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

Jonas Bushart(✉) and Christian Rossow

CISPA, Saarland University, Saarbrücken, Germany
{jonas.bushart,rossow}@cispa.saarland

Abstract. 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 authoritative name server 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 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.

Keywords: DNS · Amplification attack · Application-layer attack

1 Introduction

The Domain Name System (DNS) is at the core of today's Internet and is inevitable for networked applications nowadays. Not only is DNS the primary mean for mapping and translating domain names to IP addresses. Also, several other applications heavily depend on DNS, such as load balancing (e.g., for Content Delivery Networks), anti-spam methods (e.g., DKIM [6], SPF [19], or IP address blacklists [9]) and TLS certificate pinning [10,15]. We rely on the availability of these services for everyday communication. Yet recent incidents have demonstrated how vulnerable DNS is to Denial-of-Service (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 [14]—simply because the authoritative name servers for these services were hosted by Dyn and became unresponsive due to a successful Distributed DoS (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

M. Bailey et al. (Eds.): RAID 2018, LNCS 11050, pp. 139–160, 2018.
https://doi.org/10.1007/978-3-030-00470-5_7

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 [22,44], where miscreants abuse 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.

In this paper, we explore *application-layer* attacks against core DNS infrastructures, namely authoritative name servers (ANSs). 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 [36]. This makes them attractive for botnets.

We start by describing existing forms of application-layer attack against DNS that overload a target ANS 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* [1,47]. 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 [2] 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 ANS, 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 ANS in every other step. This has the effect that resolvers query the target ANS 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 ANS is queried per request the attacker sends.

We complete this paper with an extensive discussion how such attacks can be remedied. We foresee countermeasures that can be deployed by ANS, such as detecting malicious DNS chains or enforcing lower bounds, ensuring more caching, for TTL values. discuss how resolvers can mitigate attacks by capping DNS chains without compromising the benign usage of chains in DNS.

Our contributions can be summarized as follows:

– We present an application-layer attack against DNS that create 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 the 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.

2 Threat Model

We now define the treat 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 authoritative name server (ANS). ANSs answer DNS queries for a particular zone, as queried by any DNS resolver. Next to mapping domain names to IP addresses, ANSs provide several other services, e.g., anti-spam methods (e.g., DKIM [6], SPF [19], or IP address blacklists [9]) and TLS certificate pinning [10,15], making them fundamental on the Internet.

In the highly redundant DNS setting, resolvers choose between all ANSs of a particular zone [33,53]. Yet even a single unresponsive ANS 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 networks routing to this site, therefore the responsiveness of every single ANS matters.

In our model, the attacker targets a specific ANS, e.g., to render domains hosted by this ANS unreachable. We assume that the attacker can host at least one attacker-controlled zone on the target ANS. 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 an account.

The only other requirement on 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 [11]. Internet scans are not a limiting factor, as there are also lists of resolvers available for download [4]. Also, in contrast to amplification DDoS attacks [44], 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.

3 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. Application-layer attacks can target more different resources like CPU time and upstream bandwidth, while volumetric attacks can only consume downstream bandwidths, making them interesting for many cases. 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.

3.1 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 authoritative name server (ANS) 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 naïve 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 ANS. 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, ANS are typically not provisioned to withstand many unique and thus non-cached requests—leaving ANS vulnerable to water torture attacks.

Water torture attacks were observed for the first time in early 2014 [1,41,51] and have since been launched repeatedly. The main ingredient for this attack is sufficient attack bandwidth, which overloads the target ANS with "too many" requests. As this does not require IP spoofing, attackers can easily facilitate botnets to maximize their attack bandwidth. In fact, several large DDoS botnets (e.g., Mirai [2] or Elknot [28]) support DNS water torture.

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

1. 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 exists. `NXDOMAIN` responses are normally caused by configuration error and therefore often monitored.

2. Water torture attacks provide no amplification, as every query by the attacker eventually results in only a single query to the target ANS—unless queries are resent in case of packet loss. The victim-facing attack traffic is thus bound by number of queries that the attacker can send. This is in stark contrast to volumetric attacks that offer more than tenfold amplification [44].

3.2 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 (Sect. 2). The main intuition of our attack is that an attacker can utilize request chains that amplify the attack volume towards a target ANS. This is achieved via aliases, i.e., a popular feature defined in the DNS specification and frequently used in practice.

`CNAME` Records DNS request chains exist due to the functionality of creating aliases in DNS, e.g., using standard `CNAME` resource records (RR) [31,32]. A `CNAME` RR, short for *canonical name*, works similar 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 which overloads the target ANS with "`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 ANS 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 ANS returns this status.

Resource Records Exists. The desired resource record's data is immediately returned by the ANS. 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 request 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 ANSs.

The last case allows chaining of several requests. In 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` records 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.

Note that `CNAME` records provide delegation between arbitrary domains, i.e., also to domains in unrelated zones. If all the `CNAME` records are hosted in the

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

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

Listing 1. Two zones "`target-ans.com.`" and "`intermediary.org.`", which contain a `CNAME` that ends at the ith element in `TXT` records.

same zone, the ANS can provide multiple `CNAME`s in one answer, by already providing the next records in the chain. By chaining `CNAME` records between two different ANSs, i.e., by alternating between them, an ANS can only know the next `CNAME` entry in the chain.

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 ANSs. The first domain will be hosted by the target ANS, and the second (or optionally further) domain(s) by some *intermediary* ANS(s). The zones are configured to contain long `CNAME` chains alternating between both domains. An example can be found in Listing 1, where a chain ping-pongs between the target and an intermediary ANS, 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 ANS's upstream bandwidth. Figure 1 shows the queries sent between the attacker A, a resolver R, and both ANSs. The dashed arrows represent the `CNAME` pointers between the different 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 ANS repeatedly.

This provides severe amplification, as a single request by the attacker results in several requests towards the target ANS. 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 Sect. 4.2). The *amplification*, as observed by the target ANS, is $\lceil N/2 \rceil$, as every second chain record is served by the target ANS.

For illustrative purposes, Fig. 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 ANS is the victim (not the resolver). Furthermore, given two domains, an attacker can

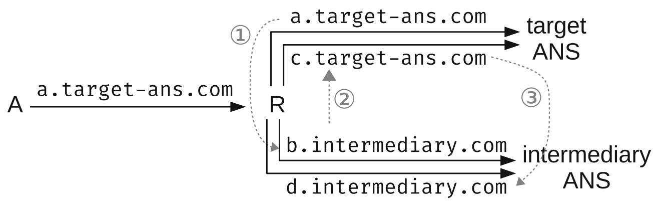

Fig. 1. Attacker A uses resolver R to attack the target ANS. The dashed arrows represent the `CNAME` pointers between the domain. ①–③ show the order of `CNAME` records in the chain. The setup is according to Listing 1.

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 ANS.

There are no strict requirements for the intermediary ANS. In general, intermediary ANSs 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 ANS should be not the same server. Some ANSs will follow `CNAME` chains if the ANS is authoritative for all domains in the chain. Requiring at least one dedicated intermediary ANSs ensures that only one answer can be returned. If the ANS is configured to only return one `CNAME` record, the same ANS can be used, doubling the amplification achieved with this attack. On the other extreme, it is perfectly possible to use *multiple* intermediate ANS, as long as every second element in the chain still points to the target ANS. Distributing the intermediary ANS will increase the reliability and reduce the load for each intermediary ANS, and raise the complexity in 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 are *fast-flux networks* [16] which provide resilience against law-enforcement take-downs and work similar to CDNs. Attackers use fast changing DNS entries to distribute traffic across sometimes hundreds of machines.

3.3 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 Time-to-Live (TTL) value, which specifies how long it may be cached by a resolver, i.e., be answered without querying the ANS. Caching has a large influence on the DNS chaining attack, as it determines how frequent resolvers will query target and intermediary ANSs.

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

Avoiding Caching at Target ANS: Determining the overall impact on ANS requires an understanding 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 ANS. To solve this problem, attackers can disable caching for records hosted by the target ANS. Specifying a TTL value of zero indicates that the resource should never be cached [32, 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 Sect. 4.1.

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 ANS. Queries by other clients for the same RR may arrive in the meantime. 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 ANSs, 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 which make use of the "birthday attack", such as the Kaminsky attack [7,18].

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 equals the optimal attack speed: Fewer queries would not use the resolver's full amplification potential, more queries would waste attack bandwidth.

Leveraging Caching at Intermediary ANSs: Recall that every other chain element points to a record hosted by an intermediary ANS. In principle, this would require resolvers to query the intermediary ANS for every second step in the chain, which significantly reduces the frequency in which the target ANS 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 ANSs, 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 ANS. The frequency of attack queries is thus largely determined by the round trip time (RTT) between resolver and target ANS. In contrast, the RTT between resolver and intermediary ANS is irrelevant.

3.4 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 [5,43] allows arbitrary many subdomains for the chain with only a single entry. Conceptually, DNAMEs are similar to CNAMEs and are created like CNAME records, e.g., "www.target-ans.com. IN DNAME intermediary.org.". The difference is that DNAME records allow the ANS 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 backwards compatibility, ANSs 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 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 ANS for each newly observed subdomain.

Resolvers that support DNAMEs 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 paying attention to any macro- or micro-caching. Furthermore, handling DNAME queries consumes more resources at the target ANS, as resolvers usually create and send synthetic CNAME records in addition to DNAME records.

4 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.

In our manual analysis we focus on the four resolvers Bind[1] 9.10.5,Unbound[2] 1.6.3, PowerDNS Recursor[3] 4.0.6, and Knot Resolver[4] 1.3.2, 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 ANS. We configured the resolvers to use the ANS for all queries, by setting corresponding root hints and configuring the ANS accordingly. Note that in this minimal setup the second VM hosts both the target and intermediary ANS. We thus changed Bind's configuration such that it does not follow `CNAME` chains[5], to simulate two independent ANSs.

We scanned the Internet via Zmap [11] and a custom DNS module, following their recommended scanning guidelines. Networks could opt-out from 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 ANS. We used PowerDNS with a custom back-end as the ANS authoritative for the domains we scanned for. PowerDNS will never follow `CNAME` chains and only return a single `CNAME` record, simulating the two zone setup.

4.1 Caching

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

First, we want to get a general understanding how the different implementations handle non-cacheable responses. We configured our ANS 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, Unbound, and PowerDNS do not cache the response and served 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 frequent resolvers queried the ANS. The delay was chosen such that the resolver has forwarded the previous query to the ANS, but 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 ANS, because resource records of the intermediary ANS can be cached by the resolver and thus do not limit the lookup speed.

[1] https://www.isc.org/downloads/bind/.
[2] https://www.unbound.net/.
[3] https://www.powerdns.com/recursor.html.
[4] https://www.knot-resolver.cz/.
[5] Config option `additional-from-auth` with two zones.

Next, we observe the behavior for longer delays. We delayed the second query until the resolver processed the response of the first queried (and hence started to resolve the second chain element). This simulates queries which 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 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 and Unbound can be queried at most for $\frac{\#\text{chains}}{1s+RTT_{RT}}$. 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 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 special cases the first `CNAME` RR in a query and always perform the authoritative lookup, even when it was fetched from 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.

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 ANS. The queried domain encodes the scan target's IP address, which allows us to (i) ensure that all records are fetched from our ANS and are not cached and (ii) match the scan targets with the queries observed at the ANS. All responses are recorded and filtered to remove domains which do not belong to our test. Figure 2 shows a (simplified) diagram of the connections between our scanner, resolvers, and our ANS. 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 3097203 answers had a TTL of zero. For the same day (2017-08-02), Shadowserver's DNS scan [45] 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, 14400, and 20, which taken together account for 24.8% (1033419) of all responses.

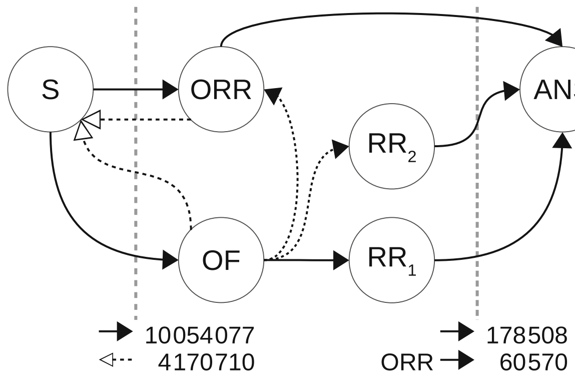

Fig. 2. Connections between different resolver types. Our scanner (S) finds open forwarder (OF) and open recursive resolver (ORR). A forwarder forwards the query to one or multiple recursive resolver (RR). Recursive resolver (RR and ORR) query the authoritative name server (ANS). Dashed arrows mark optional connections, like querying multiple recursive resolver or sending a response to our scanner. An empty arrow head marks a query response.

4.2 Amplification

After seeing that the vast majority of resolvers does 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 follows the chain 17 times, whereas PowerDNS and Unbound only perform 12 and 9 lookup steps, respectively. Knot Resolver performs 33 lookups. Bind 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 10054077 open resolvers and 178508 recursive resolvers. Figure 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 ANS. We can count and distinguish the two types of resolvers based on the traffic captured at our ANS. If the encoded IP address of the scan target and the source IP address of the resolver querying our ANS 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 resolver, because as Kührer et al. [24] found, most open resolvers are routers or other embedded devices. There is little reason for them to host a recursive resolver, because they require more resources.

Figure 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. 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.

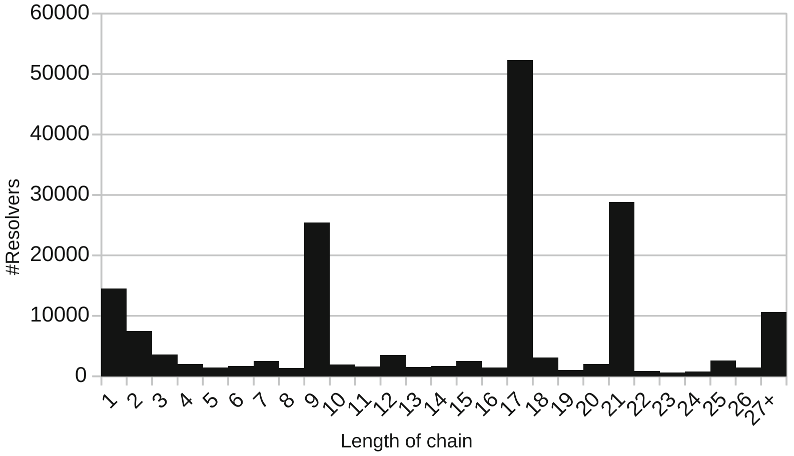

Fig. 3. Supported chain length configurations for 178 508 recursive resolvers discovered with a full Internet scan. The spike at nine corresponds to Unbound and Microsoft DNS version; Bind shows up as the spike at 17. The cause of the 21-spike is unknown to us.

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 ANS 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 ANS, which is the more beneficial setup for an attacker.

All resolvers together (178508) 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 156481 available resolvers. These numbers are lower bounds, because the early drop-off in Fig. 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 Sect. 3.2 that some ANSs do not follow `CNAME` chains, even if they are authoritative for all domains. This is a performance optimization reducing the work required to answer a query. For ANSs, which do not follow chains, all elements of the chain can be hosted on the target ANS thus an amplification factor of 14.34 can be achieved resulting in 89% stronger attacks. Effectively, this removes the intermediary ANSs from the chain and all resource records need to have a zero TTL value.

4.3 Overall Impact

Based on these observations, we conclude that `CNAME` chains enable for attacks that are an order of magnitude larger (measured in queries per second) than naïve water torture attacks. In practice, ANSs can handle 400 000 qps to 2 500 000 qps (queries per second) [3,20,34,42]. 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 ANS, because the ANS 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 ANS ($7.59 \times 178508 \times 75\% \approxeq 1016157$). 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 ANS. 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 ANS.

In case the target ANS does not follow the `CNAME` chain, the stronger attack can be used where all elements are hosted on the target ANS. A single chain causes over 1.9 million queries ($14.34 \times 178508 \times 75\% \approxeq 1919854$) reducing the required bandwidth for the attacker accordingly.

5 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 ANSs.

5.1 Identification and Remedy by ANSs

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

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, ANSs can use this as an indicator for an attack. The target ANS 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 discourage according to the specification). Exceptions are likely required for content delivery networks and cloud provider. 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 ANS 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 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.org.`". This forces a non-trivial amount of work on the ANS. 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 somewhat frequent.

Lower Limit for Time-to-Live (TTL) 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 $\frac{\#\text{chains}}{1s+RTT_{RT}}$. 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 closer 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).

5.2 Response Rate Limiting (RRL)

Response Rate Limiting (RRL) is an effective technique to counter standard DNS-based amplification attacks. If DNS servers are abused for reflective amplification attacks [24,44], 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 (e.g., Google DNS, Quad9, etc.) and especially not for ANSs, 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 ANS. Resolvers will then retry queries, lacking an answer, which again increases the load on the ANS. Filtering all resolver traffic can even increase the incoming traffic ten-fold as observed by Verisign during a water torture attack [51, p. 24].

Additionally, RRL is implemented with a slip rate, which specifies how often the ANS 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 ANS.

An ideal RRL configuration would thus never limit resolvers, as this may actually increase the required resources for the ANS in case of application-layer attacks. Filtering or rate limiting needs to be performed closer to the source. Naïvely, 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.

5.3 Back-Off Strategies

In case of packet loss at the target ANS, 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 in order to give overloaded ANSs 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 ANSs of which both are not reachable. Bind sends a total of five packets with a delay of 800 ms in between packets. The ANS is chosen at random. After the third failed packet, Bind 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 range2730. Worse, Unbound always sends two queries as a pair, which might go to the same or a different ANS. 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 ANS, 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 ANS.

Bind's and PowerDNS's behavior are not problematic, as the number of retries is small and retry delay high. Especially problematic for Unbound is that it sends two identical queries to the same ANS 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 ANSs and additional 19 773 resolvers (32.6%) have such a high RTT to one of our ANSs.

An additional strategy is serving stale cache records [26]. Stale cache records are records in the resolver's cache of which the TTL has expired. A resolver can use them based on the assumption 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 [29] and used by OpenDNS [35] and Akamai [27]. The usability improves as client will receive an answer, which likely is usable, instead of receiving an error and failing to connect.

5.4 Recursion Depth Limit

Finally, also resolvers can more strictly limit the length of `CNAME` chains. Section 4.2 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 Content Delivery Networks (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 [21] data set to identify benign chains. We extracted the `CNAME` entries from 2017-10-05 to reconstruct the longest benign chains, which consists of eight elements (seven `CNAME`s and one final RR). This fits to the shortest recursion limit of nine elements, which we observed for Unbound. Others [38] 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 [13]. 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.

6 Related Work

Application-Layer DDoS in DNS: Several application-layer DDoS defenses have been proposed in the past [12,30,39,52]. 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 which 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 [51] 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 [1] and are a known phenomenon for DNS operators [17,48,50,51]. The DNS operator community focused on implementing mitigations, mainly to stabilize recursive resolver. Takeuchi et al. [47] 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 universities network. Our attack is related to DNS water torture as both are flooding attacks using resolvers, but water torture faces several limitations—including the fact that they can be easily detected.

Reflection and Amplification Attacks: DNS also played a role in recent amplification attacks. The general risk of reflection attacks was identified by Paxson [37] and its full amplification potential presented by Rossow [44]. Different proposals to detect and defend amplification attacks [23,36,44,49] were made. They cover approaches to combat the bandwidth exhaustion, like client puzzles, or prevent source address spoofing. In the context of DNS, Kührer et al. [24,25] analyzed the amplification potential of DNS resolvers. They found millions of open DNS resolver 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 [40]. 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 to chain `CNAME` RRs is well-known and documented. For example, Shue and Kalafut [46] use differences in recursion strategies to fingerprint resolver implementations. Dagon et al. [8] 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. [38] measures the overhead for resolvers while looking up `CNAME` chains and recommend that ANSs should refuse `CNAME` chains before loading the zone files. Furthermore, they recommend that ANSs should also query destinations of `CNAME` RRs, similar to our recommendation in Sect. 5.1. In contrast to prior work our attack focuses on the authoritative name servers instead of the resolvers.

7 Conclusion

We have presented a new DDoS attack against DNS authoritatives 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 authoritative name server. 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 uncachable 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 retransmission on unavailable name servers and limit chains to nine elements. A wide deployment of any of these techniques would severely degrade the performance of the proposed attacks, and we hope that our work raises awareness for the importance of these measures.

Acknowledgement. We thank our anonymous reviewers whose useful comments helped us improving the quality of our paper. This work was supported by the German Federal Ministry of Education and Research (BMBF) through funding for the BMBF project 16KIS0656 (CAMRICS).

References

1. Andrew: Water torture: a slow drip DNS DDoS attack, February 2014. https://secure64.com/water-torture-slow-drip-dns-ddos-attack/
2. Antonakakis, M., et al.: Understanding the Mirai Botnet. In: 26th USENIX Security Symposium (2017)
3. Bellis, R.: Benchmarking DNS reliably on multi-core systems, July 2015. https://www.isc.org/blogs/benchmarking-dns/
4. Censys DNS lookup full IPv4 (2017). https://censys.io/data/53-dns-lookup-full_ipv4
5. Crawford, M.: Non-terminal DNS name redirection. Technical report, RFC Editor (1999). https://doi.org/10.17487/RFC2672
6. Crocker, D., Hansen, T., Kucherawy, M.S.: Domainkeys identified mail (DKIM) signatures. Technical report, RFC Editor (2011). https://doi.org/10.17487/RFC6376
7. CVE-2008-1447. Available from MITRE, CVE-ID CVE-2008-1447, July 2008. http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1447
8. Dagon, D., Antonakakis, M., Day, K., Luo, X., Lee, C.P., Lee, W.: Recursive DNS architectures and vulnerability implications. In: Proceedings of the Network and Distributed System Security Symposium (2009)
9. DNSBL information - spam database and blacklist check. https://www.dnsbl.info/
10. Dukhovni, V., Hardaker, W.: The DNS-based authentication of named entities (DANE) protocol: updates and operational guidance. Technical report, RFC Editor (2015). https://doi.org/10.17487/RFC7671

11. Durumeric, Z., Wustrow, E., Halderman, J.A.: ZMap: fast internet-wide scanning and its security applications. In: Proceedings of the 22th USENIX Security Symposium (2013)
12. Gilad, Y., Herzberg, A., Sudkovitch, M., Goberman, M.: CDN-on-demand: an affordable DDoS defense via untrusted clouds. In: 23rd Annual Network and Distributed System Security Symposium (2016)
13. Hallam-Baker, P.: RFC Errata for RFC 6844 “DNS Certification Authority Authorization (CAA) Resource Record”. Errata 5065, RFC Editor (2017). https://www.rfc-editor.org/errata/eid5065
14. Hilton, S.: Dyn analysis summary of friday october 21 attack. https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/
15. Hoffman, P.E., Schlyter, J.: The DNS-based authentication of named entities (DANE) transport layer security (TLS) protocol: TLSA. Technical report, RFC Editor (2012) https://doi.org/10.17487/RFC6698
16. Holz, T., Gorecki, C., Rieck, K., Freiling, F.C.: Measuring and detecting fast-flux service networks. In: Proceedings of the Network and Distributed System Security Symposium, NDSS 2008 (2008)
17. Internet Systems Consortium: Pseudo Random DNS Query Attacks & Resolver Mitigation Approaches (2015). https://www.nanog.org/sites/default/files/nanog63-dnstrack-winstead-attacks.pdf
18. Kaminsky, D.: It’s the end of the cache as we know it. Presented at Black Ops (2008)
19. Kitterman, S.: Sender policy framework (SPF) for authorizing use of domains in email, version 1. Technical report, RFC Editor (2014). https://doi.org/10.17487/RFC7208
20. Knot DNS benchmark (2017). https://www.knot-dns.cz/benchmark/
21. Kountouras, A., et al.: Enabling network security through active DNS datasets. In: Monrose, F., Dacier, M., Blanc, G., Garcia-Alfaro, J. (eds.) RAID 2016. LNCS, vol. 9854, pp. 188–208. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45719-2_9
22. Krämer, L., et al.: AmpPot: monitoring and defending against amplification DDoS attacks. In: Bos, H., Monrose, F., Blanc, G. (eds.) RAID 2015. LNCS, vol. 9404, pp. 615–636. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26362-5_28
23. Kreibich, C., Warfield, A., Crowcroft, J., Hand, S., 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)
24. Kührer, M., Hupperich, T., Bushart, J., Rossow, C., Holz, T.: Going wild: large-scale classification of open DNS resolvers. In: Proceedings of the 2015 ACM Internet Measurement Conference (2015). https://doi.org/10.1145/2815675.2815683
25. Kührer, M., Hupperich, T., Rossow, C., Holz, T.: Exit from hell? Reducing the impact of amplification DDoS attacks. In: Proceedings of the 23rd USENIX Security Symposium (2014)
26. Lawrence, D., Kumari, W.: Serving stale data to improve DNS resiliency. Internet-Draft draft-ietf-dnsop-serve-stale-00, IETF Secretariat (2017). http://www.ietf.org/internet-drafts/draft-ietf-dnsop-serve-stale-00.txt
27. Lawrence, T.: Akamai’s DNS contribution to internet resilience. https://blogs.akamai.com/2017/09/akamais-dns-contribution-to-internet-resiliency.html
28. Liu, Y., Wang, H.: The Elknot DDoS botnets we watched. Presented at VB2016 Denver. https://www.virusbulletin.com/conference/vb2016/abstracts/elknot-ddos-botnets-we-watched

29. McNally, M.: BIND 9.12.0 release notes. https://kb.isc.org/article/AA-01554/0/BIND-9.12.0-Release-Notes.html
30. Mirkovic, J., Reiher, P.L.: A taxonomy of DDoS attack and DDoS defense mechanisms. Comput. Commun. Rev. **34**, 39–53 (2004). https://doi.org/10.1145/997150.997156
31. Mockapetris, P.V.: Domain names - concepts and facilities. Technical report, RFC Editor (1987). https://doi.org/10.17487/RFC1034
32. Mockapetris, P.V.: Domain names - implementation and specification. Technical report, RFC Editor (1987). https://doi.org/10.17487/RFC1035
33. Müller, M., Moura, G.C.M., de Oliveira Schmidt, R., Heidemann, J.S.: Recursives in the wild: engineering authoritative DNS servers. In: Proceedings of the 2017 Internet Measurement Conference (2017). https://doi.org/10.1145/3131365.3131366
34. Nominum: Vantio cacheserve 7, June 2015. https://nominum.com/wp-content/uploads/2015/06/Vantio-CacheServe7-DataSheet.pdf
35. OpenDNS SmartCache. https://www.opendns.com/opendns-smartcache/
36. Ferguson, P., Senie, D.: BCP 38 on network ingress filtering: defeating denial of service attacks which employ IP source address spoofing, May 2000. http://tools.ietf.org/html/bcp38
37. Paxson, V.: An analysis of using reflectors for distributed denial-of-service attacks. Comput. Commun. Rev. **31**, 38–47 (2001). https://doi.org/10.1145/505659.505664
38. Pfeifer, G., Martin, A., Fetzer, C.: Reducible complexity in DNS. In: IADIS International Conference WWW/Internet 2008 (ICWI 2008) (2008)
39. Ranjan, S., Swaminathan, R., Uysal, M., Nucci, A., Knightly, E.W.: DDoS-shield: DDoS-resilient scheduling to counter application layer attacks. IEEE/ACM Trans. Netw. **17**, 26–39 (2009). https://doi.org/10.1145/1514070.1514073
40. van Rijswijk-Deij, R., Sperotto, A., Pras, A.: DNSSEC and its potential for DDoS attacks: a comprehensive measurement study. In: Proceedings of the 2014 Internet Measurement Conference (2014). https://doi.org/10.1145/2663716.2663731
41. Risk, V.: Resolver DDoS mitigation. https://www.isc.org/blogs/tldr-resolver-ddos-mitigation/
42. Risk, V.: BIND9 performance history, August 2017. https://www.isc.org/blogs/bind9-performance-history/
43. Rose, S., Wijngaards, W.C.A.: DNAME redirection in the DNS. Technical report, RFC Editor (2012). https://doi.org/10.17487/RFC6672
44. Rossow, C.: Amplification hell: revisiting network protocols for DDoS abuse. In: 21st Annual Network and Distributed System Security Symposium (2014)
45. Shadowserver Foundation: DNSScan Shadowserver Foundation, January 2018. https://dnsscan.shadowserver.org/stats/
46. Shue, C.A., Kalafut, A.J.: Resolvers revealed: characterizing DNS resolvers and their clients. ACM Trans. Internet Technol. **12**, 14 (2013). https://doi.org/10.1145/2499926.2499928
47. Takeuchi, Y., Yoshida, T., Kobayashi, R., Kato, M., Kishimoto, H.: Detection of the DNS water torture attack by analyzing features of the subdomain name. JIP **24**, 793–801 (2016). https://doi.org/10.2197/ipsjjip.24.793
48. Van Nice, B.: Drilling down into DNS DDoS (2015). https://www.nanog.org/sites/default/files/nanog63-dnstrack-vannice-ddos.pdf
49. Wang, X., Reiter, M.K.: Mitigating bandwidth-exhaustion attacks using congestion puzzles. In: Proceedings of the 11th ACM Conference on Computer and Communications Security (2004). https://doi.org/10.1145/1030083.1030118

50. Weber, R.: Drilling down into DNS DDoS data (2015). https://indico.dns-oarc.net/event/21/contribution/29/material/slides/0.pdf
51. Weinberg, M., Barber, P.: Everyday attacks against Verisign-operated DNS infrastructure (2015). https://indico.dns-oarc.net/event/21/contribution/24
52. Xie, Y., Yu, S.: A novel model for detecting application layer DDoS attacks. In: Interdisciplinary and Multidisciplinary Research in Computer Science (2006). https://doi.org/10.1109/IMSCCS.2006.159
53. Yu, Y., Wessels, D., Larson, M., Zhang, L.: Authority server selection in DNS caching resolvers. Comput. Commun. Rev. **42**, 80–86 (2012). https://doi.org/10.1145/2185376.2185387