The Impact of Time on DNS Security

Aanchal Malhotra[†], Willem Toorop[‡], Benno Overeinder[‡], Ralph Dolmans[‡] and Sharon Goldberg[†]
Boston University[†], NLnet Labs, Amsterdam[‡]
aanchal4@bu.edu, {willem, benno, ralph}@nlnetlabs.nl, goldbe@cs.bu.edu

July 5, 2019

Abstract

Time is an important component of the Domain Name System (DNS) and the DNS Security Extensions (DNSSEC). DNS caches rely on an absolute notion of time (*e.g.,* "August 8, 2018 at 11:59pm") to determine how long DNS records can be cached (*i.e.,* their Time To Live (TTL)) and to determine the validity interval of DNSSEC signatures. This is especially interesting for two reasons. First, absolute time is set from external sources, and is thus vulnerable to a variety of network attacks that maliciously alter time. Meanwhile, relative time (*e.g.,* "2 hours from the time the DNS query was sent") can be set using sources internal to the operating system, and is thus not vulnerable to network attacks. Second, the DNS on-the-wire protocol only uses relative time; relative time is then translated into absolute time as a part of DNS caching, which introduces vulnerabilities.

We leverage these two observations to show how to pivot from network attacks on absolute time to attacks on DNS caching. Specifically, we present and discuss the implications of attacks that (1) expire the cache earlier than intended and (2) make the cached responses stick in the cache longer than intended. We use network measurements to identify a significant attack surface for these DNS cache attacks, focusing specifically on pivots from Network Time Protocol (NTP) attacks by both on-path and off-path attackers. We therefore recommend that DNS resolvers stop using absolute time for caching, and instead start using relative time. We have implemented our recommendations as part of the popular Unbound open source resolver, and our implementation will be part of Unbound's upcoming release.

1 Introduction

Time is a crucial building component of network protocols, providing basic correctness and functionality as well as claimed security guarantees. Recently, however, it has become clear that network protocols cannot take the correctness and security of time for granted. There have been a series of works describing errors, misconfigurations and malicious attacks that are possible when time is obtained from network timing protocols such as the Network Time Protocol (NTP) and Simple NTP (SNTP), [58], [75], [59], [60], [38]. It is known that millions of hosts [38], [58], [62], [65], [68] on the Internet implement and run network timing protocols. However, there is little work exploring how a reliance on network timing protocols affects the security of other network protocols that rely on time. (Thus far, there is only the work of Selvi on SNTP's impact on the security of HSTS [48].) In this work, we explicitly look into how a reliance on time affects the security of the Domain Name System (DNS) and DNSSEC. We show how attacks on time and NTP can be used as a pivot for attacks on DNS and DNSSEC.

DNS relies on caching to provide enhanced performance and improved reliability in the face of network failures. Our work considers how time can be exploited to attack the correctness and security of caching, and to expose the DNS to Denial of Service (DoS) and cache poisoning attacks. We also consider how the DNS security extensions (DNSSEC) can be attacked by exploiting time.

Time can be represented as absolute time (*e.g.,* "midnight August 7, 2018") or relative time (*e.g.,* "20 minutes from the moment you read this"). While the relative time is typically obtained from internal sources (*e.g.,* oscillators, CPU timers), absolute time is typically obtained from an external source (*e.g.,* manual settings, network timing protocols like NTP). As we have seen from recent work [75], [58], [59], [60], this reliance on external time sources leads to vulnerabilities.

DNS and Time The DNS protocol's reliance on time comes through its use of Time To Live (TTL) values for DNS records. Fortunately, the DNS on-the-wire protocol specification uses only relative time values for the TTL, and thus is not directly attackable. However, DNS caching implements TTL as an absolute time value, and is therefore attackable. This issue is not just the result of an implementation flaw; we point out that the earliest DNS RFCs (RFC 1035) implicitly assume that DNS caches rely on absolute time (Section 2.3). Moreover, the DNSSEC on-the-wire protocol also relies on absolute time to determine the validity interval for DNSSEC signatures; thus, we show DNSSEC is also vulnerable to attacks that pivot from time.

Attacks & Implications To support these claims, we show two concrete attacks that leverage absolute time to attack DNS resolver caches. First, we present a *cache expiration attack*: when time is shifted forwards, the DNS cached responses expire sooner than expected, effectively flushing the cache. Second, we present a *cache sticking attack*: when time is shifted backwards, the cached responses stick in the cache for longer than intended. We show how these attacks can be used to harm DNS performance (introducing latency into DNS responses) and DNS availability (increasing the risk of denial of service). We also discuss how they can be used to aid for fast-fluxing, cache poisoning and other well-known threats to the DNS. We also present two similar attacks on DNSSEC signature validation; by shifting the time on the validating resolver forwards or backwards, we can force a valid signature to be deemed invalid, causing DNSSEC denial-of-service and several other problems.

Measurements How big is the attack surface? While there are many ways to attack the absolute time, in this work we specifically focus on NTP as an attack vector. In Section 4, we use DNS and NTP measurements to see how many DNS resolvers might be vulnerable to our cache expiration and cache sticking attacks via pivots from NTP. We distinguish between two types of attacks on NTP. (1) In a *Man in the Middle (MiTM)* attack on NTP, the attacker holds a privileged position on the network between a victim system and its NTP timeserver. (2) In an off-path attack on NTP, the attacker does *not* sit between the victim system and it's NTP timeserver; the off-path attacker need only have the ability to spoof NTP packets (which are sent via UDP, the transport protocol that is used by NTP). Naturally, off-path attacks are "a scarier threat", because the attack can be any remote machine on the Internet.

We measure both open resolvers identified by the Open Resolver Project [61], and private (non-open) resolvers that are accessible via RIPE Atlas Probes [20]. We find a significant attack surface. At least 33% of our measured private DNS resolvers and 5% of our measured open resolvers are vulnerable to MiTM attacks on NTP. Meanwhile, 6% of those (33%) private DNS resolvers and 19% of those (5%) open resolvers are vulnerable to "even scarier" off-path attacks on NTP [60], many of which should have been patched years ago. (Indeed, one of the relevant patches was shipped in ntp-4.2.8p6, released on 19 January 2016.)

Recommendations & Implementation Our attacks and measurements indicate that DNS should not continue to rely on the correctness of absolute time (which is often derived from NTP). Instead, we suggest using the *raw time* or *adjusted raw time* (Section 2.1) as operating-system obtained sources of relative time for DNS caching. Meanwhile, for DNSSEC signature validation, resolvers are doomed to use the vulnerable absolute time. The only way to secure DNSSEC against timing-related attacks is to secure the aboslute time (*e.g.,* to secure NTP).

Upon our disclosure and recommendations, we worked with the open-source implementation of the Unbound resolver [25] to built a prototype implementation that uses raw time (rather than absolute time) for DNS caching. Unbound is committed to release this update in the coming version of Unbound. (See section 5.1 for details on implementation.) Also, after a preliminary disclosure of our work at IETF 100, the knot resolver [14] co-incidentally also changed how it used time for DNS caching. These changes were released in knot v1.5.1. In Section 5.1 we review their changes and discuss the extent to which they actually harden knot against our attacks. (As a spoiler, we find that these changes are not sufficient to completely stop our caching attacks on DNS.)

2 Time and the DNS ecosystem

In this section, we give background on why attacks on time can be used to harm the correctness of DNS. We start with an overview of absolute time and relative time (Section 2.1) and then discuss how the DNS specifications and implementations treat time (Section 2.3).

2.1 Absolute time vs Relative time

Protocols and applications can express time in several forms, depending on whether or not universal agreement is required about that point in time. This section focuses on the differences between absolute time and relative time.

Absolute Time Absolute time expresses an absolute point in time (*e.g.,* June 13, 2018 at 1:32:09pm).For instance,"Unix Time" is seconds since midnight January 1st, 1970, while"Universal Coordinated Time" (UTC) is an international time scale that forms the basis for the coordinated dissemination of standard frequencies and time signals [26]. Absolute time is often used to express the validity of objects with a limited lifetime that are shared over the network [1]. In order to validate absolute time value, a system needs access to a reasonably close reference time, for instance one based on the UTC.

How do systems get absolute time and why is it vulnerable? The absolute time on a system is known as *system time*. One way to update system time is to manually enter the date and/or time. One can also set the system time from the local machine by using the hardware time, which is maintained by a battery-powered clock that persists upon reboot. Alternatively, one can get the absolute time from the Internet, using a variety of timing protocols including the Network Time Protocol[2] (NTP) [63], Chrony [6], SNTP [64], OpenNTP [17] and others.

There are several problems with relying on system time. First, manual configurations can be subject to errors and misconfiguration. Also, for some machines, when moving between time zones, the system time must be corrected manually. Second, because accessing the hardware time requires an I/O operation which is resource intensive, many systems use hardware time *only* upon reboot, to initialize the system time; subsequent updates to the system time are made either manually or through NTP [15], [21], [23]. However, systems like microcontrollers that operate within embedded systems (*e.g.,* Raspberry Pi, Arduino, *etc.*) often lack internal hardware to keep track of time. When embedded systems require synchronization with the absolute time, they typically initialize their base time upon reboot by obtaining the current time from an external source (*e.g.,* a timeserver or an external clock), or by asking the user to manually enter the current time [22]. Third, relying on Internet timing protocols opens up the system time to attack. Recent papers show vulnerabilities in NTP [58, 60, 59] and SNTP [75] that allow attackers to maliciously alter system time - pushing system time into the past or even into the future. Moreover, many of these *time-shifting* attacks can be performed by *off-path attackers*, who do not occupy a privileged position on the network between the victim system and its time sources on the Internet. Researchers have also demonstrated off-path *denial of service attacks* on timing protocols that prevent systems from synchronizing their clocks. The bottom line is that obtaining system time from an external sources create dependencies that can be exploited.

Relative Time *Relative time* measures the time interval that has elapsed from some reference point (*e.g.,* "20 minutes from the time of your query"). Relative time is commonly used in network protocols, *e.g.,* to determine when a packet should be considered "dropped", or *e.g.,* to set *Time To Live (TTL)* values that govern the length of time for which an object is valid or usable. Relative time does *not* require access to the UTC time, or any other absolute time metric—only the rate of passage of this time across different systems is important.

How do systems get relative time? The relative time on an operating system can either be a *raw time* or an *adjusted raw time*. In both cases, relative time is monotonic. Importantly, the key property of each type of time source is not its current value, but rather the guarantee that the time source is monotonically increasing and thus useful for calculating the difference in time between two points [56].That said, there are several caveats to each type of time source.

Raw time At its most fundamental, a system has its own perception of time; its unmodified, *raw time*. This time is typically measured by counting cycles of an oscillator, but systems can also use process CPU time or thread CPU time (via timers from the CPU). The quality of the raw time is therefore dependent on the either the stability of the oscillator or of the CPU timer. Raw time is a purely subjective time—no general meaning can be attached to any specific value. One can only obtain the relative time by comparing two values. Because raw time is unaltered by any external or manual time source, the raw time is continuous and strictly monotonically increasing; it always increases,

[1] For instance, PKIX certificates [37] carry two time values expressing their earliest and latest validity.
[2] NTP disseminates UTC time.

never decreases, never makes unexpected jumps, and never skips. It is not subject to vulnerabilities or dependencies in external time sources. Importantly, even if highly accurate oscillators are used, raw time passes at a slightly different rate than system time. This difference is called clock drift. Raw time is not adjusted for the error introduced by clock drift. Thus, the accuracy of raw time is dependent on the clock drift, which further depends on factors including oscillator quality, system load, or ambient temperature, *etc.*.

Adjusted raw time When raw time is compared to an external reference time source in order to adjust for clock drift, then the result is *adjusted raw time*. This adjustment doesn't happen sporadically but rather, the rate of advance of time is slowed down or sped up slightly until it approaches that of the external reference time source. Therefore, adjusted raw time is still monotonic. Like raw time, adjusted raw time is subjective with no specific meaning attached to its values. But how does one obtain the adjusted raw time? One way to do this to access an external time source using one of the networking timing protocols we discussed earlier. The approach, however, is thus susceptible to some of the security risks that underly these network timing protocols.

2.2 The Domain Name System (DNS)

We briefly review the aspects of DNS, address resolution and caching that are relevant to the DNS cache attacks that we present in Section 3.

Domain Name System DNS is one of the Internet's most critical components. DNS is a distributed database that provides mappings between domain names and IP addresses and other resources in the form of *resource records* (RRs). A DNS name server (NS) stores the DNS RRs for a domain and responds with answers to queries against its database. An *authoritative NS* is responsible for mapping domain names for a specific domain (*e.g.,* `example.com`) to Internet resources, by serving the relevant DNS RRs (A, CNAME, PTR, *etc.*).

Caching To resolve an address [66], a *stub resolver* typically sends DNS query from an end-user system to a recursive resolver. If the requested record is not found in the recursive resolver's cache (*i.e.,* the query 'cannot be resolved locally') then the recursive resolver recursively queries the authoritative NSs in order to resolve the name. The recursive resolver caches the final response, and any additional RRs it learned as part of the resolution process, and sends the final response to the stub resolver. The stub resolver may also cache the final response.

2.3 Time in DNS

Time is an important component of DNS RRs. Each RR contains a time interval, known as the Time to Live (TTL), which is assigned by the administrator of the DNS zone in the zone file (*aka.,* "master file" in [66]). The DNS specifications require the TTL to be a relative time. However, as we shall now see, these TTLs are converted to absolute time when they are used for DNS caching. Later, in Section 3, we show that this conversion to absolute times creates network-time-related vulnerabilities.

According to RFC 1035 (one of the earliest DNS RFCs), TTL specifies the time interval (*i.e.,* the relative time) that an RR may be cached before the source of the information should be consulted again [67]. RFC 1035 recommends that all caching resolvers obey TTL values for caching, but RFC 2181 [42] says that caches can upper-bound the TTL of any RR, and treat any TTLs larger than the upper bound as if they were equal to the upper bound. In other words, the TTL specifies a maximum time to live, *not* a mandatory time to live. RFC 1035 also advises the zone file manager about suitable TTL values for different RRs in different situations.[3]

Upon receiving a DNS RR, a caching resolver is supposed to cache the RR for the time interval specified by its TTL. While the TTL is a *relative* value, the RFCs do *not* clearly specify how the cache should determine that the TTL has elapsed. However, there is some evidence that the RFCs *assume* that the cache converts the TTL to an absolute time.

[3]For instance, RFC 1035 says that TTL values on the order of days or weeks boost Internet performance and suggests a TTL value of zero for certain records such as SOA RR that should not be cached. One may also choose to have lower TTL values for extremely volatile data or if a change in the RR is anticipated to minimize inconsistency and then later revert back to the longer TTL.

The first bit of evidence comes when RFC 1035 says "When the RR has an absolute time, it is part of a cache", which suggests that the RFC assumes that the cache will implement the TTL as an absolute time.

The second bit of evidence comes from RFC 1035's recommendations for validating the freshness of an RR that is retrieved from the cache in order to answer a query. Specifically, RFC 1035 recommends timestamping the query for an RR using the current system time, and comparing that timestamp with the TTL of the RR in the cache. Comparing the system time (an absolute time) with the TTL time implicitly implies that the cache is storing the TTL as an absolute time.[4]

We checked popular caching resolver implementations (Unbound [25], Bind [5], PowerDNS [19], Dnsmasq [9], Knot (before v1.5.1) [14]) to see how their caches were implementing time. We found that these implementations all mark the end of validity of the cached object by translating the relative time values in the TTL into absolute time values by adding an offset equal to the TTL to the current system time.

DNS resolver implementations do not come with a predefined mechanism for getting absolute time. So the best that they can do is to rely on system time (which represents some form of absolute time) from underlying OS to get these absolute time value. For instance, the POSIX function to get the system time is `gettimeofday()`, which gives the number of seconds and microseconds since the epoch `1970-01-01 00:00:00 +0000 (UTC)`. It therefore follows that DNS resolver implementations take the correctness and security of system time for granted. This creates a security risk, because the system time is often set by network timing protocols (*e.g.,* NTP), which are vulnerable to attack (see Section 2.1.)

Cache security and correctness. Finally, we note that the risks due to the cache's reliance on absolute time (or system time) are not mentioned in the DNS RFCs. RFC 1035 only minimally discusses security risks to the cache. The closest that RFCs get to this is: (a) RFC 1035 suggests implementing the cache as separate data structure so that it can be easily discarded without disturbing zone data (especially since the cache is vulnerable to corruption when a system reboots, and because it can also become full of expired RRs) and (b) RFC 3833 [33], RFC 5452 [46] and RFC 7873 [47] discuss hardening caches to DNS spoofing and cache poisoning attacks.

3 Using Time to Attack the DNS Cache

We now investigate a largely overlooked and important threat in DNS: the impact of security vulnerabilities introduced because resolver's caches are dependent on absolute time, which is obtained from the system time, which is often obtained via network timing protocols (*e.g.,* NTP), which are vulnerable to attacks. In this section, we present two attacks that exploit vulnerabilities in networking timing protocols: a *Cache Expiration Attack* and a *Cache Sticking Attack*. We also discuss how each of these attacks can harm DNS and other protocols and applications that rely on DNS.

3.1 Cache Expiration Attack

If the system time is shifted forward on a caching DNS resolver, then the RRs in the cache would expire sooner than intended. This is tantamount to flushing attacks on DNS cache and may cause a denial of DNS services to a client.

This attack is possible when the system time is updated from an external source that is attackable, *e.g.,* NTP [58], [59], [60] and SNTP [75]. The attack follows because DNS cache implementations use the system time to determine TTLs. Meanwhile, the DNS protocol gives TTL as a relative time. If the cache had instead implemented TTLs as a relative time (*i.e.,* raw time or adjusted raw time, see Section 2.1) the security of system time would have no impact on the security of the cache.

We perform the attack in the laboratory setting. For this we set up our own validating, recursive, and caching DNS resolver Unbound v1.7.3 [25] on an Arch Linux version 2018.08.01 machine. (This was the latest version of Unbound and Arch Linux at the time of our test). Our resolver uses NTP to update the system time; indeed, NTP is the most

[4]Specifically, RFC 1035 says: The "timestamp indicat[es] the time the request [for an RR] began. The timestamp is used to decide whether RRs in the database can be used or are out of date. This timestamp uses the absolute time format previously discussed for RR storage in zones and caches. ... When the RR has an absolute time, it is part of a cache, and the TTL of the RR is compared against the timestamp for the start of the request."

common method of time synchronization on GNU/Linux systems and on Arch Linux systems [16]. We reboot the system just to allow for a clean attack demonstration; this same attack is possible even if the system is not rebooted. On system reboot, the resolver cache is empty. We then perform the following experiment:

1. We initialize the empty cache by inserting an `A` record as follows. Send an `A` record query to the recursive resolver for the domain `ndss-symposium.org` using the standard `dig` query for DNS lookup. (The `dig` query bypasses the stub resolver cache lookup on our test machines.) Since the recursive resolver has an empty cache, it performs a recursive lookup for the domain. It first queries the root TLD, then .org authoritative NS and then ndss-symposium.org authoritative NS to get the final A record as the answer. In our experiment, the recorded time to complete the entire recursive address resolution process was 267 ms and TTL was 3600 sec $\approx$ 1 hour (per the"Query time" and "TTL" field respectively in the `dig` response message).

2. Next, we confirm that the `A` record has been cached by the resolver. To do this, we perform another query, within the TTL window of the `A` record, for the same domain.The record time to answer the query was just 4 ms, implying that the record is served from the resolver cache.

3. Next, we attack by manually changing the system time on the resolver to be past the TTL window of the cached `A` record by 1 hour 5 minutes. (There are several techniques to perform these attacks from off-path and on-path described in [58], [60], [59], [75]. We did not reproduce those attacks in our lab.)

4. Next, we perform the same query to the resolver. We observe now the query took 94 ms because it did not have to do the full recursion from the root. The root and .org were still cached (because they have longer TTLs than 1 hour 5 minutes.) The resolver had to refetch the `A` record for only `ndss-symposium.org` which indicates that the `A` record was flushed from the cache.

3.2 Implications of the Cache Expiration Attack

We consider the implications of cache expiration attack in three categories: (a) impact on DNS, (b) making other attacks easier, and (c) economic repercussions for outsourced domains.

Impact on DNS. Caching is critical to the DNS. *All* recursive resolvers are caching, while many stub resolvers (depending on the underlying OS) support caching. (For instance, Windows [51], Linux dnsmasq [13] and Linux systemd [1] do support caching in stub resolvers, but the libc stub resolver does not.) Our attack harms caching performance and availability.

Performance A cache fetch from the resolver is much faster than a full recursive query to one (or many) external NS(es). This is also indicated in our attack demonstration above which took 4 ms to respond from the cache as opposed to 267 ms for full recursive response. Research suggests that resolvers typically have a 70-90% cache hit rate [49], [40], so by flushing the cache using our cache expiration attack, we can harm performance significantly. Each time this attack is launched, flushing the cache, it harms the performance for the first query made for a given RR. Nevertheless, we believe this performance hit is significant. As evidence of this, [24] observed that DNS resolver operators sometimes serve RRs from the cache for 2-3 days longer than is allowed by their TTLs. In other words, the the performance hit we have demonstrated here is important enough to DNS operators that they sometimes are willing to disregard the requirements of the DNS specifications.

Availability If the cached RRs expire too soon and the local caching resolver has to go out and query the name servers for every other query it receives, the reliability level of the DNS becomes the limiting factor in availability of service. RFC 1034 [66] says that even though the Internet is built for resiliency and redundancy, nameservers can be down and/or communication link to the root server broken due to network disruptions or other reasons. In such cases, the cache of recursive resolver decides the availability of resolving services.

This is a real threat, as we have seen many instance when DNS services go down but users are unaffected because of the presence of the cache. Consider, for instance large-scale DoS attacks like as DNS flooding, which are symmetrical DoS attacks that attempt to exhaust server-side assets (e.g., memory or CPU) with a flood of UDP requests, generated

by malicious scripts running on several compromised botnet machines. In one incident [11], some root DNS servers received high query rates that caused network connections to saturate, denying service to valid, normal queries. As another example, series of DDoS attacks on DNS provider Dyn in 2016 [4] affected many major services. (News headlines went so far as to say that it "broke the Internet" [7, 10, 27]). When Dyn's targeted, authoritative DNS servers became unavailable, the attack traveled across the world at TTL speed. That is, as soon as a recursive resolver attempted to refresh its cache and discovered that Dyn wasn't available, it had no supply of IP addresses to provide. Since most of the records in question relied on short TTLs, the impact was almost immediate. Meanwhile, longer TTLs in DNS cache buys time for remedial action to be taken against the adversary. However, with our cache expiration attack if the cached responses expire, then the service is no longer available.[5]

Other Attacks made easier We can use the cache expiration attack to make other DNS attacks easier to accomplish.

DNS cache poisoning made easier. With DNS cache poisoning, an attacker attempts to insert a fake DNS record into the cache. If the querying resolver accepts the fake record and saves it, the cache is poisoned and subsequent requests for that record are answered as this fake record within its TTL window. The attacker may poison the cache with fake A or AAAA, or CNAME, or NS RR that cause traffic redirects to a server controlled by attacker. Since almost all communications on the Internet require a DNS lookup, any application that is served from the DNS cache becomes vulnerable.

Importantly, cache poisoning attacks can be difficult because they require the victim resolver to send a query to an external NS. However, if the relevant RR is already present in the victim resolver's cache, then the query to the external NS will never be sent, and the cache poisoning attack will not succeed. (Indeed, RFC5452 explains how longer TTLs for cached responses make DNS cache poisoning harder.) Meanwhile, our cache expiration attack flushes RRs out of the cache, making it easier to launch a cache poisoning attack.

Fast fluxing made easier Fast Flux [12, 45] is a DNS technique used by botnet networks to hide various malicious activities, behind a dynamic network of compromised machines acting as proxies. The basic technique behind fast fluxing is to have multiple IP addresses associated with a single domain name (controlled by the attacker) which are swapped at high frequency, using short TTL values. However, fast fluxing is commonly thwarted by DNS resolvers that defensively reject RRs with very short TTLs. However, with our cache expiration attack, one can effectively shorten TTLs short, thus thwarting this defense mechanism and aiding fast fluxing. While our attack works only on individual caching resolvers (rather than authoritative NSes), it's important to remember that some resolvers serve as core Internet infrastructure that are used by a large number of users (*e.g.,* 8.8.8.8, 9.9.9.9, 1.1.1.1). Moreover, fast fluxing via cache expiration attack is more difficult to detect than very short TTLs.

3.3 Cache Sticking Attacks

If system time is shifted backwards on DNS resolver, the validity period of cached RRs may increases. This makes them stick in the cache for longer than their intended TTL. This attack is identical to the Cache Expiration Attack we just presented, except that now, the attacker maliciously shifts time backwards, rather than forwards. That said, the implications of this attack are different.

3.4 Implications of the Cache Sticking Attack

Cache Poisoning lasts longer If the attacker is able to poison the cache at a victim resolver (by inserting a fake RR into the cache), then by sticking the cache he can keep the cached poisoned for a long time. This approach circumvents many existing defenses that try to limit the impact of cache poisoning by limiting the TTL of records that live in the cache. For instance, by default the Unbound resolver will requery for cached responses *every* 24 hours to alleviate the damage if the cache is poisoned. (This way a poisoned record will necessarily be requeried in 24 hours, forcing

[5] As a solution, there is a new draft IETF specification [55] that proposes to serve expired data from the cache to maintain availability and draft. Also [71] suggests keeping longer TTL values to improve DNS service availability during prolonged outages.

the attacker to relaunch the attack every 24 hours.) Meanwhile, our cache sticking attack will prevent this requerying, since the victim resolver will not realize that 24 hours have passed.

Domain Propagation Delay The TTL of a RR determines the rate at which changes to a DNS RR propagate through the DNS ecosystem. This is why RFC 1035 suggests that when a change to an RR is anticipated, the domain owners should reduce the TTL values of that RR prior to the change. Meanwhile, our cache sticking attack thwarts this approach, by making even those RRs with a short TTL live longer in the cache. While this attack targets resolvers (rather than authoritative NSes), it can still have a big impact if those resolvers serve as infrastructure (*e.g.,* 8.8.8.8 or 9.9.9.9 or 1.1.1.1). We now discuss several ways that this cache sticking attack can harm DNS.

1) Failover. Content Delivery Networks (CDNs) often use short TTLs on DNS records to tackle failovers. Failover is defined to be the process of moving an service from one IP to another, in the event that access to that IP fails. This failover process should happen as quickly as possible. A low TTL for the `A` record bindings can reduce the failover latency due to DNS caching and update the system more quickly, making the failover services more effective [49], [76]. However, with our cache sticking attack, this approach of adding robustness to the system can be rendered ineffective by sticking old RRs in the cache and delaying the propagation of new RRs.

2) Load Balancing and Quality of Service (QoS): Various CDNs [2] [3] rely on DNS-based server selection to balance load. The idea is that the CDN's authoritative NSes decide which RRs to serve in response to a query (*e.g.,* for `www.example.com`) based on the location of querying resolvers. (For instance, a resolver in Italy may be directed to a webserver for `www.example.com` at IP 1.2.3.4 while a resolver in China might be directed to a webserver for `www.example.com` at IP 9.8.7.6.) By serving different RRs to different resolvers (that all make the same query), the CDN can balance load across multiple webservers. Since it is a dynamic server selection scheme that must adapt to dynamically changing loads, CDNs may want to frequently change the RRs they serve to a given resolver. This requires TTLs on RRs to very short (on the order of a few minutes [70]). However, our cache sticking attack sticks a user to a particular RR, which may be outdated or saturated with load. This affects the quality of service achieved by the load balancing scheme.

3) Suppose a domain name is sold or transferred, orr that one wants to change the DNS host for a domain. Then, the RRs related to this domain need to be changed. Short TTLs allow this change to quickly propagate through the DNS ecosystem. Meanwhile, with our cache sticking attack, an old RR may be stuck in the victims cache, directing the users to an old IP which may now be malicious or may have old data. (This idea is also similar to the IP-use-after-free attacks presented by [35].)

Negative Caching RFC 1034 [66] optionally provided a negative caching service to allow negative responses (NXDOMAIN) with TTLs to be distributed by NSes and cached by resolvers. RFC 2308 [30] makes negative caching -the storage of knowledge that something does not exist - mandatory for the caching resolvers. This is important, as it not only reduces the response time of negative responses, but also helps reduce DNS traffic. However, RFC 2308 is sensitive to risk of a DoS attack that propagates via a negative response (NXDOMAIN) that has a very high TTL; as a defense, the RFC suggests a sanity check to make sure that the TTL on negative responses is not too high. Meanwhile, our cache sticking attack would thwart any such defense, by sticking even those NXDOMAIN responses with short TTLs in the cache for a very long time.

Blacklisting DNS has become the de-facto standard for the creation and distribution of blacklists of IP addresses/domains associated with spam and other anti-social behavior on the Internet. RFC 5782 says that a blacklist of IPs that sends a spam should have short TTLs as these IPs tend to change frequently (might change every few minutes). However, with the cache sticking attack, one can effectively make TTLs longer, potentially blacklisting address long after they have recovered from a compromise by the spammer.

Ranking Internet domains based on DNS: Various approaches [41], [69] for ranking Internet domains obtain information on popularity of domains based on their popularity among DNS resolvers.DNS caching in resolvers is crucial for this approach. More precisely, it is important that the TTLs or the life of records for a particular domain in a DNS cache are stable across different resolvers. However, if the cache is stuck, the resolver will not go out and query after a specific interval (as expected) and this may generate false data. Note, however, for this attack to be effective, it requires attacking a large pool of DNS resolvers.

4 Measuring the attack surface

We use measurements to find the number of DNS resolver IPs in the wild that are vulnerable to our *cache sticking* and *cache expiration* attacks.

We perform our measurement study on two kinds of DNS resolvers (1) *Open resolvers* - ones that are publicly accessible, and willing to resolve recursive queries for *anyone* on the Internet (2) *Private resolvers* - ones that are meant to provide resolver service to hosts in their respective network only. We test private resolvers only in those networks that have RIPE Atlas [20] probes.[6]

4.1 Pivoting from NTP time-shifting attacks

Our attacks rely on shifting time on the resolvers. Resolvers that use NTP to update their system time are vulnerable to off- and on-path time-shifting attacks [58], [59] and [60]. Thus we measure the attack surface of our DNS cache sticking and DNS cache expiration attacks by assuming that the attacker preforms the attack via NTP.

MiTM attacks via NTP. All NTP clients are vulnerable to Man-in-the-Middle (MiTM) time-shifting attacks. This follows because NTP packets are never encrypted and are typically not authenticated by a message authentication code (MAC) [60]. Instead, an NTP client decides whether to accept packets from a timeserver by checking a nonce called the *origin timestamp* on the timeserver's response packet; the origin timestamp on the timeserver's response packet (*aka,* an NTP *mode 4* packet) is required to match the *transmit timestamp* on the client's query packet (*aka,* an NTP *mode 3* packet). The transmit timestamp on the client query can trivially be read by a MiTM, making NTP vulnerable to MiTM attackers that spoof bogus server response packets in order to shift time on the client [58, 60]. If a resolver is vulnerable to NTP MiTM attacks, the MiTM can execute our cache expiration and cache sticking attacks on DNS. As we will soon see, we found that 33% of measured private resolvers and 5.37% of measured open resolvers speak NTP and are thus vulnerable to these MiTM attacks. Below we discuss how we find and interpret these results.

Off-path attacks via NTP. Next we consider whether NTP can be used to launch our cache expiration and cache sticking attacks from off-path. One key reason that NTP is vulnerable to off-path attacks is because it operates over UDP. Nevertheless, clients that simply speak NTP are not necessarily vulnerable to off-path time-shifting attacks (although several generic off-path time-shifting attacks have been found and patched [58, 60]). This follows because an off-path attacker cannot read the transmit timestamp nonce on an NTP client query, and therefore cannot correctly inject a bogus server response with a correct origin timestamp. To attack from off-path, the attacker somehow has to learn the origin timestamp nonce.

That said, there is a known attack [60, Section V], called the "Leaky Origin Timestamp" that causes the client to reveal his nonce. This attack is possible on clients that answer unsolicited NTP *mode 6* control queries from arbitrary IPs with information that reveals the origin timestamp nonce. The specific NTP control queries that should be answered to launch this attack are either (`rv assocID org` or `rv assocID rec`). While patches for this attack exist (namely: stop answering NTP control queries from unknown IPs), many machines remain vulnerable even as of August 2018. Thus, we will decide that a DNS resolver is vulnerable to our DNS cache expiration and cache sticking attack if they respond to either one of the above NTP control queries. Surprisingly we find that 18.28% of the open resolvers that replied to NTP mode 3 queries and 6% of private resolvers that replied to NTP mode 3 queries leaked at least one of the `org` or `rec` nonces.

Finally, we note that there is another off-path time-shifting attack called the "Zero-Origin Timestamp Attack" [60, Section IV.A]. While this attack was patched over two years ago, not all machines have deployed the patch. This attack is especially powerful because it does not require the victim NTP client to answer unsolicited mode 6 control queries from arbitrary IPs. That said, to *measure* the attack surface for this zero-origin time attack, we need to use NTP mode 6 control queries [60]. When performing this measurement, we found that of the resolvers that answered unsolicited mode 6 control queries, we found that 70.17% of open resolvers and 69.14% of private resolvers were vulnerable to the zero-origin timestamp attack.

The bottom line is that many resolvers in our dataset are vulnerable to known attacks on NTP.

[6] Since these networks are maintained by individuals or enterprises for internal use, they are generally not accessible from outside the network.

4.2 Private resolver measurements.

To find out the number of DNS resolvers that are vulnerable to attacks that pivot from NTP, we use the RIPE Atlas probes. At the time of our experiment (July 29 2018 - August 2 2018), a total of 10,338 Atlas probes were active. Each probe gets its list of DNS resolvers by either manual configuration or by default by DHCP in their respective networks. We start by obtaining the list of IPs for DNS resolvers on each probe from the publicly available data from another - long running - DNS measurement study [8] on RIPE network. This study builds the resolver IP list by sending a DNS `TXT` query for `o-o.myaddr.l.google.com` from each probe using the locally configured resolvers. The IP addresses of the locally configured resolvers are part of the measurement results. By using the results of this query measurement, we make sure to select only working local resolvers.

In the following steps we discuss our measurement methodology and the challenges that we face.

1) MiTM attacks. First we send NTP mode 3 client queries to these privates resolvers. From the ones that respond to these queries with a mode 4 server response, we can determine the number of resolvers that use NTP to update their *system time*. Importantly, this only gives us a lower bound on the number of resolvers that are NTP clients, since a security-aware NTP client will only *send* NTP mode 3 client queries but not *respond* to them [58]. NTP clients that respond to NTP mode 3 queries are (often unwittingly) acting as NTP timeservers. We emphasize that there is a significant probability that there are other resolvers that are NTP clients but are configured to ignore mode 3 queries. While an MiTM can launch our DNS cache attacks to any resolver that speaks NTP (even those who are not acting as timeservers), we know of no other technique to determine if a remote machine uses NTP to set its system time. However even this measurement comes with a challenge.

Hurdle 1: Atlas probes allow NTP mode 3 queries *only* to public IP addresses.[7]. In order to filter out the resolvers that have private address, we discard the resolvers that have IPs in the private address space as described in RFC1918 [72]. We also discard well-known open DNS resolver addresses.

Stats. We then use the remaining IP addresses as target for mode 3 query. After filtering we obtained 4,703 unique resolvers with public IP address. We then send RFC5905-compliant mode 3 NTP queries to these resolvers from the Atlas probes that had them configured. (Note: NTP mode 3 queries do *not* modify the internal state of the queried systems.) Of the 4,703 queried resolvers, a staggering 1,535 (32.64%) resolvers answered with a mode 4 response to our mode 3 NTP queries.

Discussion The fact that 33% of the scanned private resolvers are acting as NTP timeservers (by responding to NTP mode 3 queries) looks pretty odd. But if we think about it, we expect a properly-configured DNS resolver to speak NTP, since accurate time is a good operational practice, and a necessity for DNSSEC validation too.

Why then, are so many resolvers acting as NTP timeservers? This is likely because most DNS operators do not spend time thinking through the intricacies of NTP. It turns out that, by default, the ntpd daemon makes every NTP-speaking system run as an NTP server [58]; this functionality needs to be turned off manually by the administrator. Unfortunately, most DNS resolver administrators are unlikely to be aware of this default. NTP-speaking systems that act as timeservers are also more vulnerable to time-shifting and DoS attacks, when compared to NTP-speaking systems that have turned off their timeserver functionality [58].[8] Also, most DNS operators are probably unaware (or have not thought deeply about the fact) that NTP is trivially vulnerable to time-shifting attacks by an MiTM, because NTP packets are not cryptographically authenticated.

1) Off-path attacks. Which of these 1,535 NTP-speaking DNS resolvers are vulnerable to off-path time-shifting attacks? To answer this question, we use NTP mode 6 control queries to see how many of those leak their `org` and `rec` nonces. We run a script that sends the following mode 3 queries in order.

```
ntpq -c "assocID"
ntpq -c "rv assocID org"
ntpq -c "rv assocID rec"
```

[7]Note that this restriction is only for NTP queries. One can send DNS queries to both public and private IP addresses.

[8]This follows because an NTP mode 4 server response reveals the IP address of the time source used by the responding timeserver, which is useful for launching off-path time-shifting attacks [58].

Resolver type	Total IPs	replied to `mode 3`	leaked `org` OR `rec`	leaked `org` 0
Open	342,131	18,385 (5.37%)	3,434 (18.68%)	2,416 (70.17%)
Private	4,703	1,535 (32.64%)	98 (6.39%)	65 (69.14%)

Table 1: Resolver IPs vulnerable to NTP time-shifting and our DNS cache expiration and cache sticking attacks. Column 3- All IPs vulnerable to MiTM, Column 4 - All IPs vulnerable to off-path attacks, column 5 - All IPs vulnerable to off-path Zero Origin Timestamp attack.

As described earlier, systems that reply to the above queries by revealing either `org` or `rec` nonce are vulnerable to the off-path "Leaky Origin Timestamp" time-shifting attack on NTP [60]. This brings us to our next challenge.

Hurdle 2: Atlas probes *do not* allow mode 6 NTP queries to be sent. This means that we can not query these resolvers from inside their network using Atlas probes. We found two ways that can, to some extent, solve this problem.

One solution is to querying these resolvers from outside the network in which they reside using NLnet Labs network. This approach is very limited since the resolvers are inside a private network, there is a low chance that they can be queried from outside. We queried the 1,535 resolvers. 68 resolvers leaked at least one of the `org` or `rec` nonces. Of these, 49 resolvers also leaked information indicating that they are vulnerable to the NTP Zero-Origin Timestamp attack.

An additional solution is to use the NLNOG RING [77] nodes. 377 (24.56%) resolvers have a common network with at least one NLNOG ring node. Of these 42 (11.15%) resolvers leaked at least `org` or `rec`, and 38 of the responding resolvers also revealed that they are vulnerable to the Zero-Origin Timestamp attack on NTP.[9]

Putting these two approaches together, we found a total of 98 resolvers that revealed at least one of the `org` or `rec` NTP nonces (and thus are vulnerable to the off-path"Leaky Origin Timestamp" attack). Of these 65 resolvers were also vulnerable to the off-path "Zero Origin Timestamp" attack on NTP. Both of these NTP attacks should have been patched years ago. Nevertheless, these resolvers remain vulnerable, which by extension makes them vulnerable to our DNS cache attacks from off-path.

4.3 Open resolver measurements.

To measure the number of DNS resolver IPs in the wild, we obtain data from the Open Resolver Project [61] that runs weekly scans of IPv4 address space to determine the IPs that respond to DNS queries. obtained a list of 23,137,317 IPs that responded to these queries during the week of April 15, 2018. We then start our scan on DATE July 28, 2018 as follows:

1) We send a DNS `A` query for `www.example.com` to this list of 23,137,317 IPs. 1,458,194 IPs still replied to the DNS `A` query. Of the ones that replied, we found that 342,131 (23.46%) are open resolvers (those that replied with an answer) and 1,116,063 (76.54%) were *non-answering*, returning either REFUSED or some other DNS error code, indicating that they resolvers, but not open resolvers.[10]

2) Next, we send NTP mode 3 query to 342,131 answering open resolver IPs. Of these, 18,385 (5.37%) replied back with a mode 4 response packet. And 87,141 (7.80%) of 1,116,063 non-answering IPs replied back with a mode 4 response. *All* of these resolvers are vulnerable to MiTM NTP attacks and hence our DNS cache expiration and cache sticking attacks by an MiTM.

3) We then send NTP mode 6 control queries to 18,385 answering open resolver IPs and found that 3,451 (18.77%) IPs leaked at least one of the `org` or `rec` nonces. Moreover, a total of 416 (70.17%) resolvers indicated that they are vulnerable to the NTP "Zero-Origin Timestamp" attack.

Discussion 5.37% of scanned open resolvers answered NTP mode 3 queries. We don't expect open resolvers (with the exception of infrastructure open resolvers like 8.8.8.8 or 1.1.1.1) to be properly configured; the fact that they are open

[9]NLNOG RING nodes do not have inbuilt support for mode 6 NTP queries. However, NLNOG RING allows for customized queries, so we copied NTP mode 6 queries to these nodes.

[10]These non-answering resolvers are properly configured private resolvers with access control lists (ACLs). They may have been open resolvers at those IPs when openresolver project collected the data, but became non-open resolvers at the time of our scan.

resolvers is sufficient evidence of misconfiguration. Why are open resolvers (5%) less vulnerable than private resolvers (33%) to NTP MiTM attacks? Again this might look odd, but this may be because nobody cares to configure NTP on open resolvers in the first place, which could be viewed as a bad DNS operational practice. Also, a staggering 18.68% open resolvers acting as NTP timeservers (as compared to only 6% private resolvers) were found to be vulnerable to off-path time-shifting attacks. This again supports our argument of misconfiguration, except that now it is a bad configuration for NTP.

4.4 Takeaways and Ethical Measurements

We observe that more private resolvers (as compared to open resolvers) have NTP configured (33%) (good for DNS(SEC) operational practice), but fewer of them are vulnerable to off-path time-shifting attack (6%) (good NTP configuration). Fewer open resolvers are NTP configured (5%) (bad DNS(SEC) operational practice), but more of them are vulnerable to off-path time-shifting attacks (19%) (bad NTP configuration). In other words, if the DNS resolver is sloppily configured, then NTP is sloppily configured as well. Therefore we ask: Is the risk of sloppily configuring NTP bigger if one configures NTP at all? Our results suggest that this is indeed the case.

Therefore, DNS resolver operators that decide to turn on NTP should also be careful with their NTP configurations. DNS resolvers should not be operating at NTP timeservers, because this increases the surface for NTP attacks [58]. DNS resolvers should also regularly update their NTP software, to avoid situations where NTP vulnerabilities remain exploitable two years after they should have been patched.

Ethical measurements. We acknowledge that a far more 'informative' measurement would have been to directly measure our DNS cache attacks in the wild by launching NTP time-shifting attacks in the wild, and then checking if the DNS cache has been flushed or stuck. Naturally, however, we did not do this, because it would be unethical to attack live resolvers. This is especially important since our DNS cache attacks indiscriminately flush or stick *all RRs* in the cache, so we could not have a done a very targeted "test attack" on our own RRs that we inject into the cache of the measured resolver (as was done *e.g.,* in [52]). Instead, we focused on using NTP side channels for measurements. None of the side channels modify the internal state of the NTP daemon (see also [58, 60]), and thus also do not modify the state of the DNS cache.

5 Recommendations

We believe that setting TTL as *relative time* value is a feature of the DNS protocol. To deal with the attacks described in Section 3, there is no fundamental change required to the DNS protocol. The only thing that needs to be dealt with is the way *relative time* values (TTL) are implemented in DNS cache. Since TTL represents a duration of time, we just need the value that gives the difference in time between two points.

We recommend to get this value from a monotonic source of time that is not subject to a) manual changes and b) adjustments by vulnerable network timing protocols. Instead, one can use raw time or adjusted raw time,[11] as discussed in Section 2, depending on their availability on different OS. If both types are available, the choice of time value to be used is application-specific. For applications that can tolerate a certain amount of clock drift or need to keep track of extremely small intervals of time (say on the order of few seconds), then raw time should be used. However, if that is an issue, then one has no choice but to fall back to adjusted raw time.

5.1 Implementing our Recommendations

Unbound Unbound is a validating, recursive, and caching DNS resolver product from NLnet Labs [25]. It is distributed free of charge in open source form under the BSD license.

[11] Note that adjusted raw time is subject to adjustments by timing protocols. However, 1) it can not be set manually, 2) it does not make large jumps, and 3) the rate of passage of time is just slowed down or sped up which means that it will take way longer to shift time. Adjusted raw time is better than system time but worse than raw time; see Section 2.

Unbound's internal operation is organized around asynchronous event notification. Different serve-threads wait for events to happen (*i.e.,* network sockets to become readable, writable, closed or to fail) to take appropriate action. Events are registered with a so called "event-base" (or loop) together with a timeout value. Fired timeouts are also events upon which (different) appropriate actions are taken (*i.e.,* retry scheduling of queries at different authoritatives *etc.*).

Unbound serve-threads read and record system time, every time events become available (or timed out). (This is similar to the recommendation in RFC1035 to timestamp every request/query for an RR, that we discussed in Section 2.3.) The timestamp is subsequently used with Unbound's operations dealing with the fired events: to set expiration time for newly cached RRsets, to check the validity of already cached RRsets, to limit the rate at which requests are send to authoritative servers, and to check validity of DNSSEC signatures. The function which is used to read and record system time is `gettimeofday()`.

We have a preliminary implementation of our recommendations for Unbound, and the implementation is publically available on Github. The implementation is currently performed on the "monotonic" branch of the repository here: `https://github.com/ralphdolmans/unbound/tree/monotonic`. At the time of writing, Unbound has been altered as follows. Instead of one, two timestamps are recorded when events are available, one absolute time value from `gettimeofday()` which is used for DNSSEC validation only, and one relative time value from `clock_gettime()` — currently with the `clock_id` parameter set to `CLOCK_MONOTONIC_RAW` (the unadjusted raw time on Linux systems) — which is used for all other timing related functions, such as dealing with RRset's TTL values and rate-limiting requests to authoritative servers. Several issues still remain to be resolved:

1. Those tests — that are part of the Unbound source tree — which are testing timing behavior of Unbound (*i.e.,* cache expiration *etc.*) are failing, because the scripts for those tests are based on absolute time values.

2. The current implementation uses the Linux specific `CLOCK_MONOTONIC_RAW` clock which is an extension on the POSIX standard [18], which defines `CLOCK_MONOTONIC` only and leaves whether or not the value is adjusted for drift, undefined. In general, portability of the implementation has to be evaluated.

3. Furthermore, Unbound has a built-in event-base system based on select. The registration and handling of timeout values with this event-base are still based on absolute time values from `gettimeofday()`. This has to be rewritten to use relative time values too.

NLnet Labs has committed to incorporate our changes into Unbound's main development branch once all issues have been resolved and the implementation has proven to be stable and not negatively affecting current deployments.

Knot-resolver After a preliminary disclosure of our work at IETF 100, the Knot resolver [14] co-incidentally also changed how it used time for DNS caching. This was released as Knot version 1.5.1. Because Knot's cache is persistent, it needs a defined epoch to be meaningful between restarts and reboots. The cache still has expiration values in absolute time values, but timing discontinuities are detected by comparing progression of system time with monotonic time. In default configuration, the Knot clears the cache if time jumps more than 10 minutes into the past. This has two implications.

1) This opens the DNS cache to an alternate cache expiration/flushing attack. The attacker needs to shift time backwards by *only* 10 minutes and the cache will be flushed, which is equivalent to our cache expiration attack in which the attacker has to shift the time forwards to pass the TTL window of the cached response. In fact, this new attack could be easier, since it is independent of the TTL of a particular RR.

2) Cache sticking attacks, on the other hand, would now require more persistent work by the attacker. Remember, for sticking the RRs in the cache the attacker has to shift time backwards within the TTL window of the RR to stick the RR in the cache (Section 3.3). Meanwhile, knot has now imposed a restriction of 10 minutes. The attacker can not shift time backwards by more than 10 minutes; if she did, the cache will be flushed instead of getting stuck. As result, performing a cache sticking attack is harder, because the attacker has to keep attacking at least every 10 minutes for as long as he wants to stick the RRs in the cache.

We conjecture that the Knot could not fix this problem entirely because Knot uses a persistent cache. It is not possible to store relative time values (raw time or adjusted raw time) in a persistent cache.

6 DNSSEC and System Time

DNSSEC described in RFCs [31] [73] and [32] is a set of protocols that adds a layer of trust by providing *data origin authentication* and *data integrity* to DNS lookups and exchanges. For authentication, DNSSEC associates cryptographic digital signatures, called the Resource Record Signature (RRSIG) records, generated using the zone's private key with the DNS Resource Record Set (RRSet)[12]. By checking its associated RRSIG, one can verify that a requested DNS RRset comes from its authoritative NS and wasn't altered en-route. A *security-aware* resolver understands DNSSEC and is capable of using DNSEC RR types to provide DNSSEC services. When a security-aware resolver learns the zone's public key, it can validate the RRSIG as described in [54].

RRSIG RRs have defined the *inception time* and *expiration time* to establish a validity period for the signatures and the RRsets covered by the signature. These inception and expiration fields are specified as absolute time values. This is *unlike* DNS TTLs, that are expressed as relative time values. Thus, unlike the DNS on-the-wire protocol, the DNSSEC on-the-wire protocol uses the notion of absolute time.

How do implementations deal with timestamps? In a typical software implementation (Unbound, Bind, PowerDNS, DNSMasq, *etc.*), the two absolute time values on RRSIG RRs are compared against the current system time. The current system time of the resolver MUST be between these two time values; otherwise the response is discarded either as an expired or not-yet-valid record.

DNSSEC RRs expired/not-yet-valid attacks If the system time of the security-aware validating resolver is shifted forward such that it is past the expiration time on RRSIG, then we can effectively make RRSIG RRs expire. On the other side, if the time is shifted backwards such that it is before the inception time on the RRSIG, then we can make them "not-yet-valid". In such a case, most resolver implementations (including but not limited to Unbound) shows the SERVFAIL status to the client and cache this failed response for 60 seconds after five retries with the same NS. This DNSSEC DoS is tantamount to DNS DoS attack. (Note, there is no fallback to DNS in case of too many SERVFAILs.)

So how can our attacks make DNSSEC DoS (aka DNS DoS) easier when there are multiple NSes? First, we note that the Unbound resolver [39] (and other resolvers) keep some statistics about NSes, and will stop querying an NS that causes too many SERVFAILs (or other failures) and start querying other NSes instead. Thus, a traditional DNSSEC DoS works only if *all* relevant NSes are causing SERVFAILs (or other failures). Meanwhile, our attacks directly DoS DNSSEC at the resolver by shifting time, without controlling *any* NSes, lowering the bar for the attack. Our attacks would invalidate all signed data from all correctly configured honest NSes, and leave the resolver with no NSes to fall back to.

6.1 Recommendations

The use of absolute time is inherent to security protocols. This raises the question: Why don't we use relative time values to define the validity of cryptographic objects just like we do for non-cryptographic records? There are two reasons we think that absolute time is needed for DNSSEC.

Argument 1 The authority determining and setting the validity period on the object can be different from the one delivering the object. For example, setting the TTL value on DNS records is an operational matter and is thus left to the operators of the DNS zone. Zone operators can change the TTL values on non-signed or non cryptographic records even when they don't own it. The content of the cryptographically signed records (RRSIG RRs) are, however, determined by the signer of the records. When the signer is not also the zone operator, signer has no way to determine when the records will be queried for, and thus has to depend on cryptographically signed absolute time values to limit the validity of the record stored in the zone[13].

Argument 2 In a replay attack, an attacker records the data and maliciously resends it later. However if the data has absolute time values on it, that limits its lifetime. This is not possible with relative time values.

[12] RRset is a group of records with same label, class and type, but with different data. Typically RRsets are signed as opposed to signing individual RRs.

[13] Note however that DNSSEC signatures do contain the original TTL of an RRset, restricting the maximum TTL value with which the operator may deliver the RRs.

Thus, it becomes imperative for the validating resolver to use system time to validate the absolute time values on RRSIG RRs. The only way to ensure the security of such a system is to secure the way system time is updated. On the bright side, there is an ongoing effort at Internet Engineering Task Force (IETF) community to secure NTP. *e.g.,* [43] is a proposal to secure NTP's client/server mode using Public Key Infrastructure and another Internet-draft [57] that deprecates MD5-based symmetric key authentication, which is considered to be too weak, and recommends the use of secure AES-CMAC as a replacement. DNSSEC resolver operators should be amoung the first to adopt these NTP security updates.

7 Related Work

Attacks Several works study DNS spoofing and cache poisoning attacks and propose potential solutions [50, 44, 53, 29, 74, 79]. These attacks focus on *how* the DNS protocol can be exploited to poison DNS resolver caches. By contrast our attacks show how to pivot from vulnerabilities in absolute time (*e.g.,* in NTP), to attacks that flush/stick the DNS cache. We also discuss how our DNS cache attacks can lower the bar for classic cache poisoning attacks by a) making them easier to launch, b) making them stick in the cache for longer. Finally, we note that existing work tends to propose security fixes to DNS on-wire protocol, while our recommendations focus on the DNS resolver cache implementations.

DNS measurement Many previous studies measure DNS resolvers. For instance, studies have analyzed resolver performance [36, 49], the effectiveness of resolver caching [49, 34], or the classification of open DNS resolvers and their non-legitimate responses. [28] measures local resolvers from 50 commercial ISPs for responsiveness and correctness and compares them with third-party popular open DNS resolvers. [78] measure how current caching resolver implementations distribute queries among a set of authoritative NSs. Our measurements are different because we focus on NTP-related vulnerabilities at DNS resolvers.

NTP measurement. In [58], [59], [60], the authors scan the IPv4 address space to find systems that are vulnerable to NTP attacks. While our attack surface measurement uses similar NTP queries (as in [60]) our measurements are specifically focused on open and private DNS resolvers.

8 Conclusion

We showed that DNS resolvers' dependence on absolute time makes them vulnerable to DNS caching attacks and DNS(SEC) DoS attacks. We studied the attack surface using network measurements, focusing specifically on the Network Time Protocol (NTP) as a vector for remotely attacking time. Our network measurements indicate that the attack surface is large . From total 342,131 measured open resolvers, we identified 18,385 open DNS resolvers that use NTP to set the absolute time. From total 4703 measured private resolvers, we find 1,535 private DNS resolvers that use NTP to set the absolute time. All these NTP-speaking resolvers are vulnerable to attacks by a MiTM that pivot from NTP time-shifting into DNS cache sticking / cache expiration attacks and DNSSEC downgrade attacks. In fact, we identified 3,451 out of 18,385 measured open DNS resolvers and 98 out of 1,535 measured private DNS resolvers that are vulnerable to NTP time-shifting attacks that can be launched from off-path, by any remote machine on the Internet with the ability to spoof UDP packets. While these off-path NTP attacks should have been patched over two years ago, many resolvers in the wild remain vulnerable.

Our attacks indicate that DNS caches should lessen their reliance on absolute time. Indeed, the DNS on-wire protocol does not actually need absolute time; instead, DNS packets only use relative time. We therefore recommend that DNS caching resolvers use relative time rather than absolute time. Specifically, we suggest using the operating system's raw time, which is monotonically increasing and not subject to adjustment by external sources (and thus immune to the network attacks on time explored here). We are currently working with Unbound resolver to implement and roll out our recommendations; our implementation is publicly available on Github.

References

[1] https://www.freedesktop.org/software/systemd/man/systemd-resolved.service.html(Accessed: July 2018).

[2] https://www.cloudflare.com/load-balancing/(Accessed: July 2018).

[3] https://www.akamai.com/uk/en/resources/server-load-balancing.jsp(Accessed: July 2018).

[4] "2016 dyn cyberattack," https://en.wikipedia.org/wiki/2016_Dyn_cyberattack(Accessed: July 2018).

[5] "Bind," https://www.isc.org/downloads/bind/(Accessed: July 2018).

[6] "chrony," https://chrony.tuxfamily.org/(Accessed: July 2018).

[7] "The day ddos attacks broke the internet," https://www.tektonikamag.com/index.php/2016/10/28/the-day-ddos-attacks-broke-the-internet/(Accessed: July 2018).

[8] "Dns measurement," https://atlas.ripe.net/measurements/8310237/(Accessed: July 2018).

[9] "Dnsmasq," http://www.thekelleys.org.uk/dnsmasq/doc.html(Accessed: July 2018).

[10] "The dyn ddos attack that broke the internet: Here's what happened," https://www.siteuptime.com/blog/2016/12/05/the-dyn-ddos-attack-2016/(Accessed: July 2018).

[11] "Events of 2015-11-30," http://www.root-servers.org/news/events-of-20151130.txt(Accessed: July 2018).

[12] "The honeynet project, know your enemy: fast-flux service networks," http://www.honeynet.org/book/export/html/130(Accessed: July 2018).

[13] "How to dnsmasq," https://wiki.debian.org/HowTo/dnsmasq#Local_Caching(Accessed: July 2018).

[14] "Knot resolver," https://www.knot-resolver.cz/(Accessed: July 2018).

[15] "Linux system administrators guide," https://www.tldp.org/LDP/sag/html/hw-sw-clocks.html(Accessed: July 2018).

[16] "Network time protocol daemon," https://wiki.archlinux.org/index.php/Network_Time_Protocol_daemon(Accessed: July 2018).

[17] "Openntpd," http://www.openntpd.org/(Accessed: July 2018).

[18] "Posix clock_gettime()," http://pubs.opengroup.org/onlinepubs/9699919799/functions/clock_gettime.html(Accessed: July 2018).

[19] "Powerdns," https://www.powerdns.com/(Accessed: July 2018).

[20] "Ripe ncc," https://atlas.ripe.net/(Accessed: July 2018).

[21] "System time," https://wiki.gentoo.org/wiki/System_time(Accessed: July 2018).

[22] "System time," https://en.wikipedia.org/wiki/System_time(Accessed: July 2018).

[23] "Time," https://wiki.archlinux.org/index.php/time(Accessed: July 2018).

[24] "TTL override by ISPs – is there a performance cost?" https://dyn.com/blog/ttl-override-by-isps-is-there-a-performance-cost/(Accessed: July 2018).

[25] "Unbound," https://nlnetlabs.nl/projects/unbound/about/(Accessed: July 2018).

[26] "What time is it?" https://www.bipm.org/en/bipm-services/timescales/time-server.html(Accessed: July 2018).

[27] "What we know about friday's cyber attack that broke the internet," https://www.inc.com/joseph-steinberg/what-we-know-about-fridays-cyber-attack-that-broke-the-internet.html(Accessed: July 2018).

[28] B. Ager, W. Mühlbauer, G. Smaragdakis, and S. Uhlig, "Comparing DNS resolvers in the wild," in *Proceedings of the 10th ACM SIGCOMM Internet Measurement Conference, IMC 2010, Melbourne, Australia - November 1-3, 2010*, 2010, pp. 15–21.

[29] Amir Herzberg and Haya Shulman, "Fragmentation considered poisonous," *CoRR*, 2012.

[30] M. P. Andrews, *Negative Caching of DNS Queries (DNS NCACHE)*, 1998. [Online]. Available: https://doi.org/10.17487/RFC2308

[31] R. Arends, R. Austein, M. Larrson, D. Massey, and S.Rose, *RFC 4033: DNS Security Introduction and Requirements*. Internet Engineering Task Force (IETF), 2005. [Online]. Available: https://www.ietf.org/rfc/rfc4033.txt

[32] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, *Protocol Modifications for the DNS Security Extensions*, 2005. [Online]. Available: https://doi.org/10.17487/RFC4035

[33] D. Atkins and R. Austein, *Threat Analysis of the Domain Name System (DNS)*, 2004. [Online]. Available: https://tools.ietf.org/html/rfc3833

[34] S. N. Bhatti and R. J. Atkinson, "Reducing dns caching," *2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 792–797, 2011.

[35] K. Borgolte, T. Fiebig, S. Hao, C. Kruegel, and GiovanniVigna, "Cloud strife: Mitigating the security risks of domain-validated certificates," in *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*, 2018.

[36] H. Boulakhrif, "Analysis of DNS Resolver Performance Measurements," Master's thesis, University of Amsterdam, the Netherlands, 2015.

[37] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. T. Polk, *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, 2008. [Online]. Available: https://doi.org/10.17487/RFC5280

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

[39] T. Dai, H. Shulman, and M. Waidner, "DNSSEC misconfigurations in popular domains," in *Cryptology and Network Security - 15th International Conference, CANS 2016, Milan, Italy, November 14-16, 2016, Proceedings*, 2016, pp. 651–660.

[40] T. DeGroote, "Lies, damn lies and dns performance statistics," https://secure64.com/lies-damn-lies-dns-performance-statistics/(Accessed: July 2018).

[41] L. Deri, S. Mainardi, M. Martinelli, and E. Gregori, "Exploiting DNS traffic to rank internet domains," in *IEEE International Conference on Communications, ICC 2013, Budapest, Hungary, June 9-13, 2013, Workshops Proceedings*, 2013, pp. 1325–1329.

[42] R. Elz and R. Bush, *Clarifications to the DNS Specification*, 1997. [Online]. Available: https://doi.org/10.17487/RFC2181

[43] D. Franke, D. Sibold, and K. Teichel, *draft-ietf-ntp-using-nts-for-ntp-12: Network Time Security for the Network Time Protocol*. Internet Engineering Task Force (IETF), July 2018, https://tools.ietf.org/html/draft-ietf-ntp-using-nts-for-ntp-12.

[44] A. Herzberg and H. Shulman, "Antidotes for DNS poisoning by off-path adversaries," in *Seventh International Conference on Availability, Reliability and Security, Prague, ARES 2012, Czech Republic, August 20-24, 2012*, 2012, pp. 262–267.

[45] T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling, "Measuring and detecting fast-flux service networks," in *Proceedings of the Network and Distributed System Security Symposium, NDSS 2008, San Diego, California, USA, 10th February - 13th February 2008*, 2008.

[46] A. Hubert and R. von Mook, *Measures for Making DNS More Resilient against Forged Answers*, 2009. [Online]. Available: https://tools.ietf.org/html/rfc5452

[47] D. E. E. III and M. P. Andrews, *Domain Name System (DNS) Cookies*, 2016. [Online]. Available: https://doi.org/10.17487/RFC7873

[48] J. Selvi, "Breaking SSL using time synchronisation attacks," *DEFCON'23*, June 2015.

[49] J. Jung, E. Sit, H. Balakrishnan, and R. T. Morris, "DNS performance and the effectiveness of caching," in *Proceedings of the 1st ACM SIGCOMM Internet Measurement Workshop, IMW 2001, San Francisco, California, USA, November 1-2, 2001*, 2001, pp. 153–167.

[50] D. Kaminsky, "It's the end of the cache as we know it. in: Black hat conference, august 2008." http://www.blackhat.com/presentations/bh-jp-08/bh-jp-08-Kaminsky/BlackHat-Japan-08-Kaminsky-DNS08-BlackOps.pdf(Accessed: July 2018).

[51] A. Klein, "Microsoft windows dns stub resolver cache poisoning (ms08-020)," http://www.securiteam.com/securityreviews/5QP022KO1E.html(Accessed: July 2018).

[52] A. Klein, H. Shulman, and M. Waidner, "Internet-wide study of dns cache injections," in *INFOCOM 2017-IEEE Conference on Computer Communications, IEEE*. IEEE, 2017, pp. 1–9.

[53] ——, "Internet-wide study of DNS cache injections," in *2017 IEEE Conference on Computer Communications, INFOCOM 2017, Atlanta, GA, USA, May 1-4, 2017*, 2017, pp. 1–9.

[54] O. M. Kolkman and R. M. Gieben, *DNSSEC Operational Practices*, 2006. [Online]. Available: https://doi.org/10.17487/RFC4641

[55] D. Lawrence and W. Kumari, *Serving Stale Data to Improve DNS Resiliency*. Internet Engineering Task Force (IETF), 2017, https://tools.ietf.org/html/draft-ietf-dnsop-serve-stale-00.

[56] R. Love, *Linux System Programming*. O'Reilly Media, Inc., 2013.

[57] A. Malhotra and S. Goldberg, *Message Authentication Codes for the Network Time Protocol*. Internet Engineering Task Force (IETF), March 2018, https://tools.ietf.org/html/draft-ietf-ntp-mac-04.

[58] A. Malhotra, I. E. Cohen, E. Brakke, and S. Goldberg, "Attacking the network time protocol," in *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*, 2016.

[59] A. Malhotra and S. Goldberg, "Attacking NTP's authenticated broadcast mode," *Computer Communication Review*, pp. 12–17, 2016.

[60] A. Malhotra, M. V. Gundy, M. Varia, H. Kennedy, J. Gardner, and S. Goldberg, "The security of ntp's datagram protocol," in *Financial Cryptography and Data Security - 21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017.*, 2017, pp. 405–423.

[61] J. Mauch, "Open resolver project," http://openresolverproject.org/(Accessed: July 2018).

[62] ——, "openntpproject: NTP Scanning Project."

[63] D. Mills, J. Martin, J. Burbank, and W. Kasch, *RFC 5905: Network Time Protocol Version 4: Protocol and Algorithms Specification*. Internet Engineering Task Force (IETF), 2010. [Online]. Available: http://tools.ietf.org/html/rfc5905

[64] D. L. Mills, *Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI*, 2006. [Online]. Available: https://doi.org/10.17487/RFC4330

[65] N. Minar, "A survey of the NTP network," 1999.

[66] P. Mockapetris, *RFC 1034: Domain Names: Concepts and Facilities*. Internet Engineering Task Force (IETF), 1987. [Online]. Available: https://tools.ietf.org/html/rfc1034

[67] P. V. Mockapetris, *Domain names - implementation and specification*. Internet Engineering Task Force (IETF), 1987. [Online]. Available: https://doi.org/10.17487/RFC1035

[68] C. D. Murta, P. R. Torres Jr, and P. Mohapatra, "Characterizing quality of time and topology in a time synchronization network." in *GLOBECOM*, 2006.

[69] P. O'leary, J. Paugh, and R. S. Wilbourn, "Dns-based ranking of domain names," Patent 20 160 065 535.

[70] J. Pan, Y. T. Hou, and B. Li, "An overview of dns-based server selections in content distribution networks," *Computer Networks*, pp. 695–711, 2003.

[71] V. Pappas, Ed., and E. E. Osterweil, *Improving DNS Service Availability by Using Long TTL Values*. Internet Engineering Task Force (IETF), 2012, https://tools.ietf.org/id/draft-pappas-dnsop-long-ttl-04.html.

[72] Y. Rekhter, B. G. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear, *Address Allocation for Private Internets*, 1996. [Online]. Available: https://doi.org/10.17487/RFC1918

[73] Roy Arends and Rob Austein and Matt Larson and Dan Massey and Scott Rose, *Resource Records for the DNS Security Extensions*. Internet Engineering Task Force (IETF), 2005. [Online]. Available: https://doi.org/10.17487/RFC4034

[74] K. Schomp, T. Callahan, M. Rabinovich, and M. Allman, "Assessing DNS vulnerability to record injection," in *Passive and Active Measurement - 15th International Conference, PAM 2014, Los Angeles, CA, USA, March 10-11, 2014, Proceedings*, 2014, pp. 214–223.

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

[76] K. Singh and H. Schulzrinne, "Failover, load sharing and server architecture in SIP telephony," *Computer Communications*, pp. 927–942, 2007.

[77] J. Snijders, "Nlnog ring," https://ring.nlnog.net/(Accessed: July 2018).

[78] Y. Yu, D. Wessels, M. Larson, and L. Zhang, "Authority server selection in DNS caching resolvers," *Computer Communication Review*, 2012.

[79] L. Yuan, K. Kant, P. Mohapatra, and C. Chuah, "Dox: A peer-to-peer antidote for dns cache poisoning attacks," in *2006 IEEE International Conference on Communications*, 2006, pp. 2345–2350.