[2]We redact the concrete TLD for anonymity.

**Table 7.1:** Capture statistics of both anycast clouds in the NetFlow dataset. The prefixes and packets are split evenly.

| Cloud | Locations | Prefixes v4 | Prefixes v6 | Packets |
|---|---|---|---|---|
| C1 | 8* | 968 915 | 105 565 | 4 563 713 484 |
| C2 | 7* | 928 893 | 101 790 | 4 683 252 517 |

hours. The anycast setup of our ccTLD consists of five anycast clouds, each having two to eight locations and visualized by Fig. 7.3. These locations are distributed over multiple countries. Overall, five to seven anycast locations are in the ccTLD country itself, Europe, and the rest of the world, respectively. Some physical locations are part of multiple clouds; we call a unique *(cloud, location)* tuple a *site* to resolve ambiguities.

Our TLD partner gave us access to four weeks of traffic statistics recorded at sites in the two largest anycast clouds. The collection period was 2022-05-13 06:00 UTC to 2022-06-09 06:00 UTC. To make data size manageable for the operator, the recorded data is randomly sampled per IP packet at a rate of one to ten. Due to processing errors, one location is missing 18 hours, and another one was not monitored at all; having said this, the dataset still represents a vast majority of the DNS query behavior observed at these two zones. The traffic and observable IP networks are pretty even between the two anycast clouds, see Table 7.1. Overall, our dataset spans 9 246 966 001 inbound packets from 1 185 657 different IP networks. The traffic per physical locations ranges from 94 950 430 packets (31 544 networks) to 2 015 510 497 packets (from 511 063 networks). Due to the 1:10 sampling rate, the real number of packets (and hence, DNS requests) will be larger by a factor of ten.

**Data Collection Ethics:** Our partnering TLD provider minimized data collection to preserve user privacy to the best extent possible. First, the dataset only contains NetFlow data, such that no DNS query names or responses are recorded. Instead, we infer the number of queries from the number of packets per flow and direction, with the approximation that an IP packet corresponds to exactly one query (modulo sampling). Second, the collected data is aggregated based on BGP routing granularity of /24 for IPv4 and /48 for IPv6 to hide individual users. Third, the datasets were protected by strict access control.

**Sampling Bias:** Due to sampling, we miss nine out of ten DNS requests. Random sampling statistically guarantees that we only miss low-volume senders, all of which will pass any filter due to the LPF. This implies that any false positives rate reported in the following is an upper bound, and would likely be a few percent lower on unsampled data. To estimate *how many* IP networks we are missing due to the sampling, we inspect a secondary dataset that consists of 24 hours of unsampled traffic recorded on 2021-01-14 from 14:57 UTC.

To this end, for each IP network in the unsampled dataset, we compute the probability that this IP network would not appear in a sampled dataset. This probability is $P = (1 - 0.1)^n$ for $n$ queries received by the IP network. Summing the individual probabilities tells us how many IP networks we are likely missing. Indeed, our main dataset does not observe 30.5 %–42.9 % of IP networks per physical location, with a single exception where only 24.8 % of IP networks would be uncaptured. However, over 99.5 % of these IP networks send fewer than 32 queries during the whole day, and 99.99 % send fewer than 64 queries during the day. Even though we miss a significant portion of IP networks, these networks only contribute 3.7 %/5.4 % of the overall traffic.

### 7.7.2 Benign Client Behavior Analysis

Before turning to the evaluation of the defense itself, we analyze the DNS query behavior of client networks in our dataset. The behavior of the AuthNS's clients is quite diverse. Most IP networks send few overall packets and are ephemeral, only active for a couple of hours during the whole time. Most IP networks send fewer than ten packets per hour yet contribute less than 7.5 % of the recorded traffic. The networks with the highest query rates are active over 90 % of the time and are likely resolvers of ISPs, cloud operators, and public DNS offerings, since these have always active and large userbases. There are some high-volume clients with overall short temporal activity, which might be problematic for anomaly detection.

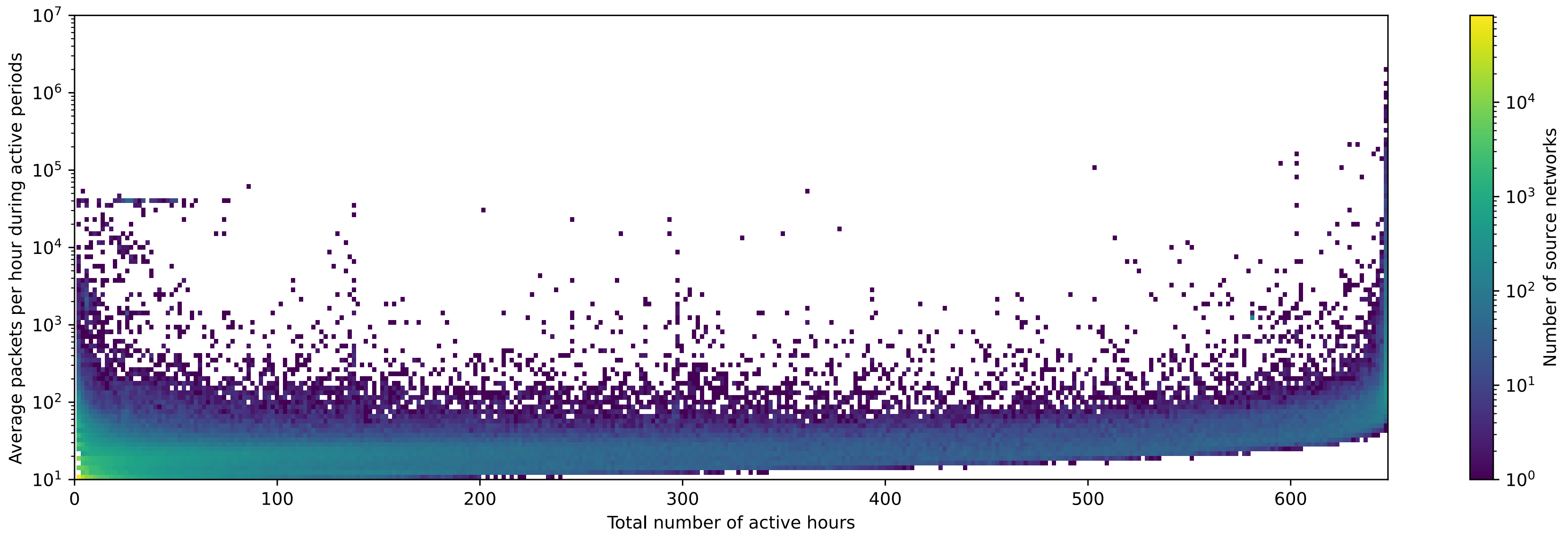


**Figure 7.4:** The histogram depicts the number of networks with packet rates and activity times. The x-axis shows during how many hours the IP network was observed, while the y-axis represents the average packet rate during the active period. The color shows how common each combination is, from non-existing white, over rare purple, to a common yellow.

The heatmap in Fig. 7.4 shows the most common combinations of average packet rate in packets per hour (y-axis in log scale) and the number of hours a network was active (x-axis; max. 682 hours). The color encodes how common these combinations are, ranging from purple (least number of networks) to yellow (high density of networks).

The graph shows the highest concentration towards the bottom-left corner. This shows that most networks in our dataset are only sporadically active and send packets at low rates. In fact, most IP networks send fewer than ten packets per hour, yet they contribute less than 7.5 % of the recorded traffic. The IP networks with the highest traffic rates are located on the right in the graph. This means that there is a stable set of IP networks, which are active throughout the observation period and send an order of magnitude more traffic than other networks. These are likely to be resolver infrastructures of ISPs, cloud operators, and public DNS offerings since these are always active and attract large user bases. There are just a few sporadic high-volume clients, as indicated by the upper left part of the histogram. These source networks can be problematic for anomaly detection because they appear briefly but have significant traffic compared to other sources.

The overall very moderate query rates are not unexpected, as the resource records in the ccTLD zone can be cached for up to 24 hours. Overall, this seems to confirm that most client networks fit the proposed low-pass filter well. Having said this, the analysis also shows that there is a need for exceptions using an allowlist. Larger query rates, as observed in the heatmap, can occur for multiple reasons. First, clients can look up millions of different domains. The more popular a resolver, the more domains it will query. Second, some recursives do not adhere to the caching period and fetch data more frequently, which can cause hundreds of millions of queries over a day due to the larger zone file size. Third, some networks have a large concentration of cache-independent resolvers, all of which are merged in our analysis.

### 7.7.3 Attack Modeling

We now model attacks that target the DNS infrastructure as outlined in our threat model (Section 7.5.1). We thereby distinguish between the type of traffic sources an attacker chooses. As a single attack source will easily be blocked by the proposed filter without causing much collateral damage, we assume distributed attacks. We envision two possible attacks. First, as we explain in Section 7.7.3.1, attackers can relay attack traffic via DNS resolvers. Second, as we discuss in Section 7.7.3.1, a large network of compromised hosts (“botnet”) may attack the AuthNSes directly. After discussing these attacker models in more detail, we use them in evaluating the proposed methodology. Figure 7.5 visualizes both attack vectors.

#### 7.7.3.1 Indirect Attack via Open Resolvers

Open resolvers represent an obvious choice for distributing an application-layer attack against DNS. The intermediary recursives hide the attacker’s identity and enable distributed attacks even for single-homed attackers. Furthermore, resolvers are topologically distributed, and thereby also distribute the attack among all anycast locations.

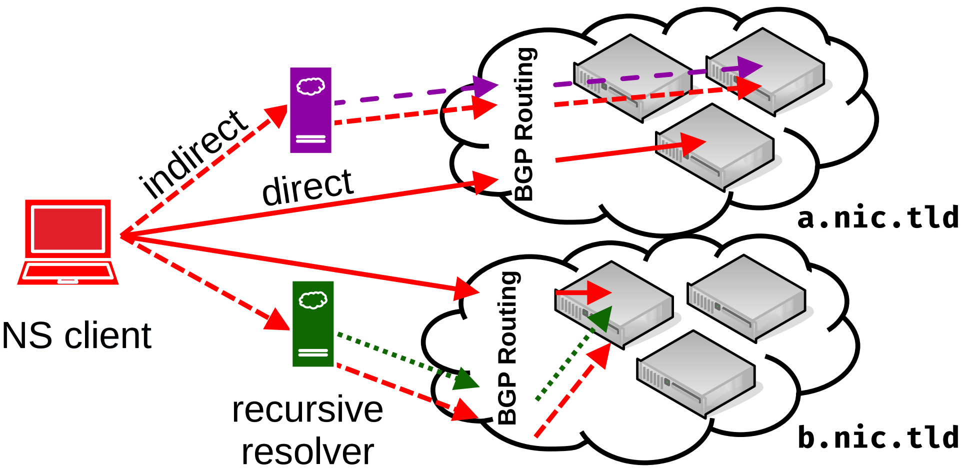


**Figure 7.5:** The attacker can attack indirectly by abusing benign resolvers (dashed lines) or query anycast clouds directly (solid lines). Both vectors lead to different sites (servers in the cloud). Indirect attacks increase the risk of false positives if regular clients use the same resolver for issuing benign requests (dotted lines).

The attacker can launch an indirect attack by abusing pre-configured resolvers of ISPs, and anycast or unicast open resolvers. The resolvers all act conceptually similar, but differences between anycast and unicast are important. Anycast resolvers, to which most public DNS offerings belong, have a wide range of locations sharing the same IP addresses. Thus it is likely that a location is "close" to the attacker and will be in the same catchment as the attacker. Alternatively, attackers can abuse unicast recursives. They can be selected such that they have diverse network views and can reach all locations of the anycasted AuthNS.

To model an attacker that abuses open resolvers, we scanned the IPv4 landscape of open resolvers via Zmap [53] and a custom DNS module. To this end, we follow the recommended scanning guidelines by Zmap and allow networks to opt out of the scans. The scans were performed on 2022-06-09 at reduced speed and lasted for 33 hours. Our scan generally identifies two types of resolvers: forwarders and recursive resolvers. That is, we find (i) the *client-facing* "scannable" recursives that receive and respond to the query, which may (optionally) forward the query to (ii) *server-facing* recursives that perform the actual iterative resolution at the AuthNS. Furthermore, by including our own AuthNS infrastructure in the DNS resolution chain, we can map the client-facing scanned IP address (by encoding it into the query name) to the server-facing IP address of the recursives that perform the authoritative lookups.

Our IPv4 scan covered all geographic regions except Russia.[3] We find 919 174 scannable IP addresses in 374 464 networks of size /24 that contact our AuthNSes. The recursives forward the DNS queries to 48 263 recursives in 24 942 networks.

[3]We excluded Russian target IP addresses from our scans as requested by our network operator due to the political situation.

#### 7.7.3.2 Direct Attack via Botnet

Instead of abusing resolvers, the attacker can also abuse a set of geographically diverse bots to steer traffic directly to the victim AuthNS. The diverse geographical locations of botnets allow the attacker to reach all locations in the anycast cloud(s) to attack all locations simultaneously.

To model botnet attackers, we analyze the population of two large in-the-wild botnets. (i) *Sality* is a peer-to-peer botnet targeting Microsoft Windows systems. On 2022-05-28 we discovered 53 824 IPv4 bots by crawling the Sality botnet. (ii) *Mirai-like* bots are all hosts showing the typical SSH and Telnet protocol scanning behavior. We collect all SSH and Telnet connection attempts to a /20 large IPv4 darknet, i.e., a network without any hosts. We collected 152 486 IP addresses in 100 366 Mirai-infected IP networks between 2022-05-16 and 2022-06-22.

We then mapped these bots to their available attack bandwidth. To this end, we leverage the Measurement Lab [132] Network Diagnostic Tool (NDT) dataset, using the `measurement-lab.ndt.*` subset. This dataset was created via a dedicated network speed measurement website, where users can measure their Internet connection. IP addresses without a measurement get attributed the performance of the next smallest supernet, i.e., we take the data associated with the /24, /22, /20, or /18. If that is insufficient, we use the average bandwidth of all bots at a specific anycast location.

Using this dataset, we find that the Sality bandwidth ranges between 201 Gbit/s to 2827 Gbit/s (average 564 Gbit/s) while Mirai has between 1149 Gbit/s to 8947 Gbit/s (average 2675 Gbit/s). We use the maximum bandwidth for the later evaluations for a worst-case analysis. The average bandwidths translate to 705 Mpps and 3343 Mpps for Sality and Mirai, respectively. This analysis reveals that attacks would have a devastating effect on the anycast setup of our partnering TLD operator.

### 7.7.4 Learning Parameters / Accuracy

We now evaluate the effects of these attackers against the DNS infrastructures of our partnering ccTLD operator. To this end, we first define the metrics we will use throughout the evaluation (Section 7.7.4.1). We then describe how we select (Section 7.7.4.2) and optimize (Section 7.7.4.3) the input parameters of the anomaly detection. Finally, we evaluate how well the defense can mitigate the attacks (Section 7.7.4.4).

#### 7.7.4.1 Evaluation Metrics

We follow standard terminology in our evaluation. We define malicious traffic as *positive* and correctly filtered DNS requests as *true positive* (TP). Conversely, benign traffic is *negative* and benign DNS requests that bypass the filter are *true negatives* (TNs). A *false positive* (FP) is erroneously blocked benign traffic, which can occur if a traffic source exceeds the allowed traffic level $T_i$ or $LPF$, respectively. A *false negative* (FN) refers to undetected malicious traffic that bypasses the filter, e.g., all attack packets per source that do not exceed $LPF$ packets. A FP negatively interferes with DNS lookups of benign clients and has negative consequences. In the best case, FPs just cause delays in DNS resolutions, potentially opening resolvers up for further attacks; in the worst case,

**Table 7.2:** Overview of the parameters we use in the methodology and evaluation.

| Parameter | Description |
|---|---|
| $HEAVY$ | Minimum traffic level (in packets per hour) before an IP network can be added to the allowlist. |
| $LPF$ | Traffic level (in packets per hour), which is allowed without an allowlist entry for the IP network. |
| $STEADY$ | Minimum number of active hours before an IP network can be added to the allowlist. |
| $TOL$ | Factor to use on $T_i$ to calculate the allowed traffic during the filtering phase, past which queries from the IP network will be blocked. |
| $W_{test}$ | Windows size during the test phase, i.e., the attack length. |
| $W_{train}$ | Windows size during the training phase. |

they render entire services (e.g., websites) unavailable. In contrast, a FN just adds to the load and does not have any immediate negative consequences if there are sufficient infrastructural resources. In the following, we argue to minimize FPs while maintaining an acceptable level of FNs.

Based on these classifications, we use standard evaluation measures, such as false positive rate $FPR = \frac{FP}{FP+TN}$ and false negative rate $FNR = \frac{FN}{FN+TP}$, and their inverses $TNR = 1 - FPR$ and $TPR = 1 - FNR$. To combine both FPs and FNs, we report on the F-score $F_\beta = \frac{(1+\beta^2)*TP}{(1+\beta^2)*TP+\beta^2*FN+FP}$ and balanced accuracy $BA = (TPR + TNR)/2$. Having said this, in the following, we optimize for and report on the FPR, as doing so still reaches an acceptable number of FNs.

#### 7.7.4.2 Parameter Values

We now perform a grid search to find reasonable values for the *training parameters* we declared in Table 7.2. To this end, we first specify concrete value ranges for each parameter. We vary the training time ($W_{train}$) between one hour and 73 hours. The minimum traffic level for allowlist inclusion ($HEAVY$) ranges between 64 pph to 256 pph (packets per hour). The minimal active hours ($STEADY$) range from one to twelve hours, with the constraint that they are always smaller than $W_{train}$. The low-level traffic threshold ($LPF$) varies between 128 pph to 2048 pph. We also allow traffic sources to exceed the traffic levels ($TOL$) recorded in the allowlist by the factor one (no exceeding), two, or four.

The attack is determined by another set of *test parameters*. The attack starts immediately following the training phase (we will evaluate more outdated training models in Section 7.7.5) and lasts between an hour and three days ($W_{test}$). For the simulation, we model different attacker bandwidths from 1 Gbit/s to terabit speed, depending on the attacker model, and simulate different attack sources, such as open resolver and different botnets.

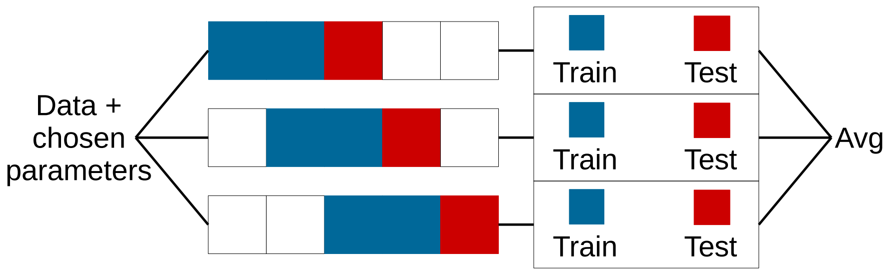


**Figure 7.6:** Symbolic representation of the grid search approach to find the best parameter combination. Given a fixed set of data and parameters, we pick distinct test intervals and compute the average of the performance.

#### 7.7.4.3 Parameter Optimization

We use a grid search on the aforementioned training parameters to find the best parameter choices. We consider those parameters ideal that minimizes the False Positive Rate (FPR). To this end, we leverage *sliding window cross-validation*, a standard cross-validation technique for time series that avoids mixing training and test data. It allows us to keep the chronological data order and still have multiple folds, as the test length is much smaller than our overall time series. In our evaluation, we test all parameter combinations and report on the one that worked best for a given site across all folds. Figure 7.6 illustrates the general idea.

As an alternative cross-validation concept, we also tried an alternative cross-validation technique that optimizes Parameters per sliding window instead of using a fixed parameter configuration for all folds. This technique yields similar results yet requires constant parameter search. Consequently, in the following, we do not differentiate between the two methods and only report on results obtained from the global parameter search.

#### 7.7.4.4 Results

Figure 7.7 shows results for an attack using the Mirai-like botnet (top), and open resolver (bottom). The same graph for Sality is Fig. 7.8. More detailed statistics about the Mirai-like botnet are presented in Table 7.3 and for Sality in Table 7.4. The botnet uses the calculated maximum bandwidth, and the open resolver graph uses a theoretical 100 Tibit/s attacker to show the worst-case behavior in both cases. It shows the malicious traffic which needs processing after filtering (x-axis) in comparison to the FPR the filtering achieves (y-axis). Each point in the graph represents the evaluation result for a given site. Color shows different values for $W_{test}$, while the symbol shape represents $LPF$. As expected, the $LPF$ has a significant impact on the false negatives (x-axis) while the respective site and $W_{test}$ influences false positives (y-axis). In fact, the four chosen $LPF$ thresholds result in four clearly separate clusters. The lower the attack duration, the lower the FPR. Some sites have a larger FPR than others, for all attack lengths.

The graph shows how we can trade better FPR scores by allowing larger amounts of malicious traffic. Larger $LPF$ values mean that benign sources, which do not occur on the allowlist, are not as likely to be blocked and thus decrease the false positives by

**Table 7.3:** Detailed evaluation results for the Mirai-like attacker from Fig. 7.7a, since it has the worst FPR values. *Normal* counts each physical location identically, while in the *weighted* columns each location is weighted by the benign traffic levels. "Mdn" denotes the median.

| | | | Normal | | Weighted | | |
|---|---|---|---|---|---|---|---|
| $W_{test}$ | $LPF$ | Min | Avg | Mdn | Avg | Mdn | Max |
| 8 | 128 | 0.06 | 0.10 | 0.10 | 0.10 | 0.10 | 0.14 |
| | 512 | 0.03 | 0.07 | 0.07 | 0.07 | 0.07 | 0.11 |
| | 2048 | 0.03 | 0.04 | 0.05 | 0.04 | 0.05 | 0.07 |
| | 8192 | 0.02 | 0.03 | 0.02 | 0.03 | 0.02 | 0.05 |
| 24 | 128 | 0.06 | 0.11 | 0.10 | 0.11 | 0.10 | 0.18 |
| | 512 | 0.03 | 0.07 | 0.07 | 0.07 | 0.07 | 0.14 |
| | 2048 | 0.02 | 0.05 | 0.04 | 0.05 | 0.04 | 0.08 |
| | 8192 | 0.01 | 0.03 | 0.03 | 0.03 | 0.03 | 0.05 |
| 72 | 128 | 0.06 | 0.13 | 0.11 | 0.13 | 0.11 | 0.23 |
| | 512 | 0.03 | 0.09 | 0.08 | 0.09 | 0.08 | 0.19 |
| | 2048 | 0.02 | 0.05 | 0.05 | 0.05 | 0.05 | 0.10 |
| | 8192 | 0.01 | 0.03 | 0.02 | 0.03 | 0.02 | 0.06 |

**Table 7.4:** Attack traffic statistics for the evaluation results of Fig. 7.7. Only results for $W_{test} = 24$ are shown, as the test length only has minimal impact on the results. The rows contain the data for **S**ality, **M**irai-like, and open **R**esolver. *Normal* counts each physical location identically, while in the *weighted* columns each location is weighted by the benign traffic levels. "Mdn" denotes the median.

| | | | Normal | | Weighted | | |
|---|---|---|---|---|---|---|---|
| | $LPF$ | Min | Avg | Mdn | Avg | Mdn | Max |
| S | 128 | 345 | 593 | 570 | 593 | 570 | 809 |
| | 512 | 1380 | 2350 | 2282 | 2350 | 2282 | 3225 |
| | 2048 | 5522 | 9378 | 9128 | 9378 | 9128 | 12 883 |
| | 8192 | 22 087 | 37 495 | 36 511 | 37 495 | 36 511 | 51 519 |
| M | 128 | 818 | 1958 | 1482 | 1958 | 1482 | 8394 |
| | 512 | 3270 | 5848 | 5778 | 5848 | 5778 | 12 535 |
| | 2048 | 13 082 | 21 401 | 22 964 | 21 401 | 22 964 | 29 100 |
| | 8192 | 52 323 | 83 603 | 91 704 | 83 603 | 91 704 | 109 446 |
| R | 128 | 733 | 1229 | 1187 | 1229 | 1187 | 1736 |
| | 512 | 2932 | 4717 | 4565 | 4717 | 4565 | 6518 |
| | 2048 | 11 728 | 18 645 | 17 965 | 18 645 | 17 965 | 25 643 |
| | 8192 | 46 913 | 74 285 | 71 362 | 74 285 | 71 362 | 102 133 |

them. As a downside, more attack traffic will simultaneously be allowed. For example, consider $LPF = 2048$ pph, the second-highest $LPF$ threshold in our evaluation. Here, the malicious traffic is generally quite low, with less than 13 000 pps (left) or 26 000 (right) at any single location. This volume is well in the realm of traffic an AuthNS can handle, and thus prioritizing further reductions are not relevant. For 8-hour long attacks and $LPF = 2048$ pph, the FPR is 4.2 % on median, ranging from 1.7 % to 6.9 % for the Sality botnet, while ranging from 0.2 % to 5.7 % with a median of 2.5 % for the open resolver evaluation.

$W_{test}$ has a negative impact on the FPR due to population churn. As the allowlist ages, it no longer accurately mirrors the current state of benign resolvers. Resolver churn leads to "dead" entries on the allowlist, which no longer match a resolver. Likewise, new resolvers appear. If they exceed the $LPF$ they get blocked increase the FPR.

### 7.7.5 Concept Drift

In the previous section, we observed a negative impact of long time spans between training and testing. The main reason for this decline is *concept drift*, which occurs because the population of active resolvers changes by hour and day. If the allowlist is not adapted accordingly, previously unseen resolvers may cause false positives. We evaluate this churn by artificially creating longer time gaps between the allowlist creation and the attack. For this, we train a static allowlist per site on the first $W_{train}$ intervals in our dataset and test it on all possible future $W_{test}$ intervals. This experiment indicates how quickly the allowlist becomes outdated and how often retraining is necessary. As the amount of unfiltered attack traffic is generally lower with the fixed allowlist, we omit an analysis of traffic changes and focus solely on the FPR.

Figure 7.9 shows the performance impact of using a static allowlist—compared to the standard procedure of constantly retraining the allowlist (as described in Section 7.7.4.2). The graph shows the evaluation performed for a single exemplary large AuthNS location for three training lengths (1 hour, 1 day, 3 days) and a fixed $W_{test} = 24$ hours interval. Positive values indicate that the fixed allowlist had a higher FPR. The shorter the training interval, the higher the maximum negative impact on the FPR. In fact, using short $W_{train}$ results in a daily repeating pattern, where the static allowlist at times performs *better*. This is a result of diurnal patterns in the resolver population. Every 24 hours, the fixed allowlist matches exactly from the time of day it was created to the test interval, while the normal allowlist is shifted by at least an hour. However, this advantage is eaten up by large FPR increases at other times. Still, it shows a potential improvement in the allowlist creation to compensate for diurnal and weekly patterns. Instead of only training on the preceding $x$ hours, we could additionally train on the previous day and week.

Longer training times result in patterns that fluctuate less. But we also see larger dips for $W_{train} = 24$, resulting from weekday to weekend shifts. The first training window starts on a Friday but captures part of the weekend too. This has positive effects on those weekend parts. Only the 3-day training interval captures the behavior of a full workday *and* the weekend, thus compensating for the resolver population cycle. But this long training period cannot account for longer shifts over weeks or months.

Independent of the training length, a static allowlist always causes significantly more FPs than periodically trained allowlists. Based on these results, we conclude that periodic retraining is highly advisable.

### 7.7.6 Evasion

Any learning mechanism has to contend with smart attackers trying to evade the defense mechanisms. In our case, we see two evasion possibilities.

First, attackers could try to have their attack sources be allowlisted. For example, consider an attacker that adds attacker-controlled IP networks to the allowlist. This corresponds to a persistent attacker capable of placing their nodes undetected into the seemingly benign resolver population. For argument's sake, we simulate that each of the attacker networks is assigned a historic traffic level $T_i$ equal to the highest benign value in the allowlist. Figure 7.10 shows the result of the evaluation with the attacker being able to transmit up to 100 Tibit/s. The graph plots the number of IP networks that successfully evaded the allowlist (x-axis) and the amount of attack traffic the AuthNSes must process (y-axis). Each line represents a different site. We showcase a single configuration for our algorithm, the best-performing combination from Fig. 7.7 ($LPF = 2048$, $W_{test} = 24$). Overall, we see the trend that such evasion indeed leads to higher attack traffic. This is expected, as the bandwidth granted by the allowlist is significantly higher than the selected $LPF$ value. The increase per location heavily depends on the source of the allowlisted attack hosts. In general, though, one can argue that thousands of poisoned allowlist entries do not hurt. Even 30 000 allowlisted attack sources seem uncritical; in the worst case, this causes a few 100 000s of attack requests per location. Having said this, this analysis indicates that care must be taken not to allowlist too many attack nodes. Indeed, the attack traffic increases roughly linearly with the number of poisoned allowlist entries.

Second, attackers may try to influence the $LPF$ and $TOL$ parameters, such that IPs not on the allowlist can send much traffic. Having said this, we observed that variations to the $LPF$ and $TOL$ parameters lead to similar performance. Thus, retraining these parameters is usually unnecessary, such that attackers cannot influence them. Furthermore, attackers cannot significantly influence the parameters without the AuthNS operator noticing, as the changes stand out compared to previous models.

### 7.7.7 Insider Threats

Insider threats to our defense are possible. We define three levels of access, each leading to different threats: (i) knowledge about the algorithm parameters, (ii) knowledge of the allowlists per site, and (iii) modifications to the parameters or allowlist.

**Knowledge of the algorithm parameters:** Knowing the exact parameters enables an attacker to inject IP addresses into the allowlist during training more efficiently than without such knowledge. For each host to inject, a traffic profile using minimal theoretical bandwidth is selected. For example, each host only needs to be active for $STEADY$ intervals per training window. An insider with parameter knowledge can thus save bandwidth by not always using the full capabilities of an attack source.

**Table 7.5:** Performance of our prototype for three sites, ranging from small to large data volume.

| Value | Site Size | $W_{train} = 1$ | 3 | 6 | 12 | 24 |
|---|---|---|---|---|---|---|
| Data in MB | small | 4.4 | 13.3 | 25.0 | 41.9 | 89.1 |
| | medium | 9.5 | 26.6 | 50.4 | 115.5 | 273.2 |
| | large | 31.4 | 90.4 | 170.6 | 379.2 | 811.9 |
| Mean Time in s | small | 6.6 | 7.0 | 7.4 | 8.3 | 10.8 |
| | medium | 15.8 | 16.4 | 17.9 | 24.2 | 41.3 |
| | large | 51.5 | 53.9 | 58.1 | 76.4 | 102.1 |

**Knowledge of the allowlist:** An attacker informed about the allowlist could aim to DoS a listed resolver by sending traffic from the resolver's IP range(s). This exhausts the resolver's traffic budget with mostly malicious traffic, drowning out the resolver's benign queries. It requires the attacker to have IP addresses in the same range as the victim[4], which is possible for resolvers hosted by cloud and shared hosting provides, but not for organizations with their own IP space. The feasibility depends on the aggregation level of the IP space, with stronger aggregation (e.g., /16 for IPv4) making it easier.

**Direct allowlist or parameter modifications:** Finally, we consider an unlikely yet strong attacker that finds a way to directly modify the allowlist of algorithm parameters. Such a strong insider attack can render the defense useless by picking rogue parameter values, enhancing the allowlists, or DoS resolvers by emptying it. Having said this, using standard access control mechanisms, only trusted parties—the AuthNS operator or its upstream providers—have such privileges. In addition, if the insider can only manipulate data at a single anycast site, resolvers could switch to other (non-compromised) sites.

## 7.7.8 Prototype and Performance

We implemented a prototype of our algorithm to test its feasibility. The prototype is written in Python and uses nfdump [141] for parsing the NetFlow data. The prototype is available with our other source code at the locations listed on Page vii.

We ran the prototype on different NetFlow data and measured its runtime performance. Based on three sites, picked from small to large, we show how the performance scales with data size and the number of input files. Each NetFlow file contains the data for one hour. Table 7.5 and Fig. 7.11 show the runtime for different training windows ($W_{train}$) and data sizes. The runtime scales well with the data size and the number of input files as long as there are enough CPU cores available. A problem arises when the individual NetFlow files grow too large, as they cannot be processed in parallel anymore. This is the main reason why the large site runs 7.7 × to 9.4 × slower than the small site.

**Table 7.6:** Comparison of different anti-DDoS solutions. A ✓indicates a good value for the criteria, and parentheses show that it might not apply in all situations.

| | Our method | Reverse Proxy | Managed DNS |
|---|---|---|---|
| App.-Layer Flood | ✓ | (✗) | (✓) |
| Vol. Attack | ✗ | (✓) | (✓) |
| Privacy | ✓ | (✗) | (✗) |
| Latency | ✓ | $\updownarrow$ | $\updownarrow$ |

## 7.7.9 Comparison with anti-DDoS Vendors

Our proposed system allows the AuthNS operator to defend against the attack itself (possibly with the help of upstreams). Commercial anti-DDoS services also aim to protect AuthNSes against DDoS attacks. Their general mode of operations falls into two categories: *reverse proxy* and *managed DNS*. We compare these third-party services to our method, regarding their effectiveness and privacy—as summarized in Table 7.6.

**Reverse Proxy:** The anti-DDoS vendor can act as a *reverse proxy*, accepting traffic and forwarding valid DNS queries to the AuthNS operator. Technically, caching reverse proxies sit between the client (resolver) and AuthNS such that clients no longer query the AuthNS themselves. In principle, reverse proxies thereby significantly reduce the AuthNSes's load in that they only forward uncached queries. Assuming that the proxies have better network connectivity than the AuthNS operators, they also provide decent protection against volumetric attacks [107, 109, 171]—only valid DNS requests are forwarded to the AuthNS operator. Having said this, reverse proxies only provide limited defense against application-layer attacks that bust the cache. Given tens of millions of DNS records to choose from, plus non-cacheable queries to non-existing domains, attackers can evade the caches and flood the AuthNS infrastructure. There is a clear privacy downside, as a third party learns about the DNS queries. Finally, reverse proxies may not be able to retain the latency of the original AuthNS setup, especially if they are further away from the clients or still have to forward queries to the AuthNS. While there are opposite cases in which clients get cached responses faster, TLD operators that want to retain SLAs have to choose reverse proxy locations wisely.

**Managed DNS:** In the *managed DNS* mode, the anti-DDoS vendor directly answers any query with their copy of the AuthNS's zone file. The anti-DDoS vendor thus operates like an AuthNS operator. The big difference is that an anti-DDoS vendor operates a shared infrastructure for many customers. This results in a different financial model and likely larger infrastructure. If so, the managed DNS provider may even be able to protect against volumetric attacks. For application-layer attacks, the vendor will ultimately be similarly vulnerable as the AuthNS operator is for larger attacks that exceed its over-provisioning—unless protected by our defense. The privacy impact is similar to the reverse proxy, as a third party learns all DNS queries; even worse, it learns the zone file that may be deemed sensitive. The latency may change and depends on

[4]Recall that our threat model assumes that AuthNS operators can enforce TCP to render spoofing attempts useless.

the anycast setup of the managed DNS operator. Again, to retain latency-based SLA guarantees, the anycast setup would have to be similar (or superior) to the original one.

## 7.8 Discussion

In this section, we first check whether our proposed defense aligns with the design goals. We then discuss the relevance of our work in light of other defenses. Finally, we iterate threats of validity in our evaluation.

### 7.8.1 Reflection on Design Goals

We first discuss how our proposed defense meets the design goals as outlined in Section 7.6.1. By design, our proposed content-agnostic (G4) anomaly detection can be applied by upstream providers (G3) and does not require client support (G2). The low-pass filter retains compatibility with low-volume resolvers (G8). Our proposed anomaly detection defense is easy to use (G7); it just requires NetFlow data (or similar statistics) to learn statistical profiles of past resolver behavior. There are no extra costs (G7); all standard routing/switching hardware nowadays offers flow monitoring, and storage costs for up to 72 hours of NetFlow data are negligible. Despite this simplicity, the method filters sufficient attack traffic to avoid packet loss at AuthNSes (G1). To further underline this, the *LPF* threshold can even be lowered to reduce the load in case of overloads. This even holds if attackers can poison the allowlist with thousands of entries, showing a high degree of resilience against the major evasion angle (G5). At the same time, there are few false positives (G6), especially when compared to the alternative defenses (e.g., null routing) or no defense at all (high packet loss → high FPR). To conclude, the proposed methodology fully fulfills the design goals and thereby represents a vital last-resort defense for TLD operators.

### 7.8.2 Limitations

Our proposed methodology still has a few limitations, as discussed in the following.

**Catchment Changes:** Some mitigations, like changes in the BGP announcements, will influence the catchment area of the sites. This affects how well the computed allowlist matches and might lead to higher false positives. Computing a single shared allowlist across all sites alleviates that, as it guarantees that each resolver is allowlisted everywhere, but it is less specific.

**Millions of attack sources:** Massive botnets are problematic, as they can overwhelm the AuthNS with their sheer volume, as each network is allowed to send *LPF* amount of traffic. This can add up to significant amounts of traffic. It can be partially counteracted by decreasing the value for *LPF*, but this increases the number of false positives. The reduction of *LPF* will affect the benign resolvers equally, thus changes have to be made carefully.

**Spoofing-capable attackers:** IP address spoofing allows the attacker to mimic arbitrary sources. This not only allows mimicking millions of attack sources. It also offers the chance to DoS a specific set of victim resolvers. When our defense is enabled, attackers can spoof victim IP addresses to consume their traffic allowance. Detecting spoofing is simple when many IP addresses are affected but can be challenging when only a few are. In either case, the AuthNS operator can enable spoofing countermeasures (see Section 7.5.1), which effectively prevents these attacks.

**Coarse-grained filters:** Network aggregation may merge traffic from both benign and malicious sources into a single mixed entity. The attribution of network traffic is still correct. But both sources now compete for the same traffic allowance. If attackers send orders of magnitude more traffic, it prevents the benign resolver from accessing any AuthNS. This requires the attacker to have an IP address close to the victim, which is possible in the cloud or shared hosting environments, but impossible if the resolver has a dedicated IP address. The amount of aggregation is another contributing factor and can be reduced, i.e., longer prefixes, when this problem occurs. Longer prefixes result in more detailed attribution but increase the size of the allowlist and computation cost.

### 7.8.3 Relevance of Our Work

So far, we assumed that we have to use upstream filters to avoid overloaded AuthNSes (i.e., packet loss) at all costs. We see our proposed defense as the last level of defense if all other DDoS countermeasures fail. Again, for volumetric DDoS attacks, upstream content filters work well. Small application-layer attacks that overload a single anycast location can only be mitigated by traffic rerouting, as suggested by Rizvi et al. [166].

However, when **DDoS attacks overload the entire anycast infrastructure**, the countermeasures mentioned before are of no help. This is a situation that will likely occur in our modeled attacks (Section 7.7.3). If a standard botnet attacks a large TLD, the total attack bandwidth (or query volume) exceeds the resources of the entire anycast infrastructure by far. This underlines that we indeed need additional countermeasures like ours that are agnostic to the content and filter attack traffic upstream.

We acknowledge that, for ethical reasons, we have not tested the consequences of successful DoS attacks against TLDs. We consider an attack successful if the overall traffic exceeds typical query capacities of AuthNSes. Especially short-term attacks may not have as severe consequences, as caching and even stale records (Section 7.5.2) provide short-term mitigation. In general, it is hard to model the clients' tolerance to resolution failures. Given many layers of caching and uncertainty, determining the final impact of DDoS attacks is difficult. Having said this, past attacks have illustrated that there *are* negative attack implications. First, Moura et al. [139] report on two DDoS attacks against the DNS root servers between 2015-11-30 and 2015-12-01. The DNS root servers are separated into multiple anycast clouds, containing a couple to hundreds of servers each. Multiple anycast clouds failed under a load of up to 5 Mpps. End-user errors were reported, even though multiple anycast clouds were still operational. Second, Sommese et al. [187] report on further DDoS activity against various DNS servers between 2020-11 and 2022-03. They achieve greater visibility by combining DoS activity from a large darknet with active DNS measurements of the OpenINTEL platform, providing them with information about resolution times and failures. Most successful attacks are against smaller DNS providers, often without anycast deployments. Yet, larger providers with anycast infrastructures, such as Hetzner, Apple, or GoDaddy, are also affected. From these reports, we learn that successful DDoS attacks against DNS infrastructure happen and represent an open research problem. We hope that our work can help to mitigate future attacks.

### 7.8.4 Threats to Validity

In our evaluation, we tried to get as close as possible to a realistic attack by using real-world background traffic of a TLD and monitoring potential attack sources. While we believe these results are representative, there are a few factors that may influence the preciseness of the results. We iterate potential threats to validity in the following.

**Requests per lookup:** We modeled that each client lookup towards an open resolver creates a single query towards the AuthNS. In practice, resolvers may send multiple queries even during normal operations. For example, resolvers often issue the same query to multiple AuthNSes, e.g., to both an IPv4 and IPv6 server. During a successful DDoS attack, resolvers may even *amplify* the attack, as they do not receive responses to their queries. The default resolver behavior then is to issue the same query again [138]. Such repeated queries are an extra load the attacked server needs to handle. After multiple unsuccessful queries, the resolver might deem the AuthNS unresponsive and switch to another IP address or AuthNS. This, in turn, alleviates the load on the attacked server but may cause cascading failures at other AuthNSes. All these effects—the number of re-transmissions and how resolvers switch between name servers—are impossible to determine without performing an actual attack, as it depends on the software, configuration, and the specific connection to all AuthNSes. Thus, we excluded this effect from our analysis. However, this underlines once more that we have to shield AuthNS infrastructures early on, not to run into these amplification issues.

**Anycast cloud sizes:** In this chapter, we only looked at medium-sized anycast clouds. Other anycast deployments range from two locations to hundreds of locations for a single cloud, which influences DDoS resilience. Smaller clouds are more vulnerable, as they have less redundancy and fewer overall resources for traffic processing. They also have fewer catchment areas, allowing an attacker to more efficiently target a specific location or deploy attacking nodes in all catchment areas. Clouds with hundreds of locations are more capable of swallowing an attack and attracting more benign traffic.

Another observation is that not all locations within an anycast setup receive the same traffic share. Resolvers prefer locations with low latency. Anycast locations with beneficial routing (e.g., close to exchange points or backbone networks) will attract traffic from many networks. Other locations will be located further from the Tier-1 providers, and thus their route announcements will have limited reach. Given the larger catchment, this imbalance makes some locations easier to attack. In contrast, other locations are "protected" since the attacker will struggle to assemble enough attack bandwidth in their catchment areas. Furthermore, such highly imbalanced anycast setups have the potential for a cascading failure. If a high-volume location is overwhelmed and traffic switches to a relatively smaller location, that location will be overloaded.

Having said this, the mid-sized anycast clouds we studied provide a good balance and are representative of anycast setups of several other large ccTLD operators. Furthermore, the defenses we studied work independently of the concrete anycast setup.

**Attack sources:** Our evaluation consistently models the attacker using *all* available traffic sources. Given our filtering algorithm, this is the most beneficial choice for the attacker, as any IP network can contribute attack traffic up to the *LPF*. In reality, the attacker might not use all available traffic sources, e.g., as nodes are in the wrong catchment area or nodes provide less bandwidth than other sources. This is not a concern for our proposed filtering, as any deviation benefits the defenders, and thus we report worst-case results in the chapter.

As an alternative to using subsets of attack sources, attackers may find *larger* attack sets than the ones we studied. We monitored reasonably large botnets that contain up to ≈ 100k attack networks. Larger botnets existed in the past (e.g., Conficker), and would demand stricter low-pass filters to protect DNS infrastructures.

**Preconfigured resolvers:** ISPs operate DNS resolvers which are only accessible to their customers. A botnet could leverage these resolvers if they have compromised hosts in the ISP's network. These preconfigured resolvers are shared with many benign users, more so maybe than many open resolvers. From the AuthNS perspective, misuse of the preconfigured resolvers is harder to detect and mitigate since benign and attack traffic is blended together. Consequently, we risk blocking legitimate traffic. Having said this, ISPs can rate limit their clients to prevent massive abuse and mitigate the resulting damage to their resolvers' reputations. We cannot measure the population of ISP-provided resolvers due to a lack of access to these networks. Therefore, we do not report on the effects of abusing such resolvers in attacks.

## 7.9 Related Work

Work about DNS-based DDoS attacks is split mainly into two broad categories: (i) classifiers and strategies to detect or evade attacks and (ii) describing novel attacks.

### 7.9.1 DDoS Defenses

Moura et al. [138] investigate how resolvers and clients behave during attacks on AuthNSes. They set up an AuthNS infrastructure and drop packets to simulate an attack. They measure the success rate, latency, and more of clients querying the "attacked" zone. The paper explains how an AuthNS DoS will affect the larger ecosystem and highlights the importance of defenses for AuthNSes. We thus iterate over DNS-specific defenses in the following.

Davis et al. [41] propose a new mitigation strategy based on reverse DNS entries. An IP address space owner can mark what kind of DNS traffic is expected to originate from the address space, e.g., specifying DNS Cookie support or noting that no DNS traffic is expected. AuthNSes can use this information to filter out traffic originating from a spoofed source and not respond. A high adoption rate in both the reverse DNS space and among all DNS servers could reduce reflective DNS attacks. This comes with a costly downside for AuthNSes, as they must now perform DNS queries, something usually only performed by clients and resolvers. This adds network, processing, and memory overheads for little protection of the AuthNS itself, but rather for the rest of the Internet.

Alonso et al. [4] suggest a more passive methodology to detect simple DNS floods. They leverage the “social structure” of resolvers, i.e., the relationship between resolvers and the queried domains. Their basic idea is to search for query patterns that increase or change the set of domains being queried, which happens, e.g., for random subdomain attacks. Unfortunately, their work is thus not content-agnostic, leaving leeway for trivial evasion techniques for attacks against TLDs. For example, in our setting, attackers can choose from querying millions of *existing* domains and could even mimic website popularity rankings to model query likelihoods for each domain. Moreover, their methodology requires a costly query analysis and subgraph detection (approximating the NP-complete problem of bipartite subgraph search).

Rizvi et al. [166] propose an active routing-based defense for attacks that overload just single AuthNS in a larger anycast setup. To mitigate these local attacks, they use BGP to steer traffic between multiple anycast servers. Anycast catchment changes as a result of altered BGP announcements are often unpredictable. As such, the paper describes the creation of a playbook of applicable BGP alterations and the likely outcome. During an attack, the alteration that best matches the current situation can be picked, bringing all servers back to capacity. This defense is elegant and free of false positives, yet can only be applied if the overall anycast network has sufficient capacities to defend against an attack. Our work applies to the exact opposite situation, i.e., when massive attacks overload the entire anycast network. In these situations, shifting traffic from one location to another would only trigger cascading collateral damage.

Finally, Trejo et al. [204] create DNS-ADVP, a nearest-neighbor-based system to detect the presence of amplification attacks against DNS infrastructures. First, the authors create visual classifiers to inspect and alert on abnormal traffic conditions. The visual classifiers are based on the traffic fraction of the largest source compared to the overall volume and the correlation of requested domain names between different resolvers. They then represent the DNS traffic as feature vectors, each vector capturing the overall traffic statistics of a second time window. The resulting system can, with 78 % accuracy, detect the presence of attacks against DNS. Having said this, their work has several limitations. First, the proposed defense is limited to detecting *that* attacks occur but cannot pinpoint the attack sources. It can complement our work in that it can be used to *activate* our defense. Second, the features are not content-agnostic, risking that targeted attacks bypass the methodology by adapting their query patterns.

### 7.9.2 DNS Attacks

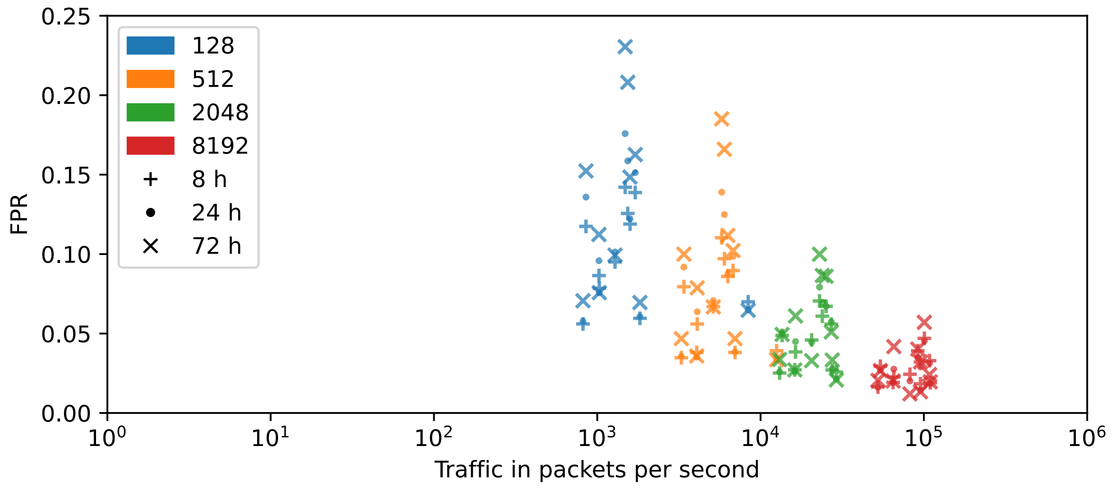
Besides the ongoing efforts on exploring attack countermeasures, other related work documents new DNS attack strategies. While not directly related to this chapter, the strategies listed below describe attacks against an AuthNS to which our anomaly detection applies.

The tsuNAME attack disclosed by Moura et al. [137] describes a software bug, where a pair of dependent records can lead to an infinite loop. Two zones in different TLDs are set up to host their AuthNSes in the other zone. This causes a cyclic dependency where, looking up the AuthNS of the first zone, the second zone needs to be resolved, but that requires finding the AuthNS in the first zone, leading to a situation where neither zone can be resolved successfully. A bug in the software implementation Google used did not detect this situation and instead kept sending requests to both domains. The requests were not rate limited, causing flooding of the TLD AuthNSes. While this specific attack is caused by a combination of bad zone files, i.e., cyclic dependencies, and a software bug, it shows that even benign services can sometimes misbehave. Any defense strategy should be careful in generally allowlisting specific sources and, as we propose, better apply limits even to allowlisted sources.

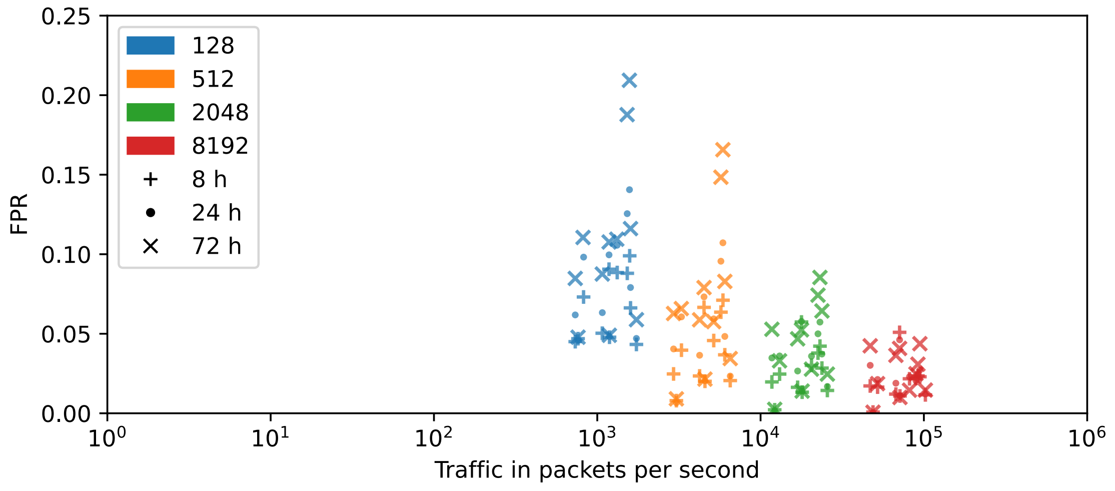
The NXNSAttack discovered by Afek et al. [1] used application-layer amplification enabled by resolvers to flood AuthNSes with over 500× the original bandwidth. The attack is enabled by resolvers, too generous in following DNS delegations, thus resulting in multiple identical queries. A DNS delegation is performed with a `NS` record pointing to a domain where the `A` and `AAAA` records for the AuthNS IP addresses are stored. Each delegation can have multiple `NS` records, and each domain has multiple `A/AAAA` records. All `A/AAAA` records can point to the same server. Resolvers were using all records simultaneously and sending hundreds of queries. Limiting the number of simultaneously followed delegations fixes the attack. This attack shows again how benign resolvers can suddenly turn bad. Our defense can cope with this attack; we would penalize "unpatched" resolvers by dropping their (partially benign) lookups. As soon as they mitigate their bogus behavior, they fall below the *LPF* or their prior allowlist threshold.

## 7.10 Conclusion

For many years, the DNS operators community has tacitly assumed that our core DNS infrastructure is sufficiently resilient against large-scale DDoS attacks. We showed that this might be a false assumption in the presence of strong adversaries. Motivated by this, we proposed a two-layer anomaly detection defense that allows upstream providers to effectively mitigate application-layer attacks against DNS. The proposed defense finds a sweet spot between collateral damage (false positives) and the remaining load on the critical DNS infrastructures (measured in requests per second). This provides a big step forward for DNS operators, especially if traffic engineering is no longer effective. Should an application-layer attack arise for which content-based filters fail, DNS operators now have a better alternative to extreme(ly bad) measures such as null routing.



**(a)** Attacker with Mirai-like



**(b)** Open Resolver Attacker

**Figure 7.7:** Effect of the *LPF* and $W_{test}$ parameters on the FPR and FNs. Each point represents a site's performance depending on *LPF* and $W_{test}$. The graphs depict attackers using a Mirai-like botnet and open resolvers.

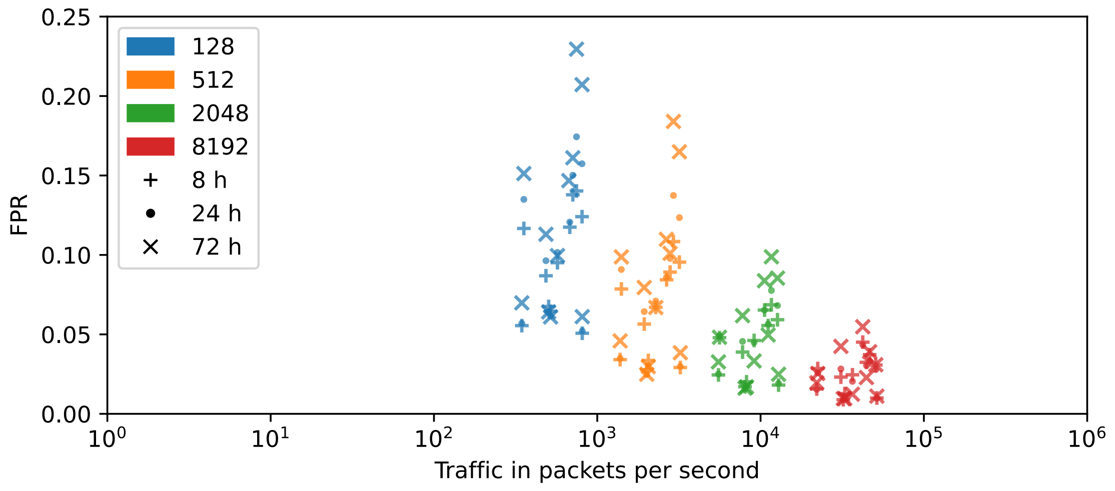


**Figure 7.8:** Effect of the *LPF* and $W_{test}$ parameters on the FPR and FNs. Each point represents a site's performance depending on *LPF* and $W_{test}$. The graphs depict attackers using Sality, a Mirai-like botnet, and open resolvers.

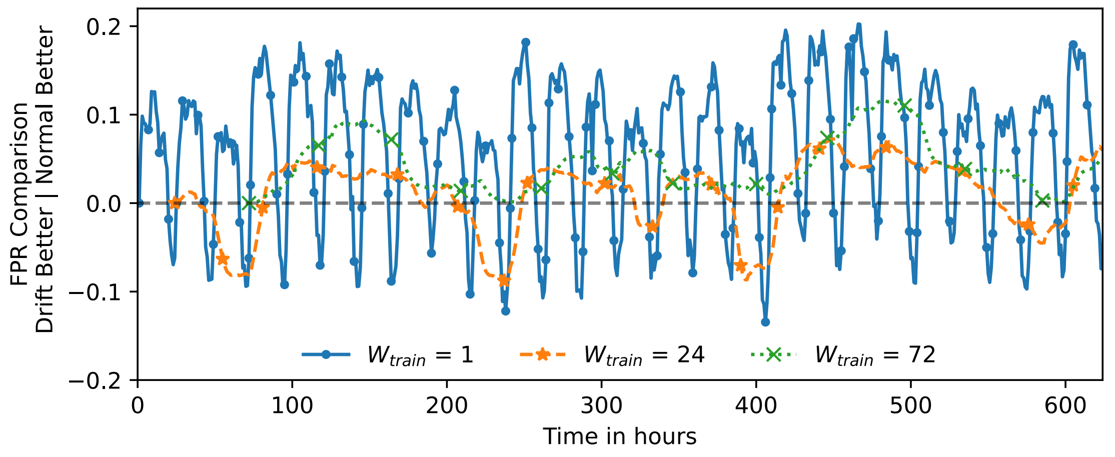


**Figure 7.9:** Impact of static (non-retrained) allowlists on the FPR based on three training lengths (one hour, one day, three days). Higher means worse.

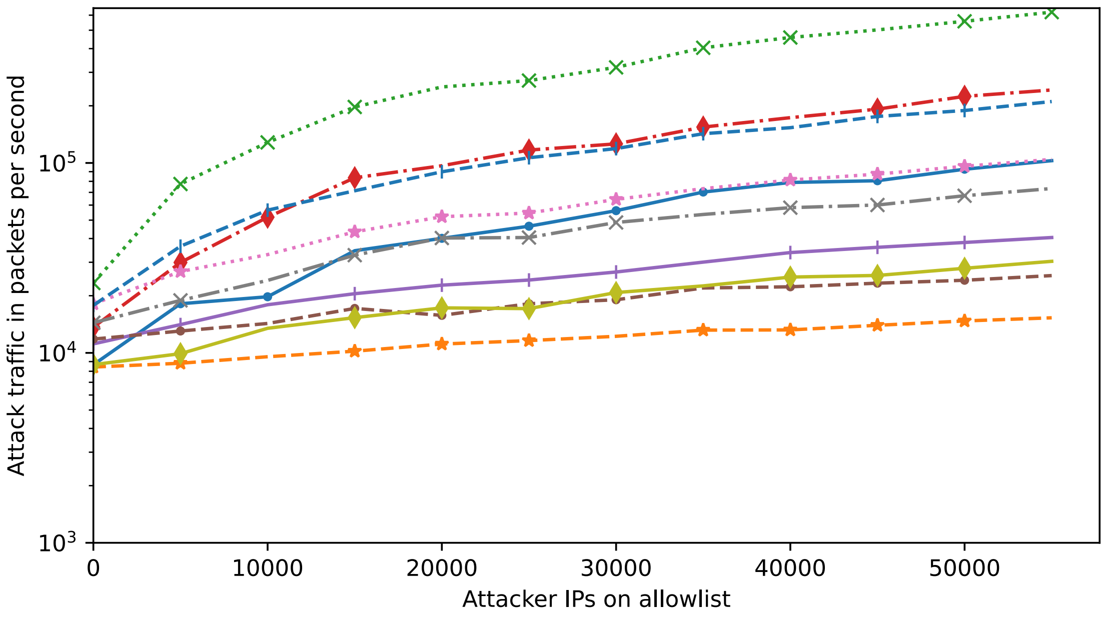


**Figure 7.10:** Impact of randomly selected attacker nodes added to the allowlist (x-axis) on the traffic in pps on a log scale (y-axis). Each line represents a different site.

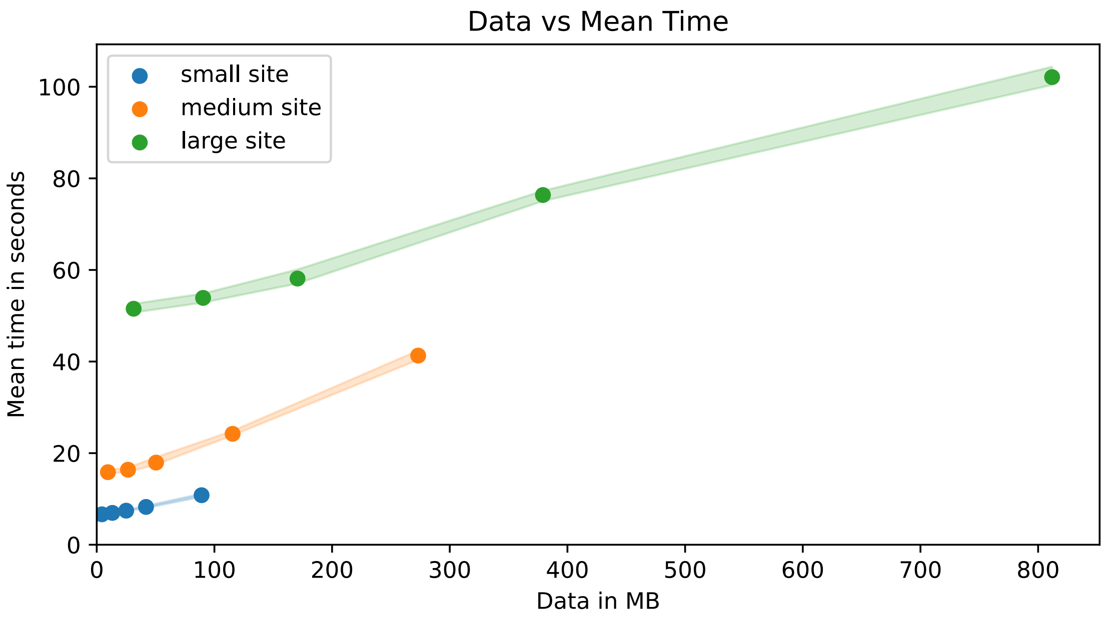


**Figure 7.11:** Performance of our prototype for three sites, ranging from small to large data volume. The training window ($W_{train}$) ranges from one (left) to 24 hours (right). The graph shows the mean runtime in seconds (y-axis) with the observed min/max values and data size (x-axis).

# 8 Conclusion

## 8.1 Summary of Contributions

In the previous chapters, we covered a wide range of topics, from DNS privacy, to DNS software correctness, to DoS attacks and mitigation on the infrastructure level. We uncovered different problems and vulnerabilities in different components. Most importantly, we proposed and evaluated solutions to these problems, enabling others to benefit from this work and improving the DNS ecosystem for everyone. The last chapters cover the research questions RQ1-RQ3, which we introduced in Chapter 1. We summarize our findings and contributions below.

### 8.1.1 (RQ1) How can we protect users' privacy and limit surveillance possibilities?

Modern life is increasingly lived online, with more and more services moving to the Internet. Companies and governments are tracking the behavior of users online, which often contradicts the users' expectations. As such it is important to give users the ability to protect their privacy. One such attack is *website fingerprinting*, where an attacker tries to infer the domain or URL a user is visiting.

DNS is one such channel that can be used to infer the visited domain, trivially so if the DNS query is unencrypted. We showed in Chapter 3 that even with encrypted DNS (DoT and DoH) and padded DNS queries (to hide size information) it is still possible to infer the visited domain. Our main insight is that a website visit oftentimes involves multiple sub- and third-party domains, which are loaded in a specific order. The suggested padding in RFC 7830 [128] is quite effective in hiding the size of the DNS query, but it does not hide the number of queries, nor the timing between queries. We showed that these differences allow an attacker to infer the visited website in up to 86.1 % of cases.

Our solution is to add dummy traffic to the DNS channel, such that the number of queries and the timing between queries is hidden. This is a known technique in the literature, but we showed how to apply it to DNS while finding a good tradeoff between privacy and performance.

With our work, we hope to inspire developers of DNS software to think about side-channel attacks and urge them to implement countermeasures. A DNS client or stub server has many options to obfuscate the DNS traffic, including the ones we proposed, and can use the dummy traffic to preload the cache. Long term, we hope the DNS standard will be extended to include other privacy-preserving features like resolverless DNS [201], in which clients no longer need to send DNS queries to a recursive resolver. Instead, the client gets the answers directly from the web servers, before they are needed. Similar to OCSP stapling [148, 160] this gets rid of many messages and the privacy and performance penalties that come with them.

### 8.1.2 (RQ2) How can we improve correctness and reduce bugs in DNS software?

Software correctness is an important aspect of any software, but especially so for security-critical software and Internet-connected software. Defects in software can lead to many different problems, such as data leakage, DoS attacks or providing wrong information. Software testing and fuzzing are two standard methods for validating program behavior and uncover bugs.

In Chapter 4 we took the challenge of fuzzing recursive resolvers, which are a critical part of the DNS infrastructure. They are exposed to the Internet and DNS clients trust them to provide correct answers. Fuzzing them is challenging, because they are inherently stateful, need access to multiple servers, and need to support many different DNS features. Further, the DNS standard allows for many ways to encode the same information, which complicates validation efforts.

Nevertheless, we build a differential fuzzer, that can test multiple resolvers at the same time and compare their answers. We create rules for normalizing DNS messages, enabling robust comparisons of different messages. Using that we fuzzed seven open-source resolvers, and found seven bugs in five different implementations. We reported the bugs to the developers and helped them fix the issues. The code is open source and can be used by others to fuzz their own resolvers.

With this work, we showed that existing and new DNS servers still have bugs and they disagree on the correct behavior for some situations. There are no standardized test suites that cover DNS servers, leaving the potential for more bugs and inconsistencies in the future. We hope that this work inspires more development in ensuring the correctness and standard compliance of DNS software.

### 8.1.3 (RQ3) How can we protect the DNS infrastructure from attacks?

Lastly, Chapters 5 and 7 cover the topic of DoS attacks and mitigation. The best DNS server is of no use when it is unavailable due to an attack. The software must be robust and not have relevant bugs, the protocol should prevent any amplification potential, and the infrastructure should be able to handle the load of legitimate traffic and attacks.

Chapter 5 covers an amplification attack, that makes use of standard DNS protocol features to increase the load on AuthNSes, by forcing many recursive resolvers to query the same AuthNS. An attacker that can send DNS queries to many resolvers and has control over two zones, one on the victim server, can use `CNAME` records that link to each other. A single query to the recursive resolver will force it to follow the chain, sending on average 8.51 queries to the victim server.

We showed that this vulnerability is a result of the protocol, which discourages but requires that recursive resolvers follow `CNAME` chains. The standards are silent on best practices for handling these chains, such as the maximum number of `CNAME` records to follow. This leaves the implementors to pick their defaults, which are often too high and exacerbate the problem. We provide recommendations for AuthNS operators on how to identify this attack before it is launched and configurations to limit the impact it has. Similarly, we make suggestions for recursive resolver operators and implementers for better defaults.

In Chapter 6 we analyzed a slight variation of the attack, which aims to overload network links close to one AuthNS server. This enables targeting victims that are in the same data center. Defending against this attack is harder since the victims might not even be aware of the attack, since they only notice a slowdown in their network traffic. But to prevent the DNS Unchained attack a transport protocol robust against packet loss can help.

Chapter 7 takes the perspective of an AuthNS operator, who wants to protect their server from DoS attacks. We analyze the traffic and network for one large European TLD and find some weaknesses in the setup that can enable an attacker to impact service availability, using many recursive resolvers or a botnet. We propose a filtering system that can be used to detect and block malicious traffic, while still allowing legitimate traffic. It works with existing NetFlow information, allowing for efficient and privacy-preserving intervention.

Our work shines a light on the DNS infrastructure and the challenges it faces in the presence of attacks. The analysis deepens the understanding of the DNS ecosystem and helps operators to better protect their servers. Our suggested mitigations are practical and can be implemented by operators with little effort.

## 8.2 Further Research Directions

With this work, we investigated multiple aspects of the DNS protocol and its implementations and enhanced the understanding of the DNS ecosystem. The topics we investigated will never be fully solved and will provide new research possibilities for the future. Like any Internet-connected service, DNS is a moving target and new threats and vulnerabilities will be discovered, forcing the research and security community to react. New features are introduced in two ways: new capabilities that were impossible before, like the introduction of encrypted DNS, or the use of existing features in new ways, like encryption between recursive resolvers and AuthNSes.

Another development direction is taking the existing work and making it more robust and usable. The work on the DNS fuzzer is a good example of this. The basic functionality is there, but more resolvers can be integrated, documentation can be improved, and the DNS standard can be covered more completely.

In Chapter 3 we tackled the privacy aspects of encrypted DNS communications. Since then new protocols like DoQ [88] were standardized and even more, like DNS over COAP (DoC) [118], are proposed. These newer protocols might suffer from new or different side channels or will inspire new DNS library implementations, which might be missing the discovered countermeasures. Besides the client-to-resolver communication, the DNS operator community is talking about extending the encryption support to cover the messages between recursive resolvers and AuthNSes [16]. Resolver to AuthNS communication has a different threat model, mainly because individual client queries are less visible, due to caching and coalescing of data.

We tested multiple mitigation algorithms to hide timing information and hinder web page fingerprinting. Other algorithms might provide better tradeoffs, especially for resource-constrained environments. While we are confident that our mitigations work and will continue so in the future, we suggest that mitigations should be periodically tested in the future. New developments will come up that might allow for better mitigations, such as newer transport protocols like QUIC [51, 93], or the problem can be side-stepped by using different approaches like resolverless DNS [201].

Lastly, the work in Chapter 3 focuses on the one specific use case of web browsing, but DNS is used on servers, Internet-of-Things (IoT) devices, and many other places, which might be vulnerable to variations of the attack.

We built a fuzzer for recursive resolvers in Chapter 4 and showed that it is possible to find bugs in the implementations. More engineering work can always make the tool easier to use, support more platforms and resolvers, and make it more robust. Such work is important for the wider adoption of tools but is usually out-of-scope for research projects. The fuzzer supports an initial set of DNS features, but more research is needed to cover the full DNS feature set, including DNSSEC and stateful operations. Both areas were not covered by our fuzzer, because of the complexity of the features, but are important for operating modern DNS infrastructures.

We hope that this work inspires other researchers and the DNS community to continue and improve on the work we started. DNS has shown to be a capable and flexible protocol in the last 35 years and will remain important for the foreseeable future. Its age and continued feature development mean that many aspects of the protocol and interactions between software with varying levels of feature support, will reveal many bugs and offer many research opportunities.

As another research direction, we covered DoS attacks against DNS infrastructure, in Chapters 5 to 7. DNS Unchained is only one attack, with other known attacks like NXNSAttack [1] and tsuNAME [137]. It is important to ensure that these known attacks are mitigated and do not regress in the future. Similarly, new software implementations need to ensure that they include the mitigations. Having a standard set of tests that can be run against DNS software would help but requires considerable effort to create and maintain. Alternatively, improving the above-mentioned fuzzer to cover these attacks and make it more user-friendly.

Infrastructure is generally less flexible than software, as changes and updates often require larger changes in hardware, network link capacities, or service agreements. This means that our analysis from Chapter 7 will remain relevant for some time. Nevertheless, infrastructure is a moving target and operators are investigating ways of improving their infrastructure, by using new tools or making it cheaper to operate. This includes the ongoing transition into the IPv6 space, the use of cloud providers for dynamic scaling, and infrastructure-as-code to automate the management of infrastructure. With these changes come new challenges and threats, such that operators must always be vigilant about protecting their systems. However, these developments come with new capabilities that make attack mitigation easier.

# Bibliography

## Author's Papers for this Thesis

[P1] **Bushart**, **J.** and Rossow, C. DNS Unchained: amplified application-layer DoS attacks against DNS authoritatives. In: *Proceedings of the 21th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*. Sept. 2018. DOI: 10.1007/978-3-030-00470-5_7 (cit. on pp. vii, 16, 78, 85, 90).

[P2] **Bushart**, **J.** Optimizing recurrent pulsing attacks using application-layer amplification of open DNS resolvers. In: *12th USENIX Workshop on Offensive Technologies (WOOT)*. Aug. 2018 (cit. on pp. vii, 16).

[P3] **Bushart**, **J.** and Rossow, C. Padding ain't enough: assessing the privacy guarantees of encrypted DNS. In: *10th USENIX Workshop on Free and Open Communications on the Internet (FOCI)*. USENIX Association, Aug. 2020 (cit. on pp. vii, 15).

[P4] **Bushart**, **J.** and Rossow, C. Anomaly-based filtering of application-layer DDoS against DNS authoritatives. In: *8th IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, July 2023. DOI: 10.1109/EUROSP57164.2023.00040 (cit. on pp. vii, 17).

[P5] **Bushart**, **J.** and Rossow, C. ResolFuzz: differential fuzzing of DNS resolvers. In: *28th European Symposium on Research in Computer Security (ESORICS)*. Springer Nature, Sept. 2023. DOI: 10.1007/978-3-031-51476-0_4 (cit. on pp. vii, 16).

## Other Papers of the Author

[S1] Kührer, M., Hupperich, T., **Bushart**, **J.**, Rossow, C., and Holz, T. Going Wild: large-scale classification of open DNS resolvers. In: *Proceedings of the 2015 ACM Internet Measurement Conference (IMC)*. Oct. 2015. DOI: 10.1145/2815675.2815683 (cit. on pp. viii, 64, 67, 69, 78).

## Other References

[1] Afek, Y., Bremler-Barr, A., and Shafir, L. NXNSAttack: recursive DNS inefficiencies and vulnerabilities. In: *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*. 2020 (cit. on pp. 17, 118, 128).

[2] AFNIC. *Statistiques*. Jan. 2022. URL: https://www.afnic.fr/observatoire-ressources/statistiques/ (cit. on pp. 92, 94).

[3] Aitken, P., Claise, B., and Trammell, B. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information*. RFC 7011. Sept. 2013. DOI: 10.17487/RFC7011 (cit. on pp. 90, 96).

[4] Alonso, R., Monroy, R., and Trejo, L. A. Mining IP to domain name interactions to detect DNS flood attacks on recursive DNS servers. *Sensors* (2016). DOI: 10.3390/S16081311 (cit. on p. 117).

[5] Andrews, M. P., Huque, S., Wouters, P., and Wessels, D. *DNS Glue Requirements in Referral Responses*. Internet-Draft draft-ietf-dnsop-glue-is-not-optional-08. Work in Progress. Internet Engineering Task Force, Feb. 2023. 12 pp. (cit. on p. 38).

[6] Andronidis, A. and Cadar, C. SnapFuzz: high-throughput fuzzing of network applications. In: *ISSTA '22: 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*. 2022. DOI: 10.1145/3533767.3534376 (cit. on pp. 48, 49).

[7] Antonakakis, M., April, T., Bailey, M. D., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J. A., Invernizzi, L., Kallitsis, M., Kumar, D., Lever, C., Ma, Z., Mason, J., Menscher, D., Seaman, C., Sullivan, N., Thomas, K., and Zhou, Y. Understanding the mirai botnet. In: *26th USENIX Security Symposium*. 2017 (cit. on pp. 54, 56).

[8] Arends, R., Sisson, G., Blacka, D., and Laurie, B. *DNS Security (DNSSEC) Hashed Authenticated Denial of Existence*. RFC 5155. Mar. 2008. DOI: 10.17487/RFC5155 (cit. on p. 12).

[9] Bella, T. *Major Internet outage along East Coast causes large parts of the Web to crash — again*. July 2021. URL: https://www.washingtonpost.com/business/2021/07/22/internet-outage-amazon-airbnb-delta/ (visited on 2023-08-15) (cit. on p. 4).

[10] Bellis, R. *Benchmarking DNS Reliably on Multi-core Systems*. July 2015. URL: https://www.isc.org/blogs/benchmarking-dns/ (visited on 2017-10-25) (cit. on p. 65).

[11] *BIND Open Source DNS Server*. URL: https://www.isc.org/bind/ (cit. on pp. 6, 61, 95).

[12] Blanton, E., Paxson, D. V., and Allman, M. *TCP Congestion Control*. RFC 5681. Sept. 2009. DOI: 10.17487/RFC5681 (cit. on pp. 74, 84).

[13] Böck, H. *Telekom beendet DNS-Hijacking*. May 2019. URL: https://www.golem.de/news/t-online-navigationshilfe-telekom-beendet-dns-hijacking-nach-strafanzeige-1905-141370.html (visited on 2023-08-15) (cit. on p. 4).

[14] Bortzmeyer, S. *DNS Query Name Minimisation to Improve Privacy*. RFC 7816. Mar. 2016. DOI: 10.17487/RFC7816 (cit. on p. 5).

[15] Brakmo, L. S., O'Malley, S. W., and Peterson, L. L. TCP vegas: new techniques for congestion detection and avoidance. In: *Proceedings of the ACM SIGCOMM 1994 Conference on Communications Architectures, Protocols and Applications*. 1994. DOI: 10.1145/190314.190317 (cit. on p. 84).

[16] Bretelle, M. *DNS over TLS: Encrypting DNS end-to-end*. Dec. 2018. URL: https://engineering.fb.com/2018/12/21/security/dns-over-tls/ (cit. on p. 127).

[17] *Bromite*. 2019. URL: https://www.bromite.org/ (visited on 2019-01-21) (cit. on p. 21).

[18] Cambus, F. *DNS related RFCs*. URL: https://www.statdns.com/rfc/ (visited on 2023-05-11) (cit. on pp. 6, 37).

[19] Cambus, F. *DNS resources: A collection of DNS related resources*. URL: https://www.statdns.com/resources/ (visited on 2024-07-30) (cit. on p. 6).

[20] Cardwell, N., Cheng, Y., Gunn, C. S., Yeganeh, S. H., and Jacobson, V. BBR: congestion-based congestion control. *Communications of the ACM* (2017). DOI: 10.1145/3009824 (cit. on p. 84).

[21] Cardwell, N., Cheng, Y., Yeganeh, S. H., Swett, I., and Jacobson, V. *BBR Congestion Control*. Internet-Draft draft-cardwell-iccrg-bbr-congestion-control-02. Work in Progress. Internet Engineering Task Force, Mar. 2022. 66 pp. (cit. on p. 84).

[22] *Censys DNS Lookup full IPv4*. 2017. URL: https://censys.io/data/53-dns-lookup-full_ipv4 (visited on 2017-04-27) (cit. on pp. 55, 76).

[23] US-CERT. *Alert (TA14-017A) UDP-Based Amplification Attacks*. Mar. 2018 (cit. on p. 75).

[24] *Chrome: Partition the HTTP Cache*. Nov. 2019. URL: https://chromestatus.com/feature/5730772021411840 (visited on 2020-02-24) (cit. on p. 30).

[25] Claise, B. *Cisco Systems NetFlow Services Export Version 9*. RFC 3954. Oct. 2004. DOI: 10.17487/RFC3954 (cit. on pp. 90, 96).

[26] Claise, B. and Trammell, B. *Information Model for IP Flow Information Export (IPFIX)*. RFC 7012. Sept. 2013. DOI: 10.17487/RFC7012 (cit. on pp. 90, 96).

[27] *1.1.1.1 – The free app that makes your internet faster*. Jan. 2022. URL: https://1.1.1.1/ (cit. on pp. 15, 67, 90).

[28] *Common Crawl CC-MAIN-2019-43*. Oct. 2019. URL: https://commoncrawl.s3.amazonaws.com/cc-index/collections/CC-MAIN-2019-43/indexes/cdx-00000.gz (visited on 2019-10-15) (cit. on p. 26).

[29] Consortium, I. S. *General DNS Reference Information*. URL: https://bind9.readthedocs.io/en/latest/general.html (visited on 2023-05-11) (cit. on pp. 6, 37).

[30] Contavalli, C., Gaast, W. van der, Lawrence, D. C., and Kumari, W. *Client Subnet in DNS Queries*. RFC 7871. May 2016. DOI: 10.17487/RFC7871 (cit. on p. 37).

[31] Cooper, A., Tschofenig, H., Aboba, D. B. D., Peterson, J., Morris, J., Hansen, M., and Smith, R. *Privacy Considerations for Internet Protocols*. RFC 6973. July 2013. DOI: 10.17487/RFC6973 (cit. on p. 5).

[32] Corporation, S. S. *Water Torture: A Slow Drip DNS DDoS Attack*. Secure64 Software Corporation. Feb. 2014. URL: https://secure64.com/water-torture-slow-drip-dns-ddos-attack/ (visited on 2021-07-14) (cit. on pp. 54, 56, 69, 90, 94).

[33] Crawford, D. M. *Non-Terminal DNS Name Redirection*. RFC 2672. Aug. 1999. DOI: 10.17487/RFC2672 (cit. on p. 60).

[34] *Creative Commons Attribution-ShareAlike 3.0 Unported*. Version 3. Creative Commons, May 20, 2018 (cit. on p. 76).

[35] *CVE-2008-1447*. Available from MITRE, CVE-ID CVE-2008-1447. July 2008 (cit. on p. 60).

[36] *CVE-2022-48256*. Available from MITRE, CVE-ID CVE-2022-48256. Jan. 2023 (cit. on p. 44).

[37] *Traffic Dashboard*. Feb. 2023. URL: https://stats.nic.cz/dashboard/en/Traffic.html (cit. on p. 93).

[38] Dagon, D., Antonakakis, M., Day, K., Luo, X., Lee, C. P., and Lee, W. Recursive DNS architectures and vulnerability implications. In: *Proceedings of the Network and Distributed System Security Symposium*. 2009 (cit. on p. 69).

[39] Dahan, A. *Business as usual for Azure customers despite 2.4 Tbps DDoS attack*. Oct. 2021. URL: https://azure.microsoft.com/en-us/blog/business-as-usual-for-azure-customers-despite-24-tbps-ddos-attack/ (cit. on p. 93).

[40] Damerau, F. A technique for computer detection and correction of spelling errors. *Communications of the ACM* (1964). DOI: 10.1145/363958.363994 (cit. on p. 24).

[41] Davis, J. and Deccio, C. T. Advertising DNS protocol use to mitigate DDoS attacks. In: *29th IEEE International Conference on Network Protocols*. 2021. DOI: 10.1109/ICNP52444.2021.9651929 (cit. on p. 116).

[42] CZ.NIC. *Deckard*. URL: https://gitlab.nic.cz/knot/deckard/ (cit. on pp. 47, 48, 49).

[43] DENIC. *Domain Statistics of .de*. Jan. 2022. URL: https://www.denic.de/en/know-how/statistics/monthly-statistics-of-de/ (cit. on pp. 92, 94).

[44] Dickinson, J., Dickinson, S., Bellis, R., Mankin, A., and Wessels, D. *DNS Transport over TCP - Implementation Requirements*. RFC 7766. Mar. 2016. DOI: 10.17487/RFC7766 (cit. on p. 94).

[45] *DNS Privacy Implementation Status*. Jan. 2019. URL: https://dnsprivacy.org/wiki/display/DP/DNS+Privacy+Implementation+Status (visited on 2019-02-04) (cit. on p. 21).

[46] DNS Privacy Project. *Follow-Up Performance Measurements (Q4 2108)*. Oct. 2018. URL: https://dnsprivacy.org/performance_measurements/follow-up_performance_measurements_q4_2108/ (cit. on pp. 93, 95).

[47] *dns-fuzz in nmap*. URL: https://nmap.org/nsedoc/scripts/dns-fuzz.html (visited on 2023-05-11) (cit. on pp. 48, 49).

[48] *DNSBL Information – Spam Database and Blacklist Check*. July 2021. URL: https://www.dnsbl.info/ (cit. on pp. 3, 53, 55).

[49] *DNSSEC Deployment Report*. Feb. 2019. URL: https://rick.eng.br/dnssecstat/ (visited on 2019-02-05) (cit. on p. 21).

[50] APNIC. *Use of DNSSEC Validation for World*. URL: https://stats.labs.apnic.net/dnssec/XA (visited on 2019-02-05) (cit. on p. 21).

[51] Duke, M. *QUIC Version 2*. RFC 9369. May 2023. DOI: 10.17487/RFC9369 (cit. on pp. 85, 127).

[52] Dukhovni, V. and Hardaker, W. *The DNS-Based Authentication of Named Entities (DANE) Protocol: Updates and Operational Guidance*. RFC 7671. Oct. 2015. DOI: 10.17487/RFC7671 (cit. on pp. 53, 55).

[53] Durumeric, Z., Wustrow, E., and Halderman, J. A. ZMap: fast internet-wide scanning and its security applications. In: *Proceedings of the 22th USENIX Security Symposium*. 2013 (cit. on pp. 55, 61, 76, 81, 104).

[54] Dyer, K. P., Coull, S. E., Ristenpart, T., and Shrimpton, T. Peek-a-boo, I still see you: why efficient traffic analysis countermeasures fail. In: *IEEE Symposium on Security and Privacy*. 2012. DOI: 10.1109/SP.2012.28 (cit. on p. 27).

[55] Dziuba, T. *When ISPs hijack your rights to NXDOMAIN*. Aug. 2009. URL: https://www.theregister.com/2009/08/17/dzuiba_virgin_media_opendns/ (visited on 2023-08-15) (cit. on p. 4).

[56] Eastlake 3rd, D. E. and Andrews, M. P. *Domain Name System (DNS) Cookies*. RFC 7873. May 2016. DOI: 10.17487/RFC7873 (cit. on pp. 16, 94).

[57] Efstathopoulos, P. Practical study of a defense against low-rate TCP-targeted DoS attack. In: *Proceedings of the 4th International Conference for Internet Technology and Secured Transactions*. 2009. DOI: 10.1109/ICITST.2009.5402593 (cit. on p. 84).

[58] Eiben, A. E. and Smith, J. E. *Introduction to Evolutionary Computing, Second Edition*. Natural Computing Series. Springer, 2015. DOI: 10.1007/978-3-662-44874-8 (cit. on p. 78).

[59] Elvira, R. Slack's DNSSEC rollout: third time's the outage. In: USENIX Association, Amsterdam, Oct. 2022 (cit. on p. 4).

[60] Elvira, R. and Nolan, L. *The Case of the Recursive Resolvers*. Nov. 2021. URL: https://slack.engineering/what-happened-during-slacks-dnssec-rollout/ (visited on 2023-08-15) (cit. on p. 4).

[61] Elvira, R. and Nolan, L. *Major DNSSEC Outages and Validation Failures*. July 2023. URL: https://ianix.com/pub/dnssec-outages.html (visited on 2023-08-16) (cit. on p. 4).

[62] Elz, R. and Bush, R. *Clarifications to the DNS Specification*. RFC 2181. July 1997. DOI: 10.17487/RFC2181 (cit. on p. 12).

[63] Ereche, M. V. *DNS Completitude and Compliance testing*. Oct. 2020. URL: https://github.com/mave007/dns_completitude_and_compliance/tree/2a18967d103d232e9072c4474e8c731dc3d79f7a (cit. on p. 49).

[64] Ermert, M. *Störerhaftung für DNS-Resolver: Quad9 verliert vor Landgericht gegen Sony*. Dec. 2021. URL: https://www.heise.de/news/Landgericht-Hamburg-entscheidet-gegen-Quad9-im-Streit-mit-Sony-6280566.html (visited on 2023-08-15) (cit. on p. 4).

[65] Farrell, S. and Tschofenig, H. *Pervasive Monitoring Is an Attack*. RFC 7258. May 2014. DOI: 10.17487/RFC7258 (cit. on p. 5).

[66] *Firefox: Top-level origin partitioning*. Oct. 2019. URL: https://bugzilla.mozilla.org/show_bug.cgi?id=1590107 (visited on 2020-02-24) (cit. on p. 30).

[67] Floyd, S. *HighSpeed TCP for Large Congestion Windows*. RFC 3649. Dec. 2003. DOI: 10.17487/RFC3649 (cit. on p. 74).

[68] Fujiwara, K., Kato, A., and Kumari, W. *Aggressive Use of DNSSEC-Validated Cache*. RFC 8198. July 2017. DOI: 10.17487/RFC8198 (cit. on p. 94).

[69] Gallagher, S. *After DNS change fails, Turkish government steps up Twitter censorship*. Mar. 2014. URL: https://arstechnica.com/tech-policy/2014/03/after-dns-change-fails-turkish-government-steps-up-twitter-censorship/ (visited on 2023-08-15) (cit. on p. 4).

[70] Gieben, R. and Mekking, M. *Authenticated Denial of Existence in the DNS*. RFC 7129. Feb. 2014. DOI: 10.17487/RFC7129 (cit. on p. 94).

[71] Gilad, Y., Herzberg, A., Sudkovitch, M., and Goberman, M. Cdn-on-demand: an affordable DDoS defense via untrusted clouds. In: *23rd Annual Network and Distributed System Security Symposium*. 2016 (cit. on p. 69).

[72] Gillmor, D. K. *Empirical DNS Padding Policy*. Mar. 2017. URL: https://dns.cmrg.net/ndss2017-dprive-empirical-DNS-traffic-size.pdf (cit. on pp. 21, 27, 30).

[73] Goldlust, S. *Using the Response Rate Limiting Feature*. Sept. 2018. URL: https://kb.isc.org/docs/aa-00994 (cit. on p. 94).

[74] Goldlust, S. and Almond, C. *Recursive Client Rate limiting*. Oct. 2021. URL: https://kb.isc.org/docs/aa-01304 (cit. on p. 94).

[75] Goldlust, S. and Almond, C. *Recursive Client Rate limiting – FAQs*. Oct. 2021. URL: https://kb.isc.org/docs/aa-01316 (cit. on p. 94).

[76] *Google Public DNS*. Jan. 2022. URL: https://developers.google.com/speed/public-dns (cit. on pp. 15, 67, 90).

[77] Hallam-Baker, P. *RFC Errata for RFC 6844 "DNS Certification Authority Authorization (CAA) Resource Record"*. Errata 5065. RFC Editor, 2017 (cit. on p. 68).

[78] Hayes, J. and Danezis, G. K-fingerprinting: A robust scalable website fingerprinting technique. In: *25th USENIX Security Symposium*. 2016 (cit. on p. 29).

[79] Hilton, S. *Dyn Analysis Summary Of Friday October 21 Attack*. Oct. 2016. URL: https://web.archive.org/web/20180224030354/https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/ (visited on 2022-10-27) (cit. on pp. 4, 53, 93).

[80] Hoang, N. P., Niaki, A. A., Dalek, J., Knockel, J., Lin, P., Marczak, B., Crete-Nishihata, M., Gill, P., and Polychronakis, M. How great is the great firewall? measuring china's DNS censorship. *arXiv preprint arXiv:2106.02167* (2021) (cit. on p. 4).

[81] Hoffman, P. E. *DNS Security Extensions (DNSSEC)*. RFC 9364. Feb. 2023. DOI: 10.17487/RFC9364 (cit. on pp. 15, 16, 47).

[82] Hoffman, P. E. and McManus, P. *DNS Queries over HTTPS (DoH)*. RFC 8484. Oct. 2018. DOI: 10.17487/RFC8484 (cit. on pp. 5, 15, 21, 30).

[83] Hoffman, P. E. and Schlyter, J. *The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA*. RFC 6698. Aug. 2012. DOI: 10.17487/RFC6698 (cit. on pp. 53, 55).

[84] Holz, T., Gorecki, C., Rieck, K., and Freiling, F. C. Measuring and detecting fast-flux service networks. In: *Proceedings of the Network and Distributed System Security Symposium, NDSS 2008*. 2008 (cit. on p. 59).

[85] Hoque, M. E., Chowdhury, O., Chau, S. Y., Nita-Rotaru, C., and Li, N. Analyzing operational behavior of stateful protocol implementations for detecting semantic bugs. In: *47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. 2017. DOI: 10.1109/DSN.2017.36 (cit. on pp. 48, 50).

[86] Houser, R., Li, Z., Cotton, C., and Wang, H. An investigation on information leakage of DNS over TLS. In: *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*. 2019. DOI: 10.1145/3359989.3365429 (cit. on pp. 22, 30).

[87] Hu, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., and Hoffman, P. E. *Specification for DNS over Transport Layer Security (TLS)*. RFC 7858. May 2016. DOI: 10.17487/RFC7858 (cit. on pp. 5, 15, 21, 30).

[88] Huitema, C., Dickinson, S., and Mankin, A. *DNS over Dedicated QUIC Connections*. RFC 9250. May 2022. DOI: 10.17487/RFC9250 (cit. on pp. 5, 127).

[89] Husák, M., Cermák, M., Jirsík, T., and Celeda, P. Network-based HTTPS client identification using SSL/TLS fingerprinting. In: *10th International Conference on Availability, Reliability and Security*. 2015. DOI: 10.1109/ARES.2015.35 (cit. on p. 23).

[90] Husák, M., Cermák, M., Jirsík, T., and Celeda, P. HTTPS traffic analysis and client identification using passive SSL/TLS fingerprinting. *EURASIP Journal on Information Security* (2016). DOI: 10.1186/S13635-016-0030-7 (cit. on p. 23).

[91] IANA. *Root Files*. URL: https://www.iana.org/domains/root/files (visited on 2023-05-11) (cit. on p. 39).

[92] Internet Systems Consortium. *Pseudo Random DNS Query Attacks & Resolver Mitigation Approaches*. 2015. URL: https://www.nanog.org/sites/default/files/nanog63-dnstrack-winstead-attacks.pdf (cit. on p. 69).

[93] Iyengar, J. and Thomson, M. *QUIC: A UDP-Based Multiplexed and Secure Transport*. RFC 9000. May 2021. DOI: 10.17487/RFC9000 (cit. on pp. 85, 127).

[94] Janardhan, S. *Major DNSSEC Outages and Validation Failures*. Oct. 2021. URL: More%20details%20about%20the%20October%204%20outage (visited on 2023-08-16) (cit. on p. 4).

[95] Jensen, T., Pashov, I., and Montenegro, G. *Windows will improve user privacy with DNS over HTTPS*. Nov. 2019. URL: https://techcommunity.microsoft.com/t5/Networking-Blog/Windows-will-improve-user-privacy-with-DNS-over-HTTPS/ba-p/1014229 (visited on 2019-11-18) (cit. on p. 21).

[96] Kakarla, S. K. R., Beckett, R., Arzani, B., Millstein, T. D., and Varghese, G. GRooT: proactive verification of DNS configurations. In: *SIGCOMM '20: Proceedings of the 2020 Annual conference of the ACM Special Interest Group on Data Communication on the applications*. 2020. DOI: 10.1145/3387514.3405871 (cit. on pp. 6, 49).

[97] Kakarla, S. K. R., Beckett, R., Millstein, T. D., and Varghese, G. SCALE: automatically finding RFC compliance bugs in DNS nameservers. In: *19th USENIX Symposium on Networked Systems Design and Implementation*. 2022 (cit. on pp. 6, 48, 49).

[98] Kaminsky, D. *It's the end of the cache as we know it*. URL: https://vimeo.com/1603602 (cit. on p. 60).

[99] Kitterman, S. *Sender Policy Framework (SPF) for Authorizing Use of Domains in Email, Version 1*. RFC 7208. Apr. 2014. DOI: 10.17487/RFC7208 (cit. on pp. 3, 53, 55).

[100] Kline, E. and Schwartz, B. *DNS over TLS support in Android P Developer Preview*. Apr. 2018. URL: https://android-developers.googleblog.com/2018/04/dns-over-tls-support-in-android-p.html (cit. on p. 21).

[101] Knot DNS. *Benchmark*. Jan. 2022. URL: https://www.knot-dns.cz/benchmark/ (cit. on pp. 65, 92, 93, 95).

[102] Knot DNS. *Old Benchmark*. Jan. 2022. URL: https://www.knot-dns.cz/benchmark-old/ (cit. on pp. 93, 95).

[103] *Knot Resolver*. URL: https://www.knot-resolver.cz/ (visited on 2018-02-07) (cit. on pp. 6, 61, 95).

[104] Kountouras, A., Kintis, P., Lever, C., Chen, Y., Nadji, Y., Dagon, D., Antonakakis, M., and Joffe, R. Enabling network security through active DNS datasets. In: *Proceedings of the 19th International Symposium on Research in Attacks, Intrusions and Defenses*. 2016. DOI: 10.1007/978-3-319-45719-2_9 (cit. on p. 68).

[105] Krämer, L., Krupp, J., Makita, D., Nishizoe, T., Koide, T., Yoshioka, K., and Rossow, C. AmpPot: monitoring and defending against amplification DDoS attacks. In: *Proceedings of the 18th International Symposium on Research in Attacks, Intrusions and Defenses*. 2015. DOI: 10.1007/978-3-319-26362-5_28 (cit. on pp. 53, 90).

[106] Kreibich, C., Warfield, A., Crowcroft, J., Hand, S., and Pratt, I. Using packet symmetry to curtail malicious traffic. In: *Proceedings of the 4th Workshop on Hot Topics in Networks (Hotnets-VI)*. College Park, MD, USA, 2005 (cit. on pp. 69, 90).

[107] Krupp, J., Backes, M., and Rossow, C. Identifying the scan and attack infrastructures behind amplification DDoS attacks. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 2016. DOI: 10.1145/2976749.2978293 (cit. on p. 112).

[108] Kucherawy, M., Crocker, D., and Hansen, T. *DomainKeys Identified Mail (DKIM) Signatures*. RFC 6376. Sept. 2011. DOI: 10.17487/RFC6376 (cit. on pp. 3, 53, 55).

[109] Kührer, M., Hupperich, T., Rossow, C., and Holz, T. Exit from hell? reducing the impact of amplification DDoS attacks. In: *Proceedings of the 23rd USENIX Security Symposium*. 2014 (cit. on pp. 69, 112).

[110] Kumari, W., Guðmundsson, Ó., and Barwood, G. *Automating DNSSEC Delegation Trust Maintenance*. RFC 7344. Sept. 2014. DOI: 10.17487/RFC7344 (cit. on p. 12).

[111] Kuzmanovic, A. and Knightly, E. W. Low-rate TCP-targeted denial of service attacks: the shrew vs. the mice and elephants. In: *Proceedings of the ACM SIGCOMM 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. 2003. DOI: 10.1145/863955.863966 (cit. on pp. 73, 74, 84).

[112] Kuzmanovic, A. and Knightly, E. W. Low-rate tcp-targeted denial of service attacks and counter strategies. *IEEE/ACM Transactions on Networking* (2006). DOI: 10.1145/1217648 (cit. on pp. 73, 74).

[113] Labs, N. *Unbound – RFC Compliance*. URL: https://nlnetlabs.nl/projects/unbound/rfc-compliance/ (visited on 2023-05-11) (cit. on pp. 6, 37).

[114] Lawler, R. *Facebook explains the backbone shutdown behind its global outage on Monday.* Oct. 2021. URL: https://www.theverge.com/2021/10/5/22710963/facebook-dns-bgp-outage-backbone-maintenance (visited on 2023-08-16) (cit. on p. 4).

[115] Lawrence, D. and Kumari, W. *Serving Stale Data to Improve DNS Resiliency.* Internet-Draft draft-ietf-dnsop-serve-stale-00. IETF Secretariat, 2017 (cit. on p. 68).

[116] Lawrence, D. C., Kumari, W., and Sood, P. *Serving Stale Data to Improve DNS Resiliency.* RFC 8767. Mar. 2020. DOI: 10.17487/RFC8767 (cit. on p. 95).

[117] Lawrence, T. *Akamai's DNS Contribution to Internet Resilience.* Sept. 6, 2017. URL: https://blogs.akamai.com/2017/09/akamais-dns-contribution-to-internet-resiliency.html (visited on 2018-02-06) (cit. on p. 68).

[118] Lenders, M. S., Amsüss, C., Gündoğan, C., Schmidt, T. C., and Wählisch, M. *DNS over CoAP (DoC).* Internet-Draft draft-ietf-core-dns-over-coap-03. Work in Progress. Internet Engineering Task Force, July 2023. 16 pp. (cit. on p. 127).

[119] Levenshtein, V. I. Binary codes capable of correcting deletions, insertions, and reversals. In: *Soviet physics doklady.* Vol. 10. 8. 1966, 707–710 (cit. on p. 24).

[120] Lin, Z., Moon, S., Zarate, C. M., Mulagalapalli, R., Kulandaivel, S., Fanti, G., and Sekar, V. Towards oblivious network analysis using generative adversarial networks. In: *Proceedings of the 18th ACM Workshop on Hot Topics in Networks.* 2019. DOI: 10.1145/3365609.3365854 (cit. on pp. 48, 50).

[121] Liu, Y. and Wang, H. *The Elknot DDoS Botnets We Watched.* Oct. 7, 2016. URL: https://www.virusbulletin.com/conference/vb2016/abstracts/elknot-ddos-botnets-we-watched (cit. on p. 56).

[122] Loibl, C., Hares, S., Raszuk, R., McPherson, D. R., and Bacher, M. *Dissemination of Flow Specification Rules.* RFC 8955. Dec. 2020. DOI: 10.17487/RFC8955 (cit. on pp. 90, 95, 98).

[123] Loibl, C., Raszuk, R., and Hares, S. *Dissemination of Flow Specification Rules for IPv6.* RFC 8956. Dec. 2020. DOI: 10.17487/RFC8956 (cit. on pp. 95, 98).

[124] Luo, X. and Chang, R. K. C. On a new class of pulsing denial-of-service attacks and the defense. In: *Proceedings of the Network and Distributed System Security Symposium.* 2005 (cit. on pp. 73, 74).

[125] Luo, X. and Chang, R. K. C. Optimizing the pulsing denial-of-service attacks. In: *2005 International Conference on Dependable Systems and Networks (DSN 2005).* 2005. DOI: 10.1109/DSN.2005.75 (cit. on pp. 73, 74).

[126] Man, K., Tang, W. K., and Kwong, S. Genetic algorithms: concepts and applications [in engineering design]. *IEEE Transactions on Industrial Electronics* (1996). DOI: 10.1109/41.538609 (cit. on p. 78).

[127] Martinho, C. and Strickx, T. *Understanding how Facebook disappeared from the Internet.* Oct. 2021. URL: https://blog.cloudflare.com/october-2021-facebook-outage/ (visited on 2023-08-16) (cit. on p. 4).

[128] Mayrhofer, A. *The EDNS(0) Padding Option*. RFC 7830. May 2016. DOI: 10.17487/RFC7830 (cit. on pp. 5, 22, 125).

[129] Mayrhofer, A. *Padding Policies for Extension Mechanisms for DNS (EDNS(0))*. RFC 8467. Oct. 2018. DOI: 10.17487/RFC8467 (cit. on pp. 5, 21, 22, 23, 27, 30).

[130] McManus, P. *Improving DNS Privacy in Firefox*. June 2018. URL: https://blog.nightly.mozilla.org/2018/06/01/improving-dns-privacy-in-firefox/ (cit. on p. 21).

[131] McNally, M. *BIND 9.12.0 Release Notes*. Jan. 23, 2018. URL: https://kb.isc.org/article/AA-01554/0/BIND-9.12.0-Release-Notes.html (visited on 2018-02-06) (cit. on p. 68).

[132] Measurement Lab. *The M-Lab NDT Data Set*. Jan. 2021. URL: https://measurementlab.net/tests/ndt (cit. on p. 105).

[133] Meinke, R. *dns-fuzzer*. Mar. 2019. URL: https://github.com/guyinatuxedo/dns-fuzzer/tree/6487b0053d9ee227b515490b9e00289b15a1bbd5 (cit. on pp. 48, 49).

[134] Mirkovic, J. and Reiher, P. L. A taxonomy of DDoS attack and DDoS defense mechanisms. *Computer Communication Review* (2004). DOI: 10.1145/997150.997156 (cit. on p. 69).

[135] Mockapetris, P. *Domain Names – Concepts and Facilities*. RFC 1034. Nov. 1987. DOI: 10.17487/RFC1034 (cit. on pp. 38, 56).

[136] Mockapetris, P. *Domain Names – Implementation and Specification*. RFC 1035. Nov. 1987. DOI: 10.17487/RFC1035 (cit. on pp. 56, 59).

[137] Moura, G. C. M., Castro, S., Heidemann, J., and Hardaker, W. *tsuNAME: public disclosure and Security Advisory*. Tech. rep. SIDN Labs, InternetNZ and USC/ISI, May 2021 (cit. on pp. 90, 117, 128).

[138] Moura, G. C. M., Heidemann, J. S., Müller, M., Oliveira Schmidt, R. de, and Davids, M. When the dike breaks: dissecting DNS defenses during DDoS. In: *Proceedings of the Internet Measurement Conference 2018*. 2018 (cit. on pp. 115, 116).

[139] Moura, G. C. M., Oliveira Schmidt, R. de, Heidemann, J. S., Vries, W. B. de, Müller, M., Wei, L., and Hesselman, C. Anycast vs. DDoS: evaluating the november 2015 root DNS event. In: *Proceedings of the 2016 ACM on Internet Measurement Conference*. 2016 (cit. on pp. 4, 93, 114).

[140] Müller, M., Moura, G. C. M., Oliveira Schmidt, R. de, and Heidemann, J. S. Recursives in the wild: engineering authoritative DNS servers. In: *Proceedings of the 2017 Internet Measurement Conference*. 2017. DOI: 10.1145/3131365.3131366 (cit. on pp. 55, 84).

[141] *nfdump*. Feb. 2023. URL: https://github.com/phaag/nfdump/ (cit. on p. 111).

[142] Nominet. *.UK Register Statistics – 2020*. Jan. 2022. URL: https://www.nominet.uk/news/reports-statistics/uk-register-statistics-2020/ (cit. on pp. 92, 94).

[143] Nominum. *Vantio CacheServe 7*. June 2015. URL: https://nominum.com/wp-content/uploads/2015/06/Vantio-CacheServe7-DataSheet.pdf (visited on 2017-09-12) (cit. on p. 65).

[144] *OpenDNS SmartCache*. URL: https://www.opendns.com/opendns-smartcache/ (visited on 2018-02-06) (cit. on p. 68).

[145] Panchenko, A., Lanze, F., Pennekamp, J., Engel, T., Zinnen, A., Henze, M., and Wehrle, K. Website fingerprinting at internet scale. In: *23rd Annual Network and Distributed System Security Symposium*. 2016 (cit. on pp. 21, 26, 29).

[146] Paxson, D. V., Allman, M., and Stevens, W. R. *TCP Congestion Control*. RFC 2581. Apr. 1999. DOI: 10.17487/RFC2581 (cit. on p. 74).

[147] Paxson, V. An analysis of using reflectors for distributed denial-of-service attacks. *Computer Communication Review* (2001). DOI: 10.1145/505659.505664 (cit. on p. 69).

[148] Pettersen, Y. N. *The Transport Layer Security (TLS) Multiple Certificate Status Request Extension*. RFC 6961. June 2013. DOI: 10.17487/RFC6961 (cit. on p. 125).

[149] Pfeifer, G., Martin, A., and Fetzer, C. Reducible complexity in DNS. In: *Proceedings of the IADIS International Conference WWW/Internet*. 2008 (cit. on pp. 68, 69, 85).

[150] Pham, V., Böhme, M., and Roychoudhury, A. AFLNET: A greybox fuzzer for network protocols. In: *13th IEEE International Conference on Software Testing*. 2020. DOI: 10.1109/ICST46399.2020.00062 (cit. on pp. 40, 48, 49).

[151] Pochat, V. L., Goethem, T. van, Tajalizadehkhoob, S., Korczynski, M., and Joosen, W. Tranco: A research-oriented top sites ranking hardened against manipulation. In: *26th Annual Network and Distributed System Security Symposium*. 2019 (cit. on p. 25).

[152] *PowerDNS Recursor*. URL: https://www.powerdns.com/powerdns-recursor (visited on 2024-06-24) (cit. on pp. 61, 95).

[153] *Public Suffix List*. Jan. 2024. URL: https://publicsuffix.org/ (visited on 2024-01-30) (cit. on p. 14).

[154] *A public and free DNS service for a better security and privacy*. Jan. 2022. URL: https://www.quad9.net/ (cit. on pp. 15, 67, 90).

[155] Quad9. *Quad9's Opinion of the Recent Court Ruling in Leipzig*. Mar. 2023. URL: https://www.quad9.net/news/press/quad9-s-opinion-of-the-recent-court-ruling-in-leipzig/ (visited on 2023-08-15) (cit. on p. 4).

[156] Quad9. *Sony's Legal Attack on Quad9, Censorship, and Freedom of Speech*. Mar. 2023. URL: https://www.quad9.net/news/blog/sony-s-legal-attack-on-quad9-censorship-and-freedom-of-speech/ (visited on 2023-08-15) (cit. on p. 4).

[157] Ranjan, S., Swaminathan, R., Uysal, M., Nucci, A., and Knightly, E. W. Ddos-shield: ddos-resilient scheduling to counter application layer attacks. *IEEE/ACM Transactions on Networking* (2009). DOI: 10.1145/1514070.1514073 (cit. on p. 69).

[158] Rasti, R., Murthy, M., Weaver, N., and Paxson, V. Temporal lensing and its application in pulsing denial-of-service attacks. In: *2015 IEEE Symposium on Security and Privacy*. 2015. DOI: 10.1109/SP.2015.19 (cit. on pp. 73, 74, 75, 77, 83, 85).

[159] Reen, G. S. and Rossow, C. Dpifuzz: A differential fuzzing framework to detect DPI elusion strategies for QUIC. In: *ACSAC '20: Annual Computer Security Applications Conference*. 2020. DOI: 10.1145/3427228.3427662 (cit. on pp. 48, 50).

[160] Rescorla, E. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. Aug. 2018. DOI: 10.17487/RFC8446 (cit. on p. 125).

[161] *RFC Editor Search DNS*. URL: https://www.rfc-editor.org/search/rfc_search_detail.php?title=DNS&page=All (visited on 2023-05-11) (cit. on pp. 6, 37).

[162] Rhee, I., Xu, L., Ha, S., Zimmermann, A., Eggert, L., and Scheffenegger, R. *CUBIC for Fast Long-Distance Networks*. RFC 8312. Feb. 2018. DOI: 10.17487/RFC8312 (cit. on pp. 74, 84).

[163] Rijswijk-Deij, R. van, Sperotto, A., and Pras, A. DNSSEC and its potential for DDoS attacks: a comprehensive measurement study. In: *Proceedings of the 2014 Internet Measurement Conference*. 2014. DOI: 10.1145/2663716.2663731 (cit. on p. 69).

[164] Risk, V. *Resolver DDOS Mitigation*. July 15, 2015. URL: https://www.isc.org/blogs/tldr-resolver-ddos-mitigation/ (visited on 2016-04-07) (cit. on p. 56).

[165] Risk, V. *BIND9 Performance History*. Aug. 2017. URL: https://www.isc.org/blogs/bind9-performance-history/ (visited on 2017-10-25) (cit. on p. 65).

[166] Rizvi, A. S. M., Bertholdo, L. M., Ceron, J. M., and Heidemann, J. S. Anycast agility: network playbooks to fight DDoS. In: *31st USENIX Security Symposium*. 2022 (cit. on pp. 90, 93, 114, 117).

[167] Rose, S., Larson, M., Massey, D., Austein, R., and Arends, R. *DNS Security Introduction and Requirements*. RFC 4033. Mar. 2005. DOI: 10.17487/RFC4033 (cit. on pp. 15, 16, 47).

[168] Rose, S., Larson, M., Massey, D., Austein, R., and Arends, R. *Protocol Modifications for the DNS Security Extensions*. RFC 4035. Mar. 2005. DOI: 10.17487/RFC4035 (cit. on pp. 15, 16, 47).

[169] Rose, S., Larson, M., Massey, D., Austein, R., and Arends, R. *Resource Records for the DNS Security Extensions*. RFC 4034. Mar. 2005. DOI: 10.17487/RFC4034 (cit. on pp. 12, 15, 16, 47).

[170] Rose, S. and Wijngaards, W. *DNAME Redirection in the DNS*. RFC 6672. June 2012. DOI: 10.17487/RFC6672 (cit. on p. 60).

[171] Rossow, C. Amplification hell: revisiting network protocols for DDoS abuse. In: *21st Annual Network and Distributed System Security Symposium*. 2014 (cit. on pp. 44, 53, 55, 56, 67, 69, 75, 76, 90, 112).

[172] Rudl, T. *Netzsperren: Vodafone muss kinox.to blockieren und Kundendaten speichern*. Feb. 2018. URL: https://netzpolitik.org/2018/netzsperren-vodafone-muss-kinox-to-blockieren-und-kundendaten-speichern/ (visited on 2023-08-15) (cit. on p. 4).

[173] *RUSTSEC-2023-0041: Remote Attackers can cause Denial-of-Service (packet loops) with crafted DNS packets*. Available from Rustsec, RUSTSEC-2023-0041. June 2023 (cit. on p. 46).

[174] Sakaguchi, T. *dns-fuzz-server*. Sept. 2019. URL: https://github.com/sischkg/dns-fuzz-server/tree/6f45079014e745537c2f564fdad069974e727da1 (cit. on pp. 48, 49).

[175] Salter, J. *Today's massive Internet outage comes courtesy of Akamai Edge DNS*. July 2021. URL: https://arstechnica.com/gadgets/2021/07/todays-massive-internet-outage-comes-courtesy-of-akamai-edge-dns/ (visited on 2023-08-15) (cit. on p. 4).

[176] Sarat, S. and Terzis, A. On the effect of router buffer sizes on low-rate denial of service attacks. In: *Proceedings of the 14th International Conference On Computer Communications and Networks*. 2005. DOI: 10.1109/ICCCN.2005.1523867 (cit. on p. 84).

[177] Schuster, R., Shmatikov, V., and Tromer, E. Beauty and the burst: remote identification of encrypted video streams. In: *26th USENIX Security Symposium*. 2017 (cit. on p. 21).

[178] Senie, D. and Ferguson, P. *Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing*. RFC 2827. May 2000. DOI: 10.17487/RFC2827 (cit. on pp. 54, 69).

[179] Shadowserver Foundation. *DNSScan Shadowserver Foundation*. Jan. 2018. URL: https://dnsscan.shadowserver.org/stats/ (visited on 2018-01-31) (cit. on p. 63).

[180] Shmatikov, V. and Wang, M. Timing analysis in low-latency mix networks: attacks and defenses. In: *11st European Symposium on Research in Computer Security*. 2006. DOI: 10.1007/11863908_2 (cit. on p. 28).

[181] Shue, C. A. and Kalafut, A. J. Resolvers revealed: characterizing DNS resolvers and their clients. *ACM Transactions on Internet Technology* (2013). DOI: 10.1145/2499926.2499928 (cit. on p. 69).

[182] Siby, S., Juarez, M., Díaz, C., Vallina-Rodriguez, N., and Troncoso, C. Encrypted DNS ⇒ privacy? A traffic analysis perspective. In: *27th Annual Network and Distributed System Security Symposium*. 2020. DOI: 10.14722/ndss.2020.24301 (cit. on pp. 22, 30, 31).

[183] SIDN LABS. *.nl statistieken*. Jan. 2022. URL: https://stats.sidnlabs.nl/nl/registration.html (cit. on pp. 92, 94).

[184] Silva Damas, J. da, Graff, M., and Vixie, P. A. *Extension Mechanisms for DNS (EDNS(0))*. RFC 6891. Apr. 2013. DOI: 10.17487/RFC6891 (cit. on p. 11).

[185] Sirinam, P., Imani, M., Juarez, M., and Wright, M. Deep fingerprinting: undermining website fingerprinting defenses with deep learning. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018. DOI: 10.1145/3243734.3243768 (cit. on pp. 21, 29).

[186] Sommese, R., Akiwate, G., Jonker, M., Moura, G. C. M., Davids, M., Rijswijk-Deij, R. van, Voelker, G. M., Savage, S., Claffy, K. C., and Sperotto, A. Characterization of anycast adoption in the DNS authoritative infrastructure. In: *5th Network Traffic Measurement and Analysis Conference*. 2021 (cit. on p. 92).

[187] Sommese, R., claffy, kc, Rijswijk-Deij, R. van, Chattopadhyay, A., Dainotti, A., Sperotto, A., and Jonker, M. Investigating the impact of DDoS attacks on DNS infrastructure. In: *Proceedings of the 22nd ACM Internet Measurement Conference*. 2022. DOI: 10.1145/3517745.3561458 (cit. on pp. 93, 114).

[188] Souppouris, A. *Turkish citizens use Google to fight Twitter ban*. Mar. 2014. URL: https://www.theverge.com/2014/3/21/5532522/turkey-twitter-ban-google-dns-workaround (visited on 2024-07-28) (cit. on p. 11).

[189] Sridharan, M., Tan, K., Bansal, D., and Thaler, D. *Compound TCP: A New TCP Congestion Control for High-Speed and Long Distance Networks*. Internet-Draft draft-sridharan-tcpm-ctcp-02. Work in Progress. Internet Engineering Task Force, Nov. 2008. 16 pp. (cit. on p. 74).

[190] Standcore. *Standcore DNS Conformance*. URL: https://www.standcore.com/dnsconformance.tgz (visited on 2023-05-11) (cit. on p. 49).

[191] *Most popular top-level domains worldwide as of June 2022*. June 2022. URL: https://www.statista.com/statistics/265677/number-of-internet-top-level-domains-worldwide/ (cit. on p. 91).

[192] Sundaram, M. *Akamai Summarizes Service Disruption (RESOLVED)*. July 2021. URL: https://www.akamai.com/blog/news/akamai-summarizes-service-disruption-resolved (visited on 2023-08-15) (cit. on p. 4).

[193] Surý, O., Toorop, W., Eastlake 3rd, D. E., and Andrews, M. P. *Interoperable Domain Name System (DNS) Server Cookies*. RFC 9018. Apr. 2021. DOI: 10.17487/RFC9018 (cit. on pp. 16, 94).

[194] Swiecki, R. *Honggfuzz BIND9*. Nov. 2020. URL: https://github.com/google/honggfuzz/tree/37e8e813c9daa94dff29654b262268481d8c53ee/examples/bind (visited on 2023-05-11) (cit. on pp. 48, 49).

[195] *DNS Statistics*. Feb. 2023. URL: https://www.nic.ch/statistics/dns/ (cit. on p. 93).

[196] Synopsys. *DNS Server Test Suite Data Sheet*. URL: https://www.synopsys.com/software-integrity/security-testing/fuzz-testing/defensics/protocols/dns-server.html (visited on 2023-05-11) (cit. on p. 49).

[197] Takeuchi, Y., Yoshida, T., Kobayashi, R., Kato, M., and Kishimoto, H. Detection of the DNS water torture attack by analyzing features of the subdomain name. *Journal of Information Processing* (2016). DOI: 10.2197/IPSJJIP.24.793 (cit. on pp. 54, 69, 90, 94).

[198] The Chromium Developers. *DNS over HTTPS (aka DoH)*. Nov. 2019. URL: https://www.chromium.org/developers/dns-over-https (cit. on p. 21).

[199] The Clang Team. *SanitizerCoverage*. URL: https://clang.llvm.org/docs/SanitizerCoverage.html (visited on 2023-05-11) (cit. on pp. 39, 41, 43).

[200] *DNS Implementations*. URL: https://dnsinstitute.com/implementations/ (cit. on pp. 6, 95).

[201] *The path to resolverless DNS*. May 2022. URL: https://blog.apnic.net/2022/05/17/the-path-to-resolverless-dns/ (visited on 2024-01-30) (cit. on pp. 125, 127).

[202] Toorn, O. van der, Krupp, J., Jonker, M., Rijswijk-Deij, R. van, Rossow, C., and Sperotto, A. ANYway: measuring the amplification DDoS potential of domains. In: *17th International Conference on Network and Service Management*. 2021. DOI: 10.23919/CNSM52442.2021.9615596 (cit. on p. 92).

[203] *Tranco list ID G63K*. Aug. 2019. URL: https://tranco-list.eu/list/G63K/10000 (visited on 2019-08-27) (cit. on pp. 24, 25).

[204] Trejo, L. A., Ferman, V., Medina-Pérez, M. A., Giacinti, F. M. A., Monroy, R., and Ramirez-Marquez, J. E. DNS-ADVP: A machine learning anomaly detection and visual platform to protect top-level domain name servers against DDoS attacks. *IEEE Access* (2019). DOI: 10.1109/ACCESS.2019.2924633 (cit. on p. 117).

[205] *Unbound*. URL: https://nlnetlabs.nl/projects/unbound/about/ (cit. on pp. 6, 61, 95).

[206] Van Nice, B. *Drilling Down into DNS DDoS*. Feb. 2015. URL: https://www.nanog.org/sites/default/files/nanog63-dnstrack-vannice-ddos.pdf (cit. on pp. 69, 90).

[207] Verisign. *Zone Files for Top-Level Domains (TLDs)*. Jan. 2022. URL: https://www.verisign.com/en_US/channel-resources/domain-registry-products/zone-file/index.xhtml (cit. on pp. 89, 92, 94).

[208] Vixie, P. and Schryver, V. *DNS Response Rate Limiting (DNS RRL)*. Apr. 2012. URL: https://ftp.isc.org/isc/pubs/tn/isc-tn-2012-1.txt (cit. on p. 94).

[209] Wal, M. van der. *Introducing IBDNS: The Intentionally Broken DNS Server*. Oct. 2022. URL: https://indico.dns-oarc.net/event/44/contributions/949/ (cit. on pp. 48, 49).

[210] Wang, T., Cai, X., Nithyanand, R., Johnson, R., and Goldberg, I. Effective attacks and provable defenses for website fingerprinting. In: *Proceedings of the 23rd USENIX Security Symposium*. 2014 (cit. on p. 21).

[211] Wang, X. and Reiter, M. K. Mitigating bandwidth-exhaustion attacks using congestion puzzles. In: *Proceedings of the 11th ACM Conference on Computer and Communications Security*. 2004. DOI: 10.1145/1030083.1030118 (cit. on p. 69).

[212] Weber, R. *Drilling down into DNS DDoS Data*. 2015. URL: https://indico.dns-oarc.net/event/21/contribution/29/material/slides/0.pdf (cit. on p. 69).

[213] *WebKit: Optionally partition cache to prevent using cache for tracking*. Nov. 2019. URL: https://bugs.webkit.org/show_bug.cgi?id=110269 (visited on 2020-02-24) (cit. on p. 30).

[214] Weiler, S. and Blacka, D. *Clarifications and Implementation Notes for DNS Security (DNSSEC)*. RFC 6840. Feb. 2013. DOI: 10.17487/RFC6840 (cit. on pp. 15, 16, 47).

[215] Weiler, S. and Stenstam, J. *Minimally Covering NSEC Records and DNSSEC On-line Signing*. RFC 4470. Apr. 2006. DOI: 10.17487/RFC4470 (cit. on p. 94).

[216] Weinberg, M. and Barber, P. *Everyday Attacks Against Verisign-Operated DNS Infrastructure*. 2015. URL: https://indico.dns-oarc.net/event/21/contribution/24 (cit. on pp. 56, 67, 69).

[217] Wright, C. V., Coull, S. E., and Monrose, F. Traffic morphing: an efficient defense against statistical traffic analysis. In: *Proceedings of the Network and Distributed System Security Symposium*. 2009 (cit. on p. 21).

[218] Xie, Y. and Yu, S. A novel model for detecting application layer DDoS attacks. In: *Interdisciplinary and Multidisciplinary Research in Computer Science*. 2006. DOI: 10.1109/IMSCCS.2006.159 (cit. on p. 69).

[219] Yang, G., Gerla, M., and Sanadidi, M. Y. Defense against low-rate TCP-targeted denial-of-service attacks. In: *Proceedings of the 9th IEEE Symposium on Computers and Communications*. 2004. DOI: 10.1109/ISCC.2004.1358428 (cit. on p. 84).

[220] Yu, Y., Wessels, D., Larson, M., and Zhang, L. Authority server selection in DNS caching resolvers. *Computer Communication Review* (2012). DOI: 10.1145/2185376.2185387 (cit. on pp. 55, 84).

[221] Zou, Y., Bai, J., Zhou, J., Tan, J., Qin, C., and Hu, S. TCP-Fuzz: detecting memory and semantic bugs in TCP stacks with fuzzing. In: *2021 USENIX Annual Technical Conference*. 2021 (cit. on pp. 48, 50).