
# Timeless Timing Attacks and Preload Defenses in Tor's DNS Cache

**Rasmus Dahlberg and Tobias Pulls,** ***Karlstad University***

https://www.usenix.org/conference/usenixsecurity23/presentation/dahlberg

**This paper is included in the Proceedings of the 32nd USENIX Security Symposium.**

**August 9–11, 2023 • Anaheim, CA, USA**

978-1-939133-37-3

**Open access to the Proceedings of the 32nd USENIX Security Symposium is sponsored by USENIX.**

# Timeless Timing Attacks and Preload Defenses in Tor's DNS Cache

Rasmus Dahlberg
*Karlstad University*
rasmus.dahlberg@kau.se

Tobias Pulls
*Karlstad University*
tobias.pulls@kau.se

## Abstract

We show that Tor's DNS cache is vulnerable to a timeless timing attack, allowing anyone to determine if a domain is cached or not without any false positives. The attack requires sending a single TLS record. It can be repeated to determine when a domain is no longer cached to leak the insertion time. Our evaluation in the Tor network shows no instances of cached domains being reported as uncached and vice versa after 12M repetitions while only targeting our own domains. This shifts DNS in Tor from an unreliable side-channel—using traditional timing attacks with network jitter—to being perfectly reliable. We responsibly disclosed the attack and suggested two short-term mitigations.

As a long-term defense for the DNS cache in Tor against all types of (timeless) timing attacks, we propose a redesign where only an allowlist of domains is preloaded to always be cached across circuits. We compare the performance of a preloaded DNS cache to Tor's current solution towards DNS by measuring aggregated statistics for four months from two exits (after engaging with the Tor Research Safety Board and our university ethical review process). The evaluated preload lists are variants of the following top-lists: Alexa, Cisco Umbrella, and Tranco. Our results show that four-months-old preload lists can be tuned to offer comparable performance under similar resource usage or to significantly improve shared cache-hit ratios (2–3x) with a modest increase in memory usage and resolver load compared to a 100 Mbit/s exit. We conclude that Tor's current DNS cache is mostly a privacy harm because the majority of cached domains are unlikely to lead to cache hits but remain there to be probed by attackers.

## 1 Introduction

Tor [10] is a volunteer-operated anonymity network composed of relays that route encrypted traffic with low latency. One of Tor's trade-offs is to not provide anonymity against a global passive attacker that observes traffic as it enters and leaves the network [9, 10]. A typical attacker setting is therefore to only observe encrypted traffic as it enters the network from an identifiable user, forcing traffic analysis of the encrypted packets to classify the user's behavior. An attacker that tries to classify visited websites is said to perform Website Fingerprinting (WF) [5, 16, 17, 26, 32, 47]. Many questions about the practicality of WF attacks have been raised, ranging from how to keep a trained dataset updated to managing false positives [6, 21, 33, 49]. False positives in WF may be ruled out using side-channels [21, 42]. For example, an attacker with access to (traffic to [43]) Google's public DNS resolver can use it to confirm if a website visit really happened over Tor [13].

Side-channels that leak information about exiting traffic are in fact many [42]. For example, during the course of a website visit there may be interactions with DNS resolvers, OCSP responders, real-time bidding platforms, and CDNs. An attacker that is able to query or gain access to the resulting datasets learns partial information about destination traffic, notably without ever observing any of the exiting TCP flows typically associated with correlation attacks on Tor [20, 30]. Depending on the ease of accessibility (e.g., does it require Google reach), reliability (e.g., are there any false positives), and coverage (e.g., is it only applicable for a small fraction of exit traffic), the impact of a given side-channel will be more or less urgent to address with mitigations and/or defenses [10].

### 1.1 Timeless Timing Attacks in Tor's DNS

Timing attacks exploit that an operation takes more or less time to execute depending on something secret. The attacker's goal is to infer the secret information by merely observing the non-constant execution times, e.g., to recover a private key [23], decrypt a ciphertext [1], or check if a domain is cached by a Tor exit [42]. A remote timing attack takes place over a network. Repeated measurements and statistics are usually required to account for network jitter, which adds noise to the observed timings [8]. Van Goethem *et al.* [48] proposed a technique that eliminates all network jitter in remote attacks. It is applicable if two requests can be sent to arrive at the same time, request processing is concurrent, and the order in which

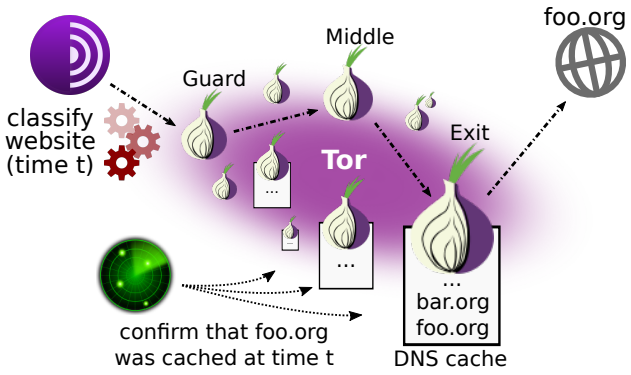



Figure 1: WF with an attacker that rules out false positives by checking that the expected DNS records were cached at the right time by conducting timeless timing attacks against exits.

responses are returned reflects differences in execution time.

We find that Tor's DNS cache at exits fulfills all three criteria of a timeless timing attack, allowing anyone to determine if a domain is cached or not by sending a single TLS record. The attack is reliable (neither false positives nor negatives), confirmed by using our prototype to make 12M network measurements against our own domains. The attack is also repeatable, making the exact time that a domain was inserted into the cache inferable due to determinism in Tor's TTL logic.

Figure 1 provides a summary of how the ability to infer whether domains are (un)cached at exits make WF attacks more practical. The attacker observes encrypted traffic from a client to a guard relay at time $t$, classifying the network trace as associated with `foo.org`. The attacker then conducts timeless timing attacks against all exits in the Tor network to determine if `foo.org` was really visited by *someone* at time $t$. If the answer is yes the classification is accepted, otherwise it is rejected. Prior work by Pulls and Dahlberg show that the capability to determine whether a website was visited from Tor at time $t$ removes virtually all false positives in WF attacks for all but the most popular websites on the web [42]. We provide further evidence that this is a realistic capability to assume by demonstrating that *any attacker with an Internet connection could have used it in attacks for the last two decades*. While it is a powerful capability to eliminate false positives, the overall success in linking users with their website visits also depends on the WF attack [6, 21, 33, 49].

## 1.2 Preload Defenses and Measurements

Patching Tor's DNS cache to resist (timeless) timing attacks is challenging without hurting performance. For example, making all DNS lookups constant time would defeat the purpose of having a cache. The idea of our long-term defense is to remove harmful cross-circuit caching that is unlikely to boost performance while still safely caching useful domains. The Tor-network measurements of Mani *et al.* [28] tell us that web-traffic from the Tor network matches that of the rest of the Internet, following popularity lists like Alexa [2]. What should boost cross-circuit performance is the upper parts of a representative popularity list; not the long tail of infrequently visited sites. This is the intuition of our defense. Preload a list of popular domains that are cached and continuously refreshed by all exits. A domain name is either always cached as part of the preload list or not shared across circuits at all.

We conduct four months of measurements in the live Tor network to evaluate 400 popularity lists derived from Alexa [2], Cisco Umbrella [7], and Tranco [25]. To put our results into perspective, we also measure a baseline of Tor's current DNS cache performance. The measurement method is to collect aggregated counters every 15 minutes, e.g., the number of lookups cache-hits, and memory overhead, from two 100 Mbit/s relays with web and permissive exit port policies.

Tor's mean *cross-circuit* cache-hit ratio is currently 11% (web) and 17% (permissive). Variants of Alexa/Tranco top-200 (web) and Alexa/Tranco top-700 (permissive) achieve the same cross-circuit cache-hit ratios. A preload list from the top-10k can achieve 2–3 times higher cross-circuit cache-hit ratios at the cost of at most 60 MiB memory and some increased resolver load (manageable in part due to RFC 8767 [24]). Throughout the entire measurement we noted only a slight decline in effectiveness while using stale preload lists (i.e., when using four-month-old lists at the end). This adds to the feasibility of using preload lists, as in practice someone has to assemble and deliver them to all exits in the Tor network.

## 1.3 Contributions and Outline

Our contributions are as follows:

- Performance measurements of the DNS cache in Tor over four months from two exits, showing an average 80–83% cache-hit ratio with approximately 10,000 entries in the cache; around 11–17% of the observed cache hits are due to the cache being shared across circuits, and the number of lookups appears weakly correlated with exit probability (Section 3).
- Demonstration of a timeless timing attack that probes for cached domains in Tor's DNS cache without any false positives or false negatives after 12M repetitions against our own domain in the Tor network (Section 4).
- Mitigations based on fuzzy TTLs and cover lookups that add some limited short-term protections (Section 5).
- A long-term redesign of Tor's DNS cache that defends against (timeless) timing attacks. Cache-hit ratios can be tuned to offer comparable performance under similar resource usage as today or to significantly improve shared cache-hit ratios (2–3x) with a modest increase in memory usage and resolver load, notably invariant to exit probability as preload lists are fixed (Section 6).

Section 2 provides necessary background on DNS and Tor, Section 7 discusses related work, and Section 8 offers conclusions, followed by the availability of our research artifacts.

We would like to highlight that Sections 3.1, 4.4, and 6.4 describe ethical and safety precautions to ensure that no users were harmed by our research and to maximize its positive impact. We responsibly disclosed our timeless timing attack to the Tor Project and engaged with the Tor Research Safety Board as well as our university's ethical review process as part of performing network measurements to inform our defenses.

## 2 Background

The remainder of the paper requires preliminaries about DNS (Section 2.1), in particular in relation to Tor (Section 2.2).

### 2.1 DNS

DNS is a hierarchical system that maps domain names ("domains") to IP addresses. The hierarchy is composed of root servers, top-level domain (TLD) servers, and authoritative name servers. Root servers are aware of TLD servers like `.com`. TLD servers are aware of authoritative name servers in their zone like `example.com`. Authoritative name servers are aware of the actual answers to a domain lookup. A domain lookup for `example.com` involves asking the root server for the TLD server of `.com`; the TLD server for the authoritative name server of `example.com`; and finally the authoritative name server for the IP address of `example.com`. The resolve process is typically performed iteratively in plaintext over UDP by a third-party resolver that caches responses, e.g., to improve performance. The default is usually to rely on ISP DNS resolvers. It is also possible to configure other ones, e.g., Google's `8.8.8.8` or self-hosted using `unbound`, `bind`, etc.

Of note is that the resolved domains are associated with a Time To Live (TTL) value. As the name suggest, it is the amount of time that a resolved domain should be considered fresh. TTL values are sometimes overridden in caches to improve reliability [24, 29] or preserve privacy [13].

### 2.2 Tor

The Tor network is composed of thousands of relays that route encrypted traffic on behalf of millions of daily users [10, 28]. Ordinary uses of Tor include preserving privacy, safety and freedom as well as facilitating dissent and circumventing censorship [14, 44]. Access to the Tor network is easy using Tor Browser (TB), which is configured to proxy all traffic through a local Tor process that takes care of routing. TB adds many other protections that are orthogonal to our work [35].

During a regular website visit a circuit is built through a guard, middle, and exit relay. The first relay is fixed in a small guard set that rarely changes once selected, while the middle and exit relays are randomly selected weighted by bandwidth for each new circuit. A circuit may have many streams (analogous to TCP/IP connections), typically corresponding to separate flows for a particular destination. Control traffic and data is transported through the network in fixed-size cells that are encrypted in layers. At each hop in a circuit, one layer of encryption is peeled-off. Outstanding cells from relay A to relay B are sent in a shared channel that is TLS protected. Public keys, relay identities, and more are discovered in Tor's consensus, which is secure if a threshold of trusted directory authorities act honestly.

We are particularly interested in how Tor interacts with DNS. To look up a domain, the user's Tor process may send a RESOLVE cell that requests resolution by the exit. Some exits are configured with their own iterative resolvers, while others rely on DNS from their ISP or other third-parties [13]. The answer to a lookup is stored in the exit's cache, but with the TTL *clipped* to 300 or 3600 seconds depending on if the TTL is $\leq 300$ seconds or not. A RESOLVED cell is then sent to the user, who only gets to see the clipped TTL regardless of how long it has been stored in the cache to avoid leaking information about past exit traffic (like the insertion time which would be trivial to infer from a counted-down TTL). If too many entries are added to Tor's DNS cache and memory becomes a scarce resource, an Out-Of-Memory (OOM) job deletes domains until freeing enough memory. This is all controlled by an event-driven single-threaded main loop.

Of further note is that TB is based on Firefox. As part of connecting to a website, DNS is handled transparently through a SOCKS proxy provided by the local Tor process. Requests to connect to a domain through the SOCKS proxy results in the user's Tor process sending a BEGIN cell to establish a connection to the destination, which in turn triggers domain resolution at the exit. In other words, there are two ways to look up domains: RESOLVE cells and BEGIN cells. At no point is any resolved IP address cached in TB or in the user's Tor process. This prevents shared state (the cache) from being used to fingerprint a user's activity across different circuits.

We continue our introduction to Tor's DNS cache next while describing the first measurement of its performance.

## 3 Tor's DNS Cache Today

To better understand the DNS cache of Tor today, we set out to collect performance metrics from exits in the live Tor network. Section 3.1 covers ethical considerations, followed by data collection in Section 3.2 and resulting metrics in Section 3.3.

### 3.1 Ethical Considerations

We submitted a proposal to the Tor Research Safety Board describing measurements that would ultimately inform the design of a long-term defense (Section 6) against our improved attack (Section 4). To be able to assess the impact of the defense we needed to better understand the DNS cache

Tor has today as a baseline. After a couple of iterations with the Tor Research Safety Board we reached consensus, and then successfully completed our university's ethical review process. The proposal also included measurements needed for our defense, described later in Section 6.3. During the measurements period of four months we consulted the Tor Research Safety Board to discuss our results.

The intuition of our measurement is as follows. Two exit relays are operated to collect counters related to domain lookups. For example, the number of lookups and cache hits (Section 3.2). These counters are the result of all traffic at the exit, aggregated over 15 minutes intervals before being written to disk and then reset in memory. Based on an exit probability of about 0.0005 ($\approx$ 100Mbit/s), we extrapolated from the measurements of Mani *et al.* [28] that we should expect about 725 website visits during 15 minutes. Each website visit typically triggers multiple domain lookups [13] that affect our global counters. A collection interval of 15 minutes should thus aggregate hundreds of website visits for a small fraction of the network, making the resulting dataset hardly useful for an attacker performing correlation or confirmation attacks on the network. This sketch appears to be confirmed by our measurement results: out of 23,632 15-minute intervals, only 18 contained less than 1,000 lookups. Our conclusion together with the Tor Research Safety Board was that the resulting dataset should be safe to make public (further discussed later).

## 3.2 Data Collection

Two 100 Mbit/s exit relays were operated on the premises of DFRI (`https://www.dfri.se`) from May 2 until September 3, 2022. One exit was configured in its exit policy with *web* ports[1]. The other relay was configured with *permissive* ports[2] to also allow non-web traffic. Otherwise the two exits were identical, running on the same VM with a dedicated `unbound` process that had caching disabled by setting the `rrset-cache-size` to zero (to avoid TTL biases). We collected the following counters every 15 minutes at both exits:

**timestamp** UNIX timestamp when the data was collected.

**lookups** Total number of observed domain lookups.

**hits_5m** Number of cache hits with a TTL of 300 seconds.

**hits_60m** Number of cache hits with a TTL of 3,600 seconds.

**hits_pending** Number of cache hits with a pending resolve, i.e., an answer has been requested but is not yet available.

**hits_same_circuit** Number of streams that looked up a domain that was previously looked up on the same circuit.

**num_cache_entries** Number of entries in Tor's DNS cache.

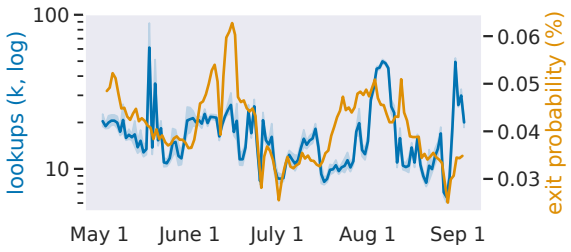
[1] Reject all ports except 80 and 443. (The exit can still do DNS for users.)

[2] Allow all ports except 25, 119, 135–139, 445, 563, 1214, 4661–4666, 6346–6429 6699, and 6881–6999.

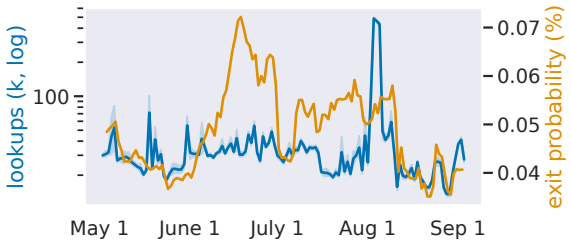
A timestamp is needed to plot metrics as a function of time. Timestamps are also crucial for the additional counters described in Section 6.3. The number of lookups and different types of cache hits are needed to get a baseline of cache-hit ratios. The number of entries in Tor's DNS cache (at the time of collection) is needed to get a baseline of memory usage. The necessary Tor changes to collect all metrics (including Section 6.3) were relatively modest: 400 lines of code.

## 3.3 Metrics

Regarding lookups per 15 minutes, the web exit processed a mean of 17,530 and median of 13,393 lookups (Figure 2a), and the permissive exit processed a mean of 41,100 and median of 26,940 lookups (Figure 2b). The permissive exit policy results in significantly more lookups. Around August 1, our exits experienced downtime, visible as dips in lookups in both figures (at times fewer than 1,000 lookups, as noted in Section 3.1). Exit probability is weakly correlated with lookups: Pearson correlation 0.30 (web) and 0.16 (permissive).




(a) web

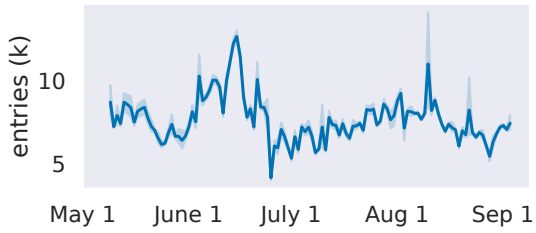



(b) permissive

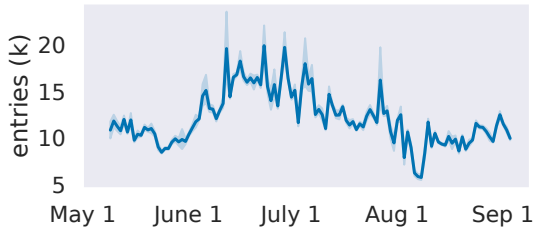
Figure 2: Lookups every 15 minutes and exit probability.

Figures 3a and 3b show the number of entries in Tor's DNS cache. The web exit has a mean of 7,672 and median of 7,325 entries, and the permissive exit a mean of 12,130 and median of 11,408 entries. Both appear relatively stable compared to the number of lookups (note log-scale y-axis in Figure 2). Likely, this is because traffic on the Tor network is not uniformly distributed, but rather concentrated to relatively few destinations, e.g., as shown with website popularity [28].

Central to a DNS cache is its *cache-hit* ratio: how often lookups can be resolved using cached entries instead of asking DNS resolvers. Figures 4a and 4b show the cache-hit ratios

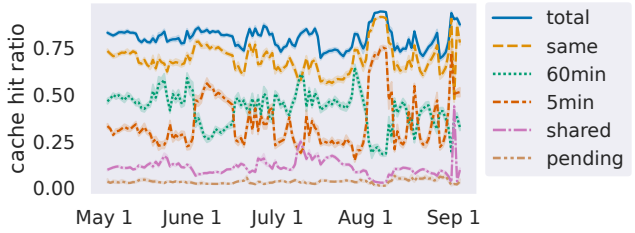



(a) web

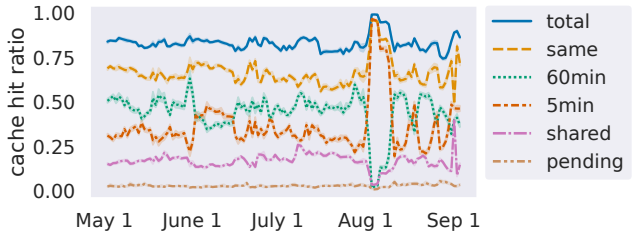



(b) permissive

Figure 3: Cache entries every 15 minutes.




(a) web




(b) permissive

Figure 4: Cache-hit ratio every 15 minutes. The total ratio can be split by same+shared hits or 60min+5min+pending hits.

for the two exits, with a mean cache-hit ratio of 0.80 (web) and 0.83 (permissive). We also show if the cache hits occurred due to a cache entry used earlier on the same circuit ("same") or from another circuit ("shared"). Further, over all the cache hits, we show if the hits were because of DNS entries with a five-minute cached TTL ("5min"), a 60-minute cached TTL ("60min"), or pending entries in the DNS cache ("pending"). Same circuit hits are likely due to Tor Browser improving performance by creating multiple streams to the same destination. The cross-circuit cache-hit ratio is much smaller ("shared") with a mean of 0.11 (web) and 0.17 (permissive). We return to these ratios in Section 6.5 to compare with our defense.

During the four months of measurements, our exits experienced sporadic downtime (early August) and the Tor-network endured significant network DDoS activities [37]. This shows in our data, e.g., with the drop to close to zero lookups in Figure 2, huge spikes of cached entries in Figure 3, and periods where the cache-hit ratio was almost one in Figure 4.

To summarize, Tor's DNS cache has a cache-hit ratio over 80% using a modestly sized DNS cache. About 11–17% of these hits are due to sharing the cache across circuits. The number of lookups are weakly correlated to exit probability.

# 4 Timeless Timing Attack

Past work demonstrated timing attacks against Tor's DNS cache [42]. In short, anyone can observe the latency of a domain lookup to determine if it is more or less likely that an answer is (not) cached. A quick response is more likely to be cached, thereby leaking information about past traffic on an exit. A downside of such a remote timing attack is that it is subject to network jitter while traversing hops in the Tor network. We show how to bypass this limitation by constructing a timeless timing attack that is immune to network jitter [48]. Notably the attack does not require any special capabilities, just Internet access and a very modest computer.

Section 4.1 outlines the attack, followed by a description of our prototype implementation in Section 4.2, evaluation in Section 4.3, as well as ethical considerations in Section 4.4.

## 4.1 Detailed Description

An exit's processing of an incoming RESOLVE cell depends on if an answer is cached or not, see Figure 5. An answer may already be available and a RESOLVED cell can be scheduled for sending immediately ("cached"). Otherwise an answer is not yet available and a resolve process needs to take place concurrently to avoid blocking ("uncached"). We construct a timeless timing attack by exploiting the fact that scheduling RESOLVED cells for sending with different concurrent timings depend on if an answer is cached (send immediately) or uncached (send based on an event later on) [38].

### 4.1.1 Attack Outline

Suppose that we craft two RESOLVE cells for `example.com` and `evil.com` such that they are processed by an exit *directly after each other without any events in between*. Further suppose that `evil.com` is cached. The first RESOLVE cell is `example.com`. The second RESOLVE cell is `evil.com`. Following from the flow in Figure 5, we can determine if `example.com` is (un)cached by observing only the order in which the two RESOLVED cells come back. The order will

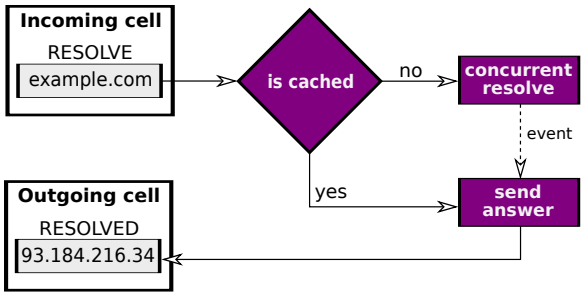



Figure 5: Processing of an incoming RESOLVE cell at an exit relay. Answers of concurrent resolves are triggered by events.

be switched if `example.com` needs concurrent resolving because *the answer is not available until after an event* (uncached). Otherwise the order is preserved (cached). Sending two requests to be processed at the same time and exploiting concurrency as well as differences in processing time that affects the response order is what makes it *timeless* [48].

Figure 6 provides a detailed description on how to satisfy the presumed setup. The attacker starts by looking up its own domain name for a selected exit. This ensures that `evil.com` is cached. Next, two RESOLVE cells are sent in the same TLS record from a hop proceeding the exit. Both cells will be unpacked at the same time by TLS [39], and when processing starts all available cells will be handled before giving control back to Tor's main loop [40]. Now recall that Tor is single-threaded. An event from any concurrent DNS resolve can thus not be completed before all unpacked cells were fully processed. This ensures that the order in which our two RESOLVED cells come back in is guaranteed to leak if `example.com` is (un)cached as long as both RESOLVE cells arrived together in-order and `evil.com` is really cached.

It should be noted that an attacker can gain full control of how their TLS records are packed to exits by either running a modified Tor relay or creating one-hop circuits. In practise, it is also possible to omit the step of caching `evil.com` and instead send a `RESOLVE` cell containing an IP address. Tor will simply echo the IP as if it was cached [41]. We describe the attack without this optimization because it is more general.

#### 4.1.2 Repeated Attack to Learn Insertion Time

So far we described how to determine if a domain is (un)cached at an exit. Figure 7 shows how to derive the exact time that a domain was added to an exit's DNS cache. First determine whether the domain's TTL will be clipped to 300 or 3,600 seconds by observing the TTL returned from the authoritative name server or the configured resolvers of the exit [13]. Then repeat the timeless timing attack periodically until the domain is no longer cached, say, once per second. Suppose the expected clip is 300 seconds and the attack started at time $t$. If it takes $x < 300$ seconds for the entry to become uncached, it was added to the exit's DNS cache at time $t + x - 300$s. Observing $x > 300$ seconds means that a different party inserted the entry into the cache between probes (may happen for some of the most frequently looked-up domains, depending on probing frequency). To recover, the attacker can perform the same steps again until they succeed. For example, with two tries the initial insertion happened at $t + x - 600$s. Notably these estimates cannot be more precise than the attacker's repetition interval.

#### 4.1.3 Discussion

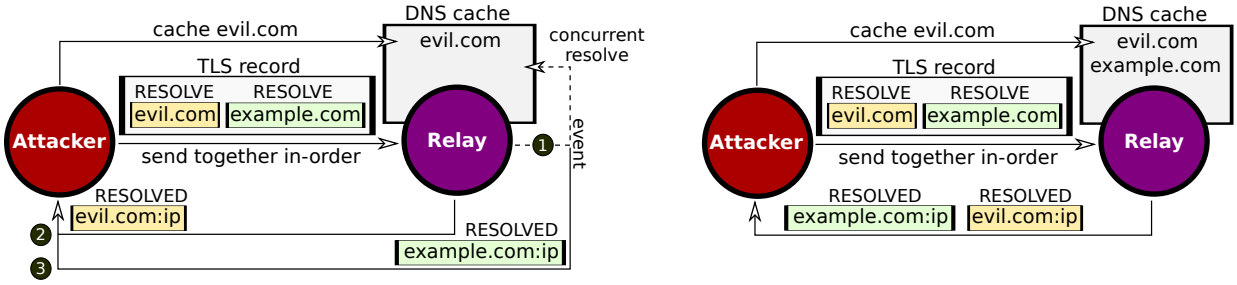
While an attacker can determine if a domain is cached by an exit and if so the exact time it was added, the attacker cannot determine the number or timings of lookups for a domain after entering the cache. In isolation, the attacker also cannot determine which identifiable user cached a given domain.

It is easy to conduct the attack in parallel because probing for the status of `foo.org` is completely independent from `bar.org` at the same relay as well as other probes on different relays. In other words, an attacker can probe a single domain on all exits simultaneously, many different domains at a single exit, or both. Network-wide probes for the same domain may be detectable by observing the DNS caches of multiple relays and correlating their contents. However, note that a risk-averse attacker [3] may spread their probes over time (five or sixty minutes) and domains (expected twelve domains per website on Alexa top-1M websites [13]), if the goal is to confirm a website visit.

An example use-case for a parallel attack is answering network-wide queries, for example, "is `foo.org` visited more frequently than `bar.org`, or did any Tor user visit `baz.org` at a particular point in time?" The latter is an instantiation of a so-called website oracle [42]. Website oracles remove virtually all false positives in WF attacks for all but the most popular websites on the web, and WF attacks may connect identifiable users with visited websites. See Figure 1 in Section 1 for an overview of this attack setting.

### 4.2 Prototype Implementation

We prototyped our timeless timing attack so that it runs for a given exit and a list of domains. Figure 8 shows the overall setup which consists of `carml`, `tor-resolve`, a locally patched Tor process, and a Python script automating the entire setup. First Tor is started, a *one-hop circuit* is built to the selected exit, and all streams are attached to it using `carml`. Next, `tor-resolve` is used to send a special lookup query for `example.com` by simply appending a magic string `--sc`. The patched Tor process splits such requests into two RESOLVE cells in the same TLS record: one for the specified domain, and another one that is guaranteed to not need any concurrent resolving. Finally Tor sets the output to `0.0.0.0` if the resulting RESOLVED cells switched order, otherwise `1.1.1.1` (arbitrary constants). After processing all domains




(a) uncached

(b) cached

Figure 6: The attacker ensures a domain `evil.com` is cached. Next, two RESOLVE cells are sent to arrive at the same time in-order. The relay processes both cells before triggering any resolve event. This means that answers can only be sent directly if no resolving is needed. The order of RESOLVED cells switch if `example.com` is uncached. Otherwise the order is preserved.

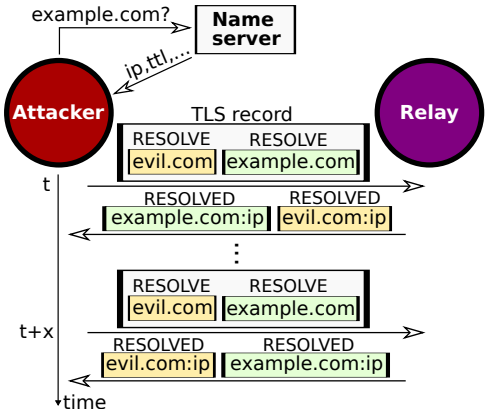



Figure 7: Repeated timeless timing attack to infer the exact time that a domain was cached by someone at an exit relay. For example, if the expected clip is 300s (ttl $\leq$ 300s), the attack is repeated every second, and the observed $x$ is 40s, then caching of `example.com` happened at time $\approx t - 260$s.

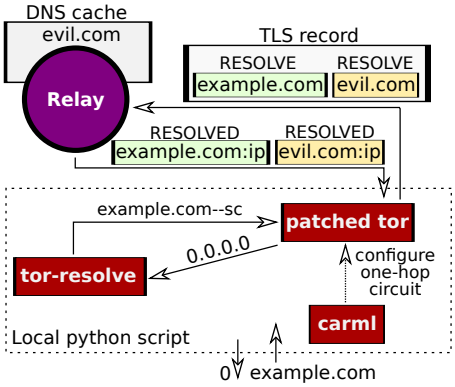



Figure 8: Local attack setup consisting of `carml` to build one-hop circuits, `tor-resolve` to inject queries, and a patched tor process that transforms them into timeless timing attacks.

Tor is closed and the output is a list where each item is zero (uncached), one (cached), or negative (unknown, e.g., due to a resolve timeout, a stream attach failure, or a vanished circuit). The complete attack required less than 100 lines of C to patch Tor, as well as 200 lines of Python to make it fully automated.

## 4.3 Network Measurements

We conducted measurements in the live Tor network to evaluate the reliability of our prototype with four parallel instances of the setup in Figure 8 on a system with an Intel(R) Xeon(R) CPU E5-2630 @ 2.30GHz and 4GB of DRAM. All targeted domains were our own, see ethical considerations in Section 4.4. In total there were 14,446 runs between May 17–26, 2022. Each run used an exit that was sampled uniformly at random. Assuming 1,000 exits at all times (conservative), the individual select probability should not exceed 0.004 per run. Each run performed up to 1,000 timeless timing attacks, chunked into 500 attacks per circuit and alternating between uncached and cached lookups by specifying a unique domain twice in a row: `<counter>.<timestamp>.<instance id>.example.com`. The maximum runtime was set to ten minutes. Each query also had a ten second timeout. In the advent of errors like circuit failure or timeouts, the remainder of the run was canceled but all results up until that point were collected. The average number of DNS requests leaving the Tor network from *all four combined instances* was 8.6 per second. The effective queries per second was slightly higher due to brief pauses while setting up a new run. For reference, Sonntag reported in 2018 that the DNS resolver of an exit with 200Mbit/s received an average and maximum of 18 and 81 requests per second [45]. Earlier, Figure 2 also showed significant variability in lookups. Handling our per-exit overhead during a couple of minutes should thus be insignificant when compared to regular patterns for DNS traffic in the network.

Table 1 summarizes our results. After 12M timeless timing attacks, there were no cases of uncached lookups being reported as cached and vice versa. This is consistent with the

description in Section 4.1: neither false positives nor false negatives are expected. The observed probability to not get an answer due to detectable failures were 0.00025.

Table 1: Timeless timing attack results. Neither false negatives nor false positives were observed with 6M repetitions each.

| Type | Got uncached | Got cached | Failures |
|---|---|---|---|
| Uncached | 6,034,779 | 0 | 2,858 |
| Cached | 0 | 6,034,594 | 142 |

### 4.4 Ethical Considerations

We responsibly disclosed our attack to the Tor Project through their security list. The submitted disclosure included a theoretical attack description, a prototype implementation with measurements showing how reliable it was, as well as a sketch of short-term and long-term defenses. As part of our dialog, we also coordinated with the Tor Project on submitting this paper to USENIX Security to get peer review.

The conducted network measurements targeted domains under our own control. This ensured that we did not learn anything about real Tor users. Performance overhead on exits and the Tor network at large was also modest, see Section 4.3. In other words, the downsides were negligible while the significance of evaluating *real-world reliability* was helpful to inform and motivate the need for mitigations and defenses.

## 5 Mitigations

Until a more comprehensive defense can be deployed we propose two short-term mitigations that require little (fuzzy TTLs) or no (cover lookups) changes to Tor. The former adds some uncertainty with regards to when a domain was added to an exit's DNS cache. The latter can remove or reduce the attacker's ability to conduct attacks against specific domains but is limited in its scalability.

### 5.1 Fuzzy TTLs

Recall that it is possible to determine when a domain was inserted into an exit's DNS cache (Section 4.1) once you know the time $t$ when the timeless timing attack started, the duration until the domain was no longer cached $x$ (repeated probes), and the expected clip value clipped_ttl of the domain. The idea of fuzzy TTLs is to add uncertainty by randomizing the length of time that an entry is cached.

In more detail, keep Tor's DNS cache as-is but sample the cache duration uniformly at random from $[m, \text{clipped\_ttl}]$, where $m$ is the minimum duration to cache. Upon observing the exact time of removal $t+x$, the attacker now learns that the domain has been in the cache for the duration $x$ and was thus cached between $[t+x-\text{clipped\_ttl}, t-m]$. Note that if $m = \text{clipped\_ttl}$, then $x = 0$; the same as in Tor today.

The reality of websites is unfortunately that they consist of multiple domains, reducing the effectiveness of fuzzy TTLs because the attacker uses the most lucky sample. For a list of domains $d_1, \ldots, d_k$ that were added at the same time with identical clips, then $x \leftarrow \max(x_1, \ldots, x_k)$. Based on our preload list measurements presented in Section 6.2, we expect around 8–13 domains per site available for an attacker to potentially query for. Earlier work found a median of two unique domains out of ten domains in total per website on Alexa top 1M [13].

Fuzzy TTLs are an ineffective mitigation if the attacker just wants to confirm suspected activity with a low base rate, i.e., the mere existence of cached domains anywhere in the network is enough of a signal [42]. Fuzzy TTLs are a plus for websites that are modestly popular in the network, since the attacker has to determine which of several exits with cached domains is the correct one. Having to consider multiple domains and exits (to narrow down the exact time) is more noisy in the network and increases the risk of detection [3]. Attackers may be forced to consider a time-window of several seconds or even minutes, which is a big win for defending against correlation and confirmation attacks [13, 42].

### 5.2 Cover Lookups

The idea of the cover lookups mitigation is to simply inject domains into DNS caches in the Tor network to create false positives. Injected domains must be indistinguishable from domains cached from real Tor user activity. For this, a distribution that models website visits for a particular base rate should be used rather than running, e.g., a deterministic cron job. Further, care has to be taken to capture all predictable domains for each website to defend.

A more drastic mitigation would be to keep a site's domains cached at every exit all the time, e.g., by running `exitmap` [50] every five minutes. This obviously scales poorly. The network overhead would already be significant for a few hundred sites, e.g., estimates based on Alexa top-1k would add about 26.7 requests per second to each exit.

Cover lookups do not scale, even if just injected at few exits probabilistically according to some target base rate. It is a last resort mitigation for site operators that fear that their users are targeted by motivated attackers and where, for some reason, the site cannot transition to being completely (no resources loaded from other domains) hosted as an onion service.

## 6 Redesigning Tor's DNS Cache

To address (timeless) timing attacks in Tor's DNS cache we considered a number of possible smaller changes. All of them failed for different reasons, however. Section 6.1 presents a straw-man design that is helpful to understand *why*, while at the same time being closely related to the properties achieved

by the preload DNS cache design in Section 6.2. Section 6.5 presents an extensive evaluation that answers questions about how feasible and performant our proposal is.

## 6.1 Straw-man Design

We omit all but one straw-man design that is particularly important to understand the proposed redesign in Section 6.2: *simply remove Tor's DNS cache*. If there is no DNS cache to speak of in Tor, it is easy to see that there cannot be any (timeless) timing attacks against Tor's DNS cache (because it does not exist). What these attacks would instead target is the exit's DNS resolver which also has a cache. At a first glance it may seem like an insignificant improvement that just moves the problem elsewhere. This would be the case if every exit used its own dedicated DNS resolver. However, an exit may share a resolver with other exits or most importantly clients outside of the Tor network. A prime example is the resolver of the ISP of the exit. Any inference made from the state of shared resolvers would thus not be directly attributable to activity on the Tor network. This would therefore make *false positives* a reality with regards to if a domain was cached or not as a consequence of activity in the Tor network.

Introducing false positives to the timeless timing attack itself is in general challenging because an answer needs to be available at the same time regardless of there being a cache hit or miss. False negatives may seem easier and could make the attacker discard otherwise correct classifications, e.g., because an attack only works half of the time. However, without false positives, attackers are still able to reliably remove otherwise incorrect classification through confirmation [42]. Because websites typically make use of multiple domain names, defenses that add random delays to responses (to cause false negatives) would need to consistently add similar delays for all relevant domains tied to websites or other user activity the attacker is trying to infer. The semantics surrounding user activity is hard if not impossible to capture at the DNS level. Therefore, all false negative defenses we could think of failed.

Now suppose that Tor has no DNS cache and exits always use a shared resolver that may introduce false positives. A major downside is that performance would take a significant hit due to the lack of a cache in Tor, especially since a shared resolver is likely not running locally, but provided by the ISP or some third-party. It is likely that both page-load latency and resolver load would increase. Worsening performance and especially latency is the opposite of what the Tor project is working towards [10, 34]. Next we show how to get the good properties of not having a DNS cache in Tor (potential for false positives) while improving performance.

## 6.2 The Preload DNS Cache

This is not only a defense against (timeless) timing attacks in the DNS cache, but a complete redesign of Tor's DNS cache. Ultimately, *what we want to achieve is false positives for an attacker trying to determine client activity in the Tor network with the help of DNS*. The only way to achieve this—upon learning that a domain associated with some activity has been looked up—is if there is a possibility that this domain lookup was caused from outside of the Tor network. Therefore, as a starting point, we assume that the Tor Project would strongly encourage exit operators to not run local resolvers dedicated to exits. Instead, exit operators should configure their systems to use their ISP resolvers or use a third-party provider. Greschbach *et al.* [13] investigated the effect of DNS on Tor's anonymity, including resolver configuration, and found that using the ISP's resolver would be preferable.

First remove all of Tor's current DNS caching as in our straw-man design. The preloaded DNS cache instead contains two types of caches: a same-circuit cache and a shared preload cache, see Figure 9. The preloaded cache only contains domains from an allowlist. This allowlist is compiled by a central party (e.g., by the Network Health team in the Tor Project) by visiting popular sites from several different vantage points. The allowed domains are then delivered to exits and continuously resolved to IPs by each exit. During domain resolution on a circuit, the client's lookup first hits the preload cache. If the domain is preloaded, a cache hit is guaranteed regardless of if anyone performed a lookup before. Therefore, it is safe to share this cache across circuits without leaking information about past exit traffic. On a cache miss, the circuit's same-circuit cache is consulted. As the name suggests, this cache is shared for streams on the same circuit but not across different circuits. Due to Tor's circuit isolation, an attacker is unable to probe any other cache than their own. Therefore, (timeless) timing attacks are eliminated (similar to if Tor did not have a DNS cache at all), but without removing the possibility of cache hits.

Including a same-circuit cache in the defense is motivated by Tor's significant same-circuit caching to retain performance, see Figures 4a and 4b in Section 3.3. One can confirm that this is most likely due to Tor Browser opening several concurrent connections by referring to the `network.http.max-persistent-connections-per-proxy` option and/or enabling debug logging[3], observing that multiple streams are often created to the same destination. Note that these destinations are domains and not IPs, and that neither TB nor the client-side Tor process has any notion of a DNS cache to prevent cross-circuit fingerprinting (see Section 2.2). While a hypothetical *per-circuit client-side cache* would be an option, it would per definition not be able to generate cache hits for concurrent resolves (without violating circuit isolation, potentially leading to cross-circuit fingerprinting) and put pressure on exits unless they do the appropriate caching. This is why our design places the same-circuit cache at exits instead of clients.

[3] https://gitlab.torproject.org/tpo/applications/tor-browser/-/wikis/Hacking#debugging-the-tor-browser

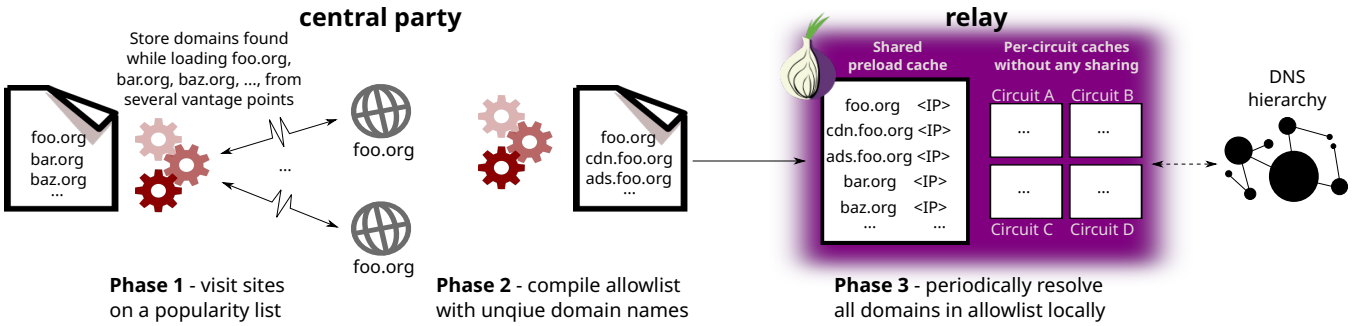



Figure 9: Overview of the preloaded DNS cache design. A central party visits sites on a popularity list from different vantage points to compile an allowlist of domains that each relay keeps preloaded at all times by resolving them continuously. DNS looks-ups start in the shared preload cache and moves on to a dynamic cache that is never shared across circuits on cache misses.

A preload cache is also motivated by performance, however without any of the harmful cross-circuit sharing. The remainder of this section explores the performance impact of compiling an allowlist from popularity lists—like Alexa [2], Cisco Umbrella [7], and Tranco [25]—by comparing the resulting cache-hit ratios to baseline Tor today. The preloaded DNS cache is inspired by RFC 8767 [24] which allows resolvers to serve stale data in some cases (see Section 7). Here, exits keep domains on a preloaded allowlist fresh on a best-effort level, serving stale records if necessary. Upon shutdown, exits could persist IPs in the preload cache to disk as a starting point on startup. Upon startup, if the preload cache have yet to be populated with IPs, simply treat lookups as cache misses. We discuss periodic refresh overhead further in Section 6.5.3.

## 6.3 Data Collection

As part of understanding Tor's DNS cache (Section 3) we also collected data to be able to evaluate the performance of the preload design. In particular, we evaluate different popularity lists, the impact on cache-hit ratio, estimated DNS cache size, and how these change over time.

Central to the preload design is domain popularity lists. We included the Alexa list [2] because that is what Mani *et al.* showed to be accurate for Tor [28], the newer Tranco list because it may be more accurate [25], and the Cisco Umbrella list because it also contains "non-web" domains [7].

In addition to considering the popularity lists, we also created *extended* lists from Alexa and Tranco by visiting each domain on those lists using the Chromium browser and recording all requests for additional domains. We repeated this three times from Europe, twice from the US, and twice from Hong Kong by using a popular VPN service. Each visit was given a timeout of 20 seconds. No pruning of the resulting extended lists of domains was done. Much can likely be done to make these lists of domains significantly more comprehensive (e.g., by considering popular subpages that might contain domains not on the front-page of websites) and smaller (e.g., by pruning unique tracking domains: in one of our biggest lists, `*.safeframe.googlesyndication.com` makes up 8% of domains with unique tracking subdomains with no value for caching). Another direction to explore that could result in lists that are smaller and/or more comprehensive would be to tailor them specifically for relays in certain regions. For example, website visits from Europe may be treated differently by website operators due to regulations like the GDPR. (In other words, there could be differences with regards to *domains*—not to be confused with IPs that each relay already resolves locally—that are encountered during website visits.)

Based on the regular and extended popularity lists, we made several lists from top-10 up to and including top-10,000 in increments. Further, the weekend before each of the *first four weeks* of data collection (see Section 3), we downloaded fresh popularity lists (Fridays) and generated new extended lists (Saturdays and Sundays). We generated in total $4*20*5=400$ lists: for the first four weeks, 20 lists each for {Alexa, Tranco, Umbrella, extended Alexa, extended Tranco}.

Our data collection involving the lists took place in three phases. The first phase consisted of the first four weeks with increasingly more lists, which was followed by two weeks of analysis of our results and dialog with the Tor Research Safety Board. This lead us to the third and final phase of data collection where we excluded the vast majority of lists, focusing only on getting extended data for about eleven more weeks on the most informative and useful lists (see Section 6.5).

## 6.4 Further Ethical Considerations

We discussed the preload additions as part of our other data collection, received feedback from the Tor Research Safety Board, and passed our university's ethical review process.

Our rationale for why including counters for preload lists is safe was as follows. We collect counters of aggregate lookups that would have been cache-hits on each list over 15 minutes. Except for the top-10 lists (non-extended), all other lists contain in the order of 100–1,000 unique domains aggregated into a single counter. The main harm associated with the dataset is if they enable an attacker to *rule out* that a particular web-

site or Tor-user activity took place at our exits (see following paragraph). So, little to no zero counters in our data is what we set out to achieve. As an additional safety precaution our exits only have a 0.1% exit probability, further making any zero counter less useful.

Let us look more closely at the potential harm. For websites, the results of Mani *et al.* [28] tell an attacker to expect a power-law-like distribution of website popularity in the network. As discussed in Section 3.1, we expect on average about 725 website visits to each exit per 15 minute period. This is *the prior of an attacker* wishing to perform confirmation or correlation attacks. Most of the visits should be to popular websites (per definition) and if the dataset allows an attacker to rule such visits out it may cause harm because it is useful information to the attacker [42]. Because of this, we grouped our lists into intervals of 100s (for top-?00) and 1000s (for top-?000). We stopped at top-10k because we estimated little utility of caching domains of even less popular websites. Further, to illustrate when the type of data we collect can be harmful, the results of Mani *et al.* [28] and Pulls and Dahlberg [42] tell us that at some point the logic becomes flipped in terms of attacker utility: confirming that it was possible that a visit took place to a *rarely visited website* is useful. The popularity (i.e., network base rate) of websites is central. We set out to only collect data on the most popular of websites/domains, so for us, the focus is on when the attacker can rule out website visits or some user activity: an attacker already expects that popular websites/domains are visited.

We analyzed the 1,330,400 sample counters we collected over the first four weeks for different popularity lists. We found 33 zero counters. All of them belonged to Alexa top-10 lists from different weeks! Besides Alexa top-10, the next list with the lowest counter was Tranco top-100 from 20 May with 39 hits. Finding empty counters for Alexa top-10 was at first very surprising, because the list contains the most popular websites on the web (e.g., from 20 May: `google.com`, `youtube.com`, `baidu.com`, `facebook.com`, `instagram.com`, `bilibili.com`, `qq.com`, `wikipedia.org`, `amazon.com`, and `yahoo.com`). However, note how the actual *domains* on the list (of *websites*) do not contain the www prefix nor any other popular subdomain associated with the sites. This highlights how poor the regular non-extended lists are at capturing actual *website* traffic. We can further see this for both Alexa and Tranco in Figure 10, presented next in Section 6.5.1. Even the top-10k lists have very low cache-hit ratios.

By comparing a list with a more popular list (which should be a strict subset) and observing the same counter value it is also possible to infer that *likely* no visits took place to the unique domains on the less popular list. (This could happen by chance though.) We found 16,055 (1.2%) such samples: 5,073 to top-10k lists, 3,703 to top-[1k,10k) lists, and 7,279 to top-[200,1k) lists. None of them were to top-100 lists. This might seem alarming at first glance, but taking a closer look at the lists we find that only 135 of the samples were to extended lists (77 to x-Tranco top-10k, the highest rank list was x-Tranco top-600 with one sample). Further, only five of the samples belonged to a list from Umbrella. The remaining 15,915 samples were to the regular (non-extended) Alexa and Tranco lists. This is of limited use to attackers for popular domains, because while the lists capture popular *websites*, our dataset contains counters of *aggregate domain lookups*. An inferred zero counter *does not mean that no visits took place* to *websites* for the *non-extended* lists. For example, if you enter `www.google.com` or `www.wikipedia.org` into Tor Browser, neither `google.com` nor `wikipedia.org` are actually connected to. The recursive resolver of the exit may perform the lookup, but Tor will not, so it is not captured in our dataset for the non-extended lists. The extended lists, due to being generated from actual website visits, include domains typically connected to by Tor Browser. Another example is users visiting direct links to websites and not entering the domain manually in the browser, such as when following links from search engines or sent through social media.

When designing our measurements the above detail was not considered. We included the regular popularity lists for sake of comparison. Ideally the non-extended lists would have been effective sources for preload lists. This was evidently not the case for Alexa and Tranco (see later results), but was the case for Umbrella. So while what we did learn helped us understand the value of using extended lists to improve cache hits, in hindsight we could have come to the same conclusion without the same granularity for non-extended lists.

In the second phase of our data collection (see Section 6.3), we discussed the above detail with the Tor Research Safety Board and concluded to stop collecting data for (non-extended) Alexa and Tranco, and to minimize the lists for future collection to those necessary to determine the longevity of potentially useful preload lists (based on our findings). Out of an abundance of caution, we will only share the collected counters for non-extended Alexa and Tranco lists with researchers for research purposes (the rest of the data is public). The counters collected during the second phase were consistent with the data from the first phase.

During the third phase of data collection, we limited the collection to extended Tranco top-{10, 100, 1k, 2k, 4k, 5k, 7k, 10k} lists and the Umbrella top-10k list, all from April 29. The goal was to learn how cache hits get worse over time with old lists. Out of 141,624 sample counters collected, three were zero and 59 were relatively zero when compared to the more popular list.

## 6.5 Performance Evaluation

The goal of our evaluation is to determine over time: cache-hit ratio of potential preload lists (Section 6.5.1), memory usage at exits (Section 6.5.2), and resolver load (Section 6.5.3).

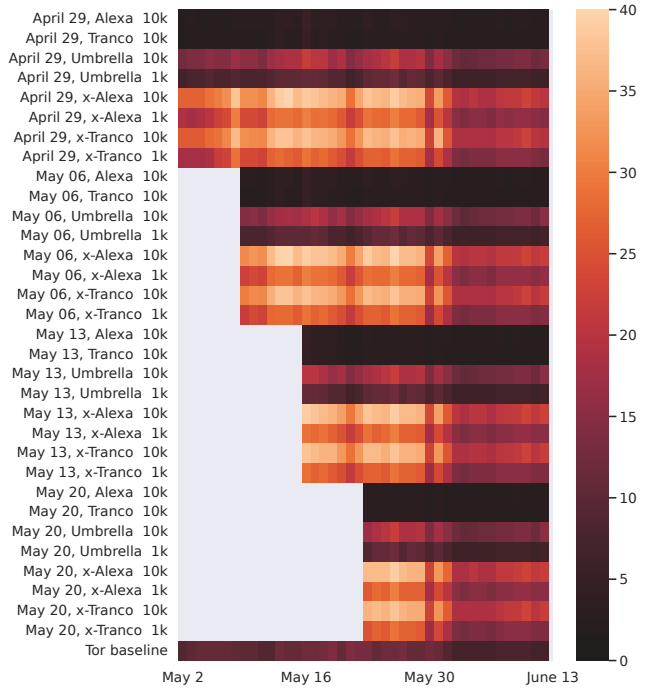
### 6.5.1 Results: Preload Lists

Our dataset is extensive with 2,498,424 sample counters from 400 popularity lists spanning about four months. Figure 10 shows comprehensive heatmaps of shared cross-circuit cache-hit ratios for the web (Figure 10a) and permissive (Figure 10b) exits over the first six weeks of data collection (first and second phases). Cache-hit ratios are medians (very similar to the mean) for 24h periods. In each figure, the groupings of the four weeks when we added new lists are visible (top to bottom), as well as baseline Tor at the bottom row for sake of comparison. Note how the regular Alexa and Tranco top-10k lists perform poorly: the two black ($< 5\%$ cache-hit ratio) lines at the top of each grouping. Even Umbrella 1k is better, with Umbrella 10k being comparable to baseline Tor. The extended lists clearly improve over baseline Tor, with the extended 10k-lists even reaching over 30% cross-circuit cache-hit ratios some days. Look at how the lists change over time: we see no real difference between lists generated at end of April and those generated during May, but consistent changes across all lists over time, likely due to varying traffic at the exits. The differences between using Alexa or Tranco to generate extended lists are negligible, so we focus on Tranco for the remainder of this analysis as it is open, maintained, and a more recent source of website popularity [25].

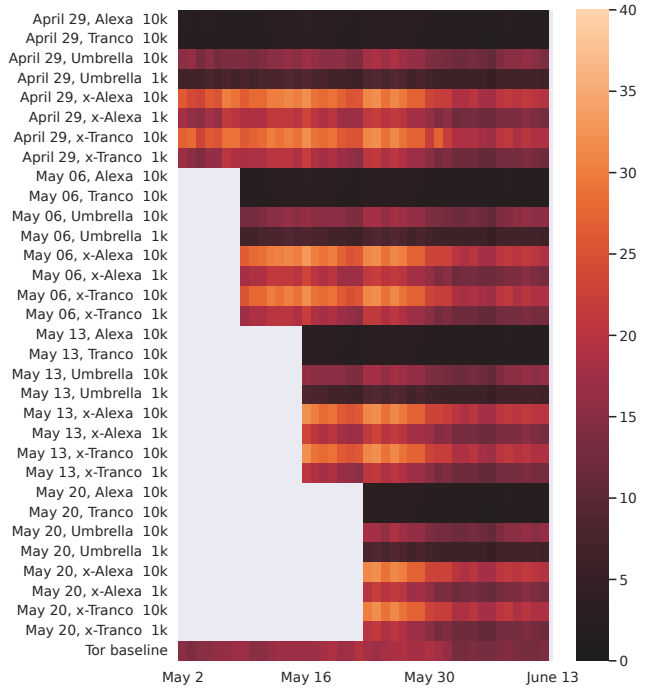
Figure 11 shows the observed *cross-circuit* cache-hit ratios for eight different extended Tranco lists, Umbrella top-10k, and Tor baseline. We used lists from the end of April because they have the most data. As a baseline, Tor's current DNS cache has a mean cache-hit ratio of 11% for web and 17% for permissive. In terms of different popularity lists, the regular (non-extended) Tranco and Alexa lists are ineffective: the top-10k lists are regularly below 5% for web and permissive (see Figure 10). Umbrella top-10k does much better with mean 17% (web) and 16% (permissive). This is slightly worse (permissive) and comparable (web) to baseline Tor.

The extended lists show a further improvement, comparable in terms of *average* (full duration of lists) cross-circuit cache-hit ratios to baseline Tor at top-200 for Alexa and Tranco for web and at top-700 for permissive. The extended lists from top-1k get (depending on which of the compiled extended Tranco lists) 20–24% (web) and 15–18% (permissive) and up to 27–32% (web) and 22–27% (permissive) at 10k. There is very little gain between top-7k and top-10k. In general, the extended lists do relatively worse on the permissive exit and the Tor baseline is higher: this makes sense, since Alexa and Tranco are focused on websites. This is further confirmed by Umbrella doing better as a more general-purpose domain popularity list.

Note that Figure 11 shows the cross-circuit cache-hit ratios for a selection of the very first preload lists we created on the April 29. The results are very encouraging: time seems to have only a slight detrimental impact on cache hits. After four months the larger extended lists show a noticable performance

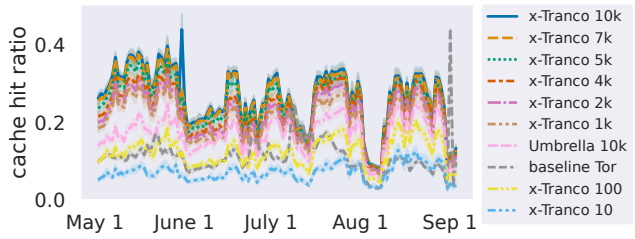



(a) web

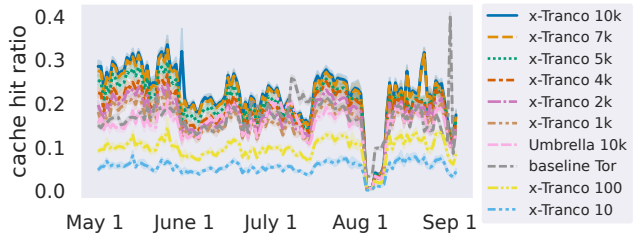



(b) permissive

Figure 10: Shared cross-circuit cache-hit ratios (%) for selected preload lists during the first six weeks (x-axis) of data collection. The plotted values are medians over 24h, and dates on the y-axis show the date of original list download.

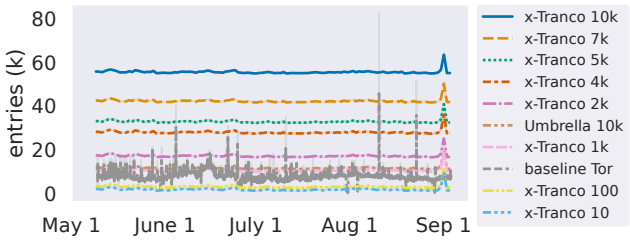



(a) web

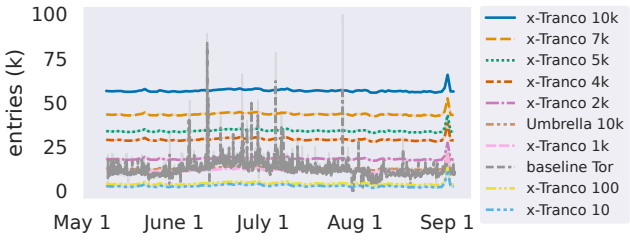



(b) permissive

Figure 11: Shared cross-circuit cache-hit ratios for eight different extended Tranco lists, Umbrella top-10k, and Tor baseline during four months in 2022.




(a) web




(b) permissive

Figure 12: Estimated cache entries for eight different extended Tranco lists, Umbrella top-10k, and Tor baseline.

improvement over baseline, with the exception of an odd spike in baseline in early September (we speculate that this is DDoS-related). The robustness of preload lists removes one of the main downsides of the preload design, i.e., to maintain and deliver a current list to exits. It is likely feasible to ship hard-coded preload lists as part of regular Tor releases and still improve performance, assuming that exit operators upgrade their software a couple of times per year.

### 6.5.2 Results: Cache Entries

Figure 12 shows the number of cache entries needed in Tor as-is ("baseline Tor") and for the preload design for a range of different popularity lists. We can accurately estimate an upper bound because we collected the total number of entires in all same-circuit caches as part of our measurements. This count is an upper bound, because some of those entries would have already been cached in the preload cache. The popularity lists have static sizes, and to be an accurate upper bound we used the largest observed size for each list over the four weeks.

Starting with the same-circuit cache, look at the line for extended Tranco top-10 ("x-Tranco 10") in Figure 12: this extended list contains only 90 entries, so the lines at the bottom show mostly the number of entries used by the same circuit cache. The size of the same-circuit caches should be proportional to the number of open circuits, and therefore follow exit probability. Based on the data from Figure 12, we do not suspect this to be a significant burden. It would be trivial to cap the size and/or prune the size as part of OOM-management, or dropping entries based on their age would probably have little impact on performance (presumably most value is at the start of the circuit when most streams are attached).

Recall from Section 3.3 and Figures 3a and 3b that the permissive exit had a mean of 12,130 entries compared to the web exit's 7,672 mean. We see the same figures for the baseline in Figure 12. Albeit slightly higher on average for the web exit but more stable, we see that Umbrella 10k as well as extended Tranco top-1k are about the same as Tor baseline. So with about the same memory usage as now the preload design would offer slightly (permissive) or noticeably (web) better cache-hit ratios. Looking at the top-2k up until top-10k extended lists we see a significant higher memory usage (only slightly sublinear) but that comes with significantly higher cache-hit ratios, as seen in Figure 11. In absolute terms, for extended Tranco top-10k, about 60,000 cache entries—even if pessimistically assuming 1 KiB per entry—would end up using about 60 MiB of memory for the cache. Since domains can be at most 255 bytes and most domains are much shorter, one could clearly implement the cache more memory-efficiently. Also, as mentioned earlier, it is likely possible to reduce the size of the extended top-10k lists by removing useless tracking domains. Further note that the memory needed to cache the preload list—unlike the same-circuit cache—only depends on the size of the list, not the number circuits or streams created at the exit.

### 6.5.3 Results: Resolver Load

In general, on the one hand, improving cache-hit ratios will reduce resolver load and scale well with increased traffic. On the other hand, continuously refreshing domains on the preload

list increases resolver load. Consider the mean number of lookups at the web exit, 17,529, and its mean/median cache-hit ratio of 0.80 (see Section 3). This implies an expected $3.9 \leftarrow \frac{17529(1-0.80)}{15 \cdot 60}$ requests per second to the exit's resolver. For the permissive exit we observed about 7.8 requests per second. As a source of comparison, Sonntag [45, 46] reports for a DNS resolver dedicated to a 200 Mbit/s exit in 2018 an average of 18.5 requests per second.

The resolver load for the different preload lists should be proportional to the estimated number of cache entries shown in Figure 12. The estimated load for an extended top-1k list would be similar to current Tor, while the extended top-10k list would see about a seven-fold increase without changes. This may or may not be problem. Given the variability of lookups we observed throughout our data collection (Figure 2) and reported by Sonntag, resolvers are clearly capable of dealing with increased loads. Requests due to the preload list should be predictable, consistent, and cheap in terms of bandwidth even for a low-capacity exit.

Regardless, the load on resolvers could be lowered by reducing the number of domains, e.g., the increased cache-hit ratio from top-7k to top-10k is very small (≈1%) for a 20–30% increase in entries. One could also increase the internal TTLs, i.e., the frequency of refreshing the entries in the preload cache. In Tor, this is especially practical since circuits use random exits. In the rare case of stale data causing issues, simply create a fresh circuit. Serving stale data is not uncommon in DNS [24], further discussed next in Section 7.

## 7 Related Work

Van Goethem *et al.* [48] originally proposed timeless timing attacks, showing significant improvements against HTTP/2 web servers, Tor onion services, and EAP-pwd. All timeless timing attacks exploit concurrent processing, e.g., in HTTP/2, by filling buffers at the relay closest to an onion service, or packing two authentication requests in EAP-pwd into the same RadSec (TLS over TCP) packet. The latter was the inspiration for our timeless timing attack on Tor's DNS cache, i.e., packing two RESOLVE cells into a single TLS record.

There has been a long body of work on how to safely perform measurements of the Tor network [11, 12, 18, 27], laying the foundation for safely performing large-scale measurements [19, 28]. Our timeless timing attack enables anyone to do network-wide measurements for exact domains on specific exits with a precision of at least one second. This is highly invasive and a useful resource to deanonymize Tor-users, discussed further shortly. Our network measurements to inform the design of defenses have been focused around the DNS in Tor. Similar to other related work (see below), we focused on how to make those limited measurements safe; not on how to broadly perform a much wider range of measurements safely.

Greschbach *et al.* [13] investigated the effect of DNS on Tor's anonymity. They quantified the use of DNS resolvers in the network, the impact of choice of resolver on correlation and confirmation attacks, and how to incorporate observed DNS traffic with website fingerprinting attacks [5, 16, 17, 26, 32, 47] to make improved correlation attacks. In their construction, DNS traffic is used to either reduce the number of websites to consider during classification or to confirm classification. A key observation was that Tor, due to a bug, clipped all TTLs to 60 seconds. This was resolved and lead to the current approach of clipping to 300 or 3,600 seconds. One of our short-time mitigations update these clips to be fuzzy.

Greschbach *et al.* [13] also measured DNS requests from an exit for both web and a more permissive exit policy in 2016. The collection was done by observing DNS requests to the exit's resolver and aggregating results into five-minute buckets. Similarly, we aggregate over time in 15-minute buckets and do not directly collect resolved domains. They found a small difference between exit policies, with the permissive exit having slightly fewer (3% smaller median) lookups. Our results are very different: the permissive exit policy resulted in significantly more (double the median) lookups.

Pulls and Dahlberg [42] generalized the classifier confirmation attack of Greschbach *et al.* [13] into a new security notion for website fingerprinting attacks, and further explored the use of DNS. They showed that timing attacks were possible in Tor's DNS cache, performing network-wide measurements on a domain under their control with a true positive rate of 17.3% when attempting to minimize false positives. We use a similar method for measurements, but our attack is significantly better with a 100% true positive rate and no false positives at all.

Sonntag collected hourly data from the resolver of an exit during five months in 2018 [45, 46]. In terms of frequency, they noted about 18.5 requests per second, with a peak of 291,472 requests in an hour. The average is higher than ours (3.9 and 7.8 requests per second) while the peak significantly smaller (1,183,275 requests in 15 minutes). Sonntag also analyzed the actual domains looked up, including categorization (porn, social network, shopping, advertisement etc). We do not collect domains; only cache-hits as part of popularity lists by aggregating domains into buckets like top-100, top-1k, etc.

Mani *et al.* [28] used PrivCount [18] and PSC [12] to safely make extensive network-wide measurements of the Tor network. They measured, e.g., circuits, streams, destination ports, and exit domains at exits, as well as client connections, churn, composition, and diversity at clients. Their exit probability ranged between 1.5–2.2%, compared to our peak of 0.1%. While our data is much more limited and targeted around DNS, there are two interesting comparisons to consider:

- Mani *et al.* observed 2.1 billion exit streams inferred in the network every 24 hours. Extrapolating on our lookup statistics we have an average of 6.3 billion lookups, which corresponds to the number of streams[4]. This sug-

[4] Streams either do a lookup with RELAY_BEGIN or are closed after a

gests a significant increase ($\approx 3x$) in the number of streams in the Tor network since 2018.

- Mani *et al.* measured the frequency of how well the primary domain on a circuit matched the Alexa top-one-million list. We transform their reported relative counts and compare it to the relative count of average lookups in the same intervals in our dataset for top-10k, shown in Figure 13. Note that this only uses data from phase one of our data collection. Broadly, we see that their results show significantly more traffic to top-10 than any of the lists we use. That said, our data supports one of Mani *et al.*'s conclusion that the popularity lists are reasonably accurate representations of traffic from the Tor network.

The relatively recent RFC 8767 [24] allows for DNS data to be served "stale", i.e., after expiry according to its TTL, in the exceptional circumstance that a recursive resolver is unable to refresh the information. In case data goes stale, RFC 8767 suggests serving it for at most one to three days. The background of RFC 8767 aptly motivates this with the saying that *"stale bread is better than no bread"*. In addition to serving potentially stale data, modern resolvers like Unbound [31] further support prefetching: preemptively refreshing domains in the cache before TTL expiry. These measures all serve to improved reliability and have been found to be used for sake of resiliency [29]. Tor already clips TTLs, in a sense serving stale data for the vast majority of domains. Our preload design takes this further by introducing continuous prefetching of domains on a fixed allowlist.

Two decades ago, Jung *et al.* [22] found that cache-hit ratios on the order of 80–87% are achieved if a resolver has ten or more clients and TTLs are at least ten minutes. More recently Hao and Wang [15] reported that 100k cached entries are required to achieve a baseline of 86% cache-hits for a first-come first-serve cache in a university network. Their dataset had similar characteristics to a DNS trace collected for an ISP resolver by Chen *et al.* [4] with regards to *disposable domains* that are never requested more than once in the long-tail of DNS; out of the 11% of domains that are not disposable, 5% and 30% of them have cache-hit ratios of at least 95% and 80% respectively. It appears that fewer disposable domains are resolved in Tor because the observed cache sizes are not large enough for 89% unique lookups. Achieving an 80% cache-hit ratio with a cache of 10k entries does not seem to be an outlier.

## 8 Conclusion

Our timeless timing attack on Tor's DNS cache is virtually perfect, significantly improving over earlier timing attacks [42]. Based on 12 million measurements in the live Tor network, we only observed a 0.00025 failure rate due to vanished circuits and other transient networking errors that are easy to account for. We responsibly disclosed the attack to the Tor Project and coordinated the process around defenses with them.

Our proposed mitigations are just that—mitigations—and do not completely address the underlying issues. The fuzzy TTLs mitigation primarily addresses confirmation with WF attacks involving moderately popular domains. Cover lookups, while valuable if done, does not scale and requires continuous efforts that are not easily automated on a large scale.

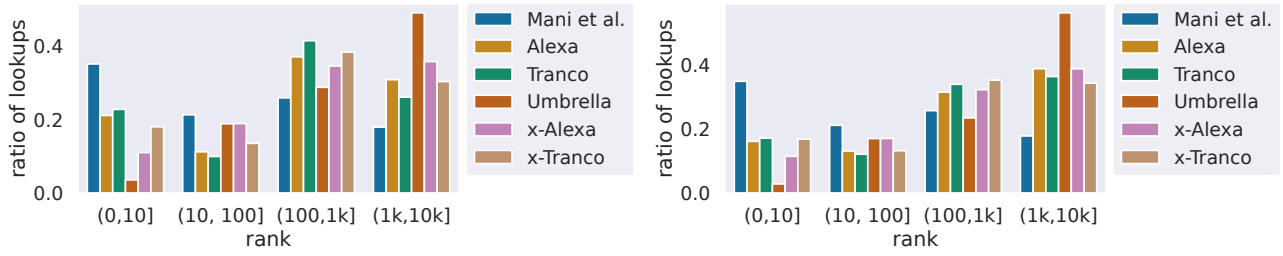
Setting out to find long-term solutions, we landed in redesigning Tor's DNS cache completely with a preload design. To inform the design and to evaluate its feasibility, we ran a four-month experiment starting in May 2022 measuring key performance metrics. To ensure that our measurements were safe, we repeatedly consulted the Tor Research Safety Board and completed our university ethical review process. We received positive feedback as well as invaluable suggestions along the way to minimize any potential harm to the Tor network and its users.

First, the preload design is immune to timing and timeless attacks due to never sharing any data in the DNS cache injected due to user activity across circuits. Secondly, the preload lists of domains based on extended Alexa, extended Tranco, and Cisco Umbrella all show impressive cache-hit ratios. Depending on list, it is possible to get comparable cache-hit ratios, memory usage, and resolver load as Tor today. More extensive lists can trade modest increases in memory and resolver load with significantly higher cache-hit ratios, especially for web traffic. Important future work is improving how the extended lists are generated—e.g., by tailoring them specifically for relays in certain regions (location sensitivity), excluding unique tracking domains, or crawling websites to discover subdomains—which is likely to lead to higher cache-hit ratios and smaller lists.

One of the biggest downsides of the preload design is that the most effective preload lists are extended lists based on Alexa or Tranco, requiring continuous efforts to update. Fortunately, our measurements show that even four-month-old extended lists remain effective with significant improvement over baseline Tor. It is likely feasible for the Tor Project to generate and ship hard-coded preload lists as part of regular Tor releases and still improve performance compared to today.

Like Mani *et al.* [28], we see that traffic in the Tor network appears to reasonably match website/domain popularity lists like Alexa, Tranco, and Umbrella. This is fundamental for the preload design, and likely also a contributing factor for the observed long stability of the extended preload lists, since the most popular sites see relatively little churn [36]. Finally, our measurements indicate that the Tor network has grown by about 300% in terms of number of streams since 2018, and that the large majority of Tor's current DNS caching is a privacy harm rather than a cross-circuit performance boost.

RELAY_RESOLVE cell. Timeout and retries are possible on resolver failure, but the way we measure hides those extra lookups.




(a) web

(b) permissive

Figure 13: Comparison of *relative popularity* of popularity rankings with the results of Mani *et al.* [28].

## Acknowledgments

Many thanks to Georg Koppen and Marc Juarez for engaging with us in continuous Tor Research Safety Board discussions, as well as Elias Rudberg, Johan Nilsson, and Linus Nordberg who operated our modified Tor relays at the premises of DFRI. We would further like to thank our shepherd, the anonymous reviewers, Mike Perry, Nick Mathewson, Paul Syverson, and Roger Dingledine for their valuable feedback. The authors were supported by Mullvad VPN, the Swedish Foundation for Strategic Research, and the Swedish Internet Foundation.

## Availability

We make the following three artifacts available:

1. Patches to Tor, associated scripts and data, and documentation for performing timeless timing attacks.

2. The measurement data from our two exits, a detailed timeline of operations, scripts for creating extended preload lists, and associated Python scripts for parsing all stats and generating figures. Sharing of the dataset was discussed as part of the contact with the Tor Research Safety Board and our university ethical review process. Relevant parts of our research safety board contact are included in our artifact.

3. Contributions to the Tor Project, including source code and associated tooling for our Fuzzy TTLs mitigation and preload defense.

See https://gitlab.torproject.org/rgdd/ttapd/-/tree/main/artifact to locate our artifacts and this paper.

## References

[1] Nadhem J. AlFardan and Kenneth G. Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols. In *IEEE S&P*, 2013.

[2] Amazon. Alexa top 1 million. https://www.alexa.com/, 2022.

[3] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In *TCC*, 2007.

[4] Yizheng Chen, Manos Antonakakis, Roberto Perdisci, Yacin Nadji, David Dagon, and Wenke Lee. DNS noise: Measuring the pervasiveness of disposable domains in modern DNS traffic. In *IEEE/IFIP DSN*, 2014.

[5] Heyning Cheng and Ron Avnur. Traffic analysis of SSL encrypted web browsing. *Project paper, University of Berkeley*, 1998.

[6] Giovanni Cherubin, Rob Jansen, and Carmela Troncoso. Online website fingerprinting: Evaluating website fingerprinting attacks on tor in the real world. In *USENIX Security*, 2022.

[7] Cisco. Umbrella popularity list. https://umbrella-static.s3-us-west-1.amazonaws.com/index.html, 2022.

[8] Scott A. Crosby, Dan S. Wallach, and Rudolf H. Riedi. Opportunities and limits of remote timing attacks. *ACM Trans. Inf. Syst. Secur.*, 12(3), 2009.

[9] Debajyoti Das, Sebastian Meiser, Esfandiar Mohammadi, and Aniket Kate. Anonymity trilemma: Strong anonymity, low bandwidth overhead, low latency—choose two. In *IEEE S&P*, 2018.

[10] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In *USENIX Security*, 2004.

[11] Tariq Elahi, George Danezis, and Ian Goldberg. PrivEx: Private collection of traffic statistics for anonymous communication networks. In *CCS*, 2014.

[12] Ellis Fenske, Akshaya Mani, Aaron Johnson, and Micah Sherr. Distributed measurement with private set-union cardinality. In *CCS*, 2017.

[13] Benjamin Greschbach, Tobias Pulls, Laura M. Roberts, Philipp Winter, and Nick Feamster. The effect of DNS on Tor's anonymity. In *NDSS*, 2017.

[14] Gustavo Gus. Responding to Tor censorship in Russia. https://blog.torproject.org/tor-censorship-in-russia/, 2021.

[15] Shuai Hao and Haining Wang. Exploring domain name based features on the effectiveness of DNS caching. *Comput. Commun. Rev.*, 47(1), 2017.

[16] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *CCSW*, 2009.

[17] Andrew Hintz. Fingerprinting websites using traffic analysis. In *PETS*, 2002.

[18] Rob Jansen and Aaron Johnson. Safely measuring Tor. In *CCS*, 2016.

[19] Rob Jansen, Matthew Traudt, and Nicholas Hopper. Privacy-preserving dynamic learning of Tor network traffic. In *CCS*, 2018.

[20] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul F. Syverson. Users get routed: traffic correlation on Tor by realistic adversaries. In *CCS*, 2013.

[21] Marc Juárez, Sadia Afroz, Gunes Acar, Claudia Díaz, and Rachel Greenstadt. A critical evaluation of website fingerprinting attacks. In *CCS*, 2014.

[22] Jaeyeon Jung, Emil Sit, Hari Balakrishnan, and Robert Tappan Morris. DNS performance and the effectiveness of caching. *IEEE/ACM Trans. Netw.*, 10(5), 2002.

[23] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *CRYPTO*, 1996.

[24] David C Lawrence, Warren Kumari, and Puneet Sood. Serving stale data to improve DNS resiliency. RFC 8767, 2020.

[25] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. Tranco: A research-oriented top sites ranking hardened against manipulation. In *NDSS*, 2019.

[26] Marc Liberatore and Brian Neil Levine. Inferring the source of encrypted HTTP connections. In *CCS*, 2006.

[27] Akshaya Mani and Micah Sherr. Historε: Differentially private and robust statistics collection for Tor. In *NDSS*, 2017.

[28] Akshaya Mani, T. Wilson-Brown, Rob Jansen, Aaron Johnson, and Micah Sherr. Understanding Tor usage with privacy-preserving measurement. In *IMC*, 2018.

[29] Giovane C. M. Moura, John S. Heidemann, Moritz Müller, Ricardo de Oliveira Schmidt, and Marco Davids. When the dike breaks: Dissecting DNS defenses during DDoS. In *IMC*, 2018.

[30] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. Deepcorr: Strong flow correlation attacks on Tor using deep learning. In *CCS*, 2018.

[31] NLnet Labs. Serving stale data—unbound 1.14.0 documentation. https://unbound.docs.nlnetlabs.nl/en/latest/topics/serve-stale.html, 2022.

[32] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. Website fingerprinting in onion routing based anonymization networks. In *WPES*, 2011.

[33] Mike Perry. A critique of website traffic fingerprinting attacks. https://blog.torproject.org/critique-website-traffic-fingerprinting-attacks, 2013.

[34] Mike Perry. Congestion control arrives in Tor 0.4.7-stable! https://blog.torproject.org/congestion-contrl-047/, 2022.

[35] Mike Perry, Erinn Clark, Steven Murdoch, and Georg Koppen. The design and implementation of the Tor Browser [draft]. https://2019.www.torproject.org/projects/torbrowser/design/, 2018.

[36] Victor Le Pochat, Tom van Goethem, and Wouter Joosen. Evaluating the long-term effects of parameters on the characteristics of the Tranco top sites ranking. In *USENIX Security*, 2019.

[37] Tor Project. Tor Project status. https://web.archive.org/web/20220906145324/https://status.torproject.org/, 2022.

[38] Tor Project. Tor source code. https://gitlab.torproject.org/tpo/core/tor/-/blob/tor-0.4.7.7/src/feature/relay/dns.c#L600-689, 2022.

[39] Tor Project. Tor source code. https://gitlab.torproject.org/tpo/core/tor/-/blob/tor-0.4.7.7/src/lib/tls/buffers_tls.c#L43-100, 2022.

[40] Tor Project. Tor source code. https://gitlab.torproject.org/tpo/core/tor/-/blob/tor-0.4.7.7/src/core/or/connection_or.c#L2361-2426, 2022.

[41] Tor Project. Tor source code. https://gitlab.torproject.org/tpo/core/tor/-/blob/tor-0.4.7.7/src/feature/relay/dns.c#L725-732, 2022.

[42] Tobias Pulls and Rasmus Dahlberg. Website fingerprinting with website oracles. *PETS*, 2020.

[43] Sandra Siby, Marc Juárez, Claudia Díaz, Narseo Vallina-Rodriguez, and Carmela Troncoso. Encrypted DNS -> privacy? A traffic analysis perspective. In *NDSS*, 2020.

[44] Al Smith. Tor and the humans who use it. https://blog.torproject.org/tor-and-the-humans-who-use-it/, 2021.

[45] Michael Sonntag. DNS traffic of a Tor exit node—an analysis. In *SpaCCS*, 2018.

[46] Michael Sonntag. Malicious DNS traffic in Tor: Analysis and countermeasures. In *ICISSP*, 2019.

[47] Qixiang Sun, Daniel R. Simon, Yi-Min Wang, Wilf Russell, Venkata N. Padmanabhan, and Lili Qiu. Statistical identification of encrypted web browsing traffic. In *IEEE S&P*, 2002.

[48] Tom van Goethem, Christina Pöpper, Wouter Joosen, and Mathy Vanhoef. Timeless timing attacks: Exploiting concurrency to leak secrets over remote connections. In *USENIX Security*, 2020.

[49] Tao Wang and Ian Goldberg. On realistically attacking Tor with website fingerprinting. *PETS*, 2016(4), 2016.

[50] Philipp Winter, Richard Köwer, Martin Mulazzani, Markus Huber, Sebastian Schrittwieser, Stefan Lindskog, and Edgar R. Weippl. Sopoiled onions: Exposing malicious Tor exit relays. In *PETS*, 2014.