A Flushing Attack on the DNS Cache

Yehuda Afek and Anat Bremler-Barr, *Tel-Aviv University;*
Shoham Danino, *Reichman University;* Yuval Shavitt, *Tel-Aviv University*

https://www.usenix.org/conference/usenixsecurity24/presentation/afek

This paper is included in the Proceedings of the
33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

Open access to the Proceedings of the
33rd USENIX Security Symposium
is sponsored by USENIX.

A Flushing Attack on the DNS Cache *

Yehuda Afek †
Tel-Aviv University
afek@tauex.tau.ac.il

Anat Bremler-Barr ‡
Tel Aviv University
anatbr@tauex.tau.ac.il

Shoham Danino
Reichman University
mrdaninos@gmail.com

Yuval Shavitt
Tel-Aviv University
shavitt@eng.tau.ac.il

Abstract

A severe vulnerability in the DNS resolver's cache is exposed here, introducing a new type of attack, termed DNS CacheFlush. This attack poses a significant threat as it can easily disrupt a resolver's ability to provide service to its clients.

DNS resolver software incorporates various mechanisms to safeguard its cache. However, we have identified a tricky path to bypass these safeguards, allowing a high-rate flood of malicious but seemingly existent domain name resolutions to thrash the benign DNS cache. The resulting attack has a high amplification factor, where with a low rate attack it produces a continuous high rate resource records insertions into the resolver cache. This prevents benign request resolutions from surviving in the DNS LRU cache long enough for subsequent requests to be resolved directly from the cache. Thus leading to repeated cache misses for most benign domains, resulting in a substantial delay in the DNS service. The attack rate amplification factor is high enough to even flush out popular benign domains that are requested at a high frequency ($\sim 100/1sec$). Moreover, the attack packets introduce additional processing overhead and all together the attack easily denies service from the resolver's legitimate clients.

In our experiments we observed 95.7% cache miss rate for a domain queried once per second under 8,000 qps attack on a resolver with 100MB cache. Even on a resolver with 2GB cache size we observed a drop of 88.3% in the resolver benign traffic throughput.

A result of this study is a recommendation to deny and drop any authoritative replies that contain many server names, e.g., a long referral response, or a long CNAME chain, before the resolver starts any processing of such a response.

*Supported by a grant from the Blavatnik Interdisciplinary Cyber Research Center (ICRC), Tel Aviv University.

†Member of the Checkpoint Institute of Information Security.

‡Member of the Checkpoint Institute of Information Security.

1 Introduction

Several DDoS attacks on the DNS system have been discovered by attackers and researchers in the past decade [10, 11, 14, 42]. These attacks have targeted the communication or/and computation load of either resolver or authoritative DNS servers, by generating high packets or CPU load amplification factors, or simply using a large botnet. None of these attacks have succeeded in effectively thrashing the DNS resolvers' benign cache (as oppose to the negative cache, NCache), making the resolver ineffective.

In this paper we present a low rate, e.g., $\sim 8Kqps$ (queries per sec.), carefully crafted attack requests that generate a high rate (cors. $\sim 12M$ per sec.) of record insertions into the benign cache of the resolver, that is, a high amplification factor attack on the benign cache, which we call *cache amplification factor (CAF)*. Such a continuous high rate stream of records insertion into the benign cache ($12Mrecords\ ps$) evicts from a cache of size $10MB$ any domain name that is not queried at least 800 times per second, thus effectively leaving the resolver without a cache. This requires the resolver to re-query the authoritative servers on each client query, except perhaps the absolutely most popular one, thus deteriorating the resolver operation until it denies service to its clients.

Mounting such an attack on the benign cache of a resolver is not trivial. Simply using a huge botnet to request hundreds of thousands of different domain names at a rate of millions or even thousands per second is expensive and hard. It would choke the communication lines before reaching the resolver itself, in addition to being easily detected and blocked by behavioral analysis systems, such as IPS's and/or firewalls. Furthermore, using randomly generated fake domain names, as in the water torture attack [42], would go into the NCache (Negative Cache, or NX - NonExistent) limited size portion of the cache, thus leaving the benign cache unaffected.

Here we discovered a vulnerability by which an attacker can cause the insertion of a large number of records into the benign cache with just one query to the resolver, that is over a million records per second by an attack of 1K malicious

queries per second to the resolver. More precisely, in a maximum size TCP message ($\sim 65KB$) response by a malicious NS, about 2000 NS names that can fit (the exact number depends on the number of characters in each NS name) which results in the insertion of 2000 resource records into the cache. As each benign query to a resolver adds at least one resource record into the resolver's cache, the cache amplification factor is 2000. In bytes each malicious request may result in the insertion of at least $100KB$ into the cache since a minimum resource record of a NS name in the cache is about $50B$. The query our attacker employs generates a response from an authoritative server hosting an attacker zone file, that instructs the resolver to delegate and recursively resolve a large number (up to 2000) name server (NS) names. Such a technique was used by [10, 11] to create a large communication and computation amplification factor attacks. As a result of these attacks the vendors have issued a number of CVE's limiting the number of recursive resolutions (to 5 and 20) each client query may result in [36, 37]. However, these mitigations do not limit the number of NS names stored in the cache while resolving the limited number of NS names. Thus leaving the door open to the current DNS CacheFlush attack.

In this paper we point out two types of such large responses by authoritative servers that require recursive resolution of many additional names; the CNAME and the referral types.

In addition to the high cache miss rate, these large referral responses from the authoritative server come with another complexity price tag due to the extra processing each such referral response (and as we show in this paper also long CNAME chains) requires. These processing include, opening a TCP session (the response does not fit in a UDP packet), checking the consistency and whether an IP address is already available for each name in the response, inserting the list into a memory buffer (in addition to inserting them into the benign cache), and issuing the corresponding extra (20 rather than 1800) resolution requests to resolve the IP address of each of the first 20 names in the list. This overhead together with the extra cache misses easily takes the resolver down and denies service to legitimate clients.

In this paper we conclude and suggest that the DNS system should be changed to eliminate large referral responses (> 13 NS names) and long (> 8 NS names) long CNAME chains. Resolvers should discard such responses before performing any processing on them, and authoritative servers should prohibit zone files that generate such responses. A few more modest mitigation suggestions are provided here, such as, trimming the long responses upon their reception.

In this paper we built a test bed and measured the effect of DNS CacheFlush on the latest versions of BIND9 [24] and UNBOUND [26] with default cache sizes ranging from 10MB to 100MB, as well as with a 2GB cache size that simulates our university environment, using two methods: CNAME chains containing 17 domains and a referral response with 1,500 NS names. To determine whether a benign domain is likely to experience a high rate of cache misses, we developed a model that predicts whether the cache entry of a domain is going to be flushed between two queries of the same benign domain, and the likelihood of a high cache miss rate as a function of the attacker and benign rates, and the cache size. It is also possible to apply the model to the distribution of benign domains, which means that, given a resolver's benign domain distribution and cache size, we can determine for each attack rate the benign domain rate starting from which all benign domains with a lower rate are highly likely to have a cache miss. Furthermore, we carried experiments to estimate the effect of some mitigation suggestions, to separate the contribution of the cache amplification factor from that of the long response processing cost, to the attack strength.

A review of the benign and negative cache importance, as well as cache management techniques is presented in Section 2 followed by a discussion of the threat model in Section 3. The CacheFlush Attack versions and the attack model are discussed in Section 4. The experimental set-up and comparison between our model prediction and the experimental results are presented in 5. Alternative mechanisms to mitigate the CacheFlush attack are suggested in Section 6. Additional vulnerabilities we found during our research are shortly mentioned in Section 7 and related work is shared in Section 8. We review the responsible disclosure procedure in Section 9, and draw conclusions in Section 10.

2 Preliminaries

2.1 DNS Resolution Process

DNS Resolver Server (sometimes called recursive resolver server, or just recursive server). To obtain the IP address of hello.world.com a client (e.g., a browser) queries a DNS resolver with the name hello.world.com. If the answer is not in the resolver cache, the resolver then issues several queries to the authoritative hierarchy to obtain the desired IP address, which it then stores in its cache and returns to the querying client.

Authoritative Name Servers. An authoritative name server maintains and provides official up-to-date DNS records (containing the IP addresses) for domain names.

Resource Records. Data from DNS is organized and stored in resource records (RRs). Each RR consists of an owner name, which is the fully qualified domain name of the tree node where the RR is located, a type, a time-to-live field (TTL), and a value. Value fields are structured differently according to the type of record. It is noteworthy that multiple records with the same owner name and type can coexist, provided they contain distinct data, together forming an RRset. The various record types, including TXT, SOA, NS, A, CNAME, serving different purposes.

2.2 Delegation Response

In response to a resolver query an authoritative server may delegate (refer) the resolver to obtain the answer from a different authoritative server (the delegation can also be to a different name in the same authoritative server). For example, ".com" can delegate the resolution of hello.world.com to world.com.

In a referral response with a list of NS names, the resolver is delegated to obtain its answer from any one of the name servers in the list. The number of NS names may be large, and their IP addresses (known as glue record) may not always be included. The referral-list response does not include glue records because authoritative servers cannot provide IP addresses for domains whose origins reside outside their zone. This policy protects the servers from DNS poisoning attacks by identifying them as Out-of-Bailiwick name servers [23]. For example, in Figure 1, attack.com delegates the resolution of e1.attack.com to the name servers (NS) cacheflush1.delegation.attack in a referral response. Hence, the resolver would send a new query to resolve the NS name cacheflush1.delegation.attack.

2.3 CNAME Record

A Canonical Name (CNAME) record in the Domain Name System (DNS) is a pivotal element that enables aliasing or redirection of one domain to another. Essentially, a CNAME record serves as a pointer, allowing a domain to function as an alias or nickname for another domain's canonical or primary name. This redirection is not limited to a single level; it can create a CNAME chain, where one CNAME points to another, forming a sequential chain of aliases. When a DNS query encounters a CNAME record, it redirects to the domain specified in the record, inheriting its associated DNS information. This functionality is commonly employed for various purposes, such as creating subdomains, facilitating domain migration, CDNs, providing alternate names for existing domains etc. For example, in Figure 2 the CNAME record e1.attack.com points to e2.attack.com

2.4 DNS Resolver Cache: Benign vs. Negative Cache

DNS resolver cache plays a critical role in making the Domain Name System (DNS) resolution process more efficient. The cache serves as a temporary storage repository within the resolver, storing previously resolved DNS queries. It retains a variety of DNS resource records (RRs) types, including domain names and their IP addresses, NS records, and CNAME records, which alias one domain to another. The resource records may be classified into two classes: benign cache records, which store successful resolutions, and negative cache [21] , which store information about non-existent domains or names for which a NXDOMAIN or NODATA is received in the resolution attempt.

Resolvers provide a special limited size section of the cache for the negative replies, called Ncache. It is crucial in preventing unnecessary repetition of failed queries, which could be an attack such as Water torture [42]. While negative caching was previously optional, it is now part of the DNS specification [21]. A large proportion of DNS traffic on the Internet is eliminated by the negative cache [21]. Chen et al. [16] analyze real-life DNS traces and show that the Non-Existent Domain (NXDomain) traffic constitutes almost 40% of the traffic from the authoritative structure to the recursive resolvers.

There is a dynamic distribution of memory allocation between the positive and negative caches in the resolvers we examined, with the positive cache having a priority advantage [27]. When the cache reaches its size limit and a positive record arrives, a record from the negative cache will be evicted from the cache, in the event that the cache is full and a non-existent record arrives, a record will be evicted from the negative cache, or from the positive cache if its TTL value is zero.

2.5 BIND Queue Management

BIND resolvers systematically push DNS queries into their management queue based on their arrival. By default, the queue size is determined based on the resources allocated to the resolver, but this configuration can also be customized as necessary on specific requirements. The queue is governed by a recursive client deletion mechanism, which imposes both a soft limit and a hard limit on its size. Upon reaching the soft limit, the resolver selectively drops pending requests, allowing it to manage and prioritize ongoing queries. However, when the hard limit is reached, the resolver takes a more stringent approach by dropping all requests [13].

3 Threat Model

To mount a DNS CacheFlush attack an attacker has to:

1. Control one or more clients from which it issues the malicious queries.

2. Control an authoritative name server configured to respond with a CNAME chain or with referral responses with the crafted list of NS names.

It is easy and affordable to acquire authoritative name servers by first buying and registering new domain names. In our experiment, we bought 2 domain names for just under 1 USD each in less than five minutes and dynamically connected them to our authoritative server in our cloud setup.

The attacker associates 1500 NS names with each malicious domain name in the authoritative zone file, leading to a large zone file. Each record of NS name looks for example like this: "e1 8600 IN NS cacheflush1.delegation.attack", that is,

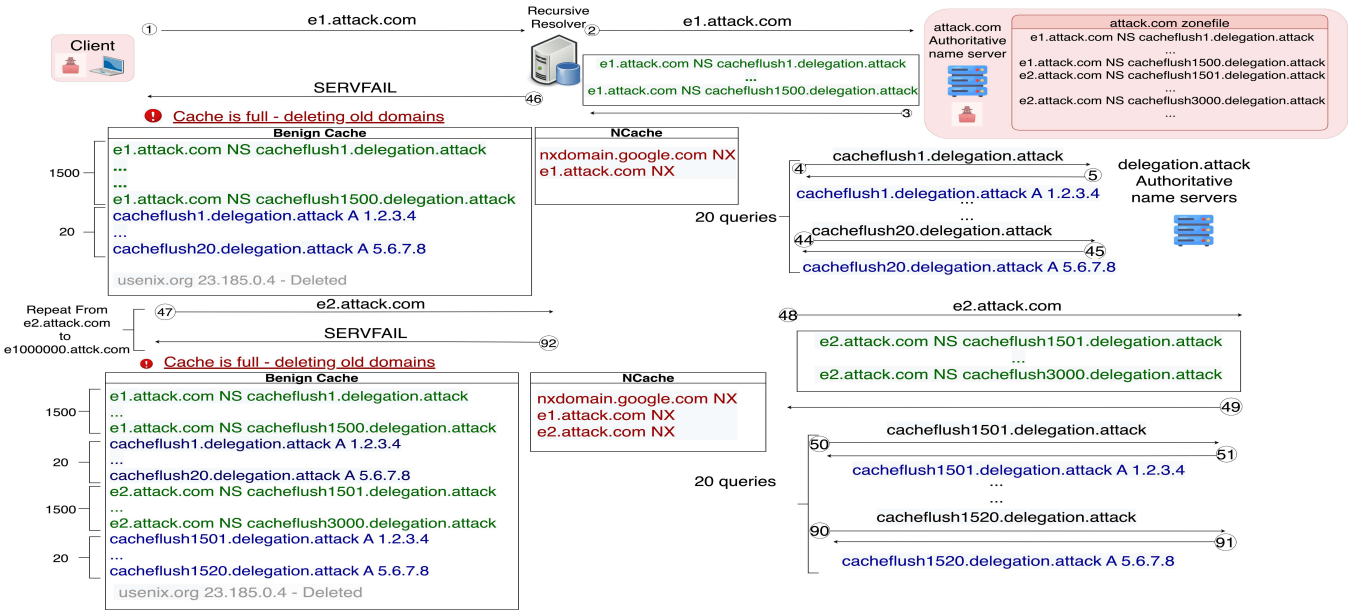


Figure 1: NS records type of CacheFlush Attack; An attacker requests e1.attack.com following by e2.attack.com and gets referral responses with 1500 names from the authoritative name server. The resolver queries the first 20 names in each list and evicts from the benign cache the RR corresponding to the referral list and the 20 responses. This evicts usenix.org from the cache.

contains 42 bytes. Therefore, a referral list of length 1500, has $42 * 1500 = 63KB$ in the zone file. Clearly, the larger the cache of the target resolver the more malicious domains the attacker needs to maintain in the zone file. For a 100MB target cache the required zone file is $1,000 * 63KB = 63MB$ since each line of the zone file results in flushing $1KB$ in the cache.

The cost of operating the malicious authoritative DNS server is relatively low. There are two options (1) self-managing either in the cloud or on-premises or, (2) outsourcing to a managed DNS service provided by companies such as Cloudns, Cloudflare, GoDaddy and Namecheap [18, 19, 22, 35]. The main expense of self-managed authoritative DNS in the cloud is the cost of outgoing traffic. It typically amounts to around $0.6 per minute for an attack that flushes a 2GB resolver cache so benign domains that are requested once per second or less have to be re-fetched on each query. See Appendix A for more details.

The four managed DNS services, Cloudns, Cloudflare, GoDaddy and Namecheap, allow the configuration of long referral lists (with 1000 NS or more) or long CNAME chains for each domain. Each service was tested by configuring a long referral list or long CNAME chain for one domain, and making sure it returned a large response that contained the entire referral list or CNAME chain, all while adhering to ethical guidelines. However, there is a limit on the free-of-charge authoritative services, including the total number of supported domains and resource records (RR) in the Zone. Performing an attack as in our experiments requires a zone file size larger than the free version allowances. For example, to flush a 1GB cache, 10,000 domains each containing 1,500 RRsets are required. Nevertheless, the cost to upgrade and support an unlimited number of requests in Cloudns [19], is $14.95 per month. For the others, customized plans are available, with costs varying according to usage.

An alternative common practice for example, is for attackers to compromise DNS operators' credentials and manipulate zone files, sometimes even getting access to their registrar records, as demonstrated by recent DNS hijacking attacks [30, 32]. Thus getting all the required services for free.

Adaptive attacker: An adaptive attacker can measure the target cache size and determine the required zone file size. To measure it the attacker requests a seemingly benign test domain name at rate r at which it wants to flush the target cache. So that any domain name that is queried at a rate lower than r would then be evicted. The attacker authoritative is the authoritative for the test domain. During the attack, the attacker expects to receive queries for the test domain in its authoritative at rate r, or otherwise, if it receives fewer queries, the cache is not flushed fast enough. In which case the attacker should increase the number of distinct malicious domains in the zone file and increase the attack rate.

Multi-threading: All our tests and measurements use BIND and Unbound multi-threading packages. Clearly if we used a single thread version, then the attack would be more harmful.

The default cache size of most resolvers is between 8MB to 200MB [3–7, 9], in practice, medium size resolvers (e.g.,

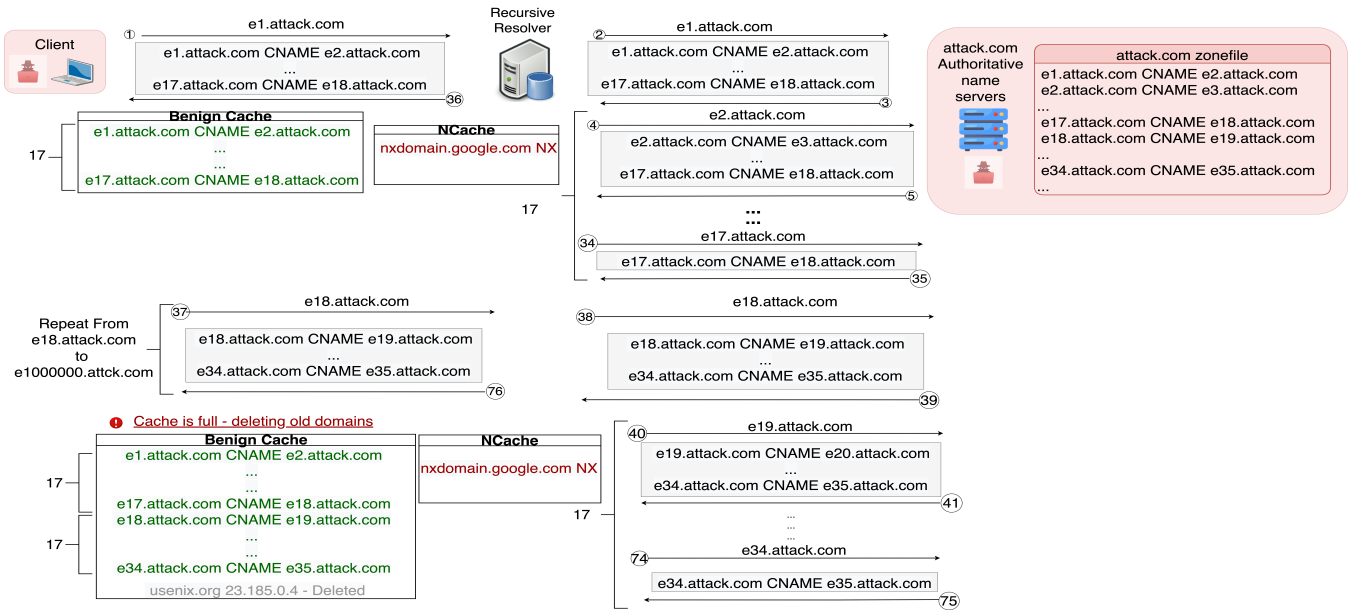


Figure 2: CacheFlush Attack using CNAME records example; An attacker requests the e1.attack.com domain and fills the benign cache with 17 RR sets from his authoritative name server

University resolver) use a 2*GB* to 4*GB* cache. We therefore use in our experiments, a cache size of 10*MB*, 100*MB*, and 2*GB*.

4 CacheFlushAttack

Two variations of the CacheFlush attack are presented here: the Name-servers based, NSCacheFlush, where the attackers' authoritative response includes a large referral response of NS delegations, and the CNAME based, CNAMECacheFlush, where the response includes a long chain of CNAMEs.

Notice that in either attack variant the malicious RR records (NS or CNAME) are inserted into the benign cache (and not to the ncache) regardless of whether the DNS resolution of the original query from the attacker results in an IP address, failure, or NS response.

4.1 NSCacheFlush Attack

Upon receiving a response with a referral response with many NS names from the authoritative name server, the resolver caches the entire list, even though it is not going to resolve the names in the list except the first 5.

Figure 1 illustrates the attack: the attacker sends malicious requests from the client to the targeted resolver, such as e1.attack.com, e2.attack.com, etc. For each query, the resolver queries the relevant authoritative (controlled by the attacker) and receives a referral response containing 1,500 names (step 3). As a result, all 1,500 names are stored in the resolver's benign cache.

Subsequently, in order to resolve the original query, the resolver queries the first k NS and caches them as well in the benign cache (steps 4-45). In BIND and UNBOUND implementations, the resolver queries the first k=5 names [11]. These NS names can be domains of DNS servers, or domains that correspond to DNS non-responsive servers, which are not controlled by the attacker.

Finally, the resolver queries the NS (e.g., e1.attack.com) to resolve the original query. If the resolution succeeds, it stores it in the benign cache; otherwise, it stores it in the Ncache (if it received SERVFAIL, as in step 92 our example).

In the attack, the attacker controlled authoritative server repeatedly responds with a referral response with n new NS names from the zone file, until all the names in the zone file have been used at which point it starts all over. For each such referral response, 1520 RR are evicted from the benign cache, for a total of $n \times 1520$ RRs, regardless of whether the resolution was successful or failed. The attacker can use the adaptive attack zone file (see Section 3 to adjust the zone file size until flushing the entire cache).

In our example, the benign domain usenix.org is queried less frequently than the cache flushing rate, causing it to be evicted from the cache by the LRU algorithm before it is queried again.

4.2 CNAMECacheFlush Attack

This attack version relies on the fact that a resolver resolves a CNAME chain sent from an authoritative name server until it reaches the limit of p CNAMES set by its vendor and store only the p first CNAMEs in its benign cache. BIND9

sets a resolution limit for a CNAME chain up to 17 times (UNBOUND allows 9, Google 15, and Cloudflare 20) [15], aiming to prevent infinite loops.

Figure 2 illustrates the attack on BIND implementation by an attacker who sends a malicious query to the attacked resolver, the authoritative name server controlled by the attacker responds with a chain of 17 CNAME records (the maximum number of records that will be stored also in the benign cache). The resolver saves the first CNAME record in its benign cache and queries the same authoritative name server for the next name in the chain even though it received the entire chain in the first answer [28], so it continues until BIND's limit is reached, saving a total of 17 records in its benign cache (steps 4-35). This version differs from NSCacheFlush in that only the limit 17 names are stored in the cache, regardless of the length of the CNAME chain. Therefore, this attack is not as effective at filling the cache as NSCacheFlush. In 7.1, we describe the effect of a CNAME chain with a length of 1,500 on the resolver. This version works the same when an IP address is returned at the end of the chain or not, and in either case the chain is not inserted into the negative cache but into the benign cache, as depicted in the figure, the negative cache does not change during an attack.

4.3 Effective Flush Rate

What rate of attack causes a high cache miss rate ($> 80\%$) for a benign domain d that is queried at a rate r_d? A cache of size s is flushed at a rate of $\frac{attack\_rate * CAF}{s}$, where CAF is the attack cache amplification factor, the amount of cache memory evicted by each attacker request. For simplicity, we assume the benign and the attacker querying rates are at a constant uniform rate. Thus a cache miss would occur iff, the benign rate is lower than the flushing rate:

$$\frac{\text{Attacker Request Rate} \times \text{CAF}}{\text{Cache Size}} > \text{Benign Rate} \qquad (1)$$

CAF is a function of the number of RRs sent in a response to the resolver, times the size of each corresponding RR. In the conducted NSCacheFlush experiments, every response results in the insertion of $1,520$ RRs, of size (~ 67 bytes), i.e., CAF = $100KB$.

4.4 Resolver Cache Miss

Next we calculate the average cache miss on a resolver. The distribution of domain queries can be characterized by a power law distribution [44] (see details in Section 5.3): $y = ax^{-b} + c$, where y is the domain rate in qps, x is the domain rank (1 is the most frequent domain), b is the exponent and it determines the slope of the power-law line on a log-log scale plot (the steeper the slope, the greater the variation in the rate between the top ranked domains). a and c are both related to the popularity of the resolver, the number of clients and the frequency with which it is used. a scales the function vertically and c is a constant that shift the function vertically.

Similar to Equation (1) above, we find the *border rate* of an attack, which is the rate such that all benign domains with lower rate than this border rate with high probability experience a cache miss under this attack rate, using the response size (rsize) and cache size (csize):

$$\text{Border rate}_{\text{csize, rsize}}(\text{attacker rate}) = \frac{\text{Attacker rate} \times \text{rsize}}{\text{csize}} \qquad (2)$$

We will then place the *border rate* as y in the power-law function $y = ax^{-b} + c$ and find the domain rank x. We denote it as m and name it the *border rank*.

The sum of the rates (y) over the domains with rate up to the *border rate* (n denotes the total number of domains the resolver was queried), represents the number of cache misses predicted by the model:

$$\sum_{x=m}^{n} ax^{-b} + c$$

Thus the average cache miss percentage for the resolver, is:

$$\frac{\sum_{x=m}^{n}(ax^{-b}+c)}{\sum_{x=1}^{n}(ax^{-b}+c)} \qquad (3)$$

[1] In the next section 5.4, we present experiments that show the accuracy of our simple modeling.

5 Experimental Results

5.1 Experiment Setup

Our experiment setup resides in Azure cloud and included DNS recursive resolvers, Authoritative name servers, an attacker, and two benign machines that issue requests in parallel to demonstrate multiple users simultaneously. This allowed us to evaluate the impact of the attack on benign users. The following machines, each Intel(R) Xeon(R) CPU E5-2673 v4 @ 2.30GHz x64 with 2 vCPUs 8 GiB RAM and Linux (Ubuntu 20.04) operating system, were used:

1. Resolver machine with most recent versions, BIND9 (9.18.21) or UNBOUND (1.19.0) resolvers

[1] It is possible to approximate the function using generalized harmonic numbers $H_n^{(r)}$:

$$\frac{\sum_{x=m}^{n}(ax^{-b}+c)}{\sum_{x=1}^{n}(ax^{-b}+c)} = 1 - \frac{\sum_{x=1}^{m}(ax^{-b}+c)}{\sum_{x=1}^{n}(ax^{-b}+c)} = 1 - \frac{aH_m^{(b)}+cm}{aH_n^{(b)}+cn} \qquad (4)$$

which can be expressed alternatively with Riemann [8] and Hurwitz [2] zeta functions:

$$1 - \frac{aH_m^{(b)}+cm}{aH_n^{(b)}+cn} = 1 - \frac{a(-\zeta(b,m+1))+a\zeta(b)+cm}{a(-\zeta(b,n+1))+a\zeta(b)+cn} \qquad (5)$$

2. Client machines, benign clients and an attacker client, each equipped with a Resperf tool.

3. Authoritative server that is used to craft the malicious CNAME and referral response.

4. Authoritative server that owns the domain referred to by the NS names in the referral response.

5. Authoritative server to which benign users are referred.

We placed the client, resolver, and authoritative servers in the same Azure region so our measurements would not be impacted by any significant Internet delays. In order to simulate as closely as possible to real world situation, the machines communicated through their internet interfaces rather than through a local area network.

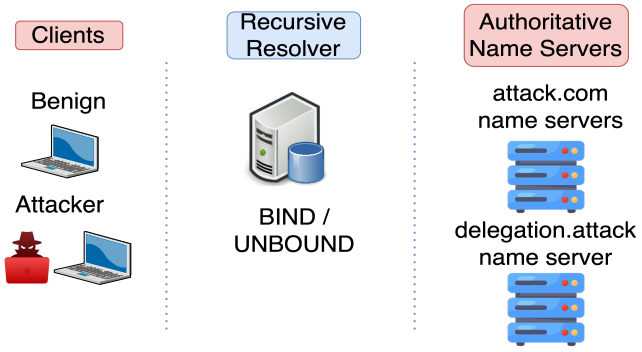


Figure 3: Test environment

5.2 Isolated Lab Setup

In addition to the cloud environment, we have created an isolated lab environment for conducting research and reproducing experiments. This setup includes a BIND resolver with the latest version (9.18.21), and four authoritative servers: a local root authoritative server and three others to simulate the "attack.com" and "delegation.attack" authoritative servers depicted in Figures 1 and 3. The authoritative name servers are implemented with Name Server Daemon (NSD) version 4.3.3. To ensure the setup poses no external risk, it operates locally within a closed Docker container environment. The clients are deployed on the same machine, configured to send DNS queries directly to the local recursive resolver. The setup configuration and environment are available on GitHub [43].

5.3 Domain Rate Distribution

To test the effectiveness of our attack in a realistic environment, we collected statistics on the DNS environment of our university over a period of 39 days and used it to model typical clients' behavior. We also examine two data sets that

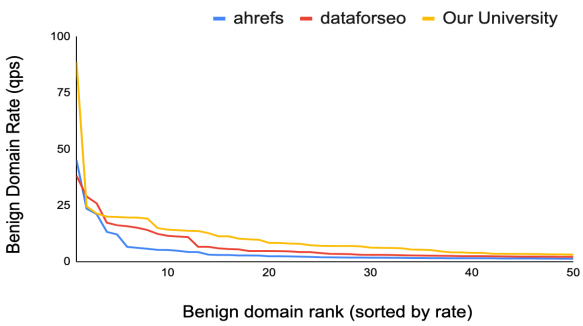


Figure 4: Domain Rate Distribution for three datasets, our university, AHREFS, DATAFORSEO (on 50 highest-ranked domains).

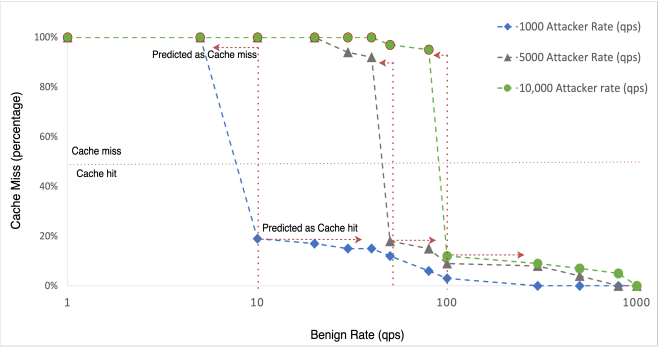
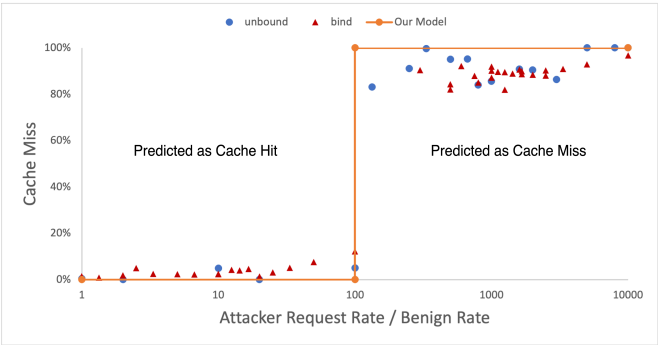
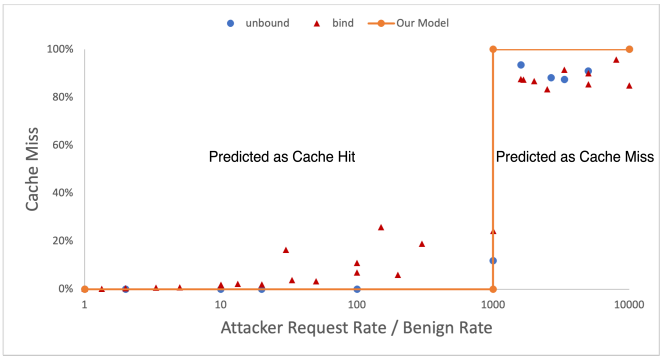


Figure 5: Cache miss percentage measured on different benign domain request rates with different NSCacheFlush attacker request rates on BIND resolver with 10MB cache size.

indicate how frequently the most common domains are visited [12,20]. The distribution of domains from all the data sets was characterized by a power law distribution [44]: $ax^{-b}+c$ where $b = 0.79$ for our university data, b = 0.93 for [12] and $b = 0.7$ for [20] (a and c relate to the number of clients which is reflected in the variety of Attacker rates in our experiments). Figure 4 illustrates the similarity between our university and the other data set we checked. Due to its similarity to real-world distributions, our university's distribution was used to test the domain distribution model. We use these distributions on our own domains in order not to overload public domains on the internet. The attack file was constructed in which each domain is evenly distributed in the file according to its distribution (e.g., a common domain that accounts for 20% of all domains appears 20% times at equal intervals in the attack file). By running the attack file at a higher rate in different experiments, we were able to control the rate at which each domain was queried.



(a) Results for 10MB cache size; According to our model, there is a high probability of cache misses when the $\frac{\text{attacker request rate}}{\text{benign rate}} > 100$, similar to the results obtained in the experiments.



(b) Results for 100MB cache size; According to our model, there is a high probability of cache misses when the $\frac{\text{attacker request rate}}{\text{benign rate}} > 1,000$, similar to the results obtained in the experiments.

Figure 6: The results of the experiments on BIND9 and UNBOUND implementations compared to our model predictions on NSCacheFlush attack; The percentage of cache miss as a function of attacker request rate divided by benign rate for a different cache sizes.

5.4 Attack model vs Experimental results

Here we show the experiments that were conducted to test the model's results. First, we measured (Figure 5) the cache miss percentages for one benign domain under various benign rates between 1 to 1000 qps and attacker request rates of 1000, 5000 and 10,000 qps using a BIND9 resolver with a cache size of 10 MB. According to our model, a cache miss is expected with high probability for benign domain with a rate less than 10qps and attack rate of 1000qps. An attack rate of 5000qps should result in a cache miss for any domain with a rate below 50qps and an attack rate of 10,000 should result in a cache miss for any domain with a rate below 100qps. Figure 5 shows that the model predicts well the cache misses under different attack rates.

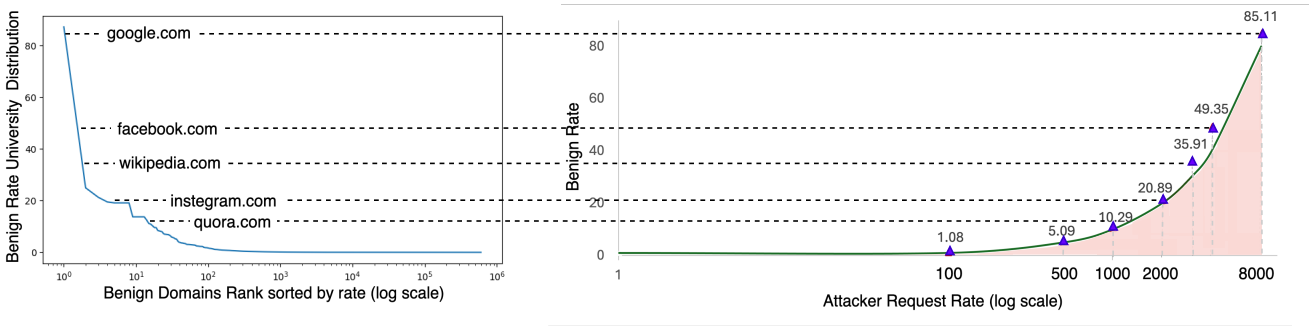
The second experiment (Figure 6) expands upon the first experiment and validated the predictions made by the model for BIND and UNBOUND implementations using two cache sizes of 10mb and 100mb for different attacker request rates between 100 and 10,000 qps and benign domain queries between 1 and 1000 qps. The analysis of Figure 6 shows that the model correctly predicts the cache miss rates for all the different experiments; For example, according to equation (1), for cache size of 10MB (10,000 KB): $\frac{\text{attacker request rate}x100}{\text{benign rate}} > 10,000$ so $\frac{\text{attacker request rate}}{\text{benign rate}} > 100$; and as shown in Figure 6a, 100 is indeed the value after which all cache misses occur.

As can be seen in both experiments, the cache miss rate is high or very low based on our model, but mostly it was not 0% or 100%, this is because the resolver manages a multi-threaded queue that holds queries since they arrive at the resolver until the client receives a response. When the queue reaches its quota (hard limit) , see Section 2.5 for details, it is reset and all malicious queries waiting for an answer in the queue are deleted. At this point, if the cache has not yet filled up and a benign domain is queried, a cache hit occurs.

The third experiment (Figure 7) included testing the validity of equation (2) on our university domains' distribution, when different domains were queried at different rates in an attempt to illustrate a real-world scenario. Each attacker request rate in the model is matched with a *border rate*, which all benign domains with a lower rate have a high probability of missing the cache. In Figure 7, we present the results of our comparison between the graph model and the experiment results on BIND9 implementation with a cache size of 10MB.

Figure 7a shows the distribution of domains in our university; for example, we observe that the most popular domain, google.com, is requested an average of 85 qps. For each domain rate in Figure 7a, we draw a parallel line to Figure 7b, where the green graph within it represents the prediction of the distribution model (2) we described earlier. In Figure 7b, for each benign domain rate (y) the x-axis indicates the attack rate, from which the domain will most likely be deleted from the cache. For example, for Instagram.com from Figure 7a which has a rate of 20 qps, a line is drawn to Figure 7b, and where it meets the model graph point [2000,20], so the rate of attack most likely needed to remove Instagram.com and all domains with lower rates from the cache is 2000. The purple triangles indicate the results of the experiment so that it is easy to see that our model's prediction (the green graph) matches the experiment's results.

Furthermore, the predictions of Equation (3) was also tested (Table 1) at five different attack rates, using the university distribution presented in Section 4. As an example, in the case of Instagram.com again, which is the 5th most frequently queried domain (rank 5), with a query rate of 20 qps, an attack of 2000 qps is required to remove this domain, and all domains with lower query rates, from the cache (according to Equation (2)). Using Equation (3), we predicted an average



(a) The distribution of the domains measured on the resolver of our university, sorted by the rate at which they were requested.

(b) The green line estimates for an attacker request rate what is the border rate from which all domains with a lower rate will be evicted with a high probability from the cache. The purple triangle points represent the results of our experiments on this distribution of domains, i.e, in what attack rate the specific domain was evicted from the cache.

Figure 7: The attacker request rate necessary to remove each benign domain with a high probability from the cache along with all benign domains with a lower rate; As an example, for the benign domain Instagram.com, which received 20 requests per second, a rate of 2000 attacker requests per second removed in high probability the domain and all domains with a lower rate (the area created under the graph) from the cache.

cache miss rate of 46.2%, and observed a cache miss rate of 48.6% during our experiment.

Attack Rate	Domain Rank with 100% cache miss	Overall experiment cache miss	Overall predicted cache miss (by Equation (3))
8000	1-n	100.0%	96.4%
2000	5-n	48.6%	46.2%
1500	10-n	41.4%	40.1%
800	20-n	22.2%	21.7%
600	30-n	12.5%	11.3%

Table 1: The overall average cache miss predicted by equation (3) compared to five experiments overall average cache miss results, with five different attack rates, on a BIND9 resolver with a 10MB cache size, using the domain distribution of our university.

5.5 Resolver Cache Miss and Throughput under CacheFlush attack

This section examines the impact of both CacheFlushAttack versions (CNAME and NS) on benign domains in a resolver under attack. Using different attack rates and cache sizes, we tested the latest version of BIND9 (9.18.21) and UNBOUND (1.19.0). We utilized two Resperf [40] tools in our cloud set-up environment: the first modeled the attacker and generated malicious domains at a fixed rate for each experiment, and the second ramped up the university benign domain requests rate until failure was encountered. For each combination of attacker request rate and cache size, we tested the impact of the following attacks:

1. NSCacheFlush
2. CNAMECacheFlush
3. Water-torture [42] - floods the resolver with pseudo-randomly generated nonexistent sub-domains.
4. NRDelegationAttack - floods with 50 different packets with the same 1500 names length referral response. (This is a complexity attack, to which the two resolver versions are patched, it generates a high CPU load but does not overload the cache. In total $50 \times 1520 = 76,000$ records are inserted into the cache in this attack.)
5. Existing domains - floods the resolver with queries for existing domains that return A records as an answer, similar to Water-torture [42] (point 3 above) except that the domains are stored in the benign cache.

Both Bind and Unbound suffer from a significant degradation in the benign domain request throughput measurements and increase in the average cache miss rate under CacheFlush, as shown in Figures (8,9).

NRDelegation has little impact on the throughput and on the cache miss rate on these patched versions of the resolver, because in the patch (that CacheFlush circumvents) it inserts

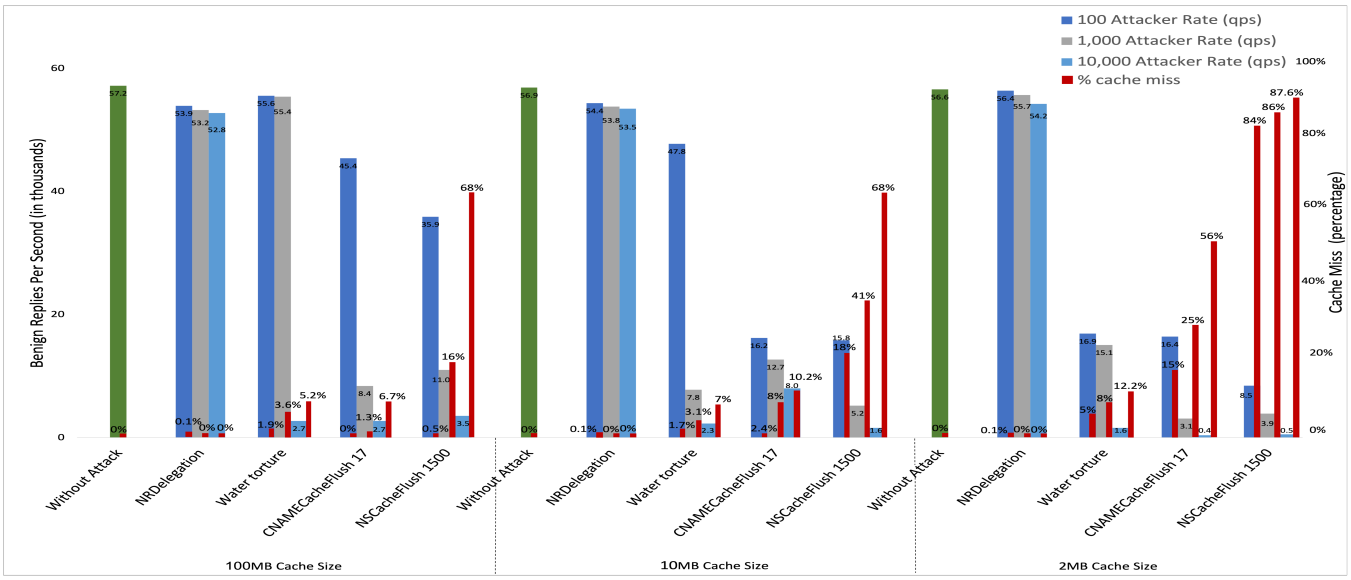


Figure 8: Cache miss and throughput for BIND9 resolver using two CacheFlush attack variants with a variety of attacker request rates and cache sizes, in comparison to NRDelegation on the fixed version of the resolver and water-torture attacks.

only a limited number of records into the benign cache. Water torture negatively impacts resolver throughput at high attack rates and has a low impact on the benign domain cache miss rate since the records are inserted into the negative cache. The Existing domains attack did not affect the resolvers because it cached only a single A record for each "malicious" query. Thus taking approximately 30 Bytes in the cache, whereas in the CacheFlushNS attack, each query consumes 100KB (3,333 times more), which means that in order to achieve the same cache depletion as in our attack, the attacker would have to issue 33.3M ($3,333 * 10,000$) queries, which is well beyond the capacity of the resolver. As a result, using the Exisiting Domains attack in the rate of our cacheFlush attack, there are no cache misses, and the throughput remained unchanged since both benign queries and existing domains were successfully resolved, hence we did not add this attack to our figures.

The attack does not affect High Rate Domains (HRD) since the attack does not fill up the cache before the HRD domain is queried again. At the same time the attack has a little impact on benign domains that are queried at a very low rate, even without our attack, since these domains are removed from the cache once it is full or their TTL has expired. Furthermore, we investigated the latency variation between our attacks and the average latency observed for all benign domains. On average, the latency during an attack is 15.6 times higher than the average latency of the benign domains without the attack, which was increased from 8.53 milliseconds to 133.1 milliseconds.

Testing a 2GB Cache Size: Since, as noted in Section 3, in practice medium size resolvers (e.g., University resolver) use a 2*GB* to 4*GB* cache we experimented with a 2*GB* cache as well. However, as predicted by equation (1), the larger the cache, the larger the attack rate required in order to flush it with in a given time interval. Therefore, the attack rate in this test is increased and had to be performed from two clients instead of one, as is done in the smaller cache sizes. Furthermore, the maximum benign rate tested was reduced from 1000 qps to 1 qps for this test. In testing BIND with the higher attack rate we encountered farther difficulties since as mentioned in Section 5.4, in high attack rates Bind's queries-queue purges many queries from the queue before being processed. This caused many benign domain queries to be removed from the queue and not queried by the resolver. In the 2*GB* experiment on Bind, most of the benign queries are deleted from the queue, and no response is received by the client, nor a query to the authoritative is observed. Therefore, we are unable to determine whether there is a cache miss or not. Hence, we performed the 2*GB* experiment only on Unbound.

Figure 9 illustrates the effect of our attack on the throughput and cache miss rate in Unbound with a cache size of 2GB based on the distribution of our university domains.

Figure 10 presents a similar experiment to Figure 5, but on 2*GB* cache and the attack rate is higher, while the benign domain rate is lower.

Equation (1) correctly predicted the cache miss rate for larger and smaller cache sizes once we filtered out malicious domains for which we did not receive a response to the clients, thus implying that these domains were not saved in the cache. Additionally, Figure 11 shows a similar experiment to Figure 6, for 2*GB* cache size equation (1) correctly predicted the results of cache misses for all attack and benign rates. By

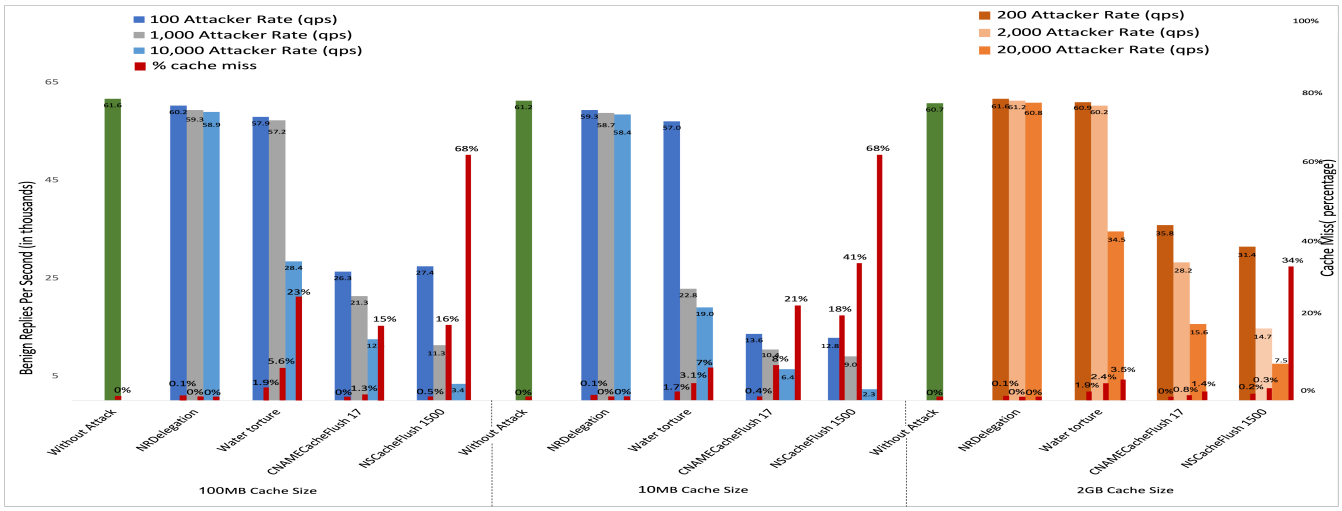


Figure 9: Cache miss and throughput for UNBOUND resolver using two CacheFlush attack variants with a variety of attacker request rates and cache sizes, in comparison to NRDelegation on the fixed version of the resolver and water-torture attacks.

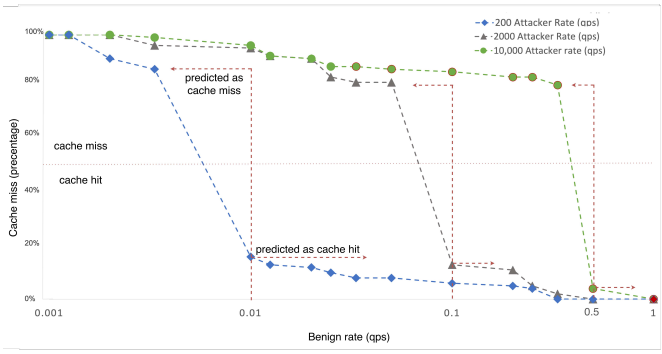


Figure 10: Cache miss percentage measured on different benign domain request rates (0.1 indicates one query every 10 seconds) with different NSCacheFlush attacker request rates on Unbound resolver with 2GB cache size.

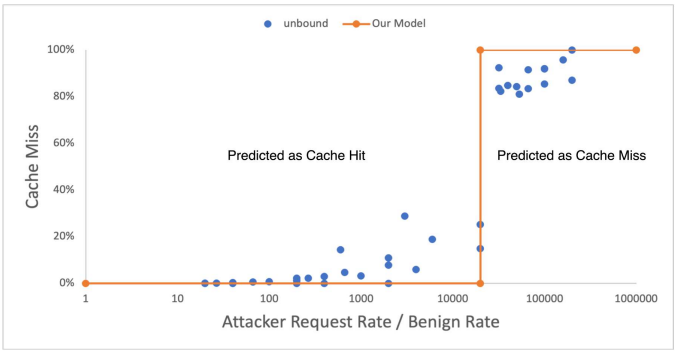


Figure 11: 2GB cache size UNBOUND resolver cache miss percentage measured on different benign domain request rates with different NSCacheFlush attacker request rates; According to our model, there is a high probability of cache misses when the $\frac{\text{attacker request rate}}{\text{benign rate}} > 20,000$, similar to the results obtained in the experiments. E.g., to flush the 2GB cache once per second, $20,000$ requests per second are required.

equation (1), for $2GB$ cache size, when $\frac{\text{attacker request rate}x100}{\text{benign rate}} > 2,000,000 \geq \frac{\text{attacker request rate}}{\text{benign rate}} > 20,000$ cache misses are almost surely to occur.

6 CacheFlush Mitigation

6.1 Bounding NS referral list

Although the resolver considers only the first p (e.g., 20) NS names from the referral response, the entire list is cached. It is noted in [10] that the top million domains have an average of 2.52 NS names in their corresponding RRs, with 99.5% fewer than 7. Since many root servers return 13 NS names in the RR, $p = 20$ was selected [36, 37] as a safe number that does not affect benign domain resolution.

As such, it is reasonable to limit the number of names stored in the cache to 20, even though this will not entirely eliminate the attack, it will result in a significant reduction in the impact of the attack, as shown in Figures (14,15). For a more complete mitigation/patch the processing complexity of the referral list (even if only 20) should be resolved. Here we only examine the impact of the limited mitigation of trimming the referral list to store at most 20 NS names.

6.2 Bounding the length of CNAME chains

One obvious solution is to bound the CNAME chain length, in order to eliminate entering multiple records to the cache. To determine what CNAME restriction is needed, we ran an experiment on the 5,000 most common domains (based on Cloudflare Radar [17]), using 8,000 resolvers in 230 countries and territories. During the period of 2022-07-10 to 2022-07-19, with 160 million queries were sent each day, using BGProtect monitoring [1]. As shown in Figure 12, 75.26% of the 5000 domains did not use CNAME at all. The maximum CNAME chain length is six with only one domain (0.02%), and only 3.64% of the domains had a CNAME chain longer than one. A recent querying of the single domain with a chain of length 6 discovered the chain is now five CNAME long.

Moreover, we also conducted a test using one machine in our Azure cloud environment located in the west-us region, on the million most popular domains in the world [17], and we checked how long their CNAME chains were. Figure 13 shows 98.7% of the domains do not have CNAME records, and only 0.0001% have a chain of length 7, while only 0.4% have a chain that exceeds one.

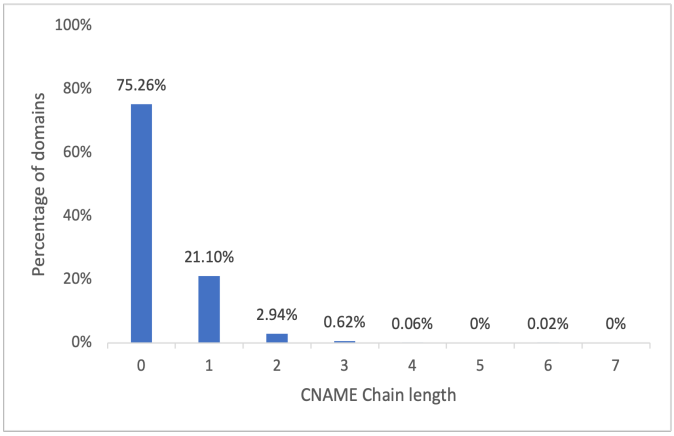


Figure 12: Percentage of domains with varying CNAME chain lengths among the 5,000 most popular domains, queried from 8,000 different resolvers spread over the world.

In Figures (14,15) we show the impact of limiting to 8 the CNAME chain as well of the CNAMECahceFlush with a length of 1,500 names to compare the effect of filling the cache to the complexity effect of our attack that we describe later in Section 7.1.

7 Additional Large DNS Messages Vulnerabilities

During the investigation of our CacheFlush attacks on the resolvers, we discovered additional implementation problems in the cache which we reported to the vendors under a responsible disclosure procedure. Some of these problems have been patched while others require further work.

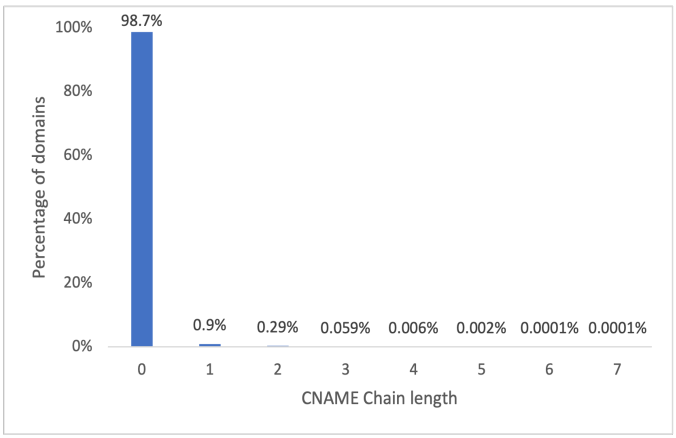


Figure 13: Percentage of domains with varying CNAME chain lengths among the million most popular domains, queried from one location.

7.1 CPU quadratic complexity when parsing large DNS messages

The CNAMECacheFlush Attack allows you to send a CNAME chain with 1,500 names, similar to NSCacheFlush, but unlike NSCacheFlush, only 17 records are saved in the benign cache and the cache miss percentage remains similar. Despite this, we detected differences in the throughput of the resolver when we sent these large messages and identified a vulnerability (CVE-2023-4408 [39]) in the validation check for large responses. In the event that a message arrives at the resolver, it performs validation checks to ensure that there are no conflicting information between the records (for example, the same domain should not point to two different names in the CNAME chain). Each domain in the response is checked to determine if it has already appeared in the answer so as to unify all of its answers into one data structure and eliminate duplicates. In this check, each domain is compared to all previous domains that have appeared in the CNAME chain before, causing a quadratic search for each message sent by the attacker, which results in approximately $\frac{17\times1500^2}{2} = 19million$ tests for each message sent by the attacker, which causes the machine to execute more than 1 billion clock instructions for each message sent by the attacker, Figure 16 present a test from the CVE disclosure discussion. As can be seen in Figures (14, 15) the resolver throughput dropped by average of 22.1% when CNAME chains of length 17 were compared to chains of length 1500, while it increased by average of 333.7% when chains of length 8, indicating that cache misses are the primary determinant of throughput.

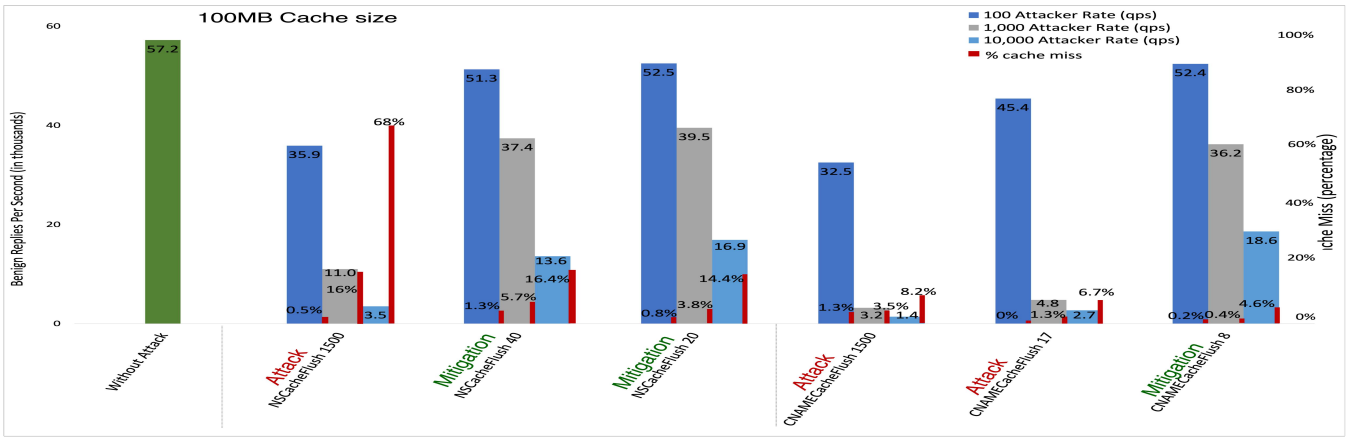


Figure 14: Cache miss and throughput mitigations compared to the CacheFlush attacks on BIND resolver with 100MB cache size; CNAMECacheFlush attacks with 8, 17 and 1500 chain length and NSCacheFlush using 1500, 20 and 40 RRs

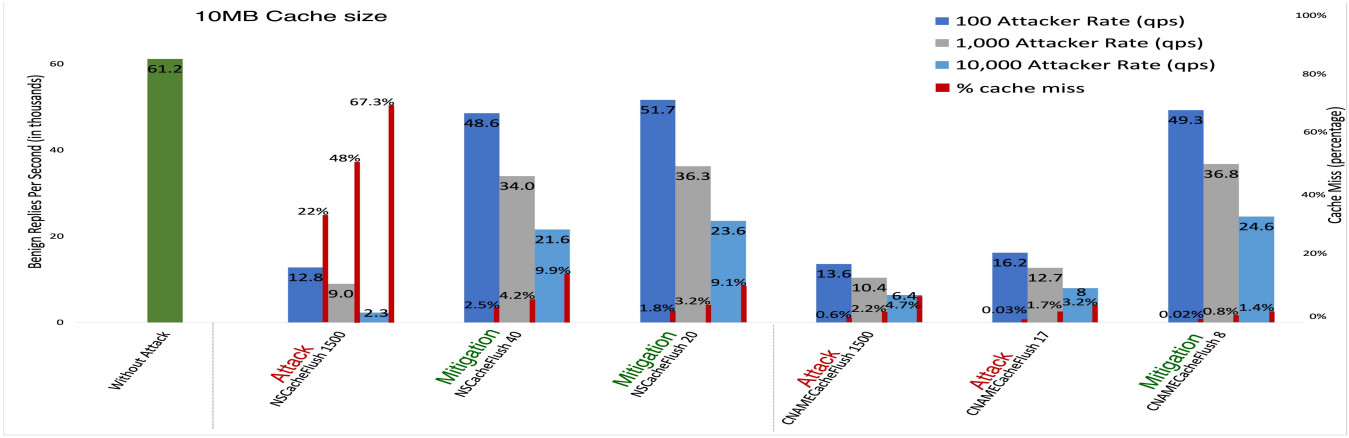


Figure 15: Cache miss and throughput mitigations compared to the CacheFlush attacks on UNBOUND resolver with 10MB cache size; CNAMECacheFlush attacks with 8, 17 and 1500 chain length and NSCacheFlush using 1500, 20 and 40 RRs

7.2 Overallocating cache memory until the resolver crashes

Another issue was discovered in the cache in that large DNS messages were not taken into account when the replacement of an old domain with a new one when the cache is full. While there is a limit to the size of the cache, it is possible to replace some small messages with larger ones in order to increase the cache until it exceeds the amount of RAM the machine has and the resolver crashes. This bug was fixed under a responsible disclosure procedure, and a CVE-2023-2828 [38] has been issued.

8 Related Work

In recent years, DNS amplification attacks have received a lot of attention. Moura et al. introduced an attack called TsuName [34], which created a loop of queries between two malicious authoritative servers, based on NS records. An adversary can register two or more domains, later reconfigure them to create a cyclic dependency, and then inject client traffic from a botnet.

Bushart et al. [15] demonstrated the amplification of packets by chaining CNAME records, causing a resolver to overload a target authoritative name server with valid requests.

Maury [31] presented another packet amplification attack that exploits the delegations of name servers in a referral response. In the attack named iDNS, the attacker's name server sends self-delegations back and forth to the attacker's name server, potentially reaching an infinite depth.

In contrast to all the above work, our research focuses on flushing the benign cache using a small number of packets, and none of the above works address benign cache flushing.

Luo, et al. [29] analyzed the prevalence and characteristics of NXDomain and water torture attacks. Our attack is not an NXDomain attack but involves flooding the benign cache

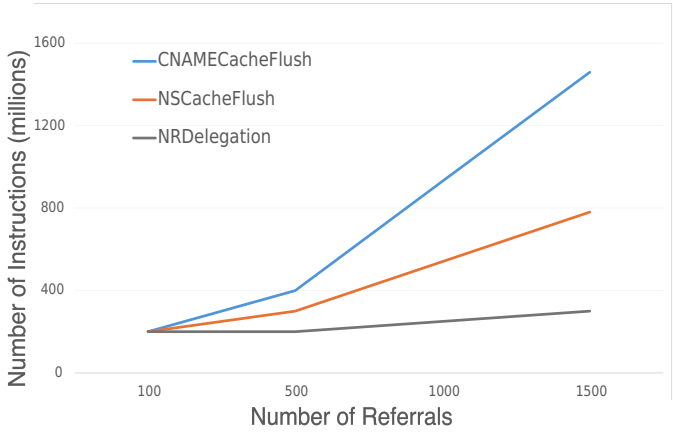


Figure 16: Instructions executed on the resolver processor per one malicious CNAMECacheFlush attack request (in millions), tested on BIND9 fixed NRDelegation attack version (9.19.13) compared to NRDelegation attack instructions.

with seemingly existent domains. NXDomain attacks flood the Ncache, and as we demonstrate, our attack results in a higher cache miss at the resolver and a greater reduction in throughput compared to water-torture [42].

NXNSAttack [10] is a packet amplification attack that exposed a vulnerability, causing a flood of queries between the recursive resolver and authoritative server, resulting in an overload on both and producing an amplified DDoS effect. However, the CacheFlush attack is not a packet amplification attack. While the NXNSAttack uses NXDomains that may be saved in the Ncache (and might flush it), our work floods the benign cache with seemingly existent domains.

NRDelegation [11] is a DNS complexity attack that uses a small number of different domains with a lengthy referral list to overload the CPU of a resolver and reduce its performance. The NRDelegation attack does not aim to attack the resolver's cache and only fills it with a few domains. In contrast, our attack emphasizes flooding the cache with different domains consistently to induce high cache miss rates and reduce the resolver's throughput. While a new complexity issue is also present in our paper, as a side problem we found, it does not play a significant role in harming the throughput as the CacheFlush, however new CVEs were issued.

In the discussion of DNS defenses during a DDoS attack, Moura et al. [33] demonstrate the impact of caching and long TTL. Since our attack removes domains from the cache before they reach the TTL by flooding the cache, the TTL does not play a significant role in our attack.

Liu et al. [28] build a DNS environment and modify the configurations of the resolver, authoritative, and zone files to discover vulnerabilities. This paper also raises the issue of the resolver continuing to query the authoritative for the CNAME chain that was already received in its entirety in the first message. However, they did not show that it can be exploited to flush the benign cache.

Kakarla et al. [25] create a verifier named GRoot that performs static analysis of DNS authoritative configuration files, enabling proactive and exhaustive checking for common DNS bugs. They develop a formal semantic model of DNS resolution and apply it to the configuration files from a campus network with over a hundred thousand records to reveal bugs. Our CacheFlush attacks do not contradict the RFC, and this paper cannot find it with their approach. The primary issue CacheFlush exploits is that the DNS RFC leaves many bounds open on the work after a DNS query, including the number of entries that can be entered into the benign cache.

9 Responsible Disclosure Procedure

We initiated a responsible disclosure procedure with several vendors after discovering the CacheFlush attack. In addition to collaborating with several parties one-on-one via encrypted email and GitLab channels, we also share our attack in a Mattermost channel with DNS-related vendors and third parties. Our cloud setup has been shared along with instructions on how to test the CacheFlush attack using the setup. Two CVEs [38, 39] have been already issued and additional ones are pending publication. The following are two quotations from one of the large parties involved in the disclosure: "It has been discovered that the effectiveness of the cache-cleaning algorithm used can be severely diminished by querying the resolver for specific RRsets" and "The rationale for this issue having a CVSS score of 7.5 (A:H) rather than 5.3 (A:L) is that the attacker can literally cause all legitimate traffic to time out when the attack is ongoing".

10 Conclusions

The vulnerability of DNS to DDoS attacks is well known and disturbing, as it is a key component of the welfare of the Internet. The Mirai attacks in 2016, which rendered services like Netflix unavailable for hours, demonstrated how vulnerable DNS can be and how much we rely on it. Since then, the research literature has made a lot of effort to make DNS more resilient and to identify and close vulnerabilities that still exist.

This paper sheds light on a key component of DNS and its resilience to attacks, which was overlooked until now: the cache. A key vulnerability stems from the fact that there is still no tight bound on the number of cache entries that can be added due to one DNS query. We show that surprisingly this can be up to 1520 records in NS and 20 in CNAME cacheFlush attacks, and it seems that much tighter bounds can be applied (8 and 20 respectively) as hinted in the mitigations suggested here.

While analyzing the Cache amplification factor of current DNS, we also discovered two additional complexity attacks.

This clearly demonstrates, once again, how important it is to systematically address the resource consumption in DNS servers.

Acknowledgements: We are grateful to the anonymous USENIX Security shepherd and referees for their invaluable, constructive comments and suggestions. We are thankful to Petr Špaček from ISC (BIND), Yorgos Thessalonikefs and Philip Homburg from NLnet Labs (Unbound), Vladimir Cunat from CZ.NIC (Knot), Otto Moerbeek from PowerDNS and the other members of the DNS-OARC Mattermost forum for their many comments, discussions, and ideas on an earlier version of the paper, and for their analysis and testing of the attack. We are thankful to Amit Klein for detailed insightful comments on an earlier version of the paper, and to Shahaf Pruss, and Shani Stajnrod for their indispensable help in testing and also commenting on earlier versions of the paper. We also thank BGProtect LTD, and Ariel Cohen from the School of Computer Science system, for providing us with helpful relevant data. This work was supported in part by a grant from the Blavatnik Interdisciplinary Cyber Research Center (ICRC), Tel Aviv University.

References

[1] BGProtect Company. https://bgprotect.com/.

[2] Hurwitz Zeta Function. https://en.wikipedia.org/wiki/Hurwitz_zeta_function.

[3] IBM Z/OS Documentation: Configuring Resolver (Optional). https://www.ibm.com/docs/en/zos/2.4.0?topic=caching-steps-configuring-resolver-optional.

[4] Knot Resolver Documentation: Daemon Bindings Cache. https://knot-resolver.readthedocs.io/en/stable/daemon-bindings-cache.html.

[5] Microsoft powershell: Set-dnsservercache. https://learn.microsoft.com/en-us/powershell/module/dnsserver/set-dnsservercache.

[6] MikroTik Wiki: Manual:IP/DNS. https://wiki.mikrotik.com/wiki/Manual:IP/DNS.

[7] Oracle Solaris Documentation: About the Name Service Switch and Cache Daemon. https://docs.oracle.com/cd/E19146-01/821-1834/abycw/index.html.

[8] Riemann Zeta Function. https://en.wikipedia.org/wiki/Riemann_zeta_function.

[9] Unbound Configuration Documentation. https://nlnetlabs.nl/documentation/unbound/unbound.conf/.

[10] AFEK, Y., BREMLER-BARR, A., AND SHAFIR, L. NXNSAttack: Recursive DNS inefficiencies and vulnerabilities. In *29th USENIX Security Symposium (USENIX Security 20)* (Aug. 2020), USENIX Association, pp. 631–648.

[11] AFEK, Y., BREMLER-BARR, A., AND STAJNROD, S. Nrdelegation-attack: Complexity ddos attack on DNS recursive resolvers. In *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023* (2023), J. A. Calandrino and C. Troncoso, Eds., USENIX Association, pp. 3187–3204.

[12] AHREFS. The 100+ Most Visited Websites in 2022. https://ahrefs.com/blog/most-visited-websites/, 2022.

[13] ALAYOFF, I., AND EINZIGER, G. Optimizing dns resolvers for high loads. In *2023 IFIP Networking Conference, IFIP Networking 2023* (Jan. 2023), Institute of Electrical and Electronics Engineers.

[14] ANTONAKAKIS, M., APRIL, T., BAILEY, M., BERNHARD, M., BURSZTEIN, E., COCHRAN, J., DURUMERIC, Z., HALDERMAN, J. A., INVERNIZZI, L., KALLITSIS, M., ET AL. Understanding the mirai botnet. In *26th USENIX Security Symposium* (2017), pp. 1093–1110.

[15] BUSHART, J., AND ROSSOW, C. Dns unchained: amplified application-layer dos attacks against dns authoritatives. In *Research in Attacks, Intrusions, and Defenses: 21st International Symposium, RAID 2018, Heraklion, Crete, Greece, September 10-12, 2018, Proceedings 21* (2018), Springer, pp. 139–160.

[16] CHEN, Y., ANTONAKAKIS, M., PERDISCI, R., NADJI, Y., DAGON, D., AND LEE, W. Dns noise: Measuring the pervasiveness of disposable domains in modern dns traffic. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks* (2014), IEEE, pp. 598–609.

[17] CLOUDFLARE. Cloudflare radar. https://radar.cloudflare.com/, 2023.

[18] CLOUDFLARE. Cloudflare plans. https://www.cloudflare.com/plans/, 2024.

[19] CLOUDNS. Premium dns pricing. https://www.cloudns.net/premium/, 2024.

[20] DATAFORSEO. Top 1000 websites by seo stats. https://dataforseo.com/free-seo-stats/top-1000-websites, 2023. Accessed on: June 13, 2024.

[21] ELZ, R., AND BUSH, R. Negative caching of dns queries (dns ncache). https://tools.ietf.org/html/rfc2308, 1998. RFC 2308.

[22] GODADDY. Domain transfer. https://www.godaddy.com/en-il/domains/domain-transfer, 2024.

[23] HOFFMAN, P., SULLIVAN, A., AND FUJIWARA, K. RFC 8499–DNS Terminology. https://tools.ietf.org/html/rfc8499, 2019.

[24] ISC. Bind: Internet systems consortium. https://www.isc.org/downloads/bind, May 2019.

[25] KAKARLA, S. K. R., BECKETT, R., ARZANI, B., MILLSTEIN, T., AND VARGHESE, G. Groot: Proactive verification of dns configurations. In *SIGCOMM'20* (2020), pp. 310–328.

[26] LABS, N. Unbound. https://nlnetlabs.nl/projects/unbound, 2019.

[27] LI, X., LIU, B., BAI, X., ZHANG, M., ZHANG, Q., LI, Z., DUAN, H., AND LI, Q. Ghost domain reloaded: Vulnerable links in domain name delegation and revocation. In *Proceedings of the 30th Annual Network and Distributed System Security Symposium (NDSS'23). https://doi.org/10.14722/ndss* (2023).

[28] LIU, S., DUAN, H., HEIMES, L., BEARZI, M., VIELI, J., BASIN, D., AND PERRIG, A. A formal framework for end-to-end dns resolution. In *Proceedings of the ACM SIGCOMM 2023 Conference* (2023), pp. 932–949.

[29] LUO, X., WANG, L., XU, Z., CHEN, K., YANG, J., AND TIAN, T. A large scale analysis of DNS water torture attack. In *Proceedings of the 2018 2nd International Conference on Computer Science and Artificial Intelligence* (2018), ACM, pp. 168–173.

[30] MANDIANT. Global dns hijacking campaign: Dns record manipulation at scale. https://www.mandiant.com/resources/blog/global-dns-hijacking-campaign-dns-record-manipulation-at-scale, 2024.

[31] MAURY, F. The idns attack. In *OARC 15* (2015).

[32] MERCER, W., AND RASCAGNERES, P. Dns espionage campaign targets middle east. https://blog.talosintelligence.com/2018/11/dnspionage-campaign-targets-middle-east.html, 2018.

[33] MOURA, G., HEIDEMANN, J., MÜLLER, M., DE O SCHMIDT, R., AND DAVIDS, M. When the dike breaks: Dissecting DNS defenses during DDoS. In *Proceedings of the Internet Measurement Conference 2018* (2018), ACM, pp. 8–21.

[34] MOURA, G. C., CASTRO, S., HEIDEMANN, J., AND HARDAKER, W. tsuname: exploiting misconfiguration and vulnerability to ddos dns. In *Proceedings of the 21st ACM Internet Measurement Conference* (2021), pp. 398–418.

[35] NAMECHEAP. Freedns. https://www.namecheap.com/domains/freedns/, 2024.

[36] NATIONAL VULNERABILITY DATABASE. CVE-2020-8616. https://nvd.nist.gov/vuln/detail/CVE-2020-8616, 2020.

[37] NATIONAL VULNERABILITY DATABASE. CVE-2022-2795, 2022.

[38] NATIONAL VULNERABILITY DATABASE. CVE-2023-2828, 2023.

[39] NATIONAL VULNERABILITY DATABASE. CVE-2023-4408, 2023.

[40] NOMINUM. resperf(1) - Linux man page. https://linux.die.net/man/1/dnsperf/, May. 2019.

[41] ORACLE. Oracle cloud infrastructure price list. https://www.oracle.com/cloud/price-list/#networking, 2024.

[42] SECURE64. Water torture, a slow drip dns ddos attack. https://secure64.com/water-torture-slow-drip-dns-ddos-attack/, Feb. 2014.

[43] SHOHAM DANINO. CacheFlushSimulator. https://github.com/shohamda/CacheFlushSimulator, 2024.

[44] WIKIPEDIA CONTRIBUTORS. Power law. https://en.wikipedia.org/wiki/Power_law, 2024.

A Appendix: The cost of attacking from the Cloud

The main expense of self-managed authoritative DNS in the cloud is the cost of outgoing traffic. As of June 2024, OCI [41], charges \$0.0085 for the transmission of 1 GB of outbound data from the VM. To flush a 2GB cache once per second, the largest cache tested in our experiments (rightmost column in Fig. 9), an attacker authoritative needs to send 20,000 query responses per second, which equals 1.3GB of data transfer ($20,000x65KB = 1.3GB$), which would cost \$0.01105 with OCI ($= 1.3GBx\$0.0085$). Thus it costs \$0.663 per minute for an attack that flushes a 2GB resolver cache so domains that are requested once per second or less have to be re-fetched on each query. In AWS, it costs \$0.09 to send 1GB to the Internet and \$0.01 for transfer within the same zone. Therefore, if the targeted resolver is maintained in AWS, the attacker can determine the zone based on the IP address and create the authoritative server in that zone area. In Azure, the price for internet-bound traffic is \$0.087, and \$0.02 for 1GB traffic within the same zone. In Google Cloud Platform (GCP), there is no cost for sending data within the same zone, \$0.01 for inter-zone transfers, \$0.085 for the first 10TB to the Internet, \$0.065 for 10-150TB, and \$0.045 for over 150TB. Consequently, if the target resolver is located in GCP, an attacker can exploit the attack without incurring costs. The secondary cost involves operating the machines for both the authoritative server and the client. For example running each machine in OCI costs \$0.00126 per minute. In comparison, the cost per minute for running a machine is \$0.00166 on AWS, \$0.0016 on Azure, and \$0.00158 on GCP. The most negligible cost is sending a DNS request from the attacker client. Each DNS query is approximately 100 Bytes. Therefore, sending 10,000 packets equates to 1MB of outgoing traffic. This cost is 1/1000th of the cost for the authoritative server, amounting to \$0.0000085. In total, the cost per minute to flush a 2GB cache every second for the authoritative server is $0.663 + 0.00126 = \$0.66426$. The cost for the client is $0.00126 + 0.0000085 = \$0.0012685$. Therefore, the total cost is \$0.6655285 per minute.