On Certificate Transparency Verification and Unlinkability of Websites Visited by Tor Users

Rasmus Dahlberg
Department of Mathematics and Computer Science

Abstract

Certificate Transparency is an ecosystem of logs, monitors, and auditors that hold certificate authorities accountable while issuing certificates. We show how the amount of trust that TLS clients and domain owners need to place in Certificate Transparency can be reduced, both in the context of existing gradual deployments and the largely unexplored area of Tor. Our contributions include improved third-party monitoring, a gossip protocol plugging into Certificate Transparency over DNS, an incrementally deployable gossip-audit model tailored for Tor Browser, and using certificates with onion addresses. The methods used range from proof sketches to Internet measurements and prototype evaluations. An essential part of our evaluation in Tor is to assess how the protocols used during website visits—such as requesting an inclusion proof from a Certificate Transparency log—affect unlinkability between senders and receivers. We find that most false positives in website fingerprinting attacks can be eliminated for all but the most frequently visited sites. This is because the destination anonymity set can be reduced due to how Internet protocols work: communication is observable and often involves third-party interactions. Some of the used protocols can further be subject to side-channel analysis. For example, we show that remote (timeless) timing attacks against Tor's DNS cache reliably reveal the timing of past exit traffic. The severity and practicality of our extension to website fingerprinting pose threats to the anonymity provided by Tor. We conclude that access to a so-called website oracle should be an assumed attacker capability when evaluating website fingerprinting defenses.

Keywords: Auditing, Certificate Transparency, DNS, Gossip, Side-Channels, Timing Attacks, Tor, Tor Browser, Website Fingerprinting, Website Oracles

Kring verifiering av Certificate Transparency samt länkbarhet av Tor-användare och besökta webbplatser

Rasmus Dahlberg
Institutionen för matematik och datavetenskap

Sammanfattning

Projektet *Certificate Transparency* är ett ekosystem av loggar, övervakare och granskare som håller certifikatutfärdare till svars för utfärdade webbcertifikat. Vi visar hur säkerheten kan höjas i ekosystemet för både domäninnehavare och TLS-klienter i nuvarande system samt som del av anonymitetsnätverket Tor. Bland våra större bidrag är förbättrad övervakning av loggarna, ett skvallerprotokoll som integrerats med DNS, ett skvaller- och granskningsprotokoll som utformats specifikt för Tors webbläsare och ett förslag på hur domännamn med adresser i Tor kan bli mer tillgängliga. De metoder som använts varierar från säkerhetsbevis till internetmätningar och utvärderingar av forskningsprototyper. En viktig del av vår utvärdering i Tor är att avgöra hur protokoll som används av webbläsare påverkar möjligheten att koppla ihop användare med besökta webbplatser. Detta inkluderar existerande protokoll samt nya tillägg för att verifiera om webbplatsers certifikat är transparensloggade. Våra resultat visar att i många fall kan falska positiva utslag filtreras bort vid mönsterigenkänning av Tor-användares krypterade trafik (eng: *website fingerprinting*). Orsaken är att besök till de flesta webbplatser kan uteslutas till följd av hur internetprotokoll fungerar: kommunikation är observerbar och involverar ofta interaktioner med tredjeparter. Vissa protokoll har dessutom sidokanaler som kan analyseras. Vi visar exempelvis att Tors DNS-cache kan undersökas med olika varianter av tidtagningsattacker. Dessa attacker är enkla att utföra över internet och avslöjar vilka domännamn som slagits upp vid angivna tidpunkter. De förbättrade mönsterigenkänningsattackerna mot webbplatser är realistiska och hotar därför Tors anonymitet. Vår slutsats är att framtida försvar bör utvärderas utifrån att angripare har tillgång till ett så kallat webbplatsorakel.

Nyckelord: Granskning, Certificate Transparency, DNS, Skvaller, Sidokanaler, Tidtagningsattacker, Tor, Tors webbläsare, Mönsterigenkänning, Webbplatsorakel

Acknowledgements

I am fortunate to be surrounded by people that helped me reach this milestone in one piece. First, I want to acknowledge that my brother, Victor, provided me with unlimited tutoring before my PhD studies. I would not be on my current path without him. Second, Sarah and Andreas, thank you for challenging and supporting me. You have been a determinantal part of my personal growth and well-being. Finally, Mom, thank you for being around whenever I need it.

Several doors were opened for me at Karlstad University. Stefan Alfredsson and Stefan Lindskog kick-started my undergraduate studies with programming feedback, increased-pace study plans, and the opportunity to be involved in the department. Tobias Pulls introduced me to computer security research. We have worked closely ever since, and *I strongly recommend him as an advisor and collaborator*. Simone Fischer-Hübner, Stefan Lindskog, Leonardo Martucci, and Tobias Pulls all provided sincere advice during my PhD studies. Many other individuals selflessly helped me forward. Some of them include Ala Sarah Alaqra, Linus Nordberg, Jenni Reuben, Fredrik Strömberg, and Paul Syverson.

I am further grateful for my collaborators: Matthew Finkel, Toke Høiland-Jørgensen, Andreas Kassler, Linus Nordberg, Tobias Pulls, Tom Ritter, Paul Syverson, and Jonathan Vestin. They are all driven individuals that I learned a lot from. I also appreciate the generous funding received from the Swedish Knowledge Foundation and the Swedish Foundation for Strategic Research.

Karlstad University , May 2, 2023 Rasmus Dahlberg

List of Acronyms

The introductory summary refrains from using acronyms to make it easily digested. However, a few acronyms are used without definition because their full versions are likely less familiar to most readers. These acronyms are:

DNS	Domain Name System
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
IP	Internet Protocol
P4	Programming Protocol-independent Packet Processors
RFC	Request For Comments
RIPE	Réseaux IP Européens
RSA	Rivest Shamir Adleman
TCP	Transmission Control Protocol
TLS	Transport Layer Security
XDP	eXpress Data Path

List of Appended Papers

I. **Rasmus Dahlberg** and Tobias Pulls. Verifiable Light-Weight Monitoring for Certificate Transparency Logs. NordSec (2018).

II. **Rasmus Dahlberg**, Tobias Pulls, Jonathan Vestin, Toke Høiland-Jørgensen, and Andreas Kassler. Aggregation-Based Certificate Transparency Gossip. SECURWARE (2019).

III. **Rasmus Dahlberg**, Tobias Pulls, Tom Ritter, and Paul Syverson. Privacy-Preserving & Incrementally-Deployable Support for Certificate Transparency in Tor. PETS (2021).

IV. **Rasmus Dahlberg**, Paul Syverson, Linus Nordberg, and Matthew Finkel. Sauteed Onions: Transparent Associations from Domain Names to Onion Addresses. WPES (2022).

V. Tobias Pulls and **Rasmus Dahlberg**. Website Fingerprinting with Website Oracles. PETS (2020).

VI. **Rasmus Dahlberg** and Tobias Pulls. Timeless Timing Attacks and Preload Defenses in Tor's DNS Cache. USENIX Security (2023).

Comments on my Participation

Paper I I had the initial idea and conducted most of the work myself. Tobias mainly contributed with discussions that lead to the final design.

Paper II Andreas and Tobias had the initial idea of exploring the intersection between Certificate Transparency and programmable packet processors. I did most of the design and writing with feedback from Tobias, our RIPE Atlas measurements, and our performance benchmarks with Jonathan and Toke.

Paper III I had the initial idea and was the main driver to move the work forward, first in discussion with Tobias and then together with Tom and Paul.

Paper IV Paul, Linus, and I had the initial idea of exploring how onion addresses fit into Certificate Transparency. Paul and I did most of the writing. I implemented our monitor, Linus our search engine, Matt our web extension.

Paper V Tobias is the main author and conducted most of the work. I mainly contributed by coining the name *website oracle*, evaluating sources of real-world website oracles, and performing our non-simulated network experiments.

Paper VI Tobias and I collaborated closely from start to finish with the following exceptions. I did most implementation work. Volunteers from DFRI—a Swedish non-profit and non-partisan organization that promotes digital rights—operated our exit relays. Tobias did most DNS cache data analysis. Tobias also had the initial idea, which was refined with feedback from Roger Dingledine.

List of Other Contributions

Throughout and before my PhD studies, I also contributed to the following:

- The Sigsum Project. https://www.sigsum.org/ (2020-present).

  Sigsum is a free and open-source software project that brings transparency to signed checksums. The design is simple and uses a proactive gossip-audit model. I founded Sigsum with Linus Nordberg and Fredrik Strömberg while working at Mullvad VPN as a side-line occupation.

- Tobias Pulls and **Rasmus Dahlberg**. Steady: A Simple End-to-End Secure Logging System. NordSec (2018).

  Tobias was the main driver. I mostly contributed with design discussions and a security formalization.

- **Rasmus Dahlberg**, Tobias Pulls, and Roel Peeters. Efficient Sparse Merkle Trees: Caching Strategies and Secure (Non-)Membership Proofs. NordSec (2016).

  This work started with my Bachelor's thesis. I did most writing with input from Tobias and Roel. Tobias did our benchmarking experiments.

- **Rasmus Dahlberg** and Tobias Pulls. Standardized Syslog Processing: Revisiting Secure Reliable Data Transfer and Message Compression. Technical Report, Karlstad University (2016).

  I was the main driver. Tobias contributed with continuous feedback.

Contents

Introductory Summary

1 Introduction

The security posture of the Internet increased significantly throughout the last decade. For example, the cleaned-up and formally verified TLS 1.3 protocol that underpins HTTPS has been rolled-out gradually [44], the certificates that specify which public keys to use when bootstrapping a secure connection can be obtained for free and automatically [1], and web browsers have shifted from positive to negative security indicators in favor of security-by-default [109]. The use of end-to-end encryption has further become the norm with services such as DNS-over-HTTPS [43], virtual private networks [27], Tor [24], and secure messaging [101] gaining traction. In other words, the era of attackers that can passively snoop and actively tamper with unencrypted network traffic is over.

What will remain the same is the incentive for attackers to snoop and tamper with network traffic. Therefore, the focus is (and will likely continue to be) on circumventing protocols that add security and privacy as they are deployed in the real world. For example, there is a long history of certificate mis-issuance that allows attackers to impersonate websites and thus insert themselves as machines-in-the-middle ("MitM") without actually breaking TLS [17, 96]. Or, in the case of encrypted channels that are hard to intercept, instead analyzing traffic patterns to infer user activity like which website is being visited [14, 40, 41, 58, 75, 102]. The bad news is that attackers only need to find one vulnerability in a deployed protocol or its software. Sometimes, such vulnerabilities can be purchased by zero-day brokers like Zerodium [114].

To address an attack vector, it is common to add countermeasures that frustrate attackers and/or increase the risk involved while trying to exploit a system. A good example is how the certificate authority ecosystem evolved. For background, certificate authorities are trusted parties that validate domain names before issuing certificates that list their public keys. Web browsers are shipped with hundreds of trusted certificate authorities, which means that the resulting TLS connections cannot be more secure than the difficulty of hijacking the weakest-link certificate authority [17]. A proposal eventually deployed to mitigate this issue is Certificate Transparency: an ecosystem of public append-only logs that publishes all certificates so that any mis-issuance can be *detected* by monitors [54, 55]. These logs have a cryptographic foundation that holds them and the issuing certificate authorities *accountable*, at least in theory. In practice, the logs are essentially trusted parties that must act honestly due to how web browsers shape their policies to respect user privacy [3, 33, 64, 97].

The first objective of this thesis is to better understand the current limits of Certificate Transparency by proposing and evaluating improvements which *reduce* the amount of trust that needs to be placed in third-party monitors and logs. We make a dent in the problem of Certificate Transparency verification both generally and concretely in the context of Tor Browser, which unlike Google Chrome and Apple's Safari does not support Certificate Transparency yet. For context, Tor Browser is a fork of Mozilla's Firefox that (among other things) routes user traffic through the low-latency anonymity network Tor [24, 77]. As part of our pursuit to improve the status quo for Certificate

Transparency verification in Tor Browser, the second objective of this thesis is to evaluate how the protocols used during website visits affect unlinkability between senders (web browsers) and receivers (websites). Our evaluation applies to our addition of Certificate Transparency and other protocols already in use, e.g., DNS, real-time bidding [110], and certificate revocation checking [85].

The remainder of the introductory summary is structured as follows. Section 2 introduces background that will help the reader understand the context and preliminaries of the appended papers. Section 3 defines our research questions and overall objective. Section 4 provides an overview of our research methods. Section 5 describes our contributions succinctly. Section 6 summarizes the appended papers that are published in NordSec (Paper I), SECURWARE (Paper II), PETS (Paper III and V), WPES (Paper IV), and USENIX Security (Paper VI). Section 7 positions our contributions with regard to related work. Section 8 concludes and briefly discusses future work.

2 Background

This section introduces background on Certificate Transparency and Tor.

2.1 Certificate Transparency

The web's public-key infrastructure depends on certificate authorities to issue certificates that map domain names to public keys. For example, the certificate of `www.example.com` is issued by DigiCert and lists a 2048-bit RSA key [87]. The fact that DigiCert signed this certificate means that they claim to have verified that the requesting party is really `www.example.com`, typically by first ensuring that a specified DNS record can be uploaded on request [11]. If all certificate authorities performed these checks correctly and the checks themselves were fool-proof, a user's browser could be sure that any certificate signed by a certificate authority would list a verified public key that can be used for authentication when connecting to a website via TLS. Unfortunately, there are hundreds of trusted certificate authorities and a long history of issues surrounding their operations in practice [8, 17, 96]. One of the most famous incidents took place in 2011: an attacker managed to mis-issue certificates from DigiNotar to intercept traffic towards Google and others in Iran [45]. The astonishing part is that this incident was first detected *seven weeks later*.

Certificate Transparency aims to facilitate *detection* of issued certificates, thus holding certificate authorities *accountable* for any certificates that they mis-issue [54, 55]. The basic idea is shown in Figure 1. In addition to regular validation rules, browsers ensure certificates are included in a public append-only Certificate Transparency *log*. This allows anyone to get a concise view of all certificates that users may encounter, including domain owners like Google who can then see for themselves whether any of the published certificates are mis-issued. The parties inspecting the logs are called *monitors*. Some monitors mirror all log entries [86], while others discard most of them in pursuit of

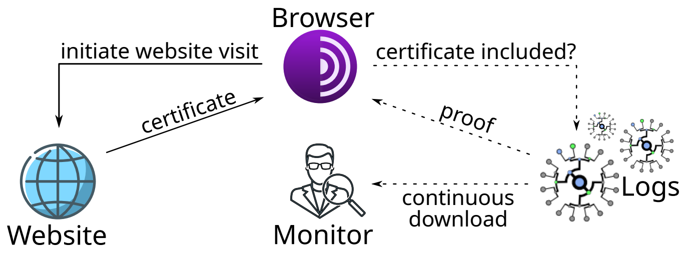


Figure 1: The idea of Certificate Transparency. Certificates encountered by users must be included in a public log so that monitors can detect mis-issuance.

finding matches for pre-defined criteria like `*.example.com` [95]. Another option is subscribing to certificate notifications from a trusted third-party [34].

What makes Certificate Transparency a significant improvement compared to the certificate authority ecosystem is that the logs stand on a cryptographic foundation that can be verified. A log can be viewed as an append-only tamper-evident list of certificates. It is efficient[1] to prove cryptographically that a certificate is in the list, and that a current version of the list is append-only with regard to a previous version (i.e., no tampering or reordering).[2] These properties follow from using a Merkle tree structure that supports *inclusion* and *consistency* proofs [19, 28, 55, 67]. The reader only needs to know that these proofs are used to reconstruct a log's Merkle tree head, often referred to as a *root hash*. It is a cryptographic hash identifying a list of certificates uniquely in a tree data structure. The logs sign root hashes with the number of entries and a timestamp to form *signed tree heads*. So, if an inconsistency is discovered, it cannot be denied. Log operators are therefore held accountable for maintaining the append-only property. A party that verifies the efficient transparency log proofs without downloading all the logs is called an *auditor*.

A log that signs two inconsistent tree heads is said to perform a *split-view*. To ensure that everyone observes the same append-only logs, all participants of the Certificate Transparency ecosystem must engage in a *gossip protocol* [16, 72]. In other words, just because Alice observes an append-only log, it is not necessarily the *same append-only log* that Bob observes. Therefore, Alice and Bob must exchange signed tree heads and verify consistency to assert that the log operators play by the rules and only append certificates. Without a secure gossip protocol, log operators would have to be trusted blindly (much like certificate authorities before Certificate Transparency). RFC 6962 defers the specification of gossip [55], with little or no meaningful gossip deployed yet.

Rolling out Certificate Transparency without breakage on the web is a challenge [98]. Certificates must be logged, associated proofs delivered to end-user software, and more. One solution RFC 6962 ultimately put forth was the introduction of *signed certificate timestamps*. A signed certificate timestamp is a log's *promise* that a certificate will be appended to the log within a *maximum*

[1]Efficient refers to space-time complexity $O(\log(n))$, where n is the number of log entries.
[2]Interested readers can refer to our Merkle tree and proof technique introduction online [21].

merge delay (typically 24 hours). Verifying if a log holds its promise is usually called *auditing*. Certificate authorities can obtain signed certificate timestamps and embed them in their final certificates by logging a *pre-certificate*. As such, there is no added latency from building the underlying Merkle tree and no need for server software to be updated (as the final certificate contains the information needed). The current policy for Google Chrome and Apple's Safari is to reject certificates with fewer than two signed certificate timestamps [3, 33]. How to request an inclusion proof for a promise without leaking the user's browsing history to the log is an open problem [64]. In other words, asking for an inclusion proof trivially reveals the certificate of interest to the log.

Other than embedding signed certificate timestamps in certificates, they can be delivered dynamically to end-users in TLS extensions and stapled certificate status responses. For example, Cloudflare uses the TLS extension delivery method to recover from log incidents without their customers needing to acquire new certificates [100]. Several log incidents have already happened in the past, ranging from split-views [6, 92, 93] to broken promises of timely logging [29, 37, 5, 91] and potential key compromise [84]. These are all good *scares* motivating continued completion of Certificate Transparency in practice.

In summary, the status quo is for web browsers to require at least two signed certificate timestamps before accepting a certificate as valid. Merkle tree proofs are not verified. Gossip is not deployed. The lack of a reliable *gossip-audit model* means that the logs are largely trusted parties.[3] We defer discussion of related work in the area of gossip-audit models until Section 7.

2.2 Tor

The Tor Project is a 501(c)(3) US nonprofit that advances human rights and defends privacy online through free software and open networks [79]. Some of the maintained and developed components include Tor Browser and Tor's relay software. Thousands of volunteers operate relays as part of the Tor network, which routes the traffic of millions of daily users with low latency [61]. This frustrates attackers like Internet service providers that may try linking *who is communicating with whom* from their local (non-global) vantage points [24].

Usage of Tor involves tunneling the TCP traffic of different destinations (such as all flows associated with a website visit to `example.com`) in fixed-size *cells* on independent *circuits*. A circuit is built through a guard, a middle, and an exit relay. At each hop of the circuit, one layer of symmetric encryption is peeled off. The used keys are ephemeral and discarded together with all other circuit state after at most 10 minutes (the maximum circuit lifetime). This setup allows guard relays to observe users' IP addresses but none of the destination traffic. In contrast, exit relays can observe destination traffic but no user IP addresses. The relays used in a circuit are determined by Tor's end-user software. Such path selection is randomized and bandwidth-weighted but starts

[3]Historical remark: the lack of verification led Google to require that all certificates be disclosed in at least one of their logs to validate [97]. The so-called *one-Google log requirement* was recently replaced. Google Chrome instead interacts with Google's trusted auditor. See Section 7.

with a largely static guard set to protect users from *eventually* entering the network from a relay an attacker volunteered to run.

Tor's *consensus* lists the relays that make up the network. As the name suggests, it is a document agreed upon by a majority of trusted *directory authorities*. Five votes are currently needed to reach a consensus. Examples of information added to the Tor consensus include tunable network parameters and uploaded relay descriptors with relevant metadata, e.g., public key, available bandwidth, and exit policy. Each relay in the consensus is also assigned different flags based on their configuration and observed performance, e.g., `Guard`, `MiddleOnly`, `Fast`, `Stable`, and `HSDir`. The latter means that the relay is a *hidden service directory*, which refers to being part of a distributed hash table that helps users lookup *onion service introduction points*.

An onion service is a self-authenticated server identified by its public key. Onion services are only reachable through the Tor network. Users that are aware of a server's *onion address* can consult the distributed hash table to find its introduction points. To establish a connection, a user builds a circuit to a *rendezvous point*. A request is then sent to one of the current introduction points, which informs the onion service that it may build its own circuit to meet the user at their rendezvous point. In total, six relays are traversed while interacting with an onion service. This setup allows not only the sender but also the receiver to be anonymous. The receiver also benefits from a large degree of censorship resistance as the server location may be hidden. The main drawback of onion services is that their non-mnemonic names are hard to discover and remember. Some sites try to overcome this by setting their onion addresses in *onion location* HTTP headers or HTML attributes [80].

Many users use Tor Browser to connect to the Tor network. In addition to routing traffic as described above, Tor Browser ships with privacy-preserving features like first-party isolation to not share any state across different origins, settings that frustrate browser fingerprinting, and *disk-avoidance* to not store browsing-related history as well as other identifying information to disk [77]. Tor Browser is a fork of Mozilla's Firefox. Unfortunately, neither Firefox nor Tor Browser supports any form of Certificate Transparency. Conducting undetected machine-in-the-middle attacks against Tor users is thus relatively straightforward: compromise or coerce the weakest-link certificate authority, then volunteer to operate an exit relay and intercept network traffic. Such interception has previously been found with self-signed certificates [113].

While global attackers are not within Tor's threat model, it is in scope to guard against various local attacks [24]. For example, the intended attacker may passively observe a small fraction of the network and actively inject their own packets. Figure 2 shows the typical attacker setting of *website fingerprinting*, where the attacker observes a user's entry traffic with the goal of inferring which website was visited solely based on analyzing encrypted traffic [14, 40, 41, 58, 75, 102]. Website fingerprinting attacks are evaluated in the *open-world* or *closed-world* settings. In the closed-world setting, the attacker monitors (not to be confused with Certificate Transparency monitoring) a fixed list of websites. A user visits one of the monitored sites, and the attacker needs to

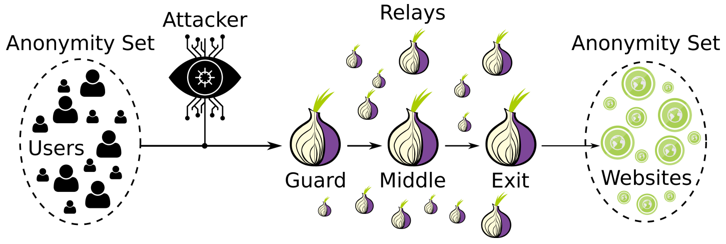


Figure 2: The setting of a website fingerprinting attack. A local passive attacker analyzes a user's encrypted network traffic as it enters the network. The goal is to infer which website is visited. (Figure reprinted from Paper V.)

determine which one. The open-world setting is the same as the closed-world setting, except that the user may also visit unmonitored sites. The practicality of website fingerprinting attacks is up for debate, e.g., ranging from challenges handling false positives to machine-learning dataset drift [15, 47, 76, 111].

In summary, Tor is a low-latency anonymity network often accessed with Tor Browser. Among the threats that Tor aims to protect against are local attackers that see traffic as it enters or leaves the network (but not both at the same time all the time). A website fingerprinting attack is an example of a passive attack that operates on entry traffic. A machine-in-the-middle attack is an example of an active attack that typically operates on exit traffic. Discussion of related work in the area of website fingerprinting is deferred until Section 7.

3 Research Questions

The overall research objective spans two different areas: transparency logs and low-latency anonymity networks. We aim to reduce trust assumptions in transparency log solutions and to apply such solutions in anonymous settings for improved security and privacy. We defined the following research questions to make this concrete in Certificate Transparency and Tor, the two ecosystems with the most history and dominant positions in their respective areas.

1. *Can trust requirements in Certificate Transparency be reduced in practice?*

   Transparency logs have a cryptographic foundation that supports efficient verification of inclusion and consistency proofs. Such proofs are useful to reduce the amount of trust that is placed in the logs. The roll-out of Certificate Transparency has yet to start using these proofs, and to employ a gossip protocol that ensures the same append-only logs are observed. Part of the challenge relates to privacy concerns as parties interact with each other, as well as deploying gradually without breakage.

   We seek practical solutions that reduce the trust requirements currently placed in the logs and third-party monitors while preserving user privacy.

2. *How can authentication of websites be improved in the context of Tor?*

   Tor Browser has yet to support Certificate Transparency to facilitate detection of hijacked websites. This includes HTTPS sites but also onion services that may be easier to discover reliably with more transparency.

   We seek incremental uses of Certificate Transparency in Tor that preserve user privacy while engaging in new verification protocols to reduce trust.

3. *How do the protocols used during website visits affect unlinkability between Tor users and their destination websites?*

   Several third-parties become aware of a user's browsing activities while a website is visited. For example, DNS resolvers and certificate status responders may be consulted for domain name resolution and verification of if a certificate has been revoked. Fetching an inclusion proof from a Certificate Transparency log would reveal the same type of information.

   We seek to explore how unlinkability between Tor users and their exit destinations is affected by the multitude of protocols used during website visits. The considered setting is the same as in website fingerprinting, except that the attacker may take additional passive and active measures. For example, the attacker may volunteer to run a Certificate Transparency log (passive) or inject carefully-crafted packets into Tor (active).

4 Research Methods

We tackle privacy and security problems in the field of computer science [23, 26]. Our work is applied, following the scientific method for security and experimental networking research. *Exactly* what it means to use the scientific method in these areas is up for debate [7, 39]. However, at a glance, it is about forming precise and consistent theories with falsifiable predictions *as in other sciences* except that the objects of study are *information systems in the real world*.

A prerequisite to formulating precise, consistent, and falsifiable theories is that there are few implicit assumptions. Therefore, scientific security research should be accompanied by definitions of security goals and attacker capabilities: what does it mean that the system is secure, and what is the attacker (not) allowed to do while attacking it [51]? Being explicit about the overall *setting* and *threat model* is prevalent in formal security work like cryptography, where an abstract (mathematical) model is used to show that security can be *proved* by reducing to a computationally hard problem (like integer factorization) or a property of some primitive (like the collision resistance of a hash function) [50]. It is nevertheless just as crucial in less formal work that deals with security of systems in the real (natural) world—the exclusive setting of the scientific method—which usually lends itself towards break-and-fix cycles in light of new observations. Where to draw the line between *security work* and *security research* is not trivial. However, a few common *failures* of past "security research" include not bringing observations in contact with theory, not making claims and assumptions explicit, or simply relying on unfalsifiable claims [39].

While deductive approaches (like formal reduction proofs) are instrumental in managing complexity and gaining confidence in different models, more than these approaches are required as a model's *instantiation* must also be secure [51]. It is common to complement abstract modeling with *real-world measurements* as well as *systems prototyping and evaluations* [7]. Real-world measurements measure properties of deployed systems like the Internet, the web, and the Tor network. For example, a hypothesis in a real-world measurement could be that (non-)Tor users browse according to the same website popularity distribution. Sometimes these measurements involve the use of research prototypes, or the research prototypes themselves become the objects of study to investigate properties of selected system parts (say, whether a packet processor with new features is indistinguishable from some baseline as active network attackers adaptively inject packets of their choosing). If it is infeasible, expensive, or unsafe (see below) to study a real-world system, a simulation may be studied instead. The downside of simulation is that the model used may not be a good approximation of the natural world, similar to formal cryptographic modeling.

The appended papers use all of the above approaches to make claims about security, privacy, and performance in different systems, sometimes with regard to an abstract model that can be used as a foundation in the natural world to manage complexity. Paper I contains a reduction proof sketch to show reliance on standard cryptographic assumptions. Paper V extends past simulation setups to show the impact of an added attacker capability. Meanwhile, Paper III models part of the Tor network with mathematical formulas to estimate performance overhead. All but Paper IV contain real-world measurements relating to Internet infrastructure, websites, certificates, Tor, or practical deployability of our proposals. All but Paper III contain research prototypes with associated evaluations, e.g., performance profiling, as well as corroborating or refuting our security definitions in experimental settings. All papers include discussions of security and privacy properties as well as their limitations and strengths in the chosen settings (where assumptions are explicit and threat models motivated).

Throughout our experiments, we strived to follow best practices like documenting the used setups, making datasets and associated tooling available, reducing potential biases by performing repeated measurements from multiple different vantage points, and discussing potential biases (or lack thereof) [7]. We also interacted with Tor's research safety board [81] to discuss the ethics and safety of our measurements in Paper VI, and refrained from measuring real (i.e., non-synthetic) usage of Tor whenever possible (Papers III and V). Finally, the uncovered bugs and vulnerabilities in Papers V–VI were responsibly disclosed to the Tor project. This included suggestions on how to move forward.

5 Contributions

The main contributions of this thesis are listed below. An overview of how they relate to our research questions and appended papers is shown in Figure 3.

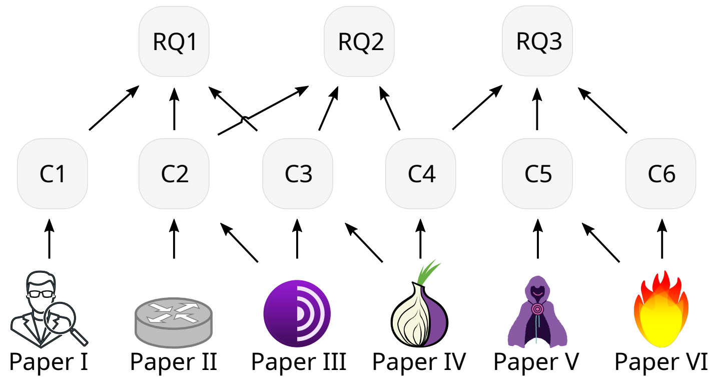


Figure 3: Overview of appended papers, contributions, and research questions.

1. *Reduced trust in third-party monitoring with a signed tree head extension that shifts trust from non-cryptographic certificate notifications to a log's gossip-audit model (or if such a model does not exist yet, the logs themselves).*

   Paper I applies existing cryptographic techniques for constructing static and lexicographically ordered Merkle trees so that certificates can be wild-card filtered on subject alternative names with (non-)membership proofs. This building block is evaluated in the context of Certificate Transparency, including a security sketch and performance benchmarks.

2. *Increased probability of split-view detection by proposing gossip protocols that disseminate signed tree heads without bidirectional communication.*

   Paper II explores aggregation of signed tree heads at line speed in programmable packet processors, facilitating consistency proof verification on the level of an entire autonomous system. Such verification can be indistinguishable from an autonomous system without any split-view detection to achieve herd immunity, i.e., protection without aggregation. Aggregation at 32 autonomous systems can protect 30-50% of the IPv4 space. Paper III explores signed tree heads in Tor's consensus. To reliably perform an undetected split-view against log clients that have Tor in their trust root, a log must collude with a majority of directory authorities.

3. *Improved detectability of website hijacks targeting Tor Browser by proposing privacy-preserving and gradual roll-outs of Certificate Transparency in Tor.*

   Paper III explores adoption of Certificate Transparency in Tor Browser with signed certificate timestamps as a starting point, then leveraging the decentralized network of relays to cross-log certificates before ultimately verifying inclusion proofs against a single view in Tor's consensus. The design is probabilistically secure with tunable parameters that result in modest overheads. Paper IV shows that Certificate Transparency logging of domain names with associated onion addresses helps provide forward censorship-resistance and detection of unwanted onion associations.

4. *An extension of the attacker model for website fingerprinting that provides attackers with the capability of querying a website oracle.*

   A website oracle reveals whether a monitored website was (not) visited by any network user during a specific time frame. Paper V defines and simulates website fingerprinting attacks with website oracles, showing that most false positives can be eliminated for all but the most frequently visited websites. A dozen sources of real-world website oracles follow from the protocols used during website visits. We enumerate and classify those sources based on ease of accessibility, reliability, and coverage. The overall analysis includes several Internet measurements.

5. *Remotely-exploitable probing-attacks on Tor's DNS cache that instantiate a real-world website oracle without any special attacker capabilities or reach.*

   Paper V shows that timing differences in end-to-end response times can be measured to determine whether a domain name is (not) cached by a Tor relay. An estimated true positive rate of 17.3% can be achieved while trying to minimize false positives. Paper VI improves the attack by exploiting timeless timing differences that depend on concurrent processing. The improved attack has no false positives or false negatives. Our proposed bug fixes and mitigations have been merged in Tor.

6. *A complete redesign of Tor's DNS cache that defends against all (timeless) timing attacks while retaining or improving performance compared to today.*

   Paper VI suggests that Tor's DNS cache should only share the same preloaded domain names across different circuits to remove the remotely-probable state that reveals information about past exit traffic. A network measurement with real-world Tor relays shows which popularity lists are good approximations of Tor usage and, thus, appropriate to preload. Cache-hit ratios can be retained or improved compared to today's Tor.

6 Summary of Appended Papers

The appended papers and their contexts are summarized below. Notably, all papers are in publication-date order except that Paper V predates Papers III–IV.

Paper I – Verifiable Light-Weight Monitoring for Certificate Transparency Logs

An often overlooked part of Certificate Transparency is that domain owners are expected to inspect the logs for mis-issued certificates continuously. The cost and required expertise to do so have led to the emergence of third-party monitoring services that notify domain owners of newly issued certificates that they subscribe to. For example, one may subscribe to email notifications whenever a certificate is issued for `*.example.com`. One downside of such third-party monitoring is that these notification services become trusted parties with little or no accountability with regard to omitted certificate notifications. We show how to add this accountability and tie it to the gossip-audit model employed by the Certificate Transparency ecosystem by proposing verifiable light-weight monitoring. The idea is for logs to batch appended certificates into an additional data structure that supports *wild-card (non-)membership proofs*. As a result, third-party monitors can prove cryptographically that they did not omit any certificate notifications selectively. Our experimental performance evaluation shows that overhead can be tuned to be small for all involved parts.

Paper II – Aggregation-Based Certificate Transparency Gossip

Another often overlooked part of Certificate Transparency is that monitors and end-users who browse websites must observe the same append-only logs. For example, if the same append-only logs are not observed, an end-user may connect to a website that serves a mis-issued certificate that no monitor will discover. This would largely defeat the purpose of public logging, which is why RFC 6962 specifies that multiple gossip protocols should be defined separately in the future. We define one such protocol that plugs into the (at the time current) idea of having end-users interact with the logs through DNS. Our work is exploratory, using recent advancements in programmable packet processors that allow turning routers, switches, and network interface cards into *aggregators* of tree heads that the logs signed and transmitted in plaintext via DNS. The aggregated tree heads are then used as a reference while challenging the logs to prove consistency, thus protecting entire vantage points from undetected split views. A different network path (like Tor) can be used to break out of a local vantage point to increase the likelihood of global consistency. If the security definition for *aggregation indistinguishability* is satisfied, vantage points without an aggregator may also receive protection due to herd immunity. Our P4 and XDP prototypes satisfy the notion of aggregation indistinguishability at line-rate with regard to throughput. Prevalent vantage points to roll out aggregation-based gossip include autonomous systems and Internet exchange points that route the traffic of many users. Our RIPE Atlas measurements show that 32 autonomous systems could protect 30-50% of the IPv4 space from undetected split views. End-users merely need to use plaintext DNS for opt-in.

Paper III – Privacy-Preserving & Incrementally-Deployable Support for Certificate Transparency in Tor

One deployment challenge of Certificate Transparency is to ensure that monitors and end-users are engaged in gossip-audit protocols. This is particularly difficult for end-users because such engagement can harm privacy. For example, verifying that a certificate is included by fetching an inclusion proof from a log reveals which website was visited. We propose a gradual roll-out of Certificate Transparency in Tor Browser that preserves privacy *due to* and *how we use* the anonymity network Tor. The complete design holds log operators accountable for certificates they promise to append by having Tor relays fetch inclusion proofs against the same view agreed upon by directory authorities in Tor's consensus. Found issues (if any) are reported to trusted auditors. The incremental design side-steps much of the practical deployment effort by replacing the audit-report pattern with cross-logging of certificates in independent logs, thus assuming that at least one log is honest as opposed to no log in the complete design. All Tor Browser needs to do is verify log signatures and then submit the encountered certificates to randomly selected Tor relays. Such submissions are probabilistic to balance performance against the risk of eventual detection of log misbehavior. Processing of the submitted certificates is also randomized to reduce leakage of real-time browsing patterns, something Tor Browser cannot do on its own due to criteria like disk avoidance and the threat model for wanting Certificate Transparency in the first place. We provide a security sketch and estimate performance overhead based on Internet measurements.

Paper IV – Sauteed Onions: Transparent Associations from Domain Names to Onion Addresses

Many prominent websites are also hosted as Tor onion services. Onion services are identified by their public keys and subject to onion routing, thus offering self-authenticated connections and censorship resistance. However, the non-mnemonic names are a limitation due to being hard to discover and remember. We explore how certificates with onion addresses may improve the status quo by proposing sauteed onions, *transparent associations from domain names to onion addresses* with the help of Certificate Transparency logs. The idea is to extend a website's regular certificate with an associated onion address. This makes it possible to offer certificate-based onion location that is no less targeted than the HTTPS connection facilitating the discovery, as well as name-to-onion search engines that use the append-only logs for verifiable population of their databases. The achieved goals are consistency of available onion associations, improved third-party discovery of onion associations, and forward censorship-resistance. To be discovered, sites must opt-in by obtaining a sauteed onion certificate. Our prototypes for certificate-based onion location and third-party search engines use an existing backward-compatible format. We discuss this trade-off and note that a certificate extension may be used in the future.

Paper V – Website Fingerprinting with Website Oracles

One of the properties Tor aims to provide against local network attackers is unlinkability between end-users (sender anonymity set) and their destinations on the Internet (receiver anonymity set). A website fingerprinting attack aims to break anonymity in this model by inferring which website an identifiable end-user is visiting based only on the traffic entering the Tor network. We extend the attacker model for website fingerprinting attacks by introducing the notion of *website oracles*. A website oracle answers the following question: was website w visited during time frame t? In other words, the attacker can query the receiver anonymity set for websites that were (not) visited. Our simulations show that augmenting past website fingerprinting attacks to include website oracles significantly reduces false positives for all but the most popular websites, e.g., to the order of 10^{-6} for classifications around Alexa top-10k and much less for the long tail of sites. Further, some earlier website fingerprinting defenses are largely ineffective in the (stronger) attacker model that includes website oracles. We discuss a dozen real-world website oracles ranging from centralized access logs to widely accessible real-time bidding platforms and DNS caches, arguing that they are inherent parts of the protocols used to perform website visits. Therefore, access to a website oracle should be an assumed attacker capability when evaluating which website fingerprinting defenses are effective.

Paper VI – Timeless Timing Attacks and Preload Defenses in Tor's DNS Cache

Tor relays cache resolved domains with constant time-to-live values not to reveal information about past exit traffic while boosting performance. We show that this caching strategy and its implementation in the live Tor network can be exploited by a *timeless timing attack* that leaks if a domain is (not) cached. Further, the time that a domain was inserted into the cache can be inferred by repeated probes. Our attack prototype's experimental evaluation in real conditions shows that there are neither false positives nor false negatives (10M repetitions). Thus, it is useful for instantiating a real-world website oracle without requiring any special attacker capabilities or reach (just a modest computer that can create a Tor circuit). One of our mitigations has been merged in Tor: probabilistic time-to-live values that make the time-of-insertion fuzzy. Long-term, Tor's DNS cache could be redesigned to *preload* the same domains at all exits. Such preloading would eliminate all (timeless) timing attacks in Tor's DNS cache because the same domains would always be (un)cached across different circuits. To retain performance within the same circuit, we propose that the preloaded domains should be complemented by a dynamic same-circuit cache that is not shared across circuits. Our four-month-long DNS cache measurement at two 100 Mbit/s exit relays informs on today's baseline performance. It is compared to a preloaded DNS cache based on different variations of three popularity lists: Alexa, Tranco, and Umbrella. A preloaded DNS cache can be as performant as today with similar resource usage or significantly improve cache-hit ratios by 2-3x. However, the increased cache-hit ratios have the cost of modest increases in memory and resolver load.

7 Related Work

This section positions the appended papers with regard to related work. For Certificate Transparency, this includes approaches towards signed certificate timestamp verification, gossip, and the problem of monitoring the logs. The related work with regard to Tor is focused on the practicality of website fingerprinting attacks and prior use of side-channels (such as timing attacks).

7.1 Certificate Transparency Verification

Approaches that fetch inclusion proofs have in common that they should preserve privacy by not revealing the link between users and visited websites. Eskandarian *et al.* mention that Tor could be used to overcome privacy concerns; however, it comes at the cost of added infrastructure requirements [31]. Lueks and Goldberg [59] and Kales *et al.* [49] suggest that logs could provide inclusion proofs using multi-server private information retrieval. This requires a non-collusion assumption while also adding significant overhead. Laurie suggests that users can fetch inclusion proofs via DNS as their resolvers already learned the destination sites [53]. While surveying signed certificate timestamp auditing, Meiklejohn *et al.* point out that Certificate Transparency over DNS may have privacy limitations [64]. For example, the time of domain lookups and inclusion proof queries are detached. Paper II uses Laurie's approach as a premise while proposing a gossip protocol. Paper III applies Certificate Transparency in a context where Tor is not additional infrastructure (Tor Browser).

Several proposals try to avoid inclusion proof fetching altogether. Dirksen *et al.* suggest that all logs could be involved in the issuance of a signed certificate timestamp [25]. This makes it harder to violate maximum merge delays without detection but involves relatively large changes to log deployments. Nordberg *et al.* suggest that signed certificate timestamps can be handed back to the origin web servers on subsequent revisits [72], which has the downside of assuming that machine-in-the-middle attacks eventually cease for detection. Nordberg *et al.* [72] as well as Chase and Meiklejohn [13] suggest that some clients/users could collaborate with a trusted auditor. Stark and Thompson describe how users can opt-in to use Google as a trusted auditor [99]. This approach was recently replaced by opt-out auditing that cross-checks a fraction of signed certificate timestamps with Google using k-anonymity [22]. Henzinger *et al.* show how such k-anonymity can be replaced with a single-server private information retrieval setup that approaches the performance of prior multi-server solutions [38]. None of the latter two proposals provide a solution for privately reporting that a log may have violated its maximum merge delay because the trusted auditor is assumed to know about all signed certificate timestamps. Eskandarian *et al.* show how to prove that a log omitted a certificate privately [31]. However, they use an invalid assumption about today's logs being in strict timestamp order [64]. Paper III suggests that Tor Browser could submit a fraction of signed certificate timestamps to randomly selected Tor relays. These relays perform further auditing on Tor Browser's behalf: much like a trusted auditor, except that no single entity is running it.

Merkle trees fix log content—not promises of logging. Therefore, inclusion proof fetching by users or their trusted parties must be accompanied by consistency verification and gossip to get a complete gossip-audit model [55]. Chuat *et al.* suggest that users and web servers can pool signed tree heads, gossiping about them as they interact [16]. Nordberg *et al.* similarly suggest that users can pollinate signed tree heads as they visit different web servers [72]. Hof and Carle suggest that signed tree heads could be cross-logged to make all logs intertwined [42]. Gunn *et al.* suggest multi-path fetching of signed tree heads [36], which may make persistent split-views hard depending on the used multi-paths. Syta *et al.* suggest that independent witnesses could cosign the logs using threshold signatures [103]. Smaller-scale versions of witness cosigning received attention in industry [18, 65], and generally in other types of transparency logs as well [60]. Larger browser vendors could decide to push the same signed tree heads to their users, as proposed by Sleevi and Messeri [94]. Paper II uses the operators of network vantage points for aggregating and verifying signed tree heads to provide their users with gossip-as-a-service, however assuming plaintext DNS traffic and a sound signed tree head frequency as defined by Nordberg *et al.* [72]. We used the multi-path assumptions of Gunn *et al.* to break out of local vantage points. In contrast, Paper III ensures that the same logs are observed in the Tor network by incorporating signed tree heads into Tor's consensus (thus making directory authorities into witnesses).

Li *et al.* argue that it would be too costly for most domains to run a monitor [57].[4] Similar arguments have been raised before, and lead to alternative data structures that could make monitoring more efficient than today's overhead [30, 66, 104]. Paper I falls into this category, as the root of an additional static lexicographically-ordered Merkle tree is added to a log's signed tree heads to encode batches of included certificates. The downside is that a non-deployed signed tree head extension is assumed [56], as well as a tree head frequency similar to those described by Nordberg *et al.* [72] to get efficiency in practise.

Paper IV uses a Mozilla Firefox web extension to verify embedded signed certificate timestamps in Tor Browser. Such verification is similar to the gradual deployments of Certificate Transparency in other browsers [97, 98], and the starting point to improve upon in Papers II–III. Moreover, the use of Certificate Transparency to associate human-meaningful domain names with non-mnemonic onion addresses (as in Paper IV) is one of many proposals for alternative naming systems and onion search solutions [48, 69, 73, 80, 88, 108].

7.2 Website Fingerprinting and Side-Channels

Several researchers outline how past website fingerprinting attacks have been evaluated in unrealistic conditions [47, 76, 111]. This includes not accounting for the size of the open-world setting, failing to keep false positive rates low enough to be useful, assuming that homepages are browsed one at a time, how to avoid dataset drift, and training classifiers on synthetic network traces. While some of these challenges were addressed [15, 111], the question of how to deal with false positives remains open. Papers V–VI make a significant dent in this problem by providing evidence that the website fingerprinting attacker model could be made *stronger* to capture *realistic real-world capabilities* that eliminate most false positives around Alexa top-10k and the long tail of unpopular sites.

Others have evaluated traffic analysis attacks against Tor beyond the website fingerprinting setting. On one side of the spectrum are end-to-end correlation/confirmation attacks that typically consider a global passive attacker that observes all network traffic [46, 71, 74, 83]. Such strong attackers are not

[4]Whether the third-party monitors in this study misbehaved or not can be questioned [4].

within the scope of Tor [24]. On the other side of the spectrum are local attackers that see a small fraction of the network, typically in a position to observe a user's encrypted entry traffic (Figure 2). Many have studied those *weak attacks* in lab settings where, e.g., advances in deep learning improved the accuracy significantly [63, 82, 90]. Others have focused on improved attacks that are *active* in the Tor network from their own local vantage points [12, 68, 70], which is similar to the techniques in Papers V–VI. Greschbach *et al.* show that an attacker who gains access to (or traffic to [89]) commonly used DNS resolvers like Google's `8.8.8.8` get valuable information to improve both end-to-end correlation and website fingerprinting attacks [35]. Paper V generalizes the attacker capability they uncovered by allowing the attacker to query Tor's receiver anonymity set with a website oracle of time-frame t. It is further shown that it is possible to instantiate such an abstraction in the real-world while *staying within Tor's threat model*. In other words, the attacker is still local but may employ passive and active measures to narrow down the receiver anonymity set. Paper III proposes Certificate Transparency verification that gives log operators website oracle access. Tor's directory authorities tune t.

Website oracles exist because Tor is designed for anonymity—not unobservable communication [78]. The instantiation of a real-world website oracle is either a direct result of observing network flows from the protocols used during website visits, or due to state of these network flows being stored and inferable. Inferring secret system state is widely studied in applied cryptography and hardware architecture [2, 32, 52, 62, 105, 107], where the goal is usually to determine a key, decrypt a ciphertext, forge a message, or similar using side-channels. A side-channel can be local or remote and ranges from analysis of power consumption to cache states and timing differences. There is a long history of remote timing attacks that are practical [9, 10, 20, 112]. A recent improvement in this area that is relevant for Tor is timeless timing attacks, which exploit concurrency and message reordering to eliminate network jitter [106]. Paper V demonstrates a remote timing attack against Tor's DNS cache that achieves up to 17.3% true positive rates while minimizing false positives. Paper VI instead uses a remote timeless timing attack with no false positives, no false negatives, and a small time-frame t. This approaches an ideal website oracle without special attacker capabilities or reach into third-parties.

8 Conclusions and Future Work

Throughout the thesis, we contributed to the understanding of how trust requirements in Certificate Transparency can be reduced. Efficient and reliable monitoring of the logs is easily overlooked. If the broader ecosystem achieves monitoring through third-parties, they should be subject to the same scrutiny as logs. We proposed a solution that makes it hard for third-party monitors to provide subscribers with selective certificate notifications. We also proposed a gossip-audit model that plugs into interacting with the logs over DNS by having programmable packet processors verify that the same append-only logs are observed. Avoiding the addition of complicated verification logic into end-user software is likely a long-term win because it reduces the number of moving parts. In other words, simple gossip-audit models will be much easier to deploy in the wide variety of end-user software that embeds TLS clients.

We also contributed to the understanding of how Certificate Transparency can be applied in the context of Tor Browser. Compared to a regular browser, this results in a different setting with its own challenges and opportunities. On the one hand, Tor Browser benefits from the ability to preserve privacy due to using the anonymity network Tor. On the other hand, data relating to website visits cannot be persisted to disk (such as signed certificate timestamps blocked by maximum merge delays). Our incrementally-deployable proposal keeps the logic in Tor Browser simple by offloading all Certificate Transparency verification to randomly selected Tor relays. The design is complete because mis-issued certificates can eventually reach a trusted auditor who acts on incidents. In addition to proposing Certificate Transparency in Tor Browser, we also explored how certificates with onion addresses may improve the association of domain names with onion addresses. Such certificates ensure domain owners know which onion addresses can be discovered for their sites, much like Certificate Transparency does the same thing for public TLS keys. This also adds censorship resistance to the discovery as logs are append-only.

As part of exploring Certificate Transparency in Tor Browser, we further contributed to the understanding of how the protocols used during website visits affect unlinkability between Tor users and their destination websites. For example, fetching an inclusion proof from a Certificate Transparency log is one such protocol. We extended the attacker model of website fingerprinting attacks with website oracles that reveal whether any network user visited a website during a specific time frame. Our results show that website oracles eliminate most false positives for all but the most frequently visited websites. In addition to the theoretical evaluation of the extended attacker model, we could exploit (timeless) timing attacks in Tor's DNS cache to instantiate real-world website oracles without any special capabilities or reach into third-parties. This led us to contribute to the understanding of how Tor's DNS cache performs today, including a proposal for a performant alternative that preloads the same popular domains on all Tor relays to withstand all (timeless) timing attacks.

As an outlook, our angle on Certificate Transparency verification has mostly been *reactive* for end-users. In other words, some or all certificate verification occurs asynchronously after a website visit. An alternative to this would be upfront delivery of inclusion proofs that reconstruct tree heads which witnesses cosigned; a form of *proactive* gossip as proposed by Syta *et al.* [103]. The significant upside is that the browser's verification could become non-interactive, eliminating privacy concerns and ensuring end-users only see certificates merged into the append-only logs. Investigating what the blockers for such a proposal are in practice—today—would be valuable as log verification quickly becomes complicated with signed certificate timestamps and reactive gossip-audit models. Are these blockers significant? Are they significant over time as other *eventual* changes will be needed, like post-quantum secure certificates? New transparency log applications are unlikely to need the complexity of Certificate Transparency, and should likely not copy something that was designed to fit into an existing system with a large amount of legacy (such as certificate authorities, their established processes for certificate issuance, and the many client-server implementations already deployed on the Internet).

Orthogonal to the verification performed by end-users, contributing to the understanding of how domains (fail to) use Certificate Transparency for detecting mis-issued certificates is largely unexplored. For example, subscribing to email notifications of newly issued certificates becomes less useful in an era where certificates are renewed frequently and automatically. Instead, domain owners need easy-to-use solutions that raise alarms only if there is a problem.

Finally, the mitigation deployed to counter our (timeless) timing attacks in Tor's DNS cache is just that: a mitigation, not a defense, that applies to modestly popular websites but not the long tail where the base rate is low. This is because the attacker's needed website oracle time frame is so large that a fuzzy time-to-live value does nothing. Practical aspects of a preloaded DNS cache need to be explored further before deployment, such as the assumption of a third-party that visits popular domains to assemble an allowlist. We may also have *underestimated* the utility of the existing Umbrella list, which in and of itself does not require any new third-party. Does the use of Umbrella impact page-load latency? Latency is the most crucial parameter to keep minimized. The question is whether frequently looked-up domains are missed or not by skipping the website-visit step, as for the non-extended Alexa and Tranco lists.

More broadly, the question of how to strike a balance between *efficiency* and *effectiveness* of website fingerprinting defenses is open. How much overhead in terms of added latency and/or bandwidth is needed? How much of that overhead is sustainable, both from a user perspective (where, e.g., latency is crucial for web browsing and other interactive activities) and a network health perspective (such as the amount of volunteered relay bandwidth that is wasted)? It is paramount to neither overestimate nor underestimate attacker capabilities, which goes back to the still-debated threat model of website fingerprinting attacks. Regardless of if Tor's DNS cache becomes preloaded or not, it will be difficult to circumvent DNS lookups from happening. Someone—be it a weak attacker like ourselves or a recursive DNS resolver at an Internet service provider—is in a position to narrow down the destination anonymity set. This is especially true when also considering other protocols that reveal information about the destination anonymity set during website visits. Accepting that sources of real-world website oracles are prevalent implies that *the world can be closed*. Therefore, a closed world is more realistic than an open world.

Acknowledgments

I received valuable feedback while writing the introductory summary from Simone Fischer-Hübner Johan Garcia, Stefan Lindskog, and Tobias Pulls. The final draft was further improved with helpful nudges from Grammarly.

References

[1] Josh Aas, Richard Barnes, Benton Case, Zakir Durumeric, Peter Eckersley, Alan Flores-López, J. Alex Halderman, Jacob Hoffman-Andrews, James Kasten, Eric Rescorla, Seth D. Schoen, and Brad Warren. Let's Encrypt: An automated certificate authority to encrypt the entire web. In *CCS*, 2019.

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

[3] Apple Inc. Apple's Certificate Transparency policy. https://support.apple.com/en-us/HT205280, accessed 2023-04-30.

[4] Andrew Ayer. Reliability of monitors | mitigations. https://groups.google.com/a/chromium.org/g/ct-policy/c/zCtQrn_7QK8, accessed 2023-04-30.

[5] Andrew Ayer. Retiring DigiCert log server (aka "CT1") in Chrome. https://groups.google.com/a/chromium.org/g/ct-policy/c/P5aj4JEBFPM/m/9AEcvY01EQAJ, accessed 2023-04-30.

[6] Andrew Ayer. Trust Asia 2021 has produced inconsistent STHs. https://groups.google.com/a/chromium.org/g/ct-policy/c/VJaSg717m9g, accessed 2023-04-30.

[7] Vaibhav Bajpai, Anna Brunström, Anja Feldmann, Wolfgang Kellerer, Aiko Pras, Henning Schulzrinne, Georgios Smaragdakis, Matthias Wählisch, and Klaus Wehrle. The dagstuhl beginners guide to reproducibility for experimental networking research. *CCR*, 49(1), 2019.

[8] Henry Birge-Lee, Yixin Sun, Anne Edmundson, Jennifer Rexford, and Prateek Mittal. Bamboozling certificate authorities with BGP. In *USENIX Security*, 2018.

[9] Billy Bob Brumley and Nicola Tuveri. Remote timing attacks are still practical. In *ESORICS*, 2011.

[10] David Brumley and Dan Boneh. Remote timing attacks are practical. In *USENIX Security*, 2003.

[11] CA/Browser Forum. Baseline requirements for the issuance and management of publicly-trusted certificates. https://cabforum.org/wp-content/uploads/CA-Browser-Forum-BR-1.8.7.pdf, accessed 2023-04-30.

[12] Sambuddho Chakravarty, Angelos Stavrou, and Angelos D. Keromytis. Traffic analysis against low-latency anonymity networks using available bandwidth estimation. In *ESORICS*, 2010.

[13] Melissa Chase and Sarah Meiklejohn. Transparency overlays and applications. In *CCS*, 2016.

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

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

[16] Laurent Chuat, Pawel Szalachowski, Adrian Perrig, Ben Laurie, and Eran Messeri. Efficient gossip protocols for verifying the consistency of certificate logs. In *CNS*, 2015.

[17] Jeremy Clark and Paul C. van Oorschot. SoK: SSL and HTTPS: revisiting past challenges and evaluating certificate trust model enhancements. In *IEEE S&P*, 2013.

[18] Sigsum Project Contributors. Witness API v0. `https://git.glasklar.is/sigsum/project/documentation/-/blob/main/witness.md`, accessed 2023-04-30.

[19] Scott A. Crosby and Dan S. Wallach. Efficient data structures for tamper-evident logging. In *USENIX Security*, 2009.

[20] 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.

[21] Rasmus Dahlberg. Transparency log preliminaries. `https://gitlab.torproject.org/rgdd/ct/-/blob/main/doc/tlog-preliminaries.md`, accessed 2023-04-30.

[22] Joe DeBlasio. Opt-out SCT auditing in Chrome. `https://docs.google.com/document/d/16G-Q7iN3kB46GSW5b-sfH5MO3nKSYyEb77YsM7TMZGE/edit`, accessed 2023-04-30.

[23] Peter J Denning. Is computer science science? *CACM*, 48(4), 2005.

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

[25] Alexandra Dirksen, David Klein, Robert Michael, Tilman Stehr, Konrad Rieck, and Martin Johns. LogPicker: Strengthening Certificate Transparency against covert adversaries. *PETS*, 2021(4).

[26] Gordana Dodig-Crnkovic. Scientific methods in computer science. In *Proceedings of the Conference for the Promotion of Research in IT at New Universities and at University Colleges in Skövde, Sweden*, 2002.

[27] Jason A. Donenfeld. Wireguard: Next generation kernel network tunnel. In *NDSS*, 2017.

[28] Benjamin Dowling, Felix Günther, Udyani Herath, and Douglas Stebila. Secure logging schemes and Certificate Transparency. In *ESORICS*, 2016.

[29] Graham Edgecombe. WoSign log failure to incorporate entry within the MMD. https://groups.google.com/a/chromium.org/g/ct-policy/c/-eV4Xe8toVk/m/pC5gSjJKCwAJ, accessed 2023-04-30.

[30] Adam Eijdenberg, Ben Laurie, and Al Cutter. Verifiable data structures. https://github.com/google/trillian/blob/master/docs/papers/VerifiableDataStructures.pdf, accessed 2023-04-30.

[31] Saba Eskandarian, Eran Messeri, Joseph Bonneau, and Dan Boneh. Certificate Transparency with privacy. *PETS*, 2017(4).

[32] Qian Ge, Yuval Yarom, David A. Cock, and Gernot Heiser. A survey of microarchitectural timing attacks and countermeasures on contemporary hardware. *JCEN*, 8(1), 2018.

[33] Google LLC. Certificate Transparency in Chrome. https://googlechrome.github.io/CertificateTransparency/ct_policy.html, accessed 2023-04-30.

[34] Google LLC. The list of existing monitors. https://certificate.transparency.dev/monitors/, accessed 2023-04-30.

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

[36] Lachlan J. Gunn, Andrew Allison, and Derek Abbott. Safety in numbers: Anonymization makes keyservers trustworthy. In *HotPETs*, 2017.

[37] Paul Hadfield. Google Aviator incident under investigation. https://groups.google.com/a/chromium.org/g/ct-policy/c/ZZf3iryLgCo/m/mi-4ViMiCAAJ, accessed 2023-04-30.

[38] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. One server for the price of two: Simple and fast single-server private information retrieval. In *USENIX Security*, 2023.

[39] Cormac Herley and Paul C. van Oorschot. SoK: Science, security and the elusive goal of security as a scientific pursuit. In *IEEE S&P*, 2017.

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

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

[42] Benjamin Hof and Georg Carle. Software distribution transparency and auditability. *CoRR*, abs/1711.07278, 2017.

[43] Paul Hoffman and Patrick McManus. DNS queries over HTTPS (DoH). RFC 8484, IETF, 2018.

[44] Ralph Holz, Jens Hiller, Johanna Amann, Abbas Razaghpanah, Thomas Jost, Narseo Vallina-Rodriguez, and Oliver Hohlfeld. Tracking the deployment of TLS 1.3 on the web: a story of experimentation and centralization. *CCR*, 50(3), 2020.

[45] Hans Hoogstraaten. Black tulip—report of the investigation into the DigiNotar certificate authority breach. Technical report, Fox-IT, 2012.

[46] 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.

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

[48] George Kadianakis, Yawning Angel, and David Goulet. A name system API for Tor onion services. `https://gitlab.torproject.org/tpo/core/torspec/-/blob/main/proposals/279-naming-layer-api.txt`, accessed 2023-04-30.

[49] Daniel Kales, Olamide Omolola, and Sebastian Ramacher. Revisiting user privacy for Certificate Transparency. In *IEEE EuroS&P*, 2019.

[50] Neal Koblitz and Alfred Menezes. Another look at "provable security". *J. Cryptol.*, 20(1), 2007.

[51] Neal Koblitz and Alfred Menezes. Another look at security definitions. *AMC*, 7(1), 2013.

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

[53] Ben Laurie. Certificate Transparency over DNS. `https://github.com/google/certificate-transparency-rfcs/blob/master/dns/draft-ct-over-dns.md`, accessed 2023-04-30.

[54] Ben Laurie. Certificate Transparency. *CACM*, 57(10), 2014.

[55] Ben Laurie, Adam Langley, and Emilia Kasper. Certificate Transparency. RFC 6962, IETF, 2013.

[56] Ben Laurie, Eran Messeri, and Rob Stradling. Certificate Transparency version 2.0. RFC 9162, IETF, 2021.

[57] Bingyu Li, Jingqiang Lin, Fengjun Li, Qiongxiao Wang, Qi Li, Jiwu Jing, and Congli Wang. Certificate Transparency in the wild: Exploring the reliability of monitors. In *CCS*, 2019.

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

[59] Wouter Lueks and Ian Goldberg. Sublinear scaling for multi-client private information retrieval. In *FC*, 2015.

[60] Harjasleen Malvai, Lefteris Kokoris-Kogias, Alberto Sonnino, Esha Ghosh, Ercan Oztürk, Kevin Lewi, and Sean F. Lawlor. Parakeet: Practical key transparency for end-to-end encrypted messaging. In *NDSS*, 2023.

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

[62] Macarena C. Martínez-Rodríguez, Ignacio M. Delgado-Lozano, and Billy Bob Brumley. SoK: Remote power analysis. In *ARES*, 2021.

[63] Nate Mathews, James K Holland, Se Eun Oh, Mohammad Saidur Rahman, Nicholas Hopper, and Matthew Wright. SoK: A critical evaluation of efficient website fingerprinting defenses. In *IEEE S&P*, 2023.

[64] Sarah Meiklejohn, Joe DeBlasio, Devon O'Brien, Chris Thompson, Kevin Yeo, and Emily Stark. SoK: SCT auditing in Certificate Transparency. *PETS*, 2022(3).

[65] Sarah Meiklejohn, Pavel Kalinnikov, Cindy S. Lin, Martin Hutchinson, Gary Belvin, Mariana Raykova, and Al Cutter. Think global, act local: Gossip and client audits in verifiable data structures. *CoRR*, abs/2011.04551, 2020.

[66] Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Edward W. Felten, and Michael J. Freedman. CONIKS: bringing key transparency to end users. In *USENIX Security*, 2015.

[67] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *CRYPTO*, 1987.

[68] Prateek Mittal, Ahmed Khurshid, Joshua Juen, Matthew Caesar, and Nikita Borisov. Stealthy traffic analysis of low-latency anonymous communication using throughput fingerprinting. In *CCS*, 2011.

[69] Alec Muffett. Real-world onion sites. `https://github.com/alecmuffett/real-world-onion-sites`, accessed 2023-04-30.

[70] Steven J. Murdoch and George Danezis. Low-cost traffic analysis of Tor. In *IEEE S&P*, 2005.

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

[72] Linus Nordberg, Daniel Kahn Gillmor, and Tom Ritter. Gossiping in CT. Internet-draft draft-ietf-trans-gossip-05, IETF, 2018.

[73] Juha Nurmi. *Understanding the Usage of Anonymous Onion Services*. PhD thesis, Tampere University, Finland, 2019.

[74] Se Eun Oh, Taiji Yang, Nate Mathews, James K. Holland, Mohammad Saidur Rahman, Nicholas Hopper, and Matthew Wright. DeepCoFFEA: Improved flow correlation attacks on Tor via metric learning and amplification. In *IEEE S&P*, 2022.

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

[76] Mike Perry. A critique of website traffic fingerprinting attacks. `https://blog.torproject.org/critique-website-traffic-fingerprinting-attacks`, accessed 2023-04-30.

[77] 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/`, accessed 2023-04-30.

[78] Andreas Pfitzmann and Marit Hansen. A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management, 2010.

[79] Tor Project. Browse privately. Explore freely. Defend yourself against tracking and surveillance. Circumvent censorship. `https://www.torproject.org/`, accessed 2022-04-30.

[80] Tor Project. Onion-Location. `https://community.torproject.org/onion-services/advanced/onion-location/`, accessed 2023-04-30.

[81] Tor Project. Research safety board. `https://research.torproject.org/safetyboard/`, accessed 2023-04-30.

[82] Mohammad Saidur Rahman, Payap Sirinam, Nate Mathews, Kantha Girish Gangadhara, and Matthew Wright. Tik-Tok: The utility of packet timing in website fingerprinting attacks. *PETS*, 2020(3).

[83] Vera Rimmer, Theodor Schnitzler, Tom van Goethem, Abel Rodríguez Romero, Wouter Joosen, and Katharina Kohls. Trace oddity: Methodologies for data-driven traffic analysis on Tor. *PETS*, 2022(3).

[84] Jeremy Rowley. CT2 log compromised via Salt vulnerability. https://groups.google.com/a/chromium.org/forum/#!topic/ct-policy/aKNbZuJzwfM, accessed 2023-04-30.

[85] Stefan Santesson, Michael Myers, Rich Ankney, Ambarish Malpani, Slava Galperin, and Carlisle Adams. X.509 Internet public key infrastructure online certificate status protocol—OCSP. RFC 6960, IETF, 2013.

[86] Sectigo Limited. crt.sh: certificate search. https://github.com/crtsh, accessed 2023-04-30.

[87] Sectigo Limited. crt.sh: certificate search ID = '8913351873'. https://crt.sh/?id=8913351873, accessed 2023-04-30.

[88] SecureDrop. Getting an onion name for your SecureDrop. https://securedrop.org/faq/getting-onion-name-your-securedrop/, accessed 2023-04-30.

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

[90] Payap Sirinam, Mohsen Imani, Marc Juárez, and Matthew Wright. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *CCS*, 2018.

[91] Ryan Sleevi. StartCom log misbehaving: Failure to incorporate SCTs. https://groups.google.com/a/chromium.org/g/ct-policy/c/92HIh2vG6GA/m/hBEHxcpoCgAJ, accessed 2023-04-30.

[92] Ryan Sleevi. Upcoming CT log removal: Izenpe. https://groups.google.com/a/chromium.org/forum/#!topic/ct-policy/qOOrKuhL1vA, accessed 2023-04-30.

[93] Ryan Sleevi. Upcoming log removal: Venafi CT log server. https://groups.google.com/a/chromium.org/forum/#!topic/ct-policy/KMAcNT3asTQ, accessed 2023-04-30.

[94] Ryan Sleevi and Eran Messeri. Certificate Transparency in Chrome: Monitoring CT logs consistency. https://docs.google.com/document/d/1FP5J5Sfsg0OR9P4YT0q1dM02iavhi8ix1mZlZe_z-ls/edit?pref=2&pli=1, accessed 2023-04-30.

[95] SSLMate Inc. Cert spotter—Certificate Transparency monitor. https://github.com/SSLMate/certspotter, accessed 2023-04-30.

[96] SSLMate Inc. Timeline of certificate authority failures. https://sslmate.com/resources/certificate_authority_failures, accessed 2023-04-30.

[97] Emily Stark, Joe DeBlasio, Devon O'Brien, Davide Balzarotti, William Enck, Samuel King, and Angelos Stavrou. Certificate Transparency in Google Chrome: Past, present, and future. *IEEE S&P*, 19(6), 2021.

[98] Emily Stark, Ryan Sleevi, Rijad Muminovic, Devon O'Brien, Eran Messeri, Adrienne Porter Felt, Brendan McMillion, and Parisa Tabriz. Does Certificate Transparency break the web? Measuring adoption and error rate. In *IEEE S&P*, 2019.

[99] Emily Stark and Chris Thompson. Opt-in SCT auditing. https://docs.google.com/document/d/1G1Jy8LJgSqJ-B673GnTYIG4b7XRw2ZLtvvSlrqFcl4A/edit, accessed 2023-04-30.

[100] Nick Sullivan. Understanding use-cases for SCTs delivered via OCSP stapling for TLS extension. https://groups.google.com/a/chromium.org/g/ct-policy/c/WX6iZt7uJBs, accessed 2023-04-30.

[101] Nick Sullivan and Sean Turner. Messaging layer security: Secure and usable end-to-end encryption. https://www.ietf.org/blog/mls-secure-and-usable-end-to-end-encryption/, accessed 2023-04-30.

[102] 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.

[103] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. Keeping authorities "honest or bust" with decentralized witness cosigning. In *IEEE S&P*, 2016.

[104] Alin Tomescu, Vivek Bhupatiraju, Dimitrios Papadopoulos, Charalampos Papamanthou, Nikos Triandopoulos, and Srinivas Devadas. Transparency logs via append-only authenticated dictionaries. In *CCS*, 2019.

[105] Yukiyasu Tsunoo, Teruo Saito, Tomoyasu Suzaki, Maki Shigeri, and Hiroshi Miyauchi. Cryptanalysis of DES implemented on computers with cache. In *CHES*, 2003.

[106] 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.

[107] Mathy Vanhoef and Tom Van Goethem. HEIST: HTTP encrypted information can be stolen through TCP-windows. In *Black Hat US Briefings*, 2016.

[108] Jesse Victors, Ming Li, and Xinwen Fu. The onion name system. *PETS*, 2017(1).

[109] Emanuel von Zezschwitz, Serena Chen, and Emily Stark. "It builds trust with the customers"—exploring user perceptions of the padlock icon in browser UI. In *IEEE SPW*, 2022.

[110] Jun Wang, Weinan Zhang, and Shuai Yuan. Display advertising with real-time bidding (RTB) and behavioural targeting. *Foundations and Trends in Information Retrieval*, 2017.

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

[112] Yingchen Wang, Riccardo Paccagnella, Elizabeth Tang He, Hovav Shacham, Christopher W. Fletcher, and David Kohlbrenner. Hertzbleed: Turning power side-channel attacks into remote timing attacks on x86. In *USENIX Security*, 2022.

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

[114] Zerodium. We pay big bounties. https://zerodium.com/, accessed 2023-04-30.

Paper I

Verifiable Light-Weight Monitoring for Certificate Transparency Logs

Reprinted from

NordSec (2018)

Verifiable Light-Weight Monitoring for Certificate Transparency Logs

Rasmus Dahlberg and Tobias Pulls

Abstract

Trust in publicly verifiable Certificate Transparency (CT) logs is reduced through cryptography, gossip, auditing, and monitoring. The role of a monitor is to observe each and every log entry, looking for suspicious certificates that interest the entity running the monitor. While anyone can run a monitor, it requires continuous operation and copies of the logs to be inspected. This has lead to the emergence of monitoring as-a-service: a trusted third-party runs the monitor and provides registered subjects with selective certificate notifications. We present a CT/bis extension for verifiable *light-weight monitoring* that enables subjects to verify the correctness of such certificate notifications, making it easier to distribute and reduce the trust which is otherwise placed in these monitors. Our extension supports verifiable monitoring of wild-card domains and piggybacks on CT's existing gossip-audit security model.

1 Introduction

Certificate Transparency (CT) [18] is an experimental standard that enhances the public-key infrastructure by adding transparency for certificates that are issued by Certificate Authorities (CAs). The idea is to mandate that every certificate must be publicly logged in an append-only tamper-evident data structure [4], such that anyone can observe what has been issued for whom. This means that a subject can determine for herself if anything is mis-issued by downloading all certificates; so called *self-monitoring*. An alternative monitoring approach is to rely on a trusted third-party that *notifies* the subject if relevant certificates are ever found. Given that self-monitoring involves set-up, continuous operation, and exhaustive communication effort, the concept of subscribing for monitoring *as-a-service* is simpler for the subject. This model is already prevalent in the wild, and is provided both by CAs and industry vendors—see for example SSLMate's Cert Spotter [26] or Facebook's monitoring tool [14]. Third-party monitors can also offer related services, such as searching for certificates interactively or inspecting other log properties. The former is provided by Facebook and Comodo's `crt.sh`; the latter by Graham Edgecombe's CT monitoring tool [9].

It would be an unfortunate short-coming if CT did not change the status quo of centralized trust by forcing subjects who cannot operate a self-monitor to trust certificate notifications that are provided by a third-party monitor. While it is true that a subject could subscribe to a large number of monitors to reduce this trust, it is overall cumbersome and does not scale well beyond a handful of notifying monitors (should they exist). To this end, we suggest a CT/bis extension for verifiable Light-Weight Monitoring (LWM) that makes it easier to distribute the trust which is otherwise placed in these monitors by decoupling the notifier from the full-audit function of inspecting all certificates. Our idea is best described in terms of a self-monitor that polls for new updates, but as opposed to processing all certificates we can filter on wild-card prefixes such as `*.example.com` in a verifiable manner. LWM relies on the ability to define a new Signed Tree Head (STH) extension, and thus a CT/bis compliant log is necessary [19]. At the time of writing CT/bis have yet to be published as an IETF standard. We are not aware of any log that deploys a drafted version.

As a brief overview, each batch of newly included certificates are grouped as a static Merkle tree in LWM. The resulting snapshot (also know as a fingerprint or a root hash) is then incorporated into the corresponding STH as an extension. An LWM subject receives one verifiable certificate notification per log update from an untrusted *notifier* (who could be the log, a monitor, or anyone else), and this notification is based on the smaller static Merkle tree rather than the complete log. This is because monitoring as-a-service is mainly about identifying newly included certificates. Moreover, we can order each static Merkle tree so that verifiable wild-card filtering is possible. For security we rely on at least one entity to verify that each snapshot is correct—which is a general monitoring function that is independent of the subjects using LWM—as well as a gossip protocol that detects split-views [3]. Since our extension is part of an STH, we piggyback on any gossip-like protocol that deals with the exchange and/or distribution of (verified) STHs [6, 23, 25, 28]. Our contributions are as follows:

- The design of a backwards-compatible CT/bis extension for light-weight monitoring of wild-card prefixes such as `*.example.com` (Section 3).
- A security sketch showing that an attacker cannot omit a certificate notification without being detected, relying on standard cryptographic assumptions and piggybacking on the proposed gossip-audit models of CT (Section 4.1).
- An open-source proof-of-concept implementation written in Go, as well as a performance evaluation that considers computation time and bandwidth requirements (Section 4.2). In particular we find that the overhead during tree head construction is small in comparison to a sound STH frequency of one hour; a notifier can easily notify 288 M subjects in a verifiable manner for Google's Icarus log on a single core and a 1 Gbps

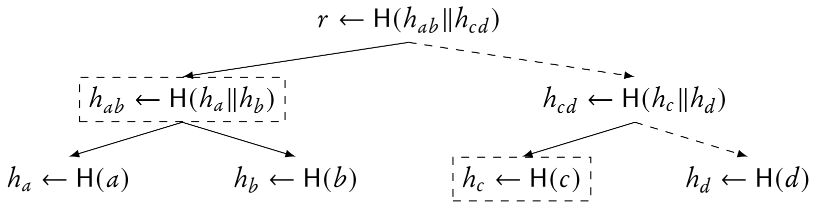

$r \leftarrow \mathsf{H}(h_{ab} \| h_{cd})$
$h_{ab} \leftarrow \mathsf{H}(h_a \| h_b)$
$h_{cd} \leftarrow \mathsf{H}(h_c \| h_d)$
$h_a \leftarrow \mathsf{H}(a)$
$h_b \leftarrow \mathsf{H}(b)$
$h_c \leftarrow \mathsf{H}(c)$
$h_d \leftarrow \mathsf{H}(d)$

Figure 1: Merkle tree containing four values a–d. The dashed arrows show the traversal used to generate an audit path for the right-most leaf (dashed nodes).

connection; and a subject receives about 24 Kb of proofs per day and log which is verified in negligible time (the order of μs for the common case of non-membership, and seconds in the extreme case of verifying membership for *an entire top-level domain*).

Background on Merkle trees and CT is provided in Section 2. Related work is discussed in Section 4.3. Conclusions are presented in Section 5.

2 Background

Suppose that a trusted content provider would like to outsource its operation to an untrusted third-party. This is often referred to as the three-party setting, in which a trusted source maintains an authenticated data structure through a responder that answers client queries on the source's behalf [29]. The data structure is authenticated in the sense that every answer is accompanied by a cryptographic proof that can be verified for correctness by only trusting the source. While there are many settings and flavors of authenticated data structures [4, 5, 7], our scope is narrowed down to CT which builds upon Merkle trees.

2.1 Merkle Trees

The seminal work by Merkle [21] proposed a *static* binary tree where each leaf stores the hash of a value and every interior node hashes its children (Figure 1). The root hash serves as a succinct snapshot of the tree's structure and content, and by revealing a logarithmic number of hashes it can be reconstructed to prove whether a value is stored in a leaf. These hashes compose an audit path for a value, and it is obtained by taking every sibling hash while traversing the tree from the root down towards the leaf being authenticated. An audit path is verified by reversing the traversal used during generation, first reconstructing the leaf hash and then every interior node recursively (using the provided sibling hashes) until finally reaching the root. Given a collision resistant hash function, an audit path proves that a given leaf contains a value iff the reconstructed root hash is known to be authentic. For example, the trusted source might sign it.

While non-membership of a value can be proven by providing the entire data structure, this is generally too inefficient since it requires linear space and time. A better approach is to structure the tree such that the node which should contain a value is known if it exists. This property is often discussed in relation to certificate revocation: as opposed to downloading a list of serial numbers that represent the set of revoked certificates, each leaf in a static Merkle tree could (for example) contain an interval $[a, b)$ where a is revoked and the open interval (a, b) current [17]. Given a serial number x, an audit path can be generated in logarithmic space and time for the leaf where $x \in [a, b)$ to prove (non-)membership. Similar constructions that are *dynamic* support updates more efficiently [5, 11, 20].

2.2 Certificate Transparency

The CA ecosystem involves hundreds of trusted third-parties that issue TLS certificates [8]. Once in a while *somebody* gets this process wrong, and as a result a fraudulent identity-to-key binding may be issued for *any* subject [12]. It is important to detect such incidents because mis-issued certificates can be used to intercept TLS connections. However, detection is hard unless the subjects *who can distinguish between anything benign and fraudulent* get a concise view of the certificates that are being served to the clients. By requiring that every CA-issued certificate must be disclosed in a public and append-only log, CT layers on-top of the error-prone CA ecosystem to provide such a view: in theory anyone can inspect a log and determine for herself if a certificate is mis-issued [18].

It would be counter-intuitive to 'solve' blind trust in CAs by suggesting that everybody should trust a log. Therefore, CT is designed such that the log can be distrusted based on two components: a dynamic append-only Merkle tree that supports verifiable membership and consistency queries [4], as well as a gossip protocol that detects split-views [3, 23]. We already introduced the principles of membership proofs in Section 2.1, and consistency proofs are similar in that a logarithmic number of hashes are revealed to prove two snapshots consistent. In other words, anyone can verify that a certificate is included in the log without fully downloading it, and whatever was in the log before still remains unmodified. Unlike the three-party setting, gossip is needed because there is no trusted source that signs-off the authenticated data structure: consistency and inclusion proofs have limited value if everybody observes different (but valid) versions of the log.

Terminology, Policy Parameters, and Status Quo

A new STH—recall that this is short for Signed Tree Head—is issued by the log at least every Maximum Merge Delay (MMD) and no faster than allowed by an STH frequency [19]. An MMD is the longest time until a certificate must be included in the log after promising to include it. This promise is referred to as a Signed Certificate Timestamp (SCT). An STH frequency is relative to the MMD, and limits the number of STHs that can be issued. These parameters (among others) are defined in a log's policy, and if a violation is detected there are non-repudiable proofs of log misbehavior that can be presented. For example, show an SCT that is not included after an MMD, too many STHs during the period of an MMD, or two STHs that are part of two inconsistent versions of the log. In other words, rather than being a trusted source a log signs statements to be held accountable.

Ideally we would have all of these components in place at once: anyone that interacts with a log audits it for correctness based on partial information (SCTs, STHs, served certificates, and proofs), subjects monitor the logs for newly included certificates to check that they are free from mis-issuance (full download), and a gossip protocol detects or deters logs from presenting split-views. This is not the case in practice, mainly because CT is being deployed incrementally [25] but also because the cost and complexity of self-monitoring is relatively high. For example, a subject that wants rapid detection of mis-issuance needs continuous operation and full downloads of the logs. It appears that the barrier towards self-monitoring have lead to the emergence of monitoring as-a-service, where a trusted third-party monitors the logs on a subject's behalf by selectively notifying her of relevant certificates, e.g., mail the operator of example.com if $*$.example.com certificates are ever found. Third-party monitoring is convenient for logs too because it reduces the bandwidth required to serve many subjects. However, for CT it is an unintuitive concept given that it requires blind trust.

3 Light-Weight Monitoring

To reduce the trust which is placed in today's third-party monitors, the idea of LWM is to lower the barrier towards self-monitoring. As shown in Figure 2, an untrusted notifier provides a subject with efficient[1] certificate notifications that can be cryptographically verified: each batch of certificates is represented by an additional Merkle tree that supports wild-card (non-)membership queries (described further in Section 3.1), and the resulting snapshot is signed by the log as part of an STH extension. As such, a subject can deal only with those certificates that are relevant, relying on wild-card proofs to verify correctness and completeness: said certificates are included and nothing is being omitted. Anyone can check that an LWM snapshot is correct by inspecting the corresponding batch of certificates. Notably this is *a general monitoring function*, rather than a *selective notification component* which is verifiable in LWM. This decoupling allows anyone to be a notifier, including logs and monitors that a subject distrust.

3.1 Authenticated Wild-Card Queries

Thus far we only discussed Merkle trees in terms of verifying whether a single value is a (non-)member: membership is proven by presenting an audit path

[1]Efficient iff less than a linear number of log entries are received per log update.

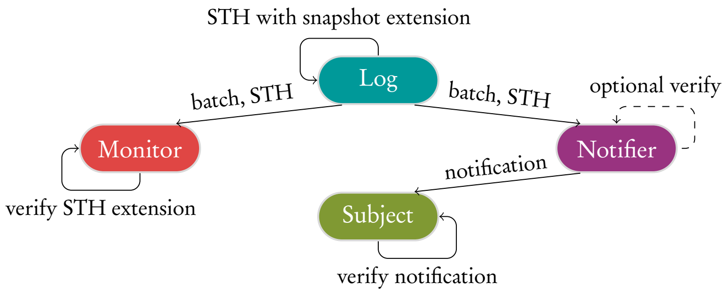


Figure 2: An overview of LWM. In addition to normal operation, a log creates an additional (smaller) Merkle tree that supports wild-card (non-)membership queries. The resulting snapshot is signed as part of an STH extension that can be verified by any monitor that downloads the corresponding batch. A subject receives one verifiable certificate notification per STH from an untrusted notifier.

$$\begin{tikzcd}
h_0 \leftarrow \mathsf{H}(\mathsf{gro.elpmaxe}) & & \\
 & h_{01} \leftarrow \mathsf{H}(h_0 \| h_1) \arrow[ul] \arrow[dl] & \\
h_1 \leftarrow \mathsf{H}(\mathsf{moc.elpmaxe}) & & \\
 & & r \leftarrow \mathsf{H}(h_{01} \| h_{23}) \arrow[uul] \arrow[ddl] \\
h_2 \leftarrow \mathsf{H}(\mathsf{moc.elpmaxe.bus}) & & \\
 & h_{23} \leftarrow \mathsf{H}(h_2 \| h_3) \arrow[ul] \arrow[dl] & \\
h_3 \leftarrow \mathsf{H}(\mathsf{ten.elpmaxe}) & &
\end{tikzcd}$$

Figure 3: Merkle tree where the leaves are ordered on reversed subject names.

down to the leaf in question, while non-membership requires a lexicographical ordering that allows a verifier to conclude that a value is absent unless provided in a particular location. The latter concept naturally extends to *prefix wild-card queries*—such as ∗.example.com and ∗.sub.example.com—by finding a suitable ordering function Ω which ensures that related leaves are grouped together as a consecutive range. We found that this requirement is satisfied by sorting on reversed subject names: suppose that we have a batch of certificates example.com, example.org, example.net, and sub.example.com. After applying Ω we get the static Merkle tree in Figure 3. A prefix wild-card proof is constructed by finding the relevant range in question, generating an audit path for the leaves that are right outside of the range [24]. Such a proof is verified by checking that (i) Ω indicates that the left (right) end is less (larger) than the queried prefix, (ii) the leaves are ordered as dictated by Ω, and (iii) the recomputed root hash is valid.

The exact details of reconstructing the root hash is a bit tedious because there are several corner cases. For example, either or both of the two audit paths may be empty depending on batch size ($\leq$1) and location of the relevant range (left/right-most side). Therefore, we omit the details and focus on the concept: given two audit paths and a sequence of data items ordered by Ω that includes the left leaf, matching range, and right leaf, repeatedly reconstruct interior nodes to the largest extent possible and then use the sibling hash which is furthest from the root to continue. For example, consider a proof for ∗sub.example.com in Figure 3: it is composed of (i) the left leaf data and its audit path h_0, h_{23} on index 1, (ii) the right leaf data and its audit path h_2, h_{01} on index 3, and (iii) the matching range itself which is a single certificate. After verifying Ω order, recompute the root hash r' and check if it matches an authentic root r as follows:

1. Compute leaf hashes h'_1, h'_2, and h'_3 from the provided data. Next, compute the interior node $h'_{23} \leftarrow \mathsf{H}(h'_2 \| h'_3)$. Because no additional interior node can be computed without a sibling hash, consider h_0 in the left audit path.

2. Compute the interior node $h'_{01} \leftarrow \mathsf{H}(h_0 \| h'_1)$, then $r' \leftarrow \mathsf{H}(h'_{01} \| h'_{23})$.[2]

Given an Ω ordered list of certificates it is trivial to locate where a subject's wild-card matches are: binary search to find the index of an exact match (if any), then up to t matches follow in order. This is not the only way to find the right range and matches. For example, a radix tree could be used with the main difference being $O(t + \log n)$ against $O(t + k)$ complexity for a batch of size n, a wild-card string of length k, and t matches. Since the complexity of generating two audit paths is $O(\log n)$ for any number of matches, the final space and time complexity for a wild-card structure based on an ordered list is $O(t + \log n)$.

3.2 Notifier

A notifier must obtain every STH to generate wild-card proofs that can be traced back to the log. Albeit error-prone in case of network issues, the simplest way to go about this is to poll the log's get-STH endpoint *frequently enough*.[3] Once an updated is spotted every new certificate is downloaded and the wild-card structure is reconstructed. A subject receives her verifiable certificate notifications from the notifier via a push ('monitoring as-a-service') or pull ('self-monitoring') model. For example, emails could be delivered after every update or in daily digests. Another option is to support queries like "what's new since STH x".

A subject can verify that a certificate notification is fresh by inspecting the STH timestamp. However, it is hard to detect missing certificate notifications unless every STH trivially follows from the previous one. While there are several methods to achieve this—for example using indices (Section 3.3) or hash chains [20]—the log must always sign a snapshot per STH using an extension.

[2]Two audit paths may contain redundancy, but we ignored this favouring simplicity.

[3]It would be better if logs supported verifiable and historical get-STH queries.

3.3 Instantiation Example

Instantiating LWM depends upon the ability to support an STH extension. In the latest version of CT, this takes the form of a sorted list of key-value pairs where the key is unique and the value an opaque byte array [19]. We could reserve the keywords *lwm* for snapshots and *index* for monotonically increasing counters.[4] Besides an LWM-compliant log, an untrusted notifier must support pushed or pulled certificate notifications that are verifiable by tracking the most recent or every wild-card structure. Examples of likely notifiers include logs (who benefit from the reduced bandwidth) and monitors (who could market increased transparency) that already process all certificates regardless of LWM.

4 Evaluation

First we discuss assumptions and sketch on relevant security properties for LWM. Next, we examine performance properties of our open-source proof-of-concept implementation experimentally and reason about bandwidth overhead in theory. Finally, we present differences and similarities between LWM and related work.

4.1 Assumptions and Security Notions

The primary threat is a computationally bound attacker that attempts to forge or omit a certificate notification without being detected. We rely on standard cryptographic assumptions, namely an unforgeable digital signature scheme and a collision resistant hash function H with 2λ-bit output for a security parameter λ. The former means that an LWM snapshot must originate from the (untrusted) log in question. While an incorrect snapshot could be created intentionally to hide a mis-issued certificate, it would be detected if at least one honest monitor exists because our STH extension piggybacks on the gossip-audit model of CT (that we assume is secure).[5] A subject can further detect missing notifications by checking the STH index for monotonic increases and the STH timestamp for freshness. Thus, given secure audit paths and correct verification checks as described in Section 3.1, no certificate notification can be forged or omitted. Our cryptographic assumptions ensure that every leaf is fixed by a secure audit path as in CT, i.e., a leaf hash with value v is encoded as $\mathsf{H}(\mathsf{0}x\mathsf{00}\|v)$ and an interior hash with children L, R as $\mathsf{H}(\mathsf{0}x\mathsf{01}\|L\|R)$ [4, 18]. To exclude any unnecessary data on the ends of a range, the value v is a subject name concatenated with a hashed list of associated certificates in LWM (subject names suffice to verify Ω order).

[4]Instead of an index to detect missing notifications (STHs), a log could announce STHs as part of a verifiable get-STH endpoint. See the sketch of Nordberg [22].

[5]Suppose that witness cosigning is used [28]. Then we rely on at least one witness to verify our extension. Or, suppose that STH pollination is used [23]. Then we rely on the most recent window of STHs to reach a monitor that verifies our extension.

CT makes no attempt to offer security in the multi-instance setting [15]. Here, an attacker that targets many different Merkle trees in parallel should gain no advantage while trying to forge *any* valid (non-)membership proof. By design there will be many different wild-card Merkle trees in LWM, and so the (strictly stronger) multi-instance setting is reasonable. We can provide full bit-security in this setting by ensuring that no node's pre-image is valid across different trees by incorporating a unique tree-wide constant c_t in leaf and empty hashes *per batch*, e.g., $c_t \leftarrow\$ \{0,1\}^\lambda$. Melera *et al.* [20] describe this in detail while also ensuring that no node's pre-image is valid across different locations within a Merkle tree.

In an ecosystem where CT is being deployed incrementally without gossip, the benefit of LWM is that a subject who subscribes for certificate notifications can trust the log only (as opposed to *also* trusting the notifier). Therefore, today's trust in third-party monitoring services can be reduced significantly. A log must also present a split-view or an invalid snapshot to deceive a subject with false notifications. As such, subjects accumulate binding evidence of log misbehavior that can be audited sometime in the future if suspicion towards a log is raised. Long-term the benefit of LWM is that it is easier to distribute the trust which is placed in third-party monitors, i.e., anyone who processes a (small in comparison to the entire log) batch of certificates can full-audit it without being a notifier.

4.2 Implementation and Performance

We implemented multi-instance secure LWM in less than 400 lines of Go [1]. Our wild-card structure uses an existing implementation of a radix tree to find leaf indices and data. To minimize proof-generation times, all hashes are cached in an in-memory Merkle tree which uses SHA-256. We benchmarked snapshot creation, proof generation, and proof verification times on a single core as the batch size increases from 1024–689,245 certificates using Go's built-in benchmarking tool, an Intel(R) Core(TM) i5-2500 CPU @ 3.30GHz, and 2x8 Gb DDR3 RAM. We assumed real subject names from Alexa's top-1M [2]. and average-sized certificates of 1500 bytes [10], where a batch of n subject names refers to the n most popular domains. Notably 689,245 certificates is the largest batch observed by us in Google's Icarus log between 2017-01-25 and 2018-08-05, corresponding to an STH interarrival time of 27.1 hours. The median (average) batch size and STH interarrival time were 22818 (23751) certificates and 60.1 (61.6) minutes. Only two batches were larger than 132077 certificates. Considering that Icarus is one of the logs that see largest loads [27], we can make non-optimistic conclusions regarding the performance overhead of LWM without inspecting other logs.

Figure 4 shows snapshot creation time as a function of batch size. Nearby the median (2^{15}) it takes 0.39 seconds to create a snapshot from scratch, initializing state from an unordered dictionary and caching all hashes for the first time. For the largest batch, the snapshot creation time is roughly 10 seconds. Arguably this overhead is still insignificant for logs, monitors, and notifiers

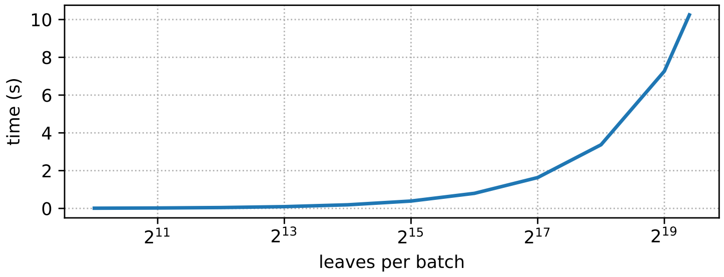


Figure 4: Snapshot creation time as a function of batch size.

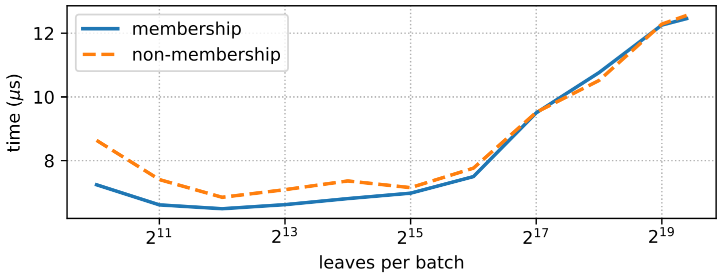


Figure 5: Membership and non-membership proof query time as a function of batch size for a single and no match, respectively.

because the associated STH interarrival times are orders of magnitude larger.

Figure 5 shows proof generation time as a function of batch size while querying for the longest wild-card prefix with a single match (membership), as well as another wild-card prefix without any match in com's top-level domain (non-membership). There is little or no difference between the generation time for these types of wild-card proofs, and nearby the median it takes around 7 μs. For the largest batch, this increased to 12.5 μs. A notifier can thus generate 288 million non-membership notifications per hour *on a single core*. Verification is also in the order of μs, which should be negligible for a subject (see Figure 6).

To evaluate the cost of generating and verifying a wild-card notification with a large number of matches, we queried for com's entire top-level domain (see Figure 7). In the largest batch where there are 352,383 matches, the proof generation time is still relatively low: 134 ms. This corresponds to 28.9k notifications per hour on a single core. The verification time is much larger: 3.5 seconds. This is expected since verification involves reconstructing the root from all the matching leaves, which is at least as costly as creating a snapshot of the same size (cf. 2^{18} in Figure 4). While these are relevant performance numbers, anyone who is interested in a top-level domain would likely just download the entire batch.

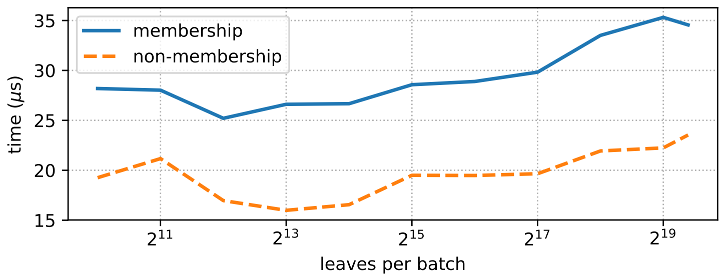


Figure 6: Membership and non-membership verification time as a function of batch size for a single and no match, respectively.

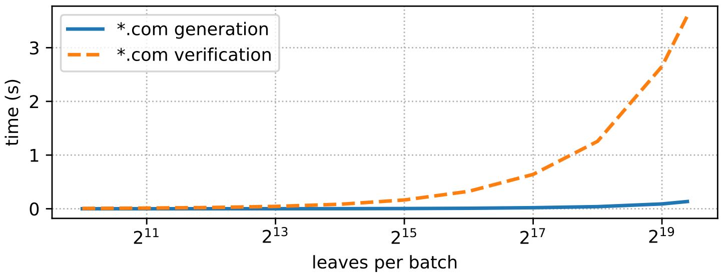


Figure 7: Membership query and verification time for $*$.com.

Finally, the space *overhead* of a verifiable wild-card notification is dominated by the two audit paths that enclose the matching subject names. Given that an audit path contains at most $\lceil \log_2 n \rceil$ sibling hashes for a batch of size n, the median overhead is roughly one Kb per STH, log, and LWM subject. Viewed from the perspective of a self-monitor, this is a significant bandwidth improvement: as opposed to downloading the median batch of 32.6 Mb, one Kb and any matching certificate(s) suffice. In the case of multiple logs, the bandwidth improvement is even greater. For the notifier we already established that it is relatively cheap to generate new notifications. Namely, in the single-core case of 288 M notifications per hour the bandwidth overhead would be 640 Mbs (i.e., all proofs must be distributed before the next STH is issued). A notifier can thus notify for a dozen of logs and a significant amount of LWM subjects without running into any CPU or bandwidth restrictions. Notably this is under the assumption of a sound STH frequency—one hour in our evaluation, as used by Icarus and many other logs.

4.3 Related Work

Earlier work related to transparent certificate and key management often use dynamic authenticated dictionaries [5, 7, 11, 16]. CONIKS maps a user's mail address to her public key in a binary Merkle prefix tree, and after each update a client self-monitors her own key-binding by fetching an exact-match (non-)membership proof [20]. While our work is conceptually similar to CONIKS since a subject receives one (non-)membership proof per log update, the main difference is that LWM builds a new Merkle tree for each update in which wild-card queries are supported. This idea is inapplicable for CONIKS because a user is potentially interested in the public key of any mail address (hence the ability to query the entire data structure on an exact-match). CONIKS is similarly inapplicable for self-monitors in CT because a subject cares about *wild-card queries* and *new certificates*. Without the need for wild-cards, any authenticated dictionary could be used as a batch building block to instantiate LWM. While a radix tree viewed as a Merkle tree could support efficient wild-card proofs [13], it is more complex than necessary. Therefore, we built upon the work of Kocher [17] and Nuckolls [24] with a twist on how to group the data for a new use-case: LWM.

5 Conclusion

We proposed a backwards-compatible CT/bis extension that enables light-weight monitoring (in short LWM). At the cost of a few hundred Kb per day, a subject can either self-monitor or subscribe to verifiable certificate notifications for a dozen of logs via an untrusted notifier. The security of LWM piggybacks on the gossip-audit model of CT, and it relies only on the existence of at least one honest monitor that verifies our extension. The cost of a compliant log is overhead during the tree head construction, and this overhead is insignificant in comparison to a log's STH frequency. A notifier can generate verifiable certificate notifications—even for wild-card queries for all domains under a top-level domain—in the order of milliseconds on a single core. Given an STH frequency of one hour and 288 M LWM subjects, the incurred bandwidth overhead is roughly 640 Mbps for proofs. As such, a log could easily be its own notifier on a 1 Gbps connection. Further, any willing third-party could notify for a dozen of logs on a 10 Gbps connection.

Acknowledgments

We would like to thank Linus Nordberg for valuable feedback. This research was funded by the Swedish Knowledge Foundation as part of the HITS research profile.

References

[1] Paper artifact. https://github.com/rgdd/lwm, 2018.

[2] Amazon. Alexa top-1M. http://s3.amazonaws.com/alexa-static/top-1m.csv.zip, accessed 2018-08-05.

[3] Laurent Chuat, Pawel Szalachowski, Adrian Perrig, Ben Laurie, and Eran Messeri. Efficient gossip protocols for verifying the consistency of certificate logs. In *CNS*, 2015.

[4] Scott A. Crosby and Dan S. Wallach. Efficient data structures for tamper-evident logging. In *USENIX Security*, 2009.

[5] Scott A. Crosby and Dan S. Wallach. Authenticated dictionaries: Real-world costs and trade-offs. *ACM TISSEC*, 14(2), 2011.

[6] Rasmus Dahlberg, Tobias Pulls, Jonathan Vestin, Toke Høiland-Jørgensen, and Andreas Kassler. Aggregation-based gossip for Certificate Transparency. *CoRR abs/1806.08817*, 2018.

[7] David Derler, Christian Hanser, and Daniel Slamanig. Revisiting cryptographic accumulators, additional properties and relations to other primitives. In *CT-RSA*, 2015.

[8] Zakir Durumeric, James Kasten, Michael Bailey, and J. Alex Halderman. Analysis of the HTTPS certificate ecosystem. In *IMC*, 2013.

[9] Graham Edgecombe. Certificate Transparency monitor. https://ct.grahamedgecombe.com/, accessed 2018-09-15.

[10] Graham Edgecombe. Compressing X.509 certificates. https://www.grahamedgecombe.com/blog/2016/12/22/compressing-x509-certificates, accessed 2018-08-15.

[11] Adam Eijdenberg, Ben Laurie, and Al Cutter. Verifiable data structures. https://github.com/google/trillian/blob/master/docs/VerifiableDataStructures.pdf, accessed 2018-09-16.

[12] ENISA. Certificate authorities—the weak link of internet security. https://www.enisa.europa.eu/publications/info-notes/certificate-authorities-the-weak-link-of-internet-security, accessed 2018-09-16.

[13] Ethereum/wiki. Patricia tree. https://github.com/ethereum/wiki/wiki/Patricia-Tree, accessed 2018-08-15.

[14] Facebook. Certificate Transparency monitoring. https://developers.facebook.com/tools/ct/, accessed 2018-09-15.

[15] Jonathan Katz. Analysis of a proposed hash-based signature standard. In *SSR*, 2016.

[16] Tiffany Hyun-Jin Kim, Lin-Shung Huang, Adrian Perrig, Collin Jackson, and Virgil D. Gligor. Accountable key infrastructure (AKI): A proposal for a public-key validation infrastructure. In *WWW*, 2013.

[17] Paul C. Kocher. On certificate revocation and validation. In *FC*, 1998.

[18] Ben Laurie, Adam Langley, and Emilia Kasper. Certificate Transparency. RFC 6962, IETF, 2013.

[19] Ben Laurie, Adam Langley, Emilia Kasper, Eran Messeri, and Rob Stradling. Certificate Transparency version 2.0. Internet-draft draft-ietf-trans-rfc6962-bis-28, IETF, 2018.

[20] Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Edward W. Felten, and Michael J. Freedman. CONIKS: Bringing key transparency to end users. In *USENIX Security*, 2015.

[21] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *CRYPTO*, 1987.

[22] Linus Nordberg. Re: [Trans] Providing the history of STHs a log has issued (in 6962-bis). https://mailarchive.ietf.org/arch/msg/trans/JbFiwO9OPjcYzXrEgh-Y7bFG5Fw, accessed 2018-09-16.

[23] Linus Nordberg, Daniel Kahn Gillmor, and Tom Ritter. Gossiping in CT. Internet-draft draft-ietf-trans-gossip-05, IETF, 2018.

[24] Glen Nuckolls. Verified query results from hybrid authentication trees. In *IFIP WG 11.3 Working Conference on Data and Application Security*, 2005.

[25] Ryan Sleevi and Eran Messeri. Certificate Transparency in Chrome: Monitoring CT logs consistency. https://docs.google.com/document/d/1FP5J5Sfsg0OR9P4YT0q1dM02iavhi8ix1mZlZe_z-ls/edit?pref=2&pli=1, accessed 2018-09-16.

[26] SSLMate. Better uptime and security with Cert Spotter. https://sslmate.com/certspotter/, accessed 2018-09-15.

[27] SSLMate. Certificate Transparency log growth. https://sslmate.com/labs/ct_growth/, accessed 2018-08-15.

[28] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. Keeping authorities "honest or bust" with decentralized witness cosigning. In *IEEE S&P*, May 2016.

[29] Roberto Tamassia. Authenticated data structures. In *ESA*, 2003.

Aggregation-Based Certificate Transparency Gossip

Reprinted from

SECURWARE (2019)

Aggregation-Based Certificate Transparency Gossip

Rasmus Dahlberg, Tobias Pulls, Jonathan Vestin, Toke Høiland-Jørgensen, and Andreas Kassler

Abstract

Certificate Transparency (CT) requires that every certificate which is issued by a certificate authority must be publicly logged. While a CT log can be untrusted in theory, it relies on the assumption that every client observes and cryptographically verifies the same log. As such, some form of gossip mechanism is needed in practice. Despite CT being adopted by several major browser vendors, no gossip mechanism is widely deployed. We suggest an aggregation-based gossip mechanism that passively observes cryptographic material that CT logs emit in plaintext, aggregating at packet processors (such as routers and switches) to periodically verify log consistency off-path. In other words, gossip is provided as-a-service by the network. Our proposal can be implemented for a variety of programmable packet processors at line-speed without aggregation distinguishers (throughput), and based on 20 days of RIPE Atlas measurements that represent clients from 3500 autonomous systems we show that significant protection against split-viewing CT logs can be achieved with a realistic threat model and an incremental deployment scenario.

1 Introduction

The HyperText Transfer Protocol Secure (HTTPS) ecosystem is going through a paradigm shift. As opposed to blindly trusting that Certificate Authorities (CAs) only issue certificates to the rightful domain owners—a model known for its weakest-link security [30]—transparency into the set of issued certificates is incrementally being required by major browser vendors [41, 52]. This transparency is forced and takes the form of Certificate Transparency (CT) logs: the idea is to reject any TLS certificate that have yet to be publicly logged, such that domain owners can monitor the logs for client-accepted certificates to *detect* certificate mis-issuance *after the fact* [45]. While the requirement of certificate logging is a significant improvement to the HTTPS ecosystem, the underlying problem of trusting CAs cannot be solved by the status quo of trusted CT logs (described further in Section 2.1). Therefore, it is paramount that nobody needs to trust these logs once incremental deployments are matured.

CT is formalized and cryptographically verifiable [28], supporting inclusion and consistency proofs. This means that a client can verify whether a log is operated correctly: said certificates are included in the log, and nothing is being removed or modified. Despite the ability to cryptographically verify these two properties, there are no assurances that everybody observes *the same log* [20, 45]. For example, certificate mis-issuance would not be detected by a domain owner that monitors the logs if fraudulently issued certificates are shown to the clients selectively. A log that serves different versions of itself is said to present a *split view* [51]. Unless such log misbehaviour can be detected, we must trust it not to happen.

The solution to the split viewing problem is a gossip mechanism which ensures that everybody observes *the same* consistent log [45]. This assumption is simple in theory but remarkably hard in practice due to client privacy, varying threat models, and deployment challenges [51, 59]. While Google started on a package that supports minimal gossip [29] and the mechanisms of Nordberg *et al.* [51], there is "next to no deployment in the wild" [34]. To this end, we propose a gossip mechanism that helps detecting split-view attacks retroactively based on the idea of packet processors, such as routers and middleboxes, that *aggregate* Signed Tree Heads (STHs)—succinct representations of the logs' states—that are exposed to the network *in plaintext*. The aggregated STHs are then used to challenge the logs to prove consistency via an off-path, such that the logs cannot distinguish between challenges that come from different aggregators. Given this indistinguishability assumption, it is non-trivial to serve a consistent split-view to an unknown location [35]. Thus, all aggregators must be on the same view, and accordingly all clients that are covered by these aggregators must also be on the same view *despite not doing any explicit gossip themselves* because gossip is provided as-a-service by the network. An isolated client (i.e., untrusted network path to the aggregator) is notably beyond reach of any retroactive gossip [59].

The premise of having STHs in plaintext is controversial given current trends to encrypt transport protocols, which is otherwise an approach that combats inspection of network traffic and protocol ossification [31, 40]. We argue that keeping gossip related material in plaintext to support aggregation-based gossip comes with few downsides though: it is easy to implement, there are no major negative privacy impacts, and it would offer significant protection for a large portion of the Internet with a realistic threat model *despite relatively small deployment efforts*. The three main limitations are no protection against isolated clients, reliance on clients that fetch STHs from the logs in plaintext, and possible concerns surrounding protocol ossification [40]. Our contributions are:

- Design and security considerations for a network-based gossip mechanism that passively aggregates STHs to verify log consistency off-path (Section 3).

- Generic implementations of the aggregation step using P4 [12] and XDP [39] for plaintext STHs, supporting line-speed packet processing on systems that range from switches, routers, network interface cards, and Linux (Section 4).
- A simulation based on RIPE Atlas measurements that evaluate the impact of deploying aggregation-based gossip at ASes and IXPs. Our evaluation shows that incremental roll-out at well-connected locations would protect a significant portion of all Internet clients from undetected split views (Section 5).

Besides the sections referenced above, the paper introduces necessary background in Section 2 and provides discussion, conclusion, and future work in Sections 6–8. A full version with additional implementation details is available online [23].

2 Background

First additional prerequisites are provided on CT and the status quo, then the techniques which allow us to program custom packet processors are introduced.

2.1 Certificate Transparency

The main motivation of CT is that the CA ecosystem is error-prone [44]: a CA can normally issue certificates for *any* domain name, and given that there are hundreds of trusted CAs an attacker only needs to target the weakest link [30]. While the requirement of CT logging all certificates cannot prevent mis-issuance proactively, it allows anyone to detect it retroactively by monitoring the logs [45]. After a log promises to include a certificate by issuing a Signed Certificate Timestamp (SCT), a new STH including the appended certificate must be issued within a Maximum Merge Delay (MMD). Typically, logs use 24 hour MMDs. Should non-included SCTs and/or inconsistent STHs be found, binding evidence of misbehaviour exists because these statements are digitally signed by the logs. Other than MMD a log's policy defines parameters such as STH frequency: the number of STHs that can be issued during an MMD, making it harder to track clients [51].

CT is being deployed across Apple's platform [41] and Google's Chrome [52]. The status quo is to trust a CA-signed certificate if it is accompanied by two or more SCTs, thereby relying on at least one log to append each certificate so that mis-issuance can be detected by monitors that inspect the logs. The next step of this incremental deployment is to *verify* that these certificates are logged by querying for inclusion [58], and that the log's append-only property is respected by challenging the log to prove STH consistency. Finally, to fully distrust CT logs we need mechanisms that detect split-views. One such mechanism which is based on programmable packet processors (introduced next) is presented in Section 3, and it is compared to related work on CT gossip in Section 6.

2.2 Programmable Data Planes

Packet processors such as switches, routers, and network interface cards are typically integrated tightly using customized hardware and application-specific integrated circuits. This inflexible design limits the potential for innovation and leads to long product upgrade cycles, where it takes *years* to introduce new processing capabilities and support for different protocols and header fields (mostly following lengthy standardization cycles). The recent shift towards flexible *match+action* packet-processing pipelines—including RMT [13], Intel Flexpipe [3], Cavium XPA [2], and Barefoot Tofino [5]—now have the potential to change the way in which packet processing hardware is implemented: it enables programmability using high-level languages, such as P4, while at the same time maintaining performance comparable to fixed-function chips.

2.2.1 P4

The main goal of P4 is to simplify programming of protocol-independent packet processors by providing an abstract programming model for the network data plane [12]. In this setting, the functionality of a packet processing device is specified without assuming any hardwired protocols and headers. Consequently, a P4 program must parse headers and connect the values of those protocol fields to the actions that should be executed based on a pipeline of reconfigurable match+action tables. Based on the specified P4 code, a front-end compiler generates a high-level intermediate representation that a back-end compiler uses to create a target-dependent program representation. Compilers are available for several platforms, including the software-based simple switch architecture [1], SDNet for Xilinx NetFPGA boards [15], and Netronome's smart network interfaces [4]. It is also possible to compile basic P4 programs into eBPF byte code [16].

2.2.2 XDP

The Berkeley Packet Filter (BPF) is a Linux-based packet filtering mechanism [47]. Verified bytecode is injected from user space, and executed for each received packet in kernel space by a just-in-time compiler. Extended BPF (eBPF) enhances the original BPF concept, enabling faster runtime and many new features. For example, an eBPF program can be attached to the Linux traffic control tool `tc`, and additional hooks were defined for a faster eXpress Data Path (XDP) [39]. In contrast to the Intel Data Plane Development Kit (DPDK), which runs in user space and completely controls a given network interface that supports a DPDK driver, XDP cooperates with the Linux stack to achieve fast, programmable, and reconfigurable packet processing using C-like programs.

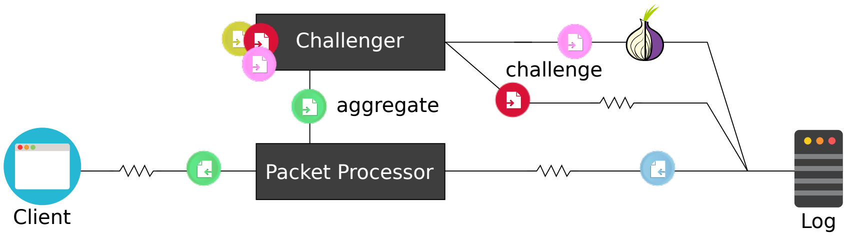


Figure 1: Packet processor that aggregates plaintext STHs for off-path verification.

3 Design

An overview of aggregation-based gossip is shown in Figure 1. The setting consists of logs that send plaintext STHs to clients over a network, and as part of the network inline *packet processors* passively aggregate observed STHs to their own off-path *challengers* which challenge the logs to prove consistency. A log cannot present split views to different clients that share an aggregating vantage point because it would trivially be detected by that vantage point's challenger. A log also cannot present a persistent split view to different challengers because they are off-path in the sense that they are indistinguishable from one another. This means that every client that is covered by an aggregator must be on the same view because at least one challenger will otherwise detect an inconsistency and report it. A client that is not directly covered by an aggregator may receive indirect protection in the form of herd immunity. This is discussed in Section 7.4.

3.1 Threat Model and Security Notion

The overarching threat is undetectable domain impersonation (ex-post) by an attacker that is capable of compromising at least one CA and a sufficient number of CT logs to convince a client into accepting a forged certificate. We assume that any illegitimately issued certificate would be detected by the legitimate domain owner through self or delegated third-party monitoring. This means that an attacker must either provide a split view towards the victim or the monitoring entity. We also assume that clients query the logs for certificate inclusion based on STHs that they acquire from the logs via plaintext mechanisms that packet processors can observe, and that some other entities than challengers process STHs using the chosen off-paths (Section 7.1). We do not account for the fact that CA compromises may be detected by other means, focusing solely on split-viewing CT logs.

3.1.1 Limitations

Our gossip mechanism is limited to STHs that packet processors can observe. As such, a client isolated by an attacker is not protected. We limit ourselves to attackers that act over a network some distance (in the sense of network path length) from a client in plaintext so that aggregation can take place. Our limitations and assumptions are further discussed in Section 7.1.

3.1.2 Attackers

Exceptionally powerful attackers can isolate clients, *but clients are not necessarily easy to isolate* for a significant number of relevant attackers. Isolation may require physical control over a device [32], clients may be using anonymity networks like Tor where path selection is inherently unpredictable [27], or sufficiently large parts of the network cannot be controlled to ensure that no aggregation takes place. This may be the case if we consider a nation state actor attacking another nation state actor, the prevalence of edge security middleboxes, and that home routers or NICs nearby the clients could aggregate. Any attacker that cannot account for these considerations is within our threat model.

3.1.3 Security Notion

To bypass our approach towards gossip an adaptive attacker may attempt to actively probe the network for aggregating packet processors. This leads us to the key security notion: *aggregation indistinguishability*. An attacker should not be able to determine if a packet processor is aggregating STHs. The importance of aggregation indistinguishability motivates the design of our gossip mechanism into two distinct components: aggregation that takes place inline at packet processors, and periodic off-path log challenging that checks whether the observed STHs are consistent.

3.2 Packet Processor Aggregation

An aggregating packet-processor determines for each packet if it is STH-related. If so, the packet is cloned and sent to a challenging component for off-path verification. The exact definition of *STH-related* depends on the plaintext source, but it is ultimately the process of inspecting multiple packet headers such as transport protocol and port number. It should be noted that the original packet must not be dropped or modified. For example, an aggregator would have a trivial aggregation distinguisher if it dropped any malformed STH.

For each aggregating packet processor we have to take IP fragmentation into consideration. Recall that IP fragmentation usually occurs when a packet is larger than the MTU, splitting it into multiple smaller IP packets that are reassembled at the destination host. Normally, an STH should not be fragmented because it is much smaller than the de-facto minimum MTU of (at least) 576 bytes [14, 24], but an attacker could use fragmentation to *intentionally* spread expected headers across multiple packets. Assuming stateless packet processing, an aggregator cannot identify such fragmented packets as STH-related because some header would be absent (cf. stateless firewalls). All tiny fragments should therefore be aggregated to account for intentional IP fragmentation, which appears to have little or no impact on normal traffic because tiny fragments are anomalies [55]. The threat of multi-path fragmentation is discussed in Section 7.1.

Large traffic loads must also be taken into account. If an aggregating packet processor degrades in performance as the portion of STH-related traffic increases, a distant attacker may probe for such behaviour to determine if a path contains an aggregator. Each *implementation* must therefore be evaluated individually for such behaviour, and if trivial aggregation distinguishers exist this needs to be solved. For example, STH-related traffic could be aggregated probabilistically to reduce the amount of work. Another option is to load-balance the traffic before aggregation, i.e., avoid worst-case loads that cannot be handled.

3.3 Off-Path Log Challenging

A challenger is setup to listen for aggregated traffic, reassembling IP fragments and storing the aggregated STHs for periodic off-path verification. Periodic off-path verification means that the challenger challenges the log based on its own (off-path fetched) STHs and the observed (aggregated) STHs to verify log consistency periodically, e.g., every day. The definition of *off-path* is that the challenger must not be linkable to its aggregating packet processor(s) or any other challenger (including itself). Without an off-path there is no gossip step amongst aggregator-challenger instances that are operated by different actors, and our approach towards gossip would only assert that clients behind the same vantage point observe the same logs. If a log cannot distinguish between different challengers due to the use of off-paths, however, it is non-trivial to maintain a targeted split-view towards an unknown location. Therefore, we get a form of *implicit gossip* [35] because at least one challenger would detect an inconsistency unless everybody observes the same log. If every challenger observes the same log, so does every client that is covered by an aggregating packet processor. Notably the challenger component *does not run inline* to avoid timing distinguishers. Note that there are other important considerations when implementing a challenger, as discussed in Section 7.1.

4 Distinguishability Experiments

There are many different ways to implement the aggregation step. We decided to use P4 and XDP because a large variety of programmable packet processors support these languages (Section 2.2). The aggregated plaintext source is assumed to be CT-over-DNS [43], which means that a client obtains STHs by fetching IN TXT resource records. Since languages for programmable packet processors are somewhat restricted, we facilitated packet processing by requiring that at most one STH is sent per UDP packet. This is reasonable because logs should only have one *most recent* STH. A DNS STH is roughly 170 bytes without any packet headers and should normally not be fragmented, but to ensure that we do not miss any intentionally fragmented STHs we aggregate every tiny fragment. We did not implement the challenging component because it is relatively easy given an existing off-path. Should any scalability issue arise for the challenger there is nothing that prevents a distributed front-end that processes the aggregated material before storage. Storage is not an issue because there are only a limited amount of unique STHs per day and log (one new STH per hour is a common policy, and browsers recognize ≈ 40 logs). Further implementation details can be found online [6, 23].

4.1 Setup

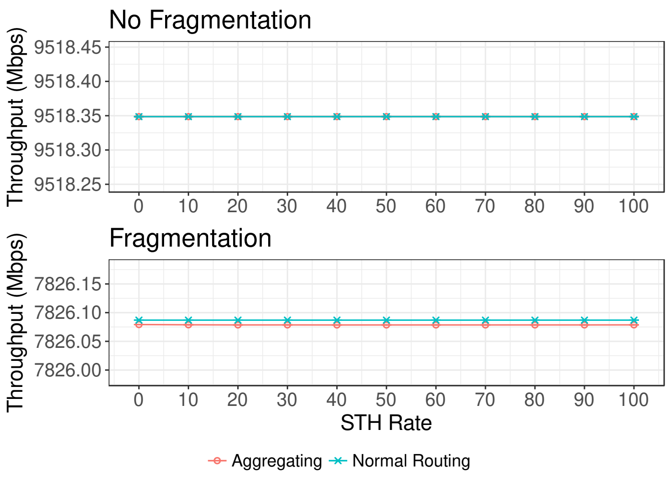
We used a test-bed consisting of a traffic generator, a traffic receiver, and an aggregating target in between. The first target is a P4-enabled NetFPGA SUME board that runs an adapted version of our P4 reference implementation. The second target is a net-next kernel v4.17.0-rc6 Linux machine that runs XDP on one core with a 10 Gb SFP+ X520 82599ES Intel card, a 3.6 GHz Intel Core i7-4790 CPU, and 16 GB of RAM at 1600 MHz (Hynix/Hyundai). We would like to determine whether there are any aggregation distinguishers as the fraction of STHs (experiment 1) and tiny fragments (experiment 2) in the traffic is increased from 0–100%, i.e., does performance degrade as a function of STH-related rate? Non-fragmented STH packets are 411 bytes (we used excessively large DNS headers to maximize the packet parsing overhead), and tiny fragments are 64 bytes. All background traffic have the same packet sizes but is not deemed STH-related.

4.2 Results

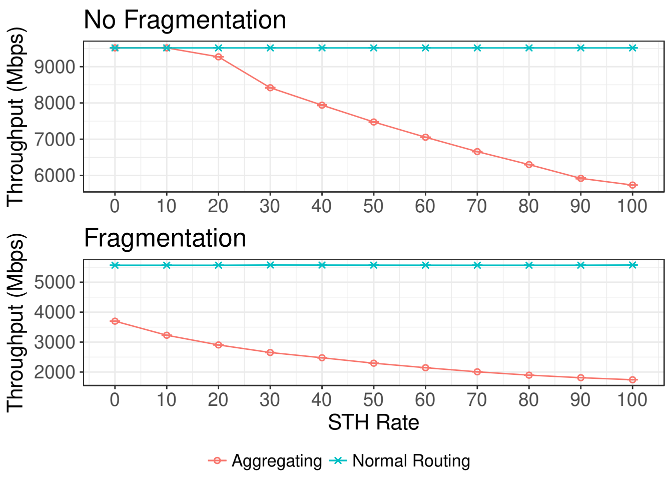
Figure 2a shows throughput as a function of STH-related rate for the P4-enabled NetFPGA. While we were unable to observe any distinguisher between normal routing and the edge case of 100% aggregation for non-fragmented STH packets, there is a small constant throughput difference for tiny fragments (7.5 Kbps). This is a non-negligible *program distinguisher* if a packet processor is physically isolated as in our benchmark, i.e., something other than a routing program is running but it is not necessarily an aggregator because performance does not degrade as a function of increased STH-related rate. However, we found such degradation behaviour for the single-core XDP case (Figure 2b). If line-speed is higher than 2 Gbps, STHs could be aggregated probabilistically or traffic could be load-balanced to *overcome* this issue.

4.3 Lessons Learned

P4-NetFPGA provides aggregation indistinguishability regardless of STH load. For XDP, it depends on the scenario: what is the line-rate criteria and how many cores are available. For example, five cores support 10 Gbps aggregation indistinguishability without probabilistic filtering or load balancing.



(a) P4 NetFPGA



(b) XDP on a single core

Figure 2: Throughput as a function of STH-related traffic that is aggregated.

5 Estimated Impact of Deployment

We conducted 20 daily traceroute measurements during spring 2018 on the RIPE Atlas platform to evaluate the effectiveness of aggregation-based gossip. The basic idea is to look at client coverage as central ASes and IXPs aggregate STHs. If any significant client coverage can be achieved, the likelihood of pulling off an undetected split-view will be small.

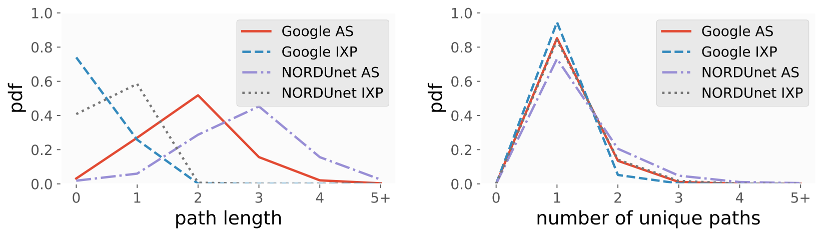


Figure 3: Path length and stability towards Google and NORDUnet.

5.1 Setup

We scheduled RIPE Atlas measurements from roughly 3500 unique ASes that represent 40% of the IPv4 space, trace-routing Google's authoritative CT-over-DNS server and NORDUnet's CT log to simulate clients that fetch DNS STHs in plaintext. Each traceroute result is a list of traversed IPs, and it can be translated into the corresponding ASes and IXPs using public data sets [7, 18]. In other words, traversed ASes and IXPs can be determined for each probe. Since we are interested in client coverage as ASes and IXPs aggregate, each probe is weighted by the IPv4 space of its AS. While an IP address is an imperfect representation of a client, e.g., an IP may be unused or reused, it gives a decent idea of how significant it is to cover a given probe.

5.2 Results

Figure 3 shows AS/IXP path length and stability from the probes to the targets. If the AS path length is one, a single AS is traversed *before reaching the target*. It is evident that an AS path tends to be one hop longer towards NORDUnet than Google because there is a rough off-by-one offset on the x-axis. A similar trend of greater path length towards NORDUnet can be observed for IXPs. For example, 74.0% of all paths traversed no IXP towards Google, but 58.5% of all paths traversed a single IXP towards NORDUnet. These results can be explained by infrastructural differences of our targets: since Google is a worldwide actor an average path should be shorter than compared to a region-restricted actor like NORDUnet. We also observed that AS and IXP paths tend to be quite stable over 20 days (the duration of our measurements). I.e., if AS a and b are traversed it is unlikely to suddenly be routed via AS c.

Figure 4 shows coverage of the RIPE Atlas network as $1 \ldots n$ actors aggregate STHs. For example, 100% and 50% coverage means that at least 40% and 20% of the full IPv4 space is covered. The aggregating ASes and IXPs were selected based on the most commonly traversed vantage points in our measurements (Pop), as well as CAIDA's largest AS ranking [17]. We found that more coverage is achieved when targeting NORDUnet than Google. This is expected given that the paths tend to be longer. If CAIDA's top-32 enabled aggregation, the coverage would be significant towards Google (31.6%) and NORDUnet (58.1%).

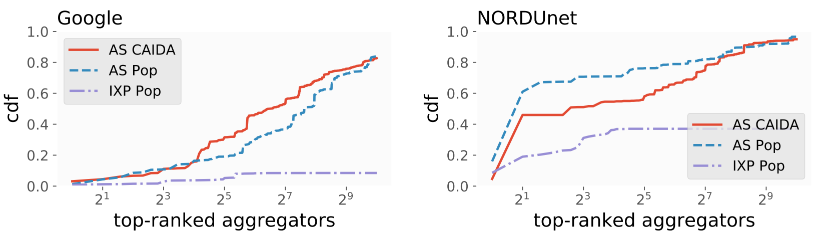


Figure 4: Coverage as a function of aggregation opt-in.

5.3 Lessons Learned

A vast majority of all clients traverse *at least* one AS that could aggregate. It is relatively rare to traverse IXPs towards Google but not NORDUnet. We also learned that paths tends to be stable, which means that the time until split view detection would be at least 20 days *if* it is possible to find an unprotected client. This increases the importance of aggregation indistinguishability. Finally, we identified vantage points that are commonly traversed using Pop, and these vantage points are represented well by CAIDA's independent AS ranking. Little opt-in from ASes and IXPs provides significant coverage against an attacker that is relatively close to a client (cf. world-wide infrastructure of Google). Although we got better coverage for NORDUnet, any weak attacker would approach Google's coverage by renting infrastructure nearby. Any weak attacker could also circumvent IXP aggregation by detecting the IXP itself [49]. As such, top-ranked AS aggregation should give the best split-view protection.

6 Related Work

Earlier approaches towards CT gossip are categorized as *proactive* or *retroactive* in Figure 5. We consider an approach proactive if gossip takes place *before* SCTs and/or STHs reach the broader audience of clients. Syta *et al.* proposed proactive witness cosigning, in which an STH is collectively signed by a *large* number of witnesses and at most a fraction of those can be faulty to ensure that a benevolent witness observed an STH [59]. STH cross-logging [29, 36, 37] is similar in that an STH must be proactively disclosed in another transparency log to be trusted, avoiding any additional cosigning infrastructure at the cost of reducing the size and diversity of the witnessing group. Tomescu and Devadas [60] suggested a similar cross-logging scheme, but split-view detection is instead reduced to the difficulty of forking the Bitcoin blockchain (big-O cost of downloading all block headers as a TLS client). The final proactive approach is STH pushing, where a trusted third-party pushes the same verified STH history to a base of clients [58].

We consider a gossip mechanism retroactive if gossip takes place *after* SCTs and/or STHs reach the broader audience of clients. Chuat *et al.* proposed that TLS clients and TLS servers be modified to pool exchanged STHs and relevant

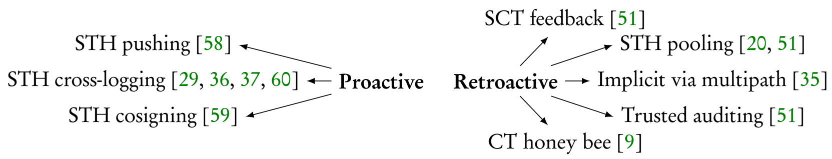


Figure 5: A categorization of approaches towards CT gossip.

consistency proofs [20]. Nordberg *et al.* continued this line of work, suggesting privacy-preserving client-server pollination of fresh STHs [51]. Nordberg *et al.* also proposed that clients feedback SCTs and certificate chains on every server revisit, and that trusted auditor relationships could be engaged if privacy need not be protected. The latter is somewhat similar to the formalized client-monitor gossip of Chase and Meiklejohn [19], as well as the CT honey bee project where a client process fetches and submits STHs to a pre-compiled list of auditors [9]. Laurie suggested that a client can resolve privacy-sensitive SCTs to privacy-insensitive STHs via DNS (which are easier to gossip) [43]. Private information retrievals could likely achieve something similar [46]. Assuming that TLS clients are indistinguishable from one another, split-view detection could also be implicit as proposed by Gunn *et al.* for the verifiable key-value store CONIKS [35, 48].

Given that aggregation-based gossip takes place after an STH is issued, it is a retroactive approach. As such, we cannot protect an isolated client from split-views [59]. Similar to STH pooling and STH pollination, we rely on client-driven communication and an existing infrastructure of packet processors to aggregate. Our off-path verification is based on the same multi-path probing and indistinguishability assumptions as Gunn *et al.* [8, 35, 61]. Further, given that aggregation is application neutral and deployable on hosts, it could provide gossip *for* the CT honey bee project (assuming plaintext STHs) and any other transparency application like Trillian [33]. Another benefit when compared to browsing-centric and vendor-specific approaches is that a plethora of HTTPS clients are covered, ranging from niche web browsers to command line tools and embedded libraries that are vital to protect but yet lack the resources of major browser vendors [10, 25]. Our approach coexists well with witness cosigning and cross-logging due to different threat models, but not necessarily STH pushing if the secure channel is encrypted (no need to fetch what a trusted party provides).

7 Discussion

Next we discuss assumptions, limitations and deployment, showing that our approach towards retroactive gossip can be deployed to detect split-views by many relevant attackers with relatively little effort. The main drawback is reliance on clients fetching STHs in plaintext, e.g., using CT-over-DNS [43].

7.1 Assumptions and Limitations

Aggregation-based gossip is limited to network traffic that packet processors can observe. The strongest type of attacker in this setting—who can completely isolate a client—trivially defeats our gossip mechanism and other retroactive approaches in the literature (see Section 6). A weaker attacker cannot isolate a client, but is located nearby in a network path length sense. This limits the opportunity for packet processor aggregation, but an attacker cannot rule it out given aggregation indistinguishability. Section 4 showed based on performance that it is non-trivial to distinguish between (non-)aggregating packet processors on two different targets using P4 and XDP. Off-path challengers must also be indistinguishable from one another to achieve *implicit gossip*. While we suggested the use of anonymity networks like Tor, a prerequisite is that this is in and of itself not an aggregation distinguisher. Therefore, we assume that other entities also use off-paths to fetch and verify STHs. The fact that a unique STH *is not audited* from an off-path could also be an aggregation distinguisher. To avoid this we could rely on a verifiable STH history [50] and wait until the next MMD to audit or simply monitor the full log so that consistency proofs are unnecessary.

The existence of multiple network paths are fundamental to the structure and functioning of the Internet. A weak attacker may use IP fragmentation such that each individual STH fragment is injected from a different location to make aggregation harder, approaching the capabilities of a stronger attacker that is located closer to the client. This is further exacerbated by the deployment of multi-path transport protocols like MPTCP (which can also be fragmented). Looking back at our RIPE Atlas measurements in Section 5, the results towards Google's world-wide infrastructure better represent an active attacker that takes *some* measures to circumvent aggregation by approaching a client nearby the edge. Given that the likelihood of aggregation is high if *any* IXP is present (Figure 4), aggregation at well-connected IXPs are most likely to be circumvented.

7.2 Deployment

Besides aggregating at strategic locations in the Internet's backbone, ISPs and enterprise networks have the opportunity to protect all of their clients with relatively little effort. Deployment of special-purpose middleboxes are already prevalent in these environments, and then the inconvenience of fragmentation tends to go away due to features such as packet reassembly. Further, an attacker cannot trivially circumvent the edge of a network topology—especially not if aggregation takes place on an end-system: all fragments are needed to reassemble a packet, which means that multi-path fragmentation is no longer a threat. If aggregation-based gossip is deployed on an end-system, STHs could be hooked using other approaches than P4/XDP. For example, shim-layers that intercept TLS certificates higher up in the networking stack were already proposed by Bates *et al.* [11] and O'Neill *et al.* [53]. In this setting, an end-system is viewed as the aggregating packet processor, and it reports back to an off-path challenger that may be a local process running on the same system or a remote entity, e.g., a TelCo could host challengers that collect aggregated STHs from smartphones.

While we looked at programming physical packet processors like routers, STH aggregation could be approached in hypervisors and software switches [54] to protect many virtual hosts. If CT-over-DNS is used to fetch STHs, it would be promising to output DNS server caches to implement the aggregation step. Similar to DNS servers, so called Tor exist relays also operate DNS caches. In other words, P4 and XDP are only examples of how to *instantiate* the aggregation step. Depending on the used plaintext source, packet processor, and network topology other approaches may be more suitable, e.g., C for vendor-specific middleboxes.

7.3 Retroactive Gossip Benefits From Plaintext

As opposed to an Internet core that only forwards IP packets, extra functionality is often embedded which causes complex processing dependencies and protocol ossification [40]. Many security and protocol issues were found for middleboxes that provides extra functionality [31, 42], resulting in the mindset that *everything* should be encrypted [42]. Our work is controversial because it goes against this mindset and advocates that STHs should be communicated in plaintext. We argue that this makes sense in the context of STHs due to the absence of privacy concerns and because the entire point of gossip is to make STHs *available* (rather than end-to-end). The idea of intentionally exposing information to the network is not new, e.g., MPQUIC is designed to support traffic shaping [21].

While we used CT-over-DNS as a plaintext source, there is a push towards DNS-over-TLS [26] and DNS-over-HTTPS [38]. Wide use of these approaches could undermine our gossip mechanism, but ironically the security of TLS could be jeopardized unless gossip is deployed. In other words, long term gossip is an essential component of CT and other transparency logs to avoid becoming yet another class of trusted third-parties. If proactive approaches such as witness cosigning are rejected in favour of retroactive mechanisms, then ensuring that STHs are widely spread and easily accessible is vital. An STH needs no secrecy if the appropriate measures are taken to make it privacy-insensitive [51]. While secure channels also provide integrity and replay protection, an STH is already signed by logs and freshness is covered by MMDs, as well as issue frequency to protect privacy. A valid argument against exposing any plaintext to the network is protocol ossification. We emphasize that our design motivates why packet processors should fail open: otherwise there is no aggregation indistinguishability. Note that there are other plaintext sources than CT-over-DNS that could be aggregated. However, if these sources require stream-reassembly it is generally hard to process in languages such as P4 and XDP [22].

7.4 Indistinguishability and Herd Immunity

An attacker that gains control over a CT log is bound to be more risk averse than an attacker that compromises a CA. There is an order of magnitude fewer logs than CAs, and client vendors are likely going to be exceptionally picky when it comes to accepted and rejected logs. We have already seen examples of this, including Google Chrome disqualifying logs that made mistakes: Izenpe used the same key for production and testing [56], and Venafi suffered from an unfortunate power outage [57]. Risk averse attackers combined with packet processors that are aggregation indistinguishable may lead to *herd immunity*: despite a significant fraction of clients that lack aggregators, indirect protection may be provided because the risk of eventual detection is unacceptable to many attackers. Hof and Carle [37] and Nordberg *et al.* [51] discussed herd immunity briefly before us. While herd immunity is promising, it should be noted that aggregation distinguishable packet processors at *the edge of a network topology* may be acceptable for some. In other words, if an aggregator cannot be circumvented but it is detectable split-views would still be deterred against covered clients if the challenger is off-path.

8 Conclusion and Future Work

Wide spread modifications of TLS clients are inevitable to support CT gossip. We propose that these modifications include challenging the logs to prove certificate inclusion based on STHs *fetched in plaintext*, thereby enabling the traversed packet processors to assist in split view detection retroactively by aggregating STHs for periodic off-path verification. Our results show that the aggregation-step can be implemented without throughput-based distinguishers for a distant attacker, and that our approach offers rapid incremental deployment with high impact on a significant fraction of Internet users. Beyond being an application neutral approach that is complementary to proactive gossip, a compelling aspect is that core packet processors are used (rather than clients) as a key building block: should a consistency issue arise, it is already in the hands of an actor that is better equipped to investigate the cause manually. Further, considering that far from all TLS clients are backed by big browser vendors (not to mention other use-cases of transparency logs in general) it is likely a long-term win to avoid pushing complex retroactive gossip logic into all the different types of clients when there are orders of magnitudes fewer packet processors that could aggregate to their own off-path challengers. Future work includes different instantiations of the aggregation step and evaluating whether aggregation indistinguishability is provided based on throughput and/or latency. The setting may also change in some scenarios, e.g., if DNS caches are aggregated the transport need not be plaintext.

Acknowledgements

We would like to thank Stefan Alfredsson and Philipp Winter for their RIPE Atlas credits, as well as Jonas Karlsson and Ricardo Santos for helping with the NetFPGA setup. We also received funding from the HITS research profile which is funded by the Swedish Knowledge Foundation.

References

[1] Behavioral model repository. https://github.com/p4lang/behavioral-model, accessed 2019-09-04.

[2] Cavium and XPliant introduce a fully programmable switch silicon family scaling to 3.2 terabits per second. https://cavium.com/newsevents-cavium-and-xpliant-introduce-a-fully-programmable-switch-silicon-family.html, accessed 2019-09-04.

[3] Intel ethernet switch FM600 series: 10/40 GbE low latency switching silicon. https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/ethernet-switch-fm6000-series-brief.pdf, accessed 2019-09-04.

[4] Programming NFP with P4 and C. https://www.netronome.com/media/redactor_files/WP_Programming_with_P4_and_C.pdf, accessed 2019-09-04.

[5] Tofino: World's fastest P4-programmable ethernet switch ASICs. https://barefootnetworks.com/products/brief-tofino/, accessed 2019-09-04.

[6] Paper artifact. https://github.com/rgdd/ctga, 2018.

[7] The Routeviews MRT format RIBs and UPDATEs dataset. http://archive.routeviews.org/bgpdata/2018.03/RIBS/, accessed 2019-09-04, March 2018.

[8] Mansoor Alicherry and Angelos D. Keromytis. DoubleCheck: Multi-path verification against man-in-the-middle attacks. In *ISCC*, 2009.

[9] Andrew Ayer. Lightweight program that pollinates STHs between Certificte Transparency logs and auditors. https://github.com/SSLMate/ct-honeybee, accessed 2019-09-04.

[10] Michael Backes, Sven Bugiel, and Erik Derr. Reliable third-party library detection in Android and its security applications. In *CCS*, 2016.

[11] Adam Bates, Joe Pletcher, Tyler Nichols, Braden Hollembaek, Dave Tian, Kevin R. B. Butler, and Abdulrahman Alkhelaifi. Securing SSL certificate verification through dynamic linking. In *CCS*, 2014.

[12] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming protocol-independent packet processors. *CCR*, 44(3), 2014.

[13] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando A. Mujica, and Mark Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. In *ACM SIGCOMM*, 2013.

[14] Robert Braden. Requirements for Internet hosts—communication layers. RFC 1122, IETF, 1989.

[15] Gordon Brebner. P4 for an FPGA target. In *P4 Workshop*, 2015. https://p4workshop2015.sched.com/event/3ZQA/p4-for-an-fpga-target, accessed 2019-09-04.

[16] Mihai Budiu. Compiling P4 to eBPF. https://github.com/iovisor/bcc/tree/master/src/cc/frontends/p4, accessed 2019-09-04.

[17] CAIDA. ARank. http://as-rank.caida.org/, accessed 2019-09-04.

[18] CAIDA. The CAIDA UCSD IXPs dataset. https://www.caida.org/data/ixps/, accessed 2019-09-04, February 2018.

[19] Melissa Chase and Sarah Meiklejohn. Transparency overlays and applications. In *CCS*, 2016.

[20] Laurent Chuat, Pawel Szalachowski, Adrian Perrig, Ben Laurie, and Eran Messeri. Efficient gossip protocols for verifying the consistency of certificate logs. In *CNS*, 2015.

[21] Quentin De Coninck and Olivier Bonaventure. Multipath QUIC: Design and evaluation. In *CoNEXT*, 2017.

[22] Rasmus Dahlberg. Aggregating Certificate Transparency gossip using programmable packet processors. Master thesis, Karlstad University, 2018.

[23] Rasmus Dahlberg, Tobias Pulls, Jonathan Vestin, Toke Høiland-Jørgensen, and Andreas Kassler. Aggregation-based gossip for certificate transparency. *CoRR*, abs/1806.08817, 2019.

[24] Steve Deering and Robert Hinden. Internet protocol version 6 (IPv6) specification. RFC 8200, IETF, 2017.

[25] Erik Derr, Sven Bugiel, Sascha Fahl, Yasemin Acar, and Michael Backes. Keep me updated: An empirical study of third-party library updatability on Android. In *CCS*, 2017.

[26] Sara Dickinson, Dan Gillmor, and Tirumaleswar Reddy. Usage profiles for DNS over TLS and DNS over DTLS. RFC 8310, IETF, 2016.

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

[28] Benjamin Dowling, Felix Günther, Udyani Herath, and Douglas Stebila. Secure logging schemes and Certificate Transparency. In *ESORICS*, 2016.

[29] David Drysdale. Minimal gossip. https://github.com/google/certificate-transparency-go/blob/master/gossip/minimal, accessed 2019-09-04.

[30] Zakir Durumeric, James Kasten, Michael Bailey, and J. Alex Halderman. Analysis of the HTTPS certificate ecosystem. In *IMC*, 2013.

[31] Zakir Durumeric, Zane Ma, Drew Springall, Richard Barnes, Nick Sullivan, Elie Bursztein, Michael Bailey, J. Alex Halderman, and Vern Paxson. The security impact of HTTPS interception. In *NDSS*, 2017.

[32] EFF. Apple challenges FBI: All writs act order (CA). https://www.eff.org/cases/apple-challenges-fbi-all-writs-act-order, accessed 2019-09-04.

[33] Adam Eijdenberg, Ben Laurie, and Al Cutter. Verifiable data structures. https://github.com/google/trillian/blob/master/docs/VerifiableDataStructures.pdf, accessed 2019-09-04.

[34] Oliver Gasser, Benjamin Hof, Max Helm, Maciej Korczynski, Ralph Holz, and Georg Carle. In log we trust: Revealing poor security practices with Certificate Transparency logs and internet measurements. In *PAM*, 2018.

[35] Lachlan J. Gunn, Andrew Allison, and Derek Abbott. Safety in numbers: Anonymization makes keyservers trustworthy. In *HotPETs*, 2017.

[36] Benjamin Hof. STH cross logging. Internet-draft draft-hof-trans-cross-00, IETF, 2017.

[37] Benjamin Hof and Georg Carle. Software distribution transparency and auditability. *CoRR*, abs/1711.07278, 2017.

[38] Paul Hoffman and Patrick McManus. DNS queries over HTTPS (DoH). RFC 8484, IETF, 2018.

[39] Toke Høiland-Jørgensen, Jesper Dangaard Brouer, Daniel Borkmann, John Fastabend, Tom Herbert, David Ahern, and David Miller. The express data path: fast programmable packet processing in the operating system kernel. In *CoNEXT*, 2018.

[40] Michio Honda, Yoshifumi Nishida, Costin Raiciu, Adam Greenhalgh, Mark Handley, and Hideyuki Tokuda. Is it still possible to extend TCP? In *IMC*, 2011.

[41] Apple Inc. Apple's Certificate Transparency policy. https://support.apple.com/en-us/HT205280, accessed 2019-09-04.

[42] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan R. Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, and Zhongyi Shi. The QUIC transport protocol: Design and internet-scale deployment. In *SIGCOMM*, 2017.

[43] Ben Laurie. Certificate Transparency over DNS. https://github.com/google/certificate-transparency-rfcs/blob/master/dns, accessed 2019-09-04.

[44] Ben Laurie. Certificate Transparency. *ACM Queue*, 12(8), 2014.

[45] Ben Laurie, Adam Langley, and Emilia Kasper. Certificate Transparency. RFC 6962, IETF, 2013.

[46] Wouter Lueks and Ian Goldberg. Sublinear scaling for multi-client private information retrieval. In *FC*, 2015.

[47] Steven McCanne and Van Jacobson. The BSD packet filter: A new architecture for user-level packet capture. In *Usenix Winter Technical Conference*, 1993.

[48] Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Edward W. Felten, and Michael J. Freedman. CONIKS: Bringing key transparency to end users. In *USENIX Security*, 2015.

[49] George Nomikos and Xenofontas A. Dimitropoulos. traIXroute: Detecting IXPs in traceroute paths. In *PAM*, 2016.

[50] Linus Nordberg. Re: [Trans] Providing the history of STHs a log has issued (in 6962-bis). https://mailarchive.ietf.org/arch/msg/trans/JbFiwO90PjcYzXrEgh-Y7bFG5Fw, accessed 2019-09-04.

[51] Linus Nordberg, Daniel Kahn Gillmor, and Tom Ritter. Gossiping in CT. Internet-draft draft-ietf-trans-gossip-05, IETF, 2018.

[52] Devon O'Brien. Certificate Transparency enforcement in Google Chrome. https://groups.google.com/a/chromium.org/forum/#!msg/ct-policy/wHILiYf31DE/iMFmpMEkAQAJ, accessed 2019-09-04.

[53] Mark O'Neill, Scott Heidbrink, Scott Ruoti, Jordan Whitehead, Dan Bunker, Luke Dickinson, Travis Hendershot, Joshua Reynolds, Kent E. Seamons, and Daniel Zappala. TrustBase: An architecture to repair and strengthen certificate-based authentication. In *USENIX Security*, 2017.

[54] Muhammad Shahbaz, Sean Choi, Ben Pfaff, Changhoon Kim, Nick Feamster, Nick McKeown, and Jennifer Rexford. PISCES: a programmable, protocol-independent software switch. In *ACM SIGCOMM*, 2016.

[55] Colleen Shannon, David Moore, and Kimberly C. Claffy. Beyond folklore: Observations on fragmented traffic. *IEEE/ACM Trans. Netw.*, 10(6), 2002.

[56] Ryan Sleevi. Upcoming CT log removal: Izenpe. https://groups.google.com/a/chromium.org/forum/#!topic/ct-policy/qOorKuhL1vA, accessed 2019-09-04.

[57] Ryan Sleevi. Upcoming log removal: Venafi CT log server. https://groups.google.com/a/chromium.org/forum/#!topic/ct-policy/KMAcNT3asTQ, accessed 2019-09-04.

[58] Ryan Sleevi and Eran Messeri. Certificate Transparency in Chrome: Monitoring CT logs consistency. https://docs.google.com/document/d/1FP5J5Sfsg0OR9P4YT0q1dM02iavhi8ix1mZlZe_z-ls/edit?pref=2&pli=1, accessed 2019-09-04.

[59] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. Keeping authorities "honest or bust" with decentralized witness cosigning. In *IEEE S&P*, 2016.

[60] Alin Tomescu and Srinivas Devadas. Catena: Efficient non-equivocation via Bitcoin. In *IEEE S&P*, 2017.

[61] Dan Wendlandt, David G. Andersen, and Adrian Perrig. Perspectives: Improving SSH-style host authentication with multi-path probing. In *USENIX ATC*, 2008.

Paper III

Privacy-Preserving & Incrementally-Deployable Support for Certificate Transparency in Tor

Reprinted from

PETS (2021)

Privacy-Preserving & Incrementally-Deployable Support for Certificate Transparency in Tor

Rasmus Dahlberg, Tobias Pulls, Tom Ritter, and Paul Syverson

Abstract

The security of the web improved greatly throughout the last couple of years. A large majority of the web is now served encrypted as part of HTTPS, and web browsers accordingly moved from positive to negative security indicators that warn the user if a connection is insecure. A secure connection requires that the server presents a valid certificate that binds the domain name in question to a public key. A certificate used to be valid if signed by a trusted Certificate Authority (CA), but web browsers like Google Chrome and Apple's Safari have additionally started to mandate Certificate Transparency (CT) logging to overcome the weakest-link security of the CA ecosystem. Tor and the Firefox-based Tor Browser have yet to enforce CT.

We present privacy-preserving and incrementally-deployable designs that add support for CT in Tor. Our designs go beyond the currently deployed CT enforcements that are based on blind trust: if a user that uses Tor Browser is man-in-the-middled over HTTPS, we probabilistically detect and disclose cryptographic evidence of CA and/or CT log misbehavior. The first design increment allows Tor to play a vital role in the overall goal of CT: detect mis-issued certificates and hold CAs accountable. We achieve this by randomly cross-logging a subset of certificates into other CT logs. The final increments hold misbehaving CT logs accountable, initially assuming that some logs are benign and then without any such assumption. Given that the current CT deployment lacks strong mechanisms to verify if log operators play by the rules, exposing misbehavior is important for the web in general and not just Tor. The full design turns Tor into a system for maintaining a probabilistically-verified view of the CT log ecosystem available from Tor's consensus. Each increment leading up to it preserves privacy due to and how we use Tor.

1 Introduction

Metrics reported by Google and Mozilla reveal that encryption on the web skyrocketed the past couple of years: at least 84% of all web pages load using HTTPS [24, 44]. An HTTPS connection is initiated by a TLS handshake where the client's web browser requires that the web server presents a valid certificate to authenticate the identity of the server, e.g., to make sure that the client who wants to visit `mozilla.org` is really connecting to Mozilla, and not, say, Google. A certificate specifies the cryptographic key-material for a given domain name, and it is considered valid if it is digitally signed by a Certificate Authority (CA) that the web browser trusts.

It is a long-known problem that the CA trust model suffers from weakest-link security: web browsers allow hundreds of CAs to sign arbitrary domain-name to key-bindings, which means that it suffices to compromise a single CA to acquire any certificate [9, 18]. Motivated by prominent CA compromises, such as the issuance of fraudulent certificates for `*.google.com`, `*.mozilla.org` and `*.torproject.org` by DigiNotar [49], multiple browser vendors mandated that certificates issued by CAs must be publicly disclosed in Certificate Transparency (CT) logs to be valid. The idea behind CT is that, by making all CA-issued certificates transparent, mis-issued ones can be detected *after the fact* [35, 36, 37]. The appropriate actions can then be taken to keep the wider web safe, e.g., by investigating the events that lead up to a particular incident, removing or limiting trust in the offending CA, and revoking affected certificates. Google Chrome and Apple's Safari currently enforce CT by augmenting the TLS handshake to require cryptographic proofs from the server that the presented certificate *will appear* in CT logs that the respective web browsers trust [3, 23].

In addition to increased encryption on the web, the ability to access it anonymously matured as well. Tor with its Tor Browser has millions of daily users [16, 40], and efforts are ongoing to mature the technology for wider use [43]. Tor Browser builds on-top of Mozilla's Firefox: it relays traffic between the user and the web server in question by routing everything through the Tor network, which is composed of thousands of volunteer-run relays that are located across the globe [64]. Just like attackers may wish to break security properties of HTTPS, it may also be of interest to break the anonymity provided by Tor. A common technique for deanonymization (known to be used in practice) is to compromise Tor Browser instead of circumventing the anonymity provided by Tor [5, 10, 22, 70]. Web browsers like Firefox (or forks thereof) are one of the most complex software types that are widely used today, leading to security vulnerabilities and clear incentives for exploitation. For example, the exploit acquisition platform Zerodium offers up to $\$100,000$ for a Firefox zero-day exploit that provides remote code execution and local privilege escalation (i.e., full control of the browser) [69].

An attacker that wishes to use such an exploit to compromise and then ultimately deanonymize a Tor Browser user has to deliver the exploit somehow. Since the web is mostly encrypted, this primarily needs to take place over an

HTTPS connection where the attacker controls the content returned by the web server. While there are numerous possible ways that the attacker can accomplish this, e.g., by compromising a web server that a subset of Tor Browser users visit, another option is to *impersonate* one or more web servers by acquiring fraudulent certificates. Due to the Tor network being run by volunteers, getting into a position to perform such an attack is relatively straightforward: the attacker can volunteer to run malicious exit relays [68]. The same is true for an attacker that wishes to man-in-the-middle connections made by Tor Browser users. In some cases a Tor Browser exploit may not even be needed for deanonymization, e.g., the attacker can observe if the user logs-on to a service linking an identity.

1.1 Introducing CTor

We propose an incrementally deployable and privacy-preserving design that is henceforth referred to as CTor. By bringing CT to Tor, HTTPS-based man-in-the-middle attacks against Tor Browser users can be detected *after the fact* when conducted by attackers that:

1. can acquire any certificate from a trusted CA,
2. with the necessary cryptographic proofs from enough CT logs so that Tor Browser accepts the certificate as valid without the attacker making it publicly available in any of the controlled logs, and
3. with the ability to gain full control of Tor Browser shortly after establishing an HTTPS connection.

The first and third capabilities are motivated directly by shortcomings in the CA ecosystem as well as how the anonymity of Tor Browser is known to be attacked. The second capability assumes the same starting point as Google Chrome and Apple's Safari, namely, that the logs are trusted to *promise* public logging, which is in contrast to being untrusted and thus forced to *prove* it. This is part of the gradual CT deployment that avoided breakage on the web [55]. Therefore, we start from the assumption that Tor Browser accepts a certificate as valid if accompanied by two independent promises of public logging. The limitation of such CT enforcement is that it is trivially bypassed by an attacker that controls two seemingly independent CT logs. This is not to say that trusting the log ecosystem would be an insignificant Tor Browser improvement when compared to no CT at all, but CTor takes us several steps further by relaxing and ultimately eliminating the trust which is currently (mis)placed in today's browser-recognized CT logs. We already observed instances of CT logs that happened to violate their promises of public logging [41], show inconsistent certificate contents to different parties [52, 53], and get their secret signing keys compromised due to disclosed remote code-execution vulnerabilities [50].

The first design increment uses the CT landscape against the attacker to ensure a non-zero (tweakable) probability of public disclosure *each time* a fraudulent certificate is used against Tor Browser. This is done by randomly adding a subset of presented certificates to CT logs that the attacker may not control (inferred from the accompanied promises of public logging). Such *certificate cross-logging* distributes trust across all CT logs, raising the bar towards unnoticed certificate mis-issuance. Motivated by factors like privacy, security and deployability, Tor Browser uses Tor relays as intermediates to cache and interact with CT logs on its behalf. Such deferred auditing is a fundamental part of our setting unless future distributed auditing mechanisms turn out to be non-interactive from the browser's perspective.

The next incremental step is to not only cross-log certificates but also their promises of public logging. While it requires an additional CT log API endpoint, it facilitates auditing of these promises if some logs are trustworthy. The full design also holds logs accountable but without any such assumption: Tor relays challenge the logs to prove correct operation with regards to a single fixed view in Tor's consensus, and potential issues are reported to auditors that investigate them further.

1.2 Contribution and Structure

Section 2 introduces background on the theory and practise of CT, as well as the anonymity network Tor. Section 3 motivates the intended attacker and presents a unified threat model for CT and Tor. Section 4 describes the full CTor design that *eliminates all trust in the browser-recognized CT logs* by challenging them to prove certificate inclusion cryptographically, and would result in a *single probabilistically-verified view of the CT log ecosystem available from Tor's consensus*. This view could be used by other browsers as the basis of trust, *greatly improving the security posture of the entire web*. The security analysis in Section 5 shows that one of the best bets for the attacker would be to take network-wide actions against Tor to avoid public disclosure of certificate mis-issuance and log misbehavior. Such an attack is trivially detected, but it is hard to attribute unless reactive defenses are enabled at the cost of trade-offs.

The full design involves many different components that add deployment burdens, such as the requirement of reliable CT auditors that investigate suspected log misbehavior further. Therefore, we additionally propose two initial increments that place *some trust in CT logs* (Section 6). The first increment *provides evidence to independent CT logs that fraudulent certificates were presented while preserving privacy*. This greatly impacts risk-averse attackers because one part of their malicious behavior becomes transparent *if the randomly selected log operator is benign*. For example, the targeted domain name is disclosed as part of the cross-logged certificate, and awareness of the event draws unwanted attention.

The next increment is minor from the perspective of Tor, but requires CT logs to support an additional API. Similar changes were proposed in the context of CT gossip [25]. If supported, Tor relays could expose both the mis-issued certificates and the operators that promised to log them publicly *without the complexity of ever distinguishing between what is benign and fraudulent*. This

API change happens to also build auditor infrastructure directly into CT log software, thereby paving the path towards the missing component of the full design. We argue that CTor can be deployed incrementally: complete Firefox's CT enforcement [4], add our cross-logging increments, and finally put the full design into operation. Each part of CTor would *greatly contribute to the open question of how to reduce and/or eliminate trust in browser-recognized log operators*, which is caused by the lack of an appropriate gossip mechanism as well as privacy issues while interacting with the logs [20, 25, 46].

We show that circuit-, bandwidth- and memory-*overheads are modest* by computing such estimates in Section 7. Therefore, we do not investigate performance further in any experimental setting. Section 8 discusses privacy aspects of our design choices with a focus on the essential role of the Tor network's distributed nature to preserve user privacy as well as the overall security. In gist, *a similar approach would be privacy-invasive without Tor*, e.g., if adopted by Google Chrome. Section 9 outlines related work. Section 10 concludes the paper.

2 Background

The theory and current practise of CT is introduced first, then Tor and its privacy-preserving Tor Browser.

2.1 Certificate Transparency

The idea to transparently log TLS certificates emerged at Google in response to a lack of proposals that could be deployed without drastic ecosystem changes and/or significant downsides [35]. By making the set of issued certificate chains[1] transparent, anyone that inspect the logs can detect certificate mis-issuance *after the fact*. It would be somewhat circular to solve issues in the CA ecosystem by adding trusted CT logs. Therefore, the cryptographic foundation of CT is engineered to avoid any such reliance. Google's *gradual* CT roll-out started in 2015, and evolved from downgrading user-interface indicators in Chrome to the current state of hard failures unless a certificate is accompanied by a signed *promise* that it will appear in two CT logs [55]. Unlike Apple's Safari [3], these two logs must additionally be operated by Google and not-Google to ensure independence [23].

The lack of mainstream verification, i.e., beyond checking signatures, allows an attacker to side-step the current CT enforcement with minimal risk of exposure *if the required logs are controlled by the attacker*. CTor integrates into the gradual CT roll-out by starting on the premise of pairwise-independently trusted CT logs, which avoids the risk of bad user experience [55] and significant system complexity. For example, web pages are unlikely to break, TLS handshake latency stays about the same, and no robust management of

[1]A domain owner's certificate is signed by an intermediate CA, whose certificate is in turned signed by a root CA that acts as a trust anchor [18]. Such a *certificate chain* is valid if it ends in a trusted anchor that is shipped in the user's system software.

suspected log misbehavior is needed. Retaining the latter property as part of our incremental designs simplifies deployment.

2.1.1 Cryptographic Foundation

The operator of a CT log maintains a tamper-evident append-only Merkle tree [36, 37]. At any time, a Signed Tree Head (STH) can be produced which fixes the log's structure and content. Important attributes of an STH include the tree head (a cryptographic hash), the tree size (a number of entries), and the current time. Given two tree sizes, a log can produce a *consistency proof* that proves the newer tree head entails everything that the older tree head does. As such, anyone can verify that the log is append-only without downloading all entries and recomputing the tree head. Membership of an entry can also be proven by producing an *inclusion proof* for an STH. These proof techniques are formally verified [17].

Upon a valid request, a log must add an entry and produce a new STH that covers it within a time known as the Maximum Merge Delay (MMD), e.g., 24 hours. This policy aspect can be verified because in response, a Signed Certificate Timestamp (SCT) is returned. An SCT is a signed promise that an entry will appear in the log within an MMD. A log that violates its MMD is said to perform an *omission attack*. It can be detected by challenging the log to prove inclusion. A log that forks, presenting one append-only version to some entities and another to others, is said to perform a *split-view attack*. Split-views can be detected by STH gossip [8, 14, 46, 58].

2.1.2 Standardization and Verification

The standardized CT protocol defines public HTTP(S) endpoints that allow anyone to check the log's accepted trust anchors and added certificates, as well as to obtain the most recent STH and to fetch proofs [36, 37]. For example, the `add-chain` endpoint returns an SCT if the added certificate chain ends in a trust anchor returned by the `get-roots` endpoint. We use `add-chain` in Section 6, as well as several other endpoints in Section 4 to fetch proofs and STHs. It might be helpful to know that an inclusion proof is fetched based on two parameters: a certificate hash and the tree size of an STH. The former specifies the log entry of interest, and the latter with regards to which view inclusion should be proven. The returned proof is valid if it can be used in combination with the certificate to reconstruct the STH's tree head.

The CT landscape provides a limited value unless it is verified that the logs play by the rules. What the rules are changed over time, but they are largely influenced by the major browser vendors that define *CT policies*. For example, what is required to become a recognized CT log in terms of uptime and trust anchors, and which criteria should pass to consider a certificate CT compliant [3, 23]. While there are several ways that a log can misbehave with regards to these policy aspects, the most fundamental forms of cheating are omission and split-view attacks. A party that follows-up on inclusion and consistency proofs is said to *audit* the logs.

Widespread client-side auditing is a premise for CT logs to be untrusted, but none of the web browsers that enforce CT engage in such activities yet. For example, requesting an inclusion proof is privacy-invasive because it leaks browsing patterns to the logs, and reporting suspected log misbehavior comes with privacy [20] as well as operational challenges. Found log incidents are mostly reported manually to the CT policy list [11]. This is in contrast to automated *CT monitors*, which notify domain owners of newly issued certificates based on what actually appeared in the public logs [12, 38].

2.2 Tor

Most of the activity of Tor's millions of daily users starts with Tor Browser and connects to some ordinary website via a circuit comprised of three randomly-selected Tor relays. In this way no identifying information from Internet protocols (such as IP address) are automatically provided to the destination, and no single entity can observe both the source and destination of a connection. Tor Browser is also configured and performs some filtering to resist browser fingerprinting, and first party isolation to resist sharing state or linking of identifiers across origins. More generally it avoids storing identifying configuration and behavioral information to disk.

Tor relays in a circuit are selected at random, but not uniformly. A typical circuit is comprised of a *guard*, a *middle*, and an *exit*. A guard is selected by a client and used for several months as the entrance to all Tor circuits. If the guard is not controlled by an adversary, that adversary will not find itself selected to be on a Tor circuit adjacent to (thus identifying) the client. And because some relay operators do not wish to act as the apparent Internet source for connections to arbitrary destinations, relay operators can configure the ports (if any) on which they will permit connections besides to other Tor relays. Finally, to facilitate load balancing, relays are assigned a weight based on their apparent capacity to carry traffic. In keeping with avoiding storing of linkable state, even circuits that share an origin will only permit new connections over that circuit for ten minutes. After that, if all connections are closed, all state associated with the circuit is cleared.

Tor clients use this information when choosing relays with which to build a circuit. They receive the information via an hourly updated *consensus*. The consensus assigns weights as well as flags such as `guard` or `exit`. It also assigns auxiliary flags such as `stable`, which, e.g., is necessary to obtain the `guard` flag since guards must have good availability. Self-reported information by relays in their *extra-info document*, such as statistics on their read and written bytes, are also part of the consensus and uploaded to *directory authorities*. Directory authorities determine the consensus by voting on various components making up the shared view of the state of the Tor network. Making sure that all clients have a consistent view of the network prevents epistemic attacks wherein clients can be separated based on the routes that are consistent with their understanding [15]. This is only a very rough sketch of Tor's design and operation. More details can be found by following links at Tor's documentation site [62].

Tor does not aim to prevent end-to-end correlation attacks. An adversary controlling the guard and exit, or controlling the destination and observing the client ISP, etc., is assumed able to confirm who is connected to whom on that particular circuit. The Tor threat model assumes an adversary able to control and/or observe a small to moderate fraction of Tor relays measured by both number of relays and by consensus weight, and it assumes a large number of Tor clients able to, for example, flood individual relays to detect traffic signatures of honest traffic on a given circuit [21]. Also, the adversary can knock any small number of relays offline via either attacks from clients or direct Internet DDoS.

3 Threat Model

We consider a strong attacker who is targeting all or a subset of users visiting a particular website over Tor. It is generally difficult to perform a targeted attack on a single particular Tor user because one needs to identify the user's connection before performing the attack—something that Tor's anonymity properties frustrate. However, it is not difficult to perform an attack on all or a subset of unknown users of a particular service. A network vantage point to perform such an attack is easily obtained by operating an exit relay (for a subset of Tor users) or by compromising the network path of multiple exit relays or the final destination. Once so positioned, the encrypted network traffic can be intercepted using a fraudulent certificate and associated SCTs. The subsequent attack on decrypted network traffic may be passive (to gather user credentials or other information) or active. Typical examples of active attacks are to change cryptocurrency addresses to redirect funds to the attacker or to serve an exploit to the user's browser for *user deanonymization*. Without the ability to intercept encrypted traffic, these attacks become more difficult as the web moves towards deprecating plaintext HTTP.

All of the components of such an attack have been seen in-the-wild numerous times. Untargeted attacks on visitors of a particular website include Syria's interception of Facebook traffic using a self-signed 512-bit RSA key in 2011 [19], Iran's interception of Bing and Google traffic using the DigiNotar CA [35, 49], and the 2018 MyEtherWallet self-signed certificate that was used as part of a BGP hijack [51]. The latter is also an example of redirecting routing as part of an attack (either suspected or confirmed). Other examples of this are Iran hijacking prefixes of Telegram (an encrypted messaging application) in 2018 [47], another attack on cryptocurrency in 2014 this time targeting unencrypted mining traffic [57], and hijacks that may have been intelligence-gathering (or honest mistakes) including hijacks by Russian ISPs in 2017 and China Telecom in 2018 and 2019 [66]. Finally, there are several examples of law enforcement serving exploits to Tor Browser users to de-anonymize and subsequently arrest individuals [28, 65].

With the attacker's profile in mind, we consider someone that controls a CA, enough CT logs to pass Tor Browser's SCT-centric CT policy, some

Tor clients, and a fraction of Tor relays. For example, it is possible to issue certificates and SCTs, dishonor promises of public logging, present split-views at will, intercept and delay traffic from controlled exit relays as well as CT logs, and be partially present in the network. This includes a weaker attacker that does not *control* CAs and CT logs, but who *gained access* to the relevant signing keys [32, 41]. A modest fraction of CTor entities can be subject to DoS, but not everyone at once and all the time. In other words, we consider the threat model of Tor and Tor Browser as a starting point [16, 48]. Any attacker that can reliably disrupt CT and/or Tor well beyond Tor's threat model is therefore not within ours.

Given that we are in the business of enforcing CT, the attacker needs to hide mis-issued certificates and SCTs from entities that audit the CT log ecosystem. As described in Section 2.1, this can either be achieved by omission or split-view attacks. Our intended attacker is clearly powerful and may successfully issue a certificate chain and associated SCTs without detection some of the time, but a CA caught in mis-issuance or a CT log that violated an MMD promise will no longer be regarded as trusted. Therefore, we assume a *risk-averse* attacker that above a relatively low probability of detection would be deterred from engaging in such activities. Note that the goal of *detection* is inherited from CT's threat model, which aims to remedy certificate mis-issuance *after the fact*; not prevent it [35].

We identify and analyze specific attack vectors that follow from our threat model and design as part of the security analysis in Section 5, namely, attack vectors related to timing as well as relay flooding and tagging.

4 Design

A complete design—a design that detects misbehavior by both CAs and CT logs within our strong threat model—requires a considerable degree of complexity. In this section we present such a full design by breaking it up into four phases as shown in Figure 1, demonstrating the need for the involved complexity in each step. Section 6 presents two incremental versions of the full design that are less complicated. The first increment comes as the cost of having a weaker threat model and security goal. The second increment does not have a weaker security goal but requires a new CT log API.

A design that starts by validating SCT signatures like Apple's Safari is promising and assumed [3, 67], but it does not stand up against a malicious CA and two CT logs that work in concert. If the logs cannot be trusted blindly, the presented SCTs need to be audited.

4.1 Phase 1: Submission

The least complicated auditing design would be one where Tor Browser receives a TLS certificate and accompanying SCTs (we will refer to this bundle as an SCT Feedback Object, or SFO for short) and talks to the corresponding logs, over Tor, requesting an inclusion proof for each SCT. In an ordinary browser,

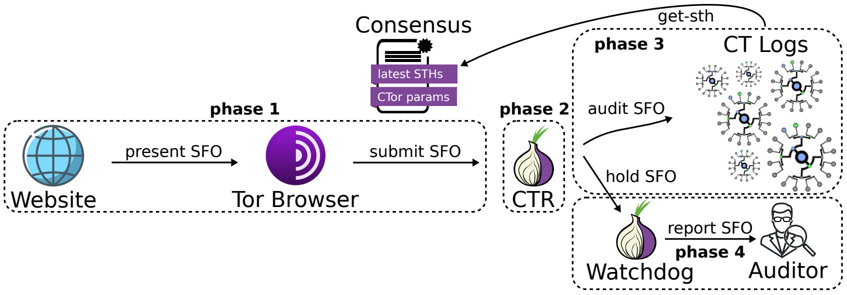


Figure 1: An overview of the four phases of the full CTor design. In phase 1 Tor Browser submits an SFO (SCT Feedback Object) to a Certificate Transparency Relay (CTR), followed by phase 2 where the CTR buffers the SFO. In phase 3 the relay attempts to audit the SFO, and in case of failure, it reports the SFO to an auditor with the help of a watchdog CTR in phase 4.

this would be an unacceptable privacy leak to the log of browsing behavior associated with an IP address; performing this request over Tor hides the user's IP address but still leaks real-time browsing behavior.

An immediate problem with this design is that a primary requirement of Tor Browser is to persist no data about browsing behavior after the application exits. If we assume that browsers are not left running for long periods of time, the inclusion proof request can be easily circumvented by the attacker by using a fresh SCT whose MMD has not completed—thus no inclusion proof needs to be provided (yet) by the log as per the CT standard. A second problem is that the STH that an inclusion proof refers to exists in a *trust vacuum*: there is no way to know that it is consistent with other STHs and not part of a split view (assuming that there is no proactive STH gossip [14, 58], which is not deployed).

We can evolve the design by adding two components: a list of STHs that Tor Browser receives over a trusted channel and the participation of a trusted third party with the ability to persist data and perform auditing actions at a later point in time.

A single third party used by all users of Tor Browser would receive a considerable aggregation of browsing behavior and would need to scale in-line with the entire Tor network. A small number of auditors presents privacy and single-point-of-failure concerns. A large number would be ideal but presents difficulties in curation and independent management and still requires scaling independent of the Tor network. These concerns do not entirely preclude the design, but they can be easily avoided by reusing relays in the Tor network as our trusted third parties: we call the relays so designated Certificate Transparency Relays (CTRs).

Now, when the browser is completing the TLS handshake, it simultaneously either passes the SFO to a CTR (if the MMD of the SCT has not elapsed) or queries the log itself for an inclusion proof to a trusted STH. However, if we presume the attacker can serve an exploit to the browser, the latter behavior is immediately vulnerable. The log, upon receiving an inclusion proof request for an SCT that it knows is malicious, can delay its response. The TLS connection in the browser, having succeeded, will progress to the HTTP request and response, at which point the exploit will be served, and the SFO (containing the cryptographic evidence of CA and log misbehavior) will be deleted by the exploit code. While blocking the TLS connection until the CT log responds is an option, experience related to OCSP hard-fail indicates that this notion is likely doomed to fail [33].

The final change of the design has Tor Browser submit the SFO to the CTR immediately upon receipt (with some probability) in all cases. A consequence of this shift is that the trusted STH list no longer needs to be delivered to the browser but rather the CTRs. To mitigate the risk of a browser exploit being able to identify the CTR to the attacker (who could then target it), we prepare *CTR circuits* ahead of time that are closed and discarded as soon as the SFO is sent. This allows the SFO submission to race with the TLS connection completion and HTTP request/response. An added detail is to block the TLS connection in the case that an SFO is unusually large, as defined by a parameter `ct-large-sfo-size`. A large SFO may indicate an attempt to win the race between SFO submission and exploitation. The parameter can be set such that it happens extremely rarely on legitimate connections, as shown in Section 7.

We summarize phase 1 with the following algorithm that provides more explicit steps and details, including the addition of a parameter `ct-submit-pr` that indicates a probability that an SFO is submitted to a CTR. This provides probabilistic security while providing the ability to adjust submission rates to account for CTR and more general network scaling/health issues. Given an incoming SFO s, Tor Browser should:

1. Raise a certificate error and stop if the certificate chain of s is not rooted in Tor Browser's trust store.

2. Raise a certificate transparency error and stop if the SCTs of s fail Tor Browser's CT policy.

3. If $\mathsf{len}(s) <$ `ct-large-sfo-size`, accept s and conduct the remaining steps in the background while the TLS connection and subsequent HTTP request/response proceed. If $\mathsf{len}(s) \geq$ `ct-large-sfo-size` pause the TLS handshake, complete the remaining steps, accept s as valid and then continue the handshake.

4. Flip a biased coin based on `ct-submit-pr` and stop if the outcome indicates no further auditing.

5. Submit s to a random CTR on a pre-built circuit. The circuit used for submission is closed immediately without waiting for any acknowledgment.

4.2 Phase 2: Buffering

Once received, the most straightforward thing for a CTR to do would be to contact the issuing log and request an inclusion proof relative to a trusted STH. (And if the SCT's MMD has not elapsed, hold the SFO until it has.) However, this proposal has two flaws, the first of which leads us to the actual design of phase 2.

Immediately contacting the log about an SFO (i) allows the log to predict when exactly it will receive a request about an SFO and (ii) discloses real-time browsing behavior to the log. The former problem means that an attacker can position resources for perpetuating an attack ahead-of-time, as well as letting it know with certainty whether a connection was audited (based on `ct-submit-pr`). The latter is some amount of information leakage that can help with real-time traffic analysis.

Because a CTR must support buffering SCTs regardless (due to the MMD), we can schedule an event in the future for when each SFO should be audited. Adding a per-SFO value sampled from `ct-delay-dist` effectively adds stop-and-go mixing [30] to the privacy protection, but where there is only one mix (CTR) between sender (client) and receiver (CT log). So there is no point in a client-specified interval-start-time such that the mix drops messages arriving before then, and there is no additional risk in having the interval end time set by the mix rather than the sender. This means both that some SFOs a client sends to a CTR at roughly the same time might be audited at different times and that SFOs submitted to that CTR by other honest clients are more likely to be mixed with these.

In addition to buffering SFOs for mixing effects, we also add a layer of caching to reduce the storage overhead, prevent unnecessary log connections, and limit the disclosure to logs. With regards to some CT circuit, an incoming SFO s is processed as follows by a CTR:

1. Close the circuit to enforce one-time use.
2. Discard all SCTs in the SFO for logs the CTR is not aware of; if no SCT remains then discard the SFO.
3. Stop if s is cached or already pending to be audited in the buffer. See caching details in Section 7.2.
4. Sample a CT log l that issued a remaining SCT in s.
5. Compute an `audit_after` time t, see Figure 2.
6. Add (l, t, s) to a buffer of pending SFOs to audit.

What makes a CT log known to the CTR is part of the Tor consensus, see Section 4.5. It implies knowledge of a trusted STH for the sampled CT log l, which refers to an entity that (i) issued an SCT in the submitted SFO, and (ii) will be challenged to prove inclusion in phase 3 sometime after the `audit_after` timestamp t elapsed. We choose one SCT (and thus log) at

```
1 :  t ← now() + MMD + random(ct-delay-dist)
2 :  if SCT.timestamp + MMD < now() :
3 :      t ← now() + random(ct-delay-dist)
```

Figure 2: Algorithm that computes an `audit_after` timestamp t.

random from the SFO because it is sufficient to suspect only one misbehaving log so long as we report the entire SFO, allowing us to identify the other malicious CT logs later on (a risk averse-attacker would not conduct an attack without controlling enough logs, i.e., one benign log would otherwise make the mis-issued certificate public).

The `audit_after` timestamp specifies the earliest point in time that an SCT from an SFO will be audited in phase 3, which adds random noise that obfuscates real-time browsing patterns in the Tor network and complicates predictions of when it is safe to assume no audit will take place. If memory becomes a scarce resource, pending triplets should be deleted at random [46]. Figure 2 shows that t takes the log's MMD into account. This prevents an *early signal* to the issuing CT logs that an SFO is being audited. For example, if an SFO is audited before the MMD elapsed, then the issuing CT log could simply merge the underlying certificate chain to avoid any MMD violation. However, by taking the MMD into account, this results in a relatively large time window during which the attacker can attempt to *flood* all CTRs in hope that they delete the omitted SFO at random before it is audited. We discuss the threat of flooding further in Section 5, noting that such an attack can be detected if CTRs publish two new metrics in the extra-info document: `ct-receive-bytes` and `ct-delete-bytes`. These metrics indicate how many SFO bytes were received and deleted throughout different time intervals, which is similar to other extra-info metrics such as `read-history` and `write-history`.

4.3 Phase 3: Auditing

As alluded to in phase 2, there is a second problem why the simple behavior of "contact the log and request an inclusion proof" is unacceptable. We include the ability to DoS an individual Tor relay in our threat model—if the log knows which CTR holds the evidence of its misbehavior, it can take the CTR offline, wiping the evidence of the log's misbehavior from its memory.

We can address this concern in a few ways. The simple proposal of contacting the log over a Tor circuit will not suffice: a log can tag each CTR by submitting unique SFOs to them all, and recognize the CTR when they are submitted (see Section 5). Even using a unique Tor circuit for each SFO might not suffice to prevent effective tagging attacks. For example, after tagging all CTRs, a malicious log could ignore all but innocuous untagged requests and tagged requests matching tags for whichever CTR it decides to respond to first. If some kind of back-off is supported (common to delay retransmissions and avoid congestion), the rest of the CTRs will likely be in back-off so that there is a high probability that the first CTR is the one fetching proofs. The log can repeat this process—alternating tagged CTRs it replies to—until it receives the offending SFO from an identifiable CTR with high probability. CTRs may report the log as inaccessible for days, but that is not the same as direct cryptographic evidence of misbehavior.

While there are ways to detect this attack after-the-fact, and there may be ways to mitigate it, a more robust design would tolerate the disclosure of a CTRs identity to the log during the auditing phase without significant security implications. A simple appealing approach is to write the data to disk prior to contacting the log; however, Tor relays are explicitly designed not to write data about user behavior to disk unless debug-level logging is enabled. Relay operators have expressed an explicit desire to never have any user data persisted to disk, as it changes the risk profile of their servers with regards to search, seizure, and forensic analysis.

The final design is to have the CTR work with a partner CTR—we call it a *watchdog*—that they choose at random and contact over a circuit. Prior to attempting to fetch a proof from a log, the CTR provides the watchdog with the SFO it is about to audit. After an appropriate response from the log, the CTR tells the watchdog that the SFO has been adequately addressed.

In more detail, each CTR maintains a single shared circuit that is used to interact with all CT logs known to the CTR (we are not using one circuit per SFO given the overhead and unclear security benefit noted above). For *each* such log l, the CTR runs the following steps:

1. Sample a delay $d \leftarrow$ `random(ct-backoff-dist)` and wait until d time units elapsed.
2. Connect to a random watchdog CTR.
3. For each pending buffer entry (l', s, t), where $l' = l$ and $t <=$ now():
   (a) Share s with the current watchdog.
   (b) Challenge the log to prove inclusion to the closest STH in the Tor consensus where $t \leq$ STH.`timestamp`. Wait `ct-log-timeout` time units for the complete proof before timing out.
      - On valid proof: send an acknowledgment to the watchdog, cache s and then discard it.
      - On any other outcome: close circuit to the watchdog CTR, discard s, and go to step 1.

4.4 Phase 4: Reporting

At any given time, a CTR may be requesting inclusion proofs from logs and act as a watchdog for one or more CTRs. A CTR acting as a watchdog will have at most one SFO held temporarily for each other CTR it is interacting with. If an acknowledgement from the other CTR is not received within `ct-watchdog-timeout`, it becomes the watchdog's responsibility to report the SFO such that it culminates in human review if need be.

Because human review and publication is critical at this end-stage, we envision that the watchdog (which is a Tor relay that cannot persist any evidence to disk and may not be closely monitored by its operator) provides the SFO to an independent CT auditor that is run by someone that closely monitors its operation. When arriving at the design of the CTR being a role played by a Tor relay, we eschewed separate auditors because of the lack of automatic scaling with the Tor network, the considerable aggregation of browsing behavior across the Tor network, and the difficulties of curation and validation of trustworthy individuals. SFOs submitted to auditors at this stage have been filtered through the CTR layer (that additionally backs-off if the logs become unavailable to prevent an open pipe of SFOs from being reported), resulting in an exponentially smaller load and data exposure for auditors. This should allow for a smaller number of them to operate without needing to scale with the network.

While we assume that most auditors are trusted to actually investigate the reported SFOs further, the watchdog needs to take precautions talking to them because the network is not trusted.[2] The watchdog can contact the auditor immediately, but must do so over an independent Tor circuit.[3] If a successful acknowledgement from the auditor is not received within `ct-auditor-timeout`, the SFO is buffered for a random time using `ct-delay-dist` before being reported to the same auditor again over a new independent Tor circuit.

When an auditor receives an SFO, it should persist it to durable storage until it can be successfully resolved to a specific STH.[4] Once so persisted, the auditor can begin querying the log itself asking for an inclusion proof. If no valid inclusion proof can be provided after some threshold of time, the auditor software should raise the details to a human operator for investigation.

Separately, the auditor should be retrieving the current Tor consensus and ensuring that a consistency proof can be provided between STHs from the older consensus and the newer. If consistency cannot be established after some threshold of time, the auditor software should raise the details to a human operator for investigation. An auditor could also monitor a log's uptime and report on excessive downtime. Finally, it is paramount that the auditor continuously monitors its own availability from fresh Tor-circuits by submitting known SFOs to itself to ensure that an attacker is not keeping watchdogs from connecting to it.

[2]While our threat model, and Tor's, precludes a global network adversary, both include partial control of the network.

[3]This is also important because CTRs are not necessarily exits, i.e., the exiting traffic must be destined to another Tor relay.

[4]The fetched inclusion proof must be against the first known STH that should have incorporated the certificate in question by using the history of STHs in Tor's consensus: the mis-issued certificate might have been merged into the log reactively upon learning that a CTR reported the SFO, such that a valid inclusion proof can be returned with regards to a more recent STH but not earlier ones that actually captured the log's misbehavior.

4.5 Setup

There are a number of additional details missing to setup phases 1–4 for the design. Most of these details relate to the Tor consensus. Directory authorities influence the way in which Tor Browser and CTRs behave by voting on necessary parameters, such as the probability of submission of an SFO (`ct-submit-pr`) and the timeout used by CTRs when auditing CT logs (`ct-log-timeout`), as introduced earlier as part of the design. See Appendix A for details on these parameters and their values that were previously used. Next, we briefly introduce a number of implicitly used parts from our design that should also be part of the consensus.

In the consensus, the existing `known-flags` item determines the different flags that the consensus might contain for relays. We add another flag named `CTR`, which indicates that a Tor relay should support CT-auditing as described here. A relay qualifies as a CTR if it is flagged as `stable` and not `exit`, to spare the relatively sparse exit bandwidth and only use relays that can be expected to stay online. Section 8 discusses trade-offs in the assignment of the `CTR` flag.

The consensus should also capture a fixed view of the CT log ecosystem by publishing STHs from all known logs. A CT log is known if a majority of directory authorities proposed a `ct-log-info` item, which contains a log's ID, public key, base URL, MMD, and most recent STH. Each directory authority proposes its own STH, and agrees to use the most recent STH as determined by timestamp and lexicographical order. Since CTRs verify inclusion with regards to SCTs that Tor Browser accepts, the CT logs recognized by Tor Browser must be in Tor's consensus.

Tor's directory authorities also majority-vote on `ct-auditor` items, which pin base URLs and public keys of CT auditors that watchdogs contact in case that any log misbehavior is suspected.

5 Security Analysis

We consider four types of impact for an attacker that conducted HTTPS-based man-in-the-middle attacks on Tor Browser. Other than *none*, these impact types are:

Minor the attack was detected due to some cover-up that involved network-wide actions against CTor. This is likely hard to attribute to the actual attacker, but nevertheless it draws much unwanted attention.

Significant the attack generated public cryptographic evidence that proves CA misbehavior.

Catastrophic the attack generated public cryptographic evidence that proves CT log misbehavior.

Our design leads to significant and catastrophic impact events, but does unfortunately not preclude minor ones. It is possible to overcome this shortcoming at different trade-offs, e.g., by tuning CTor parameters reactively (phase 2 below) or relying on different trust assumptions as in the incremental cross-logging designs (Section 6).

Probability of Detection. Suppose the attacker mis-issued a certificate that Tor Browser trusts, and that it is considered valid because it is accompanied by enough SCTs from CT logs that the attacker controls. The resulting SFO is then used to man-in-the-middle a single Tor Browser user, i.e., for the purpose of our analysis we consider *the most risk-averse scenario possible*. Clearly, none of the attacker's CT logs plan to keep any promise of public logging: that would trivially imply significant impact events. The risk of exposure is instead bound by the probability that *any* of the four phases in our design fail to propagate the mis-issued SFO to a pinned CT auditor that is benign.

Phase 1: Submission. The probability of detection cannot exceed the probability of submission (`ct-submit-pr`). We analyze the outcome of submitting the mis-issued SFO from Tor Browser to a CTR. There are two cases to consider, namely, the mis-issued SFO is either larger than `ct-large-sfo-size` or it is not.

If the SFO is larger than `ct-large-sfo-size`, Tor Browser blocks until the SFO is submitted and its CT circuit is closed. As such, it is impossible to serve a Tor Browser exploit reactively over the man-in-the-middled connection that shuts-down the submission procedure before it occurs. Assuming that forensic traces in tor and Tor Browser are unreliable,[5] the sampled CTR identity also cannot be revealed with high certainty afterwards by compromising Tor Browser. The attacker may know that the SFO is buffered by *some CTR* based on timing, i.e., blocking-behavior could be measurable and distinct. The important part is not to reveal *which CTR* received a submission: a single Tor relay may be subject to DoS.

If the SFO is smaller or equal to `ct-large-sfo-size` there is a race between (i) the time it takes for Tor Browser to submit the SFO and close its CT circuit against (ii) the time it takes for the attacker to compromise Tor Browser and identify the CTR in question. It is more advantageous to try and win this race rather than being in the unfruitful scenario above. Therefore, the attacker would maximize the time it takes to perform (i) by sending an SFO that is `ct-large-sfo-size`. Our design reduced the threat of an attacker that wins this race by using pre-built CT circuits that are closed immediately after use. This makes the attack surface *narrow*, limiting the number of reliable exploits (if any).

Note that the attack surface could, in theory, be eliminated by setting `ct-large-sfo-size` to zero. However, that is likely too costly in terms of latency [33].

Phase 2: Buffering. The probability of detection cannot exceed $1 - (f_{\text{ctr}} + f_{\text{dos}})$, where f_{ctr} is the fraction of malicious CTRs and f_{dos} the fraction of CTRs that suffer from DoS. We analyze the outcome of SFO reception at a genuine CTR.

[5]"tor" (aka "little-t tor") is the tor process Tor Browser uses to interact with the Tor network. On marking a circuit as closed in tor, tor immediately schedules the associated data structures to be freed as soon as possible.

The time that an SFO is buffered depends on if the log's MMD elapsed or not. The earliest point in time that a newly issued SCT can be audited (and the log is expected to respond) is an MMD later, whereas the normal buffer time is otherwise only governed by smaller randomness in the `audit_after` timestamp (minutes). A rational attacker would therefore maximize the buffer time by using a newly issued SCT, resulting in an attack window that is *at least* 24 hours for today's CT logs [23].

Following from Tor's threat model, the mis-issued SFO must be stored in volatile memory and not to disk. Two risks emerge due to large buffer times: the CTR in question might be restarted by the operator independently of the attacker's mis-issued SFO being buffered, and given enough time the attacker might find a way to cause the evidence to be deleted. While a risk-averse attacker cannot rely on the former to avoid detection, we emphasize that the CTR criteria must include the `stable` flag to reduce the probability of this occurring.

The latter is more difficult to evaluate. It depends on the attacker's knowledge as well as capabilities. Phase 1 ensured that the attacker *does not know which CTR to target*. As such, any attempt to intervene needs to target all CTRs. While a network-wide DoS against Tor would be effective, it is not within our threat model. A less intrusive type of DoS would be to *flood* CTRs by submitting massive amounts of SFOs: just enough to make memory a scarce resource, but without making Tor unavailable. This could potentially *flush* a target SFO from the CTR's finite memory, following from the delete-at-random strategy in Section 4.2. Assuming that a CTR has at most 1 GiB of memory available for SFOs (conservative and in favour of the attacker), Appendix C shows that the attacker's flood must involve at least 2.3 GiB per CTR to accomplish a 90% success certainty. This means that it takes 7.9–39.3 minutes if the relay bandwidth is between 8–40 Mbps. So it is impractical to flush all CTRs within a few minutes, and hours are needed not to make everyone unavailable at once.

The CTR criteria set in Section 4.5 matches over 4000 Tor relays [64]. A network-wide flush that succeeds with 90% certainty therefore involves 8.99 TiB. It might sound daunting at first, but distributed throughout an entire day it only requires 0.91 Gbps. Such an attack is within our threat model because it does not make Tor unavailable. Notably the ballpark of these numbers do not change to any significant degree by assuming larger success probabilities, e.g., a 99% probability only doubles the overhead. Further, the needed bandwidth scales linearly with the assumed memory of CTRs. This makes it difficult to rely on the finite volatile memory of CTRs to mitigate network-wide flushes. As described in Section 4.2, we ensure that flushes are *detected* by publishing the number of received and deleted SFO bytes throughout different time intervals as extra-info.

Once detected, there are several possible *reactions* that decrease the likelihood of a minor impact scenario. For example, Tor's directory authorities could lower MMDs to, say, 30 minutes, so that the SFO is reported to an auditor before it is flushed with high probability. This has the benefit of implying significant impact because the mis-issued certificate is detected, but also the drawback of allowing the logs to merge the certificate before there is any MMD violation to speak of. The most appropriate response depends on the exact attack scenario and which trade-offs one is willing to accept.

Phase 3: Auditing. By the time an SFO enters the audit phase, the log in question is expected to respond with a valid inclusion proof. There is no such proof if the log violated its MMD, and it is too late to create a split-view that merged the certificate in time because the CTR's view is already fixed by an STH in the Tor consensus that captured the log's misbehavior. In fact, creating any split-view within Tor is impractical because it requires that the consensus is forged or that nobody ever checks whether the trusted STHs are consistent. This leaves two options: the attacker either responds to the query with an invalid inclusion proof or not at all. The former is immediately detected and starts phase 4, whereas the latter forces the CTR to wait for `ct-watchdog-timeout` to trigger (which is a few seconds to avoid premature auditor reports). A rational attacker prefers the second option to gain time.

Clearly, the attacker knows that *some* CTR holds evidence of log misbehavior as it is being audited. The relevant question is whether the *exact CTR identity* can be inferred, in which case the attacker could knock it offline (DoS). Motivated by the threat of *tagging*, where the attacker sends unique SFOs to all CTRs so that their identities are revealed once queried for, we erred on the safe side and built watchdogs into our design: it is already too late to DoS the querying CTR because the evidence is already replicated somewhere else, ready to be reported unless there is a timely acknowledgement. The attacker would have to *break into an arbitrary CTR within seconds* to cancel the watchdog, which cannot be identified later on (same premise as the sampled CTR in phase 1). Such an attacker is not in Tor's threat model.

Phase 4: Reporting. At this stage the process of reporting the mis-issued SFO to a random CT auditor is initiated. Clearly, the probability of detection cannot exceed $1 - f_{\textsf{auditor}}$, where $f_{\textsf{auditor}}$ is the fraction of malicious CT auditors. Fixating the sampled CT auditor is important to avoid the threat of an eventually successful report only if it is destined to the attacker's auditor because our attacker is partially present in the network. Gaining time at this stage is of limited help because the CTR identity is unknown as noted above, and it remains the case throughout phase 4 due to reporting on independent Tor circuits (and independently of if other SFO reports succeeded or not). Without an identifiable watchdog, the attacker needs a network-wide attack that is already more likely to succeed in the buffer phase.

6 Incremental Deployment

Section 4 covered the full design that places zero-trust in the CT landscape by challenging the logs to prove certificate inclusion with regards to trusted STHs in the Tor consensus. If no such proof can be provided, the suspected evidence of log misbehavior is reported to a trusted CT auditor that follows-up on the incident, which involves human intervention if an issue persists. The proposed design modifies the Tor consensus, Tor relays, and Tor Browser. It

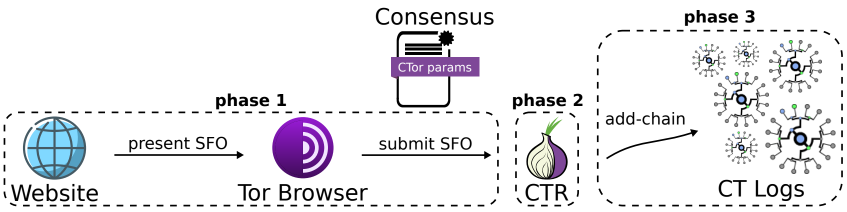


Figure 3: Incremental design that can be deployed without any trusted CT auditors. Tor Browser still submits SFOs to CTRs on independent Tor circuits for the sake of privacy and security. After CTR buffering, the submitted certificates are *cross-logged* by adding them to independent CT logs (selected at random) that the attacker does not control (inferred from accompanied SCTs).

also requires development and operation of a trusted auditor infrastructure. The current lack of the latter makes it unlikely that we will see adoption of CTor in its full potential anytime soon, and begs the question of increments that help us get there in the future. Therefore, we additionally propose two incremental designs in this section.

Without the ability to rely on CT auditors, trust needs to be shifted elsewhere because we cannot expect relay operators to take on the role. At the same time, an incremental proposal needs to improve upon the status quo of pairwise-independently trusted CT logs. These observations lead us towards the trust assumption that *at least some* of the CT logs are trustworthy. Such an assumption is suboptimal, but it does provide a real-world security improvement by significantly raising the bar from weakest-link(s) to quite the opposite.

The smallest change of the full design would be for watchdogs to report suspected certificate mis-issuance to all CT logs, simply by using the public `add-chain` API to make the SFO's certificate chain transparent. This has the benefit of holding the CA accountable if *some* log operator is benign. Given that our attacker is risk-averse, reporting to a single independent log[6] that issued none of the accompanied SCTs would likely be sufficient. There is also room for further simplification: there is no point in challenging the logs to prove inclusion if the fallback behavior of no response only makes the issued certificate public, not the associated SCTs. Thus, CTRs could opt to cross-log immediately *without ever distinguishing between certificates that are benign and possibly fraudulent*. This results in the incremental design shown in Figure 3, which initially removes several system complexities such as extra-info metrics, auditor infrastructure, watchdog collaborations, and inclusion proof fetching against trusted STHs in Tor's consensus.

The drawback of certificate cross-logging is that the misbehaving CT logs cannot be exposed. There is also a discrepancy between cross-logging and encouraging the CT landscape to deploy reliable CT auditors. We therefore

[6]The independent log need not be trusted by the browser, i.e., it could be specified separately in the Tor consensus. An operator that runs such a log would help distribute trust and facilitate auditing. Appendix B provides details on today's log ecosystem.

suggest a minimal change to the basic cross-logging design that addresses both of these concerns. This change is unfortunately to the API of CT logs and not Tor. The proposed change is to allow cross-logging of a certificate's issued SCTs, e.g., in the form of an `add-sfo` API that would replace `add-chain` in Figure 3. This means that CTRs could expose both the mis-issued certificate and the logs that violated their promises of public logging. At the same time, the infrastructural part of a CT auditor is built directly into existing CT logs: accepting SFOs that need further investigation. Such an API would be an ecosystem improvement in itself, providing a well-defined place to report suspected log misbehavior on-the-fly *casually*, i.e., without first trying to resolve an SFO for an extended time period from many different vantage points and then ultimately reporting it manually on the CT policy mailing list.

Security Sketch. There are no changes to phase 1 because cross-logging is instantiated at CTRs. Phases 3–4 are now merged, such that the encountered certificates are added to independent CT logs that the attacker does/may not control. Watchdogs are no longer needed since either the certificates are added to a log that the attacker controls, or they are not (which makes them public). The other main difference takes place in phase 2, during which CTRs buffer SFOs. The buffer time used to be lengthy due to taking early signals and MMDs into account, but it is now irrelevant as no inclusion proofs are fetched. The expected buffer time can therefore be shortened down to *minutes* that follow only from the randomness in the `audit_after` timestamp (for the sake of privacy), making network-wide flushes impractical while at the same time reducing the time that a mis-issued certificate stays unnoticed: a benign log is likely to add an entry before all MMDs elapsed.

The extended cross-logging also aims to expose log misbehavior. As such, it is paramount that no cross-logged SFO becomes public before the issuing CT logs can merge the mis-issued certificate reactively to avoid catastrophic impact. This could be assured by buffering newly issued SFOs longer as in the full design, which brings back the threat and complexity of minor impact scenarios. Another option that is appealing for Tor (but less so for CT) is to operate the `add-sfo` API with the expectation of *delayed merges* that account for MMDs before making an SFO public, effectively moving lengthy buffering from CTRs to CT logs with persistent storage. Trillian-based CT logs already support delayed merges of (pre)certificates, see `sequencer_guard_window` [26].

7 Performance

The following analysis shows that CTor's overhead is modest based on computing performance estimates from concrete parameter properties and two public data sets.

7.1 Setup

Mani *et al.* derived a distribution of website visits over Tor and an estimation of the number of circuits through the network [40]. We use their results to reason about overhead as the Tor network is under heavy load, assuming 140 million daily website visits (the upper bound of a 95% confidence interval). Our analysis also requires a distribution that captures typical SFO properties per website visit. Therefore, we collected an SFO data set by browsing the most popular webpages submitted to Reddit (r/frontpage, all time) on December 4, 2019. The data set contains SFOs from 8858 webpage visits, and it is available online as an open access artifact together with the associated scripts [13]. Notably we hypothesized that browsing actual webpages as opposed to front-pages would yield more SFOs. When compared to Alexa's list it turned out to be the case: our data set has roughly two additional SFOs per data point. This makes it less likely that our analysis is an underestimate.

We found that an average certificate chain is 5440 bytes, and it is seldom accompanied by more than a few SCTs. As such, a typical SFO is in the order of 6 KiB. No certificate chain exceeded 20 KiB, and the average number of SFOs per webpage was seven. The latter includes 1–2 SFOs per data point that followed from our client software calling home on start-up (Chromium 77).

We assume no abnormal CTor behavior, which means that there will be little or no CTR back-offs due to the high uptime requirements of today's CT logs: 99%. We set `ct-large-sfo-size` conservatively to avoid blocking in the TLS handshake (e.g., 20 KiB), and use a 10% submission probability as well as a 10 minute random buffer delay on average. It is likely unwarranted to use a higher submission probability given that the intended attacker is risk-averse. Shorter buffer times would leak finer-grained browsing patterns to the logs, while longer ones increase the attack surface in phase 2. Therefore, we selected an average for `ct-delay-dist` that satisfies none of the two extremes. The remaining CTor parameters are timeouts, which have little or no performance impact if set conservatively (few seconds).

7.2 Estimates

The incremental cross-logging designs are analyzed first without any caching. Caching is then considered, followed by overhead that appears only in the full design.

Circuit Overhead. Equation 1 shows the expected circuit overhead from Tor Browser over time, where p is the submit probability and $\bar{d}$ the average number of SFOs per website visit. The involved overhead is linear as either of the two parameters are tuned up or down.

$$p\bar{d} \tag{1}$$

Using $p \leftarrow \frac{1}{10}$ and our approximated SFO distribution $\bar{d} \leftarrow 7$ yields an average circuit overhead of 0.70, i.e., for every three Tor Browser circuits CTor adds another two. Such an increase might sound daunting at first,[7] but

[7] Circuit establishment involves queueing of onionskins [61] and it is a likely bottleneck, but since the introduction of ntor it is not a scarce resource so such overhead is acceptable if it (i) serves a purpose, and (ii) can be tuned. Confirmed by Tor developers.

these additional circuits are short-lived and light-weight; transporting 6 KiB on average. Each CTR also maintains a long-lived circuit for CT log interactions.

Bandwidth Overhead. Equation 2 shows the expected bandwidth overhead for the Tor network over time, where V is the number of website visits per time unit, p the submit probability, $\bar{d}$ the average number of SFOs per website visit, and $\bar{s}$ the average SFO byte-size.

$$6Vp\bar{d}\bar{s} \tag{2}$$

$Vp\bar{d}$ is the average number of SFO submissions per time unit, which can be converted to bandwidth by weighting each submission with the size of a typical SFO and accounting for it being relayed six times: three hops from Tor Browser to a CTR, then another three hops from the CTR to a CT log (we assumed symmetric Tor relay bandwidth). Using $V \leftarrow 140$ M/day, $p \leftarrow \frac{1}{10}$, $\bar{d} \leftarrow 7$, $\bar{s} \leftarrow 6$ KiB and converting the result to bps yields 334.5 Mbps in total. Such order of overhead is small when compared to Tor's capacity: 450 Gbps [60].

Memory Overhead. Equation 3 shows the expected buffering overhead, where V_m is the number of website visits per minute, t the average buffer time in minutes, R the number of Tor relays that qualify as CTRs, and $\bar{s}$ the typical SFO size in bytes.

$$\frac{V_m t}{R}\bar{s} \tag{3}$$

$V_m t$ represent incoming SFO submissions during the average buffer time, which are randomly distributed across R CTRs. Combined, this yields the expected number of SFOs that await at a single CTR in phase 2, and by taking the byte-size of these SFOs into account we get an estimate of the resulting memory overhead. Using $V_m \leftarrow \frac{140\text{ M}}{24\cdot 60}$, $t \leftarrow 10$ m, $R \leftarrow 4000$ based on the CTR criteria in Section 4.5, and $\bar{s} \leftarrow 6$ KiB yields 1.42 MiB. Such order of overhead is small when compared to the recommended relay configuration: at least 512 MiB [63].

A cache of processed SFOs reduces the CTR's buffering memory and log interactions proportionally to the cache hit ratio. Mani *et al.* showed that if the overrepresented `torproject.org` is removed, about one third of all website visits over Tor can be attributed to Alexa's top-1k and another one third to the top-1M [40]. Assuming 32 byte cryptographic hashes and seven SFOs per website visit, a cache hit ratio of $\frac{1}{3}$ could be achieved by a 256 KiB LFU/LRU cache that eventually captures Alexa's top-1k. Given that the cache requires memory as well, this is mainly a bandwidth optimization.

Full Design. For each CTR and CT log pair, there is an additional watchdog circuit that transports the full SFO upfront before fetching an inclusion proof. The expected bandwidth overhead is at most $9Vp\bar{d}\bar{s}$, i.e., now also accounting for the three additional hops that an SFO is subject to. In practise the overhead is slightly less, because an inclusion query and its returned proof is smaller than an SFO. We expect little or no watchdog-to-auditor overhead if the logs are available, and otherwise one light-weight circuit that reports a single SFO for each CTR that goes into back-off. Such overhead is small when compared to all Tor Browser submissions. Finally, the required memory increases because newly issued SFOs are buffered for at least an MMD. Only a small portion of SFOs are newly issued, however: the short-lived certificates of Let's Encrypt are valid for 90 days [1], which is in contrast to 24 hour MMDs [23].

8 Privacy

There is an inherent privacy problem in the setting due to how CT is designed and deployed. A browser, like Tor Browser, that wishes to validate that SFOs presented to it are *consistent* and *included* in CT logs must directly or indirectly interact with CT logs wrt. its observed SFOs. Without protections like Private Information Retrieval (PIR) [7] that require server-side support or introduction of additional parties and trust assumptions [29, 39], exposing SFOs to any party risks leaking (partial) information about the browsing activities of the user.

Given the constraints of the existing CT ecosystem, CTor is made privacy-preserving thanks to the distributed nature of Tor with its anonymity properties and high-uptime relays that make up the Tor network. First, all communication between Tor Browser, CTRs, CT logs, and auditors are made over full Tor-circuits. This is a significant privacy-gain, not available, e.g., to browsers like Chrome that in their communications would reveal their public IP-address (among a number of other potentially identifying metadata). Secondly, the use of CTRs as intermediaries probabilistically delays the interaction with the CT logs—making correlating Tor Browser user browsing with CT log interaction harder for attackers—and safely maintains a dynamic cache of the most commonly already verified SFOs. While browsers like Chrome could maintain a cache, Tor Browser's security and privacy goals (Section 2.2) prohibit such shared (persisted) dynamic state.

In terms of privacy, the main limitation of CTor is that CTor continuously leaks to CT logs—and to a *lesser extent* auditors (depending on design)—a fraction of certificates of websites visited using Tor Browser to those that operate CT logs. This provides to a CT log a partial list of websites visited via the Tor network over a period of time (determined by `ct-delay-dist`), together with some indication of distribution based on the number of active CTRs. It does not, however, provide even pseudonymously any information about which sites individual users visit, much less with which patterns or timing. As such it leaks significantly less information than does OCSP validation by Tor Browser or DNS resolution at exit-relays [27], both of which indicate visit activity in real time to a small number of entities.

Another significant limitation is that relays with the CTR flag learn real-time browser behavior of Tor users. Relays without the `exit` flag primarily only transport encrypted Tor-traffic between clients and other relays, never to destinations. If such relays are given the CTR flag—as we stated in the full design, see Section 4.5—then this might discourage some from running

Tor relays unless it is possible to opt out. Another option is to give the CTR flag only to exit relays, but this *might be* undesirable for overall network performance despite the modest overhead of CTor (Section 7). Depending on the health of the network and the exact incremental deployment of CTor, there are different trade-offs.

9 Related Work

The status quo is to consider a certificate CT compliant if it is accompanied by two independent SCTs [23, 67]. Therefore we proposed that Tor Browser should do the same, but unlike any other CT-enforcing web browser CTor also provides concrete next steps that relax the centralized trust which is otherwise misplaced in CT logs [41, 50, 52, 53]. Several proposals surfaced that aim to do better with regards to omissions and split-views.

Laurie proposed that inclusion proofs could be fetched over DNS to avoid additional privacy leaks, i.e., a user's browsing patterns are already exposed to the DNS resolver but not the logs in the CT landscape [34]. CT/bis provides the option of serving stapled inclusion proofs as part of the TLS handshake in an extension, an OCSP response, or the certificate itself [37]. Lueks and Goldberg proposed that a separate database of inclusion proofs could be maintained that supports information-theoretic PIR [39]. Kales *et al.* improved scalability by reducing the size of each entry in the PIR database at the cost of transforming logs into multi-tier Merkle trees, and additionally showed how the upper tier could be expressed as a two-server computational PIR database to ensure that any inclusion proof can be computed privately on-the-fly [29]. Nordberg *et al.* avoid inclusion proof fetching by hanging on to presented SFOs, handing them back to the same origin at a later time [46]. In contrast, CTor protects the user's privacy without any persistent browser state by submitting SFOs on independent Tor circuits to CTRs, which in turn add random noise before there is any log interaction. The use of CTRs enable caching similar to CT-over-DNS, but it does not put the logs in the dark like PIR could.

Inclusion proofs are only meaningful if everyone observes the same consistent STHs. One option is to configure client software with a list of entities that they should gossip with, e.g., CT monitors [6], or, browser vendors could push a verified view [54]. Such trusted auditor relationships may work for some but not others [46]. Chuat *et al.* proposed that HTTPS clients and HTTPS servers could pool STHs and consistency proofs, which are gossiped on website visits [8]. Nordberg *et al.* suggested a similar variant, reducing the risk of user tracking by pooling fewer and recent STHs [46]. Dahlberg *et al.* noted that such privacy-insensitive STHs need not be encrypted, which could enable network operators to use programmable data planes to provide gossip as-a-service [14]. Syta *et al.* proposed an alternative to reactive gossip mechanisms by showing how an STH can be cosigned efficiently by many independent witnesses [58]. A smaller-scale version of witness cosigning could be instantiated by cross-logging STHs in other CT logs [25], or in other append-only ledgers [59]. CTor's full design (Section 4) ensures that anyone connected to the Tor network is on the same view by making STHs public in the Tor consensus. In contrast, the first incremental design (Section 6) is not concerned with catching log misbehavior, while the second incremental design (also Section 6) exposes misbehaving logs *without* first trying to fetch inclusion proofs.

Nordberg proposed that Tor clients could enforce public logging of consensus documents and votes [45]. Such an initiative is mostly orthogonal to CTor, as it strengthens the assumption of a secure Tor consensus by enabling detection of compromised signing keys rather than mis-issued TLS certificates. Winter *et al.* proposed that Tor Browser could check self-signed TLS certificates for exact matches on independent Tor circuits [68]. Alicherry *et al.* proposed that any web browser could double-check TLS certificates on first encounter using alternative paths and Tor, again, looking for certificate mismatches and generating warnings of possible man-in-the-middle attacks [2]. The submission phase in CTor is similar to double-checking, except that there are normally no TLS handshake blocking, browser warnings, or strict assumptions regarding the attacker's location.

In parallel Stark and Thompson proposed that Chrome could submit a random subset of encountered SCTs to a trusted auditor that Google runs [56]. CTor also propagates a random subset of SCTs to a trusted auditor, but does so while preserving privacy because of and how Tor is used. Meiklejohn additionally proposed witness cosigning on-top of consistent STHs [42]. CTor adds signatures on-top of STHs too, but only as part of the Tor consensus that directory authorities sign.

10 Conclusion

We proposed CTor, a privacy-preserving and incrementally-deployable design that brings CT to Tor. Tor Browser should start by taking the same proactive security measures as Google Chrome and Apple's Safari: require that a certificate is only valid if accompanied by at least two SCTs. Such CT enforcement narrows down the attack surface from the weakest-link security of the CA ecosystem to a relatively small number of trusted log operators *without negatively impacting the user experience to an unacceptable degree*. The problem is that a powerful attacker may gain control of the required logs, trivially circumventing enforcement without significant risk of exposure. If deployed incrementally, CTor relaxes the currently deployed trust assumption by distributing it across all CT logs. If the full design is put into operation, such trust is completely eliminated.

CTor repurposes Tor relays to ensure that today's trust in CT logs is not misplaced: Tor Browser probabilistically submits the encountered certificates and SCTs to Tor relays, which cross-log them into independent CT logs (incremental design) or request inclusion proofs with regards to a single fixed view (full design). It turns out that delegating verification to a party that can defer it is paramount in our setting, both for privacy and security. Tor and the wider web would greatly benefit from each design increment. The full design turns Tor into a system for maintaining a probabilistically-verified view of the entire CT log ecosystem, provided in Tor's consensus for anyone to use as a basis of trust. The idea to cross-log certificates and SCTs further showcase how certificate mis-issuance and suspected log misbehavior could be disclosed casually without any manual intervention by using the log ecosystem against the attacker.

The attacker's best bet to break CTor involves any of the following: operating significant parts of the CTor infrastructure, spending a reliable Tor Browser zero-day that escalates privileges within a tiny time window, or targeting all Tor relays in an attempt to delete any evidence of certificate mis-issuance and log misbehavior. The latter—a so-called network-wide flush—brings us to the border of our threat model, but it cannot be ignored due to the powerful attacker that we consider. Therefore, CTor is designed so that Tor can *adapt* in response to interference. For example, in Tor Browser the `ct-large-sfo-size` could be set reactively such that all SFOs must be sent to a CTR before accepting any HTTPS application-layer data to counter zero-days, and the submit probability `ct-submit-pr` could be increased if ongoing attacks are suspected. When it comes to the storage phase, the consensus can minimize or maximize the storage time by tuning a log's MMD in the `ct-log-info` item. The distribution that adds random buffering delays could also be updated, as well as log operator relationships during the auditing phase.

Acknowledgements

We would like to thank our anonymous reviewers as well as Linus Nordberg and Eric Rescorla for their valuable feedback. Rasmus Dahlberg was supported by the Knowledge Foundation of Sweden and the Swedish Foundation for Strategic Research, Tobias Pulls by the Swedish Internet Foundation, and Paul Syverson by the U.S. Office of Naval Research (ONR).

References

[1] Josh Aas, Richard Barnes, Benton Case, Zakir Durumeric, Peter Eckersley, Alan Flores-López, J. Alex Halderman, Jacob Hoffman-Andrews, James Kasten, Eric Rescorla, Seth D. Schoen, and Brad Warren. Let's Encrypt: An automated certificate authority to encrypt the entire web. In *CCS*, 2019.

[2] Mansoor Alicherry and Angelos D. Keromytis. DoubleCheck: Multipath verification against man-in-the-middle attacks. In *ISCC*, 2009.

[3] Apple Inc. Apple's Certificate Transparency log program. https://support.apple.com/en-om/HT209255, accessed 2020-12-15.

[4] Bugzilla. Implement Certificate Transparency support (RFC 6962).

https://bugzilla.mozilla.org/show_bug.cgi?id=1281469, accessed 2020-12-15.

[5] Catalin Cimpanu. Exploit vendor drops Tor Browser zero-day on Twitter. https://www.zdnet.com/article/exploit-vendor-drops-tor-browser-zero-day-on-twitter/, accessed 2020-12-15.

[6] Melissa Chase and Sarah Meiklejohn. Transparency overlays and applications. In *CCS*, 2016.

[7] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *FOCS*, 1995.

[8] Laurent Chuat, Pawel Szalachowski, Adrian Perrig, Ben Laurie, and Eran Messeri. Efficient gossip protocols for verifying the consistency of certificate logs. In *CNS*, 2015.

[9] Jeremy Clark and Paul C. van Oorschot. SoK: SSL and HTTPS: revisiting past challenges and evaluating certificate trust model enhancements. In *IEEE S&P*, 2013.

[10] Mauro Conti, Stephen Crane, Tommaso Frassetto, Andrei Homescu, Georg Koppen, Per Larsen, Christopher Liebchen, Mike Perry, and Ahmad-Reza Sadeghi. Selfrando: Securing the Tor Browser against de-anonymization exploits. *PETS*, 2016(4).

[11] CT policy mailing list. Certificate Transparency policy. https://groups.google.com/a/chromium.org/forum/#!forum/ct-policy, accessed 2020-12-15.

[12] Rasmus Dahlberg and Tobias Pulls. Verifiable light-weight monitoring for Certificate Transparency logs. In *NordSec*, 2018.

[13] Rasmus Dahlberg, Tobias Pulls, Tom Ritter, and Paul Syverson. SFO distribution artificat. https://github.com/rgdd/ctor/tree/master/artifact, 2020.

[14] Rasmus Dahlberg, Tobias Pulls, Jonathan Vestin, Toke Høiland-Jørgensen, and Andreas Kassler. Aggregation-based Certificate Transparency gossip. In *SECURWARE*, 2019.

[15] George Danezis and Paul Syverson. Bridging and fingerprinting: Epistemic attacks on route selection. In *PETS*, 2008.

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

[17] Benjamin Dowling, Felix Günther, Udyani Herath, and Douglas Stebila. Secure logging schemes and Certificate Transparency. In *ESORICS*, 2016.

[18] Zakir Durumeric, James Kasten, Michael Bailey, and J. Alex Halderman. Analysis of the HTTPS certificate ecosystem. In *IMC*, 2013.

[19] Peter Eckersley. A Syrian man-in-the-middle attack against Facebook. https://www.eff.org/deeplinks/2011/05/syrian-man-middle-against-facebook, accessed 2020-12-15.

[20] Saba Eskandarian, Eran Messeri, Joseph Bonneau, and Dan Boneh. Certificate Transparency with privacy. *PETS*, 2017(4).

[21] Nathan S. Evans, Roger Dingledine, and Christian Grothoff. A practical congestion attack on Tor using long paths. In *USENIX Security*, 2009.

[22] firstwatch at sigaint.org. [tor-talk] javascript exploit. https://lists.torproject.org/pipermail/tor-talk/2016-November/042639.html, accessed 2020-12-15.

[23] Google LLC. Chromium Certificate Transparency policy. https://github.com/chromium/ct-policy/blob/master/README.md, accessed 2020-12-15.

[24] Google LLC. HTTPS encryption on the web. https://transparencyreport.google.com/https/overview?hl=en, accessed 2020-05-19.

[25] Google LLC. Minimal gossip. https://github.com/google/trillian-examples/blob/master/gossip/minimal/README.md, accessed 2020-12-15.

[26] Google LLC. Trillian log signer. https://github.com/google/trillian/blob/master/cmd/trillian_log_signer/main.go, accessed 2020-12-15.

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

[28] Kashmir Hill. How did the FBI break Tor? https://www.forbes.com/sites/kashmirhill/2014/11/07/how-did-law-enforcement-break-tor/#6cf2ed594bf7, accessed 2020-12-15.

[29] Daniel Kales, Olamide Omolola, and Sebastian Ramacher. Revisiting user privacy for Certificate Transparency. In *IEEE EuroS&P*, 2019.

[30] Dogan Kesdogan, Jan Egner, and Roland Büschkes. Stop-and-Go MIXes: Providing probabilistic anonymity in an open system. In *IH*, 1998.

[31] Nikita Korzhitskii and Niklas Carlsson. Characterizing the root landscape of Certificate Transparency logs. In *IFIP Networking*, 2020.

[32] Adam Langley. Enhancing digital certificate security. https://security.googleblog.com/2013/01/enhancing-digital-certificate-security.html, accessed 2020-12-15.

[33] Adam Langley. No, don't enable revocation checking. https://www.imperialviolet.org/2014/04/19/revchecking.html, accessed 2020-12-15.

[34] Ben Laurie. Certificate Transparency over DNS. https://github.com/google/certificate-transparency-rfcs/blob/master/dns/draft-ct-over-dns.md, accessed 2020-12-15.

[35] Ben Laurie. Certificate transparency. *CACM*, 57(10), 2014.

[36] Ben Laurie, Adam Langley, and Emilia Kasper. Certificate Transparency. RFC 6962, IETF, 2013.

[37] Ben Laurie, Adam Langley, Emilia Kasper, Eran Messeri, and Rob Stradling. Certificate Transparency version 2.0. Internet-draft draft-ietf-trans-rfc6962-bis-34, IETF, 2019.

[38] Bingyu Li, Jingqiang Lin, Fengjun Li, Qiongxiao Wang, Qi Li, Jiwu Jing, and Congli Wang. Certificate Transparency in the wild: Exploring the reliability of monitors. In *CCS*, 2019.

[39] Wouter Lueks and Ian Goldberg. Sublinear scaling for multi-client private information retrieval. In *FC*, 2015.

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

[41] Brendan McMillion. Un-incorporated SCTs from GDCA1. https://groups.google.com/a/chromium.org/forum/#!topic/ct-policy/Emh3ZaU0jqI, accessed 2020-12-15.

[42] Sarah Meiklejohn, Pavel Kalinnikov, Cindy S. Lin, Martin Hutchinson, Gary Belvin, Mariana Raykova, and Al Cutter. Think global, act local: Gossip and client audits in verifiable data structures. *CoRR*, abs/2011.04551, 2020.

[43] Mozilla. Mozilla research grants 2019H1. https://mozilla-research.forms.fm/mozilla-research-grants-2019h1/forms/6510, accessed 2020-12-15.

[44] Mozilla. SSL ratios. https://docs.telemetry.mozilla.org/datasets/other/ssl/reference.html, accessed 2020-05-19.

[45] Linus Nordberg. Tor consensus transparency. https://gitlab.torproject.org/tpo/core/torspec/-/blob/main/proposals/267-tor-consensus-transparency.txt, accessed 2020-12-15.

[46] Linus Nordberg, Daniel Kahn Gillmor, and Tom Ritter. Gossiping in CT. Internet-draft draft-ietf-trans-gossip-05, IETF, 2018.

[47] Patrick Howell O'Neill. Telegram traffic from around the world took a detour through Iran. https://www.cyberscoop.com/telegram-iran-bgp-hijacking/, accessed 2020-12-15.

[48] 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/, accessed 2020-12-15.

[49] J.R. Prins. DigiNotar certificate authority breach "operation black tulip". Interim report, Fox-IT, 2011.

[50] Jeremy Rowley. CT2 log compromised via Salt vulnerability. https://groups.google.com/a/chromium.org/forum/#!topic/ct-policy/aKNbZuJzwfM, accessed 2020-12-15.

[51] Aftab Siddiqui. What happened? The Amazon Route 53 BGP hijack to take over Ethereum cryptocurrency wallets. https://www.internetsociety.org/blog/2018/04/amazons-route-53-bgp-hijack/, accessed 2020-12-15.

[52] Ryan Sleevi. Upcoming CT log removal: Izenpe. https://groups.google.com/a/chromium.org/forum/#!topic/ct-policy/qOorKuhL1vA, accessed 2020-12-15.

[53] Ryan Sleevi. Upcoming log removal: Venafi CT log server. https://groups.google.com/a/chromium.org/forum/#!topic/ct-policy/KMAcNT3asTQ, accessed 2020-12-15.

[54] Ryan Sleevi and Eran Messeri. Certificate transparency in Chrome: Monitoring CT logs consistency. https://docs.google.com/document/d/1FP5J5Sfsg0OR9P4YT0q1dM02iavhi8ix1mZlZe_z-ls/edit?pref=2&pli=1, accessed 2020-12-15.

[55] Emily Stark, Ryan Sleevi, Rijad Muminovic, Devon O'Brien, Eran Messeri, Adrienne Porter Felt, Brendan McMillion, and Parisa Tabriz. Does Certificate Transparency break the web? Measuring adoption and error rate. In *IEEE S&P*, 2019.

[56] Emily Stark and Chris Thompson. Opt-in SCT auditing. https://docs.google.com/document/d/1G1Jy8LJgSqJ-B673GnTYIG4b7XRw2ZLtvvSlrqFcl4A/edit, accessed 2020-12-15.

[57] Joe Stewart. BGP hijacking for cryptocurrency profit. https://www.secureworks.com/research/bgp-hijacking-for-cryptocurrency-profit, accessed 2020-12-15.

[58] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. Keeping authorities "honest or bust" with decentralized witness cosigning. In *IEEE S&P*, 2016.

[59] Alin Tomescu and Srinivas Devadas. Catena: Efficient non-equivocation via Bitcoin. In *IEEE S&P*, 2017.

[60] Tor project. Advertised and consumed bandwidth by relay flag. https://metrics.torproject.org/bandwidth-flags.html, accessed 2020-05-30.

[61] Tor Project. Functions to queue create cells for processing. https://src-ref.docs.torproject.org/tor/onion__queue_8c_source.html, accessed 2020-12-15.

[62] Tor Project. Getting up to speed on Tor's past, present, and future. https://2019.www.torproject.org/docs/documentation.html.en, accessed 2020-12-15.

[63] Tor project. Relay requirements. https://community.torproject.org/relay/relays-requirements/, accessed 2020-05-29.

[64] Tor project. Relays by relay flag. https://metrics.torproject.org/relayflags.html, accessed 2020-05-29.

[65] U.S. Dept. of Justice. More than 400 .onion addresses, including dozens of 'dark market' sites, targeted as part of global enforcement action on Tor network. https://www.fbi.gov/news/pressrel/press-releases/more-than-400-.onion-addresses-including-dozens-of-dark-market-sites-targeted-as-part-of-global-enforcement-action-on-tor-network, accessed 2020-12-15.

[66] Wikipedia contributors. BGP hijacking—Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=BGP_hijacking&oldid=964360841, accessed 2020-12-15.

[67] Clint Wilson. CT days 2020. https://groups.google.com/a/chromium.org/g/ct-policy/c/JWVVhZTL5RM, accessed 2020-12-15.

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

[69] Zerodium. Our exploit acquisition program. https://zerodium.com/program.html, accessed 2020-05-21.

[70] Zerodium. Tor Browser zero-day exploit bounty (expired). https://zerodium.com/tor.html, accessed 2020-12-15.

A Detailed Consensus Parameters

Below, the value of an item is computed as the median of all votes.

ct-submit-pr: A floating-point in $[0, 1]$ that determines Tor Browser's submission probability. For example, `0` disables submissions while `0.10` means that every 10th SFO is sent to a random CTR on average.

ct-large-sfo-size: A natural number that determines how many wire-bytes a normal SFO should not exceed. As outlined in Section 4.1, excessively large SFOs are subject to stricter verification criteria.

ct-log-timeout: A natural number that determines how long a CTR waits before concluding that a CT log is unresponsive, e.g., 5 seconds. As outlined in Section 4.3, a timeout causes the watchdog to send an SFO to the auditor.

ct-delay-dist: A distribution that determines how long a CTR should wait at minimum before auditing a submitted SFO. As outlined in Section 4.2, random noise is added, e.g., on the order of minutes to an hour.

ct-backoff-dist: A distribution that determines how long a CTR should wait between two auditing instances, e.g., a few minutes on average. As outlined in Section 4.3, CTRs audit pending SFOs in batches at random time intervals to spread out log overhead.

ct-watchdog-timeout: A natural number that determines how long time at most a watchdog waits before considering an SFO for reporting. Prevents the watchdog from having to wait for a circuit timeout caused by an unresponsive CTR. Should be set with `ct-backoff-dist` in mind.

ct-auditor-timeout A natural number that determines how long time at most a watchdog waits for an auditor to acknowledge the submission of an SFO.

B Log Operators & Trust Anchors

The standardized CT protocol suggests that a log's trust anchors should "usefully be the union of root certificates trusted by major browser vendors" [36, 37]. Apple further claims that a log in their CT program "must trust all root CA certificates included in Apple's trust store" [3]. This bodes well for the incremental CTor designs: we assumed that the existence of independent log operators implies the ability to at least add certificate chains and possibly complete SFOs into logs that the attacker does not control. Google's CT policy currently qualifies 36 logs that are hosted by Cloudflare, DigiCert, Google, Let's Encrypt, Sectigo, and TrustAsia [23]. No log accepts all roots, but the overlap between root certificates that are trusted by major browser vendors and CT logs increased over time [31]. This trend would likely continue if there are user agents that benefit from it, e.g., Tor Browser. Despite relatively few log operators and an incomplete root coverage, the basic and extended cross-logging in CTor still provide significant value as is:

- Even if there are no independent logs available for a certificate issued by some CA, adding it again *to the same logs* would come with practical security gains. For example, if the attacker gained access to the secret signing keys but not the logs' infrastructures the mis-issued certificate trivially makes it into the public. If the full SFO is added, the log operators could also notice that they were compromised.
- Most log operators only exclude a small fraction of widely accepted root certificates: 1–5% [31]. This narrows down the possible CAs that the attacker must control by 1–2 orders of magnitude. In other words, to be entirely sure that CTor would (re)add a mis-issued SFO to the attacker-controlled CT logs, this smaller group of CAs must issue the underlying certificate. It is likely harder to take control of Let's Encrypt which some logs and operators exclude due to the sheer volume of issued certificates than, say, a smaller CA that law enforcement may coerce.

Browser-qualified or not, the availability of independent logs that accept the commonly accepted root certificates provides significant ecosystem value. Log misbehavior is mostly reported through the CT policy mailing list. Thus, it requires manual intervention. Wide support of certificate chain and SCT cross-logging allows anyone to *casually* disclose suspected log misbehavior on-the-fly.

C Flushing a Single CTR

Let n be the number of SFOs that a CTR can store in its buffer. The probability to sample a target SFO is thus $\frac{1}{n}$, and the probability to not sample a target SFO is $q = 1 - \frac{1}{n}$. The probability to not sample a target SFO after k submissions is q^k. Thus, the probability to sample the relevant buffer index at least once is $p = 1 - q^k$. Solving for k we get: $k = \frac{\log(1-p)}{\log(q)}$. Substituting q for $1 - \frac{1}{n}$ yields Equation 4, which can be used to compute the number of SFO submissions that the attacker needs to flush a buffer of $n > 2$ entries with some probability $p \in [0, 1)$.

$$k = \frac{\log(1-p)}{\log(1-\frac{1}{n})} \tag{4}$$

It is recommended that a non-exit relay should have at least 512MB of memory. If the available bandwidth exceeds 40Mbps, it should have at least 1GB [63]. Given that these recommendations are lower bounds, suppose the average memory available to store SFOs is 1GiB. Section 7 further showed that the average SFO size is roughly 6KiB. This means that the buffer capacity is $n \leftarrow 174763$ SFOs. Plugging it into Equation 4 for $p \leftarrow \frac{9}{10}$, the attacker's flood must involve $k \leftarrow 402406$ submissions. In other words, 2.3GiB must be transmitted to flush a single CTR with 90% success probability.

As a corner case and implementation detail it is important that Tor Browser and CTRs *reject* SFOs that are bogus in terms of size: it is a trivial DoS vector to load data indefinitely. If such a threshold is added the required flushing bandwidth is still 2.3GiB (e.g., use 1MiB SFOs in the above computations). What can be said about bandwidth and potential adversarial advantages is that a submitted SFO yields amplification: twofold for cross-logging, and slightly more for proof-fetching as the SFO is pushed up-front to a watchdog. Note that such amplification is smaller than a typical website visit.

Paper IV

Sauteed Onions: Transparent Associations from Domain Names to Onion Addresses

Reprinted from

WPES (2022)

Sauteed Onions: Transparent Associations from Domain Names to Onion Addresses

Rasmus Dahlberg, Paul Syverson, Linus Nordberg, and Matthew Finkel

Abstract

Onion addresses offer valuable features such as lookup and routing security, self-authenticated connections, and censorship resistance. Therefore, many websites are also available as onionsites in Tor. The way registered domains and onion addresses are associated is however a weak link. We introduce *sauteed onions*, *transparent associations from domain names to onion addresses*. Our approach relies on TLS certificates to establish onion associations. It is much like today's onion location which relies on Certificate Authorities (CAs) due to its HTTPS requirement, but has the added benefit of becoming public for everyone to see in Certificate Transparency (CT) logs. We propose and prototype two uses of sauteed onions: certificate-based onion location and search engines that use CT logs as the underlying database. The achieved goals are *consistency of available onion associations*, which mitigates attacks where users are partitioned depending on which onion addresses they are given, *forward censorship-resistance* after a TLS site has been configured once, and *improved third-party discovery of onion associations*, which requires less trust while easily scaling to all onionsites that opt-in.

1 Introduction

Onion addresses are domain names with many useful properties. For example, an onion address is self-authenticated due to encoding its own public key. It also makes integral use of the anonymity network Tor to provide secure and private lookups as well as routing [12]. A major usability concern is that onion addresses are random-looking strings; they are difficult to discover, update, and remember [49]. Existing solutions approach these limitations in different ways, e.g., ranging from setting onion addresses in HTTP headers over HTTPS with so-called *onion location* [33] and bookmarking found results to making use of manually curated third-party lists [27, 34, 41] as well as search engines like DuckDuckGo or `ahmia.fi` [31, 49].

Herein we refer to the unidirectional association from a domain name to an onion address as an *onion association*. The overall goal is to facilitate transparent discovery of onion associations. To achieve this we rely on the observation that today's onion location can be implemented in certificates issued by Certificate Authorities (CAs). This is not an additional dependency because onion location already requires HTTPS [33]. The main benefit of transitioning from HTTP headers to TLS certificates is that all such onion associations become signed and sequenced in tamper-evident Certificate Transparency (CT) logs [21, 22], further tightening the relation between CAs and onion keys [35, 36, 46] as well as public CT logging and Tor [11, 27].

Our first contribution is to make onion associations identical for all Tor users, and otherwise the possibility of inconsistencies becomes public via CT. Consistency of available onion associations mitigates the threat of users being partitioned without anyone noticing into subsets according to which onion address they received during onion association. Our second contribution is to construct a search engine that allows Tor users to look up onion associations without having to trust the service provider completely. Other than being helpful to validate onion addresses as authentic [49], such discovery can continue to work *after* a TLS site becomes censored.

Section 2 briefly covers CT preliminaries. Section 3 describes *sauteed onions*, an approach that makes discovery of onion associations more transparent and censorship-resistant compared to today. Section 4 discusses related work. Section 5 concludes the paper. Appendix A contains query examples for our search engine. Appendix B outlines an example setup. All artifacts are online [4].

2 Certificate Logging Preliminaries

CT is a system of public append-only logs that store TLS certificates issued by trusted CAs [21, 22]. If web browsers add the criterion that a certificate must be logged before accepting it as valid, certificate issuance practices by CAs effectively become transparent so that mistakes and malfeasance can be detected by anyone that observes the logs. These observers are called *monitors* because they download every certificate from all logs. One can self-host a monitor, or use a third-party service like `crt.sh`, or follow other models based on subscriptions [9, 23]. To avoid introducing more parties that are trusted blindly as in the CA ecosystem, CT stands on a cryptographic foundation that permits efficient verification of inclusion (a certificate is in the log) and the append-only property (no certificate has been removed or modified) [13]. A party engaging in verification of these (logarithmic) proofs is called an *auditor*.

In practice, CT has been rolled-out gradually to not break the web [43]. One facilitating factor has been the introduction of Signed Certificate Timestamps (SCTs). An SCT is a log's *promise* to include a certificate within a certain amount of time; typically 24 hours. This guarantees low-latency certificate issuance so that CAs can *embed* SCTs in certificates to keep web servers oblivious to CT. Google Chrome and Apple's Safari require SCTs before accepting a certificate as valid, and steps towards further SCT verification have been taken recently [42]. Tor Browser does not require CT yet [11].

3 Sauté Onions Until Discovery is Transparent and Confection is Firm

3.1 System Goals

Let an onion association be unidirectional from a traditional domain name to an onion address. Three main system goals are as follows:

Privacy-Preserving Onion Associations Users should discover the same onion associations, and otherwise the possibility of an inconsistency must become public knowledge.

Forward Censorship Resistance Unavailability of a TLS site must not impede discovery of past onion associations.

Automated Verifiable Discovery Onion association search should be possible without requiring blind trust in third-parties. It must be hard to fabricate non-empty answers, and easy to automate the setup for scalability and robustness.

For comparison, today's onion location [33] does not assure a user that the same HTTP header is set for them as for everyone else. Classes of users that connect to a domain at different times or via different links can be given targeted redirects to distinct onion addresses without detection [45]. Onion location also does not work if a regular site becomes unavailable due to censorship. The *search engine approach* is further a frequent ask by Tor users [49]. The solutions that exist in practice rely on manually curated lists [27, 34, 41], notably with little or no retroactive accountability. As specified above, we aim for a similar utility but with a setup that can be automated for all onion associations and without the ability to easily fabricate non-empty answers without trivial detection. We sketch out how these security properties are achieved in Section 3.3.4.

3.2 Threat Model and Scope

We consider an attacker that wants to trick a user into visiting a targeted onionsite without anyone noticing the possibility of such behavior. Users are assumed to know the right traditional domain name that is easy to remember (such as `torproject.org`), but not its corresponding onion address. We further assume that the attacker either controls a trusted CA sufficiently to issue certificates or is able to deceive them sufficiently during certificate issuance to obtain a valid certificate from that CA. Any misbehavior is however assumed to be detectable in CT. So, the certificate ecosystem is treated as a *building block* that we make no attempt to improve.

We permit the attacker to make TLS sites unavailable after setup, but we assume it is difficult to censor the CT log ecosystem because it can be mirrored by anyone. Also, as part of the Internet authentication infrastructure, adversaries may have equities conflicts in blocking CT logs, and if concerned at all about appearance would have a harder time justifying such a block versus, e.g., a political, journalism, or social media site. Similar to CT, we do not attempt to solve certificate revocation and especially not in relation to certificates that are connected to discovery of onion associations. This is consistent with Tor Browser's existing model for revocation with onion location, which similarly depends on the certificate for the redirecting domain. There is no formal counterpart to revoke a result in a search engine, but we outline future work related to this.

Our threat model includes countries that block direct access to HTTPS sites [25]. This is arguably a capable attacker, as no country is currently known to completely block indirect access via the Tor network (though in some places Tor bridges and/or obfuscated transport is needed). Our threat model also considers the plethora of blindly trusted parties that help users discover onion addresses with little or no retroactive accountability [6, 27, 34, 41]. In other words, it is in-scope to pave the path towards more accountability.

3.3 Description of Sauteed Onions

An observation that inspired work on sauteed onions is that onion location requires HTTPS [33]. This means that discovery of onion associations *already* relies on the CA ecosystem. By incorporating the use of CT, it is possible to add accountability to CAs and other parties that help with onion address discovery while also raising the bar for censoring sites and reducing anonymity. The name sauteed onions is a cooking pun; the association of an onion address with a domain name becomes transparent for everyone to see in CT logs.

For background, a CA-issued certificate can contain both a traditional domain name and a `.onion address` [35, 36]. This can be viewed as a mutual association because the issuing CA must verify the traditional domain name *and* the specified onion address. An immediate problem is that this would be ambiguous if there are multiple domain names; which one (if any) should be associated with an onion address with such certificate coalescence? A more appropriate path forward would therefore be to define an X.509v3 extension for sauteed onions which clearly *declares that a domain-validated name wants to be associated with an onion address*.

We describe two uses of sauteed onions that achieve our goals; first assuming it is easy to get CA-issued certificates that contain associated onion addresses for domain-validated names, and then a short-term roll-out approach that could make it a reality now. A sauteed onion is simply a CT-logged certificate that claims `example.com` wants to be associated with `<addr>.onion` but not necessarily the other way around, i.e., a unidirectional association.

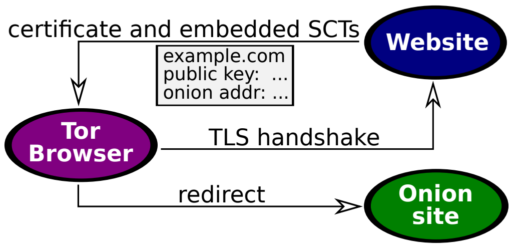


Figure 1: Onion location based on a CT-logged certificate.

3.3.1 Onion Location

Figure 1 illustrates onion location that uses certificates. A user establishes a TLS connection to a site as usual. Upon encountering a certificate that is CT-logged with an associated onion address for the visited site `example.com`, an onion-location prompt becomes available in Tor Browser or the onion site is visited automatically. This is the same type of redirect behavior as today's onion location [33], except that the possibility of such a redirect is disclosed in public CT logs. Attempts at targeted redirects would thus be visible to site owners and independent third-parties. A redirect to someone else's onion address would also be visible to the respective site owners. Notably the ability to detect inappropriate redirects acts as a deterrence while also being the first step towards remediation, e.g., if users bookmarked onion addresses [49] to achieve trust on first use or to avoid visiting a regular site *and* an onionsite in a way that might reduce a user's anonymity set.

A key observation is that onion location has always been a feature facilitated by TLS. By implementing it in certificates rather than HTTP headers that are delivered via HTTPS connections, TLS applications that are "not web" can use it too without rolling their own mechanisms. The addition of requiring CT to follow onion-location redirects is also an improvement compared to today, although one that could be achieved with an HTTP-based approach as well (or more ambitiously, for all Tor Browser certificate validations [11]).

We prototyped the above in a web extension that is free and open source [4]. The criterion for CT logging is at least one embedded SCT from a log in the policy used by Google Chrome [19]. If an onion-location redirect is followed, the path of the current webpage is preserved, similar to a typical configuration of today's HTTP-based onion location header that instead lists a complete URL [33].

3.3.2 Search Engine

A significant challenge for third-parties that help users discover TLS sites that are available as onion services is to gain confidence in the underlying dataset at scale. For example, SecureDrop onion names are scoped to news sites [41]; the list by Muffett is scoped as "no sites for tech with less than (arbitrary) 10,000 users" [27]; and `ahmia.fi` does not even attempt to give onion addresses

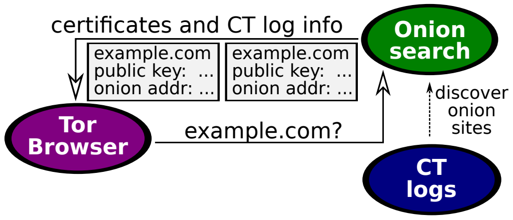


Figure 2: Verifiable domain name to onion address search.

human-meaningful names [31]. To make matters worse, solutions based on manually curated lists and third-party search are currently implemented with little or no accountability.

Figure 2 shows what our approach brings to the table. All CT logs can be monitored by a third-party to discover sauteed onions. A search API can then be presented to users for the resulting dataset, similar to existing monitoring services but scoped specifically for discovery of onion associations. The utility of such a search API is: "*what onion addresses are available for* `www.example.com`".

The expected behavior of the search API is that an answer can not be fabricated without controlling a CA or hijacking certificate issuance, and any CA malfeasance should further be caught by CT. This means that no party can fabricate inappropriate answers without detection. This is a major improvement compared to the alternative of no verifiability at all, although one that in and of itself does not prevent *false negatives*. In other words, available answers could trivially be omitted. This is a limitation with the authenticated data structure in CT that can be fixed; see security sketch in Section 3.3.4 for an intuition of how to work around it.

We specified an HTTP REST API that facilitates search using a domain name; the API also makes available additional information like the actual certificate and its exact index in a CT log. In total there are two endpoints: `search` (list of matches with identifiers to more info) and `get` (more info). The complete API specification is available online together with our implementation, which is free and open source [4]. An independent implementation from Tor's hack week is also available by Rhatto [39]. Our prototype runs against all CT logs in Google Chrome for certificates logged after July 16, 2022. A few query examples are available in Appendix A.

3.3.3 Certificate Format

Until now we assumed that a sauteed onion is easily set up, e.g., using an X.509v3 extension. The bad news is that such an extension does not exist, and it would likely be a long journey to standardize and see deployment by CAs. Therefore, our prototypes rely on a backwards-compatible approach that encodes onion addresses as subdomains [44]. To declare that `example.com` wants to be associated with `<addr>.onion`, one can request a domain-validated certificate that contains both `example.com` and `<addr>onion.example.com` [46]. The inclusion of `example.com` ensures that such a setup does not result in a dangerous label [17]. The *hack to encode an onion address as a subdomain* makes it part of the certificate without requiring changes to CAs. Appendix B details the necessary setup-steps further. The gist is the addition of a subdomain DNS record and using the `-d` option in `certbot` [14].

Although the subdomain approach is easy to deploy right now, it is by no means a perfect solution. An X.509v3 extension would not require the configuration of an additional DNS record. In other words, the unidirectional sauteed onions property works just as well if the subdomain is not domain-validated. The important part is that the CA validates `example.com`, and that the associated onion address can be declared somewhere in the issued certificate without an ambiguous intent. Another imperfection that goes hand-in-hand with backwards-compatibility is that CAs would have to *opt-out* from sauteed onions, unlike site owners that instead have to *opt-in*.

To avoid recommending a pattern that is discouraged by CAs, the Tor Project should at least have a dialog with Let's Encrypt which issues the most certificates [5]. Somewhat similar subdomain hacks related to CAs exist, but then with explicit negotiations [47]. Subdomain hacks without a relation to CAs and TLS were discouraged in the past [20]. We argue that sauteed onions is related because CA-validated names are at the heart of our approach. For example, this is unlike Mozilla's binary transparency idea that just wanted to reuse a public log [26]. Sauteed onions also do not result in more issued certificates; it is just the number of domain-validated names that increase by one for TLS sites that do the setup.

3.3.4 Security Sketch

Our threat model disallows the attacker to tamper with CT and to make the log ecosystem unavailable. Onion location as described in Section 3.3.1 therefore ensures that a redirect becomes public, achieving detectability as defined in our privacy-preserving onion association goal. The search engine in Section 3.3.2 trivially achieves the same goal because onion associations are *found* via CT. Blocking a TLS site is additionally *too late* if an association is already in a CT log, thus achieving forward censorship resistance. Our search engine approach further makes it hard to forge non-answers without detection because it requires control of a CA and defeating the tamper-evidence of CT logs. While it is possible to omit available answers, this can be mitigated by having multiple search APIs, domains that check the integrity of their own onion associations similar to the proposed verification pattern in CONIKS [24], or to represent the sauteed onion dataset as a sparse Merkle tree to get a verifiable log-backed map that additionally supports efficient non-membership proofs that CT lacks [10, 15].

3.4 Future Work

It would be valuable to implement proofs of no omissions as well as native lookups in a web extension or Tor Browser to verify everything before showing the user a result (certificates, proofs of logging, etc). The entire or selected parts of the sauteed onion dataset may further be delivered to Tor Browser similar to SecureDrop onion names [41]. The difference would be that the list is automated using a selection criteria from CT logs rather than doing it manually on a case-by-case basis. A major benefit is that the sauteed onion dataset can then be queried locally, completely avoiding third-party queries and visits to the regular site. Another approach to explore is potential integration of the sauteed onion dataset into Tor's DHT: a cryptographic source of truth for available onion associations is likely a helpful starting point so that there is *something to distribute*. It would also be interesting to consider other search-engine policies than *show everything* as in our work, e.g., only first association or last association. (These policies can be verified with *full audits* [15].)

4 Related Work

The CA/B forum accepts certificates with `.onion` addresses [35, 36]. DigiCert supports extended validation of `.onion` addresses [37], and HARICA domain validation [38]. Muffett proposed same-origin onion certificates that permit clients to omit verification of the CA trust chain for onionsites [28]. Sauteed onions help Tor users *discover* domain names with associated onion addresses. Therefore, it is a complement to approaches that bring HTTPS to onionsites.

Syverson suggested that traditional domain names and `.onion` addresses can be glued into a single registered domain [44]. Nusenu proposed long-term Tor relay identifiers based on domain names to retrieve lists of relevant public keys via HTTPS [32]. Sauteed onions may be used for such associations with the benefit of transparency, and it is further a *lighter* version of Syverson and Traudt's self-authenticated traditional addresses which favors early deployment over properties like bidirectional onion associations, guaranteed timeliness of revocation, and addressing all known threats [45, 46].

Winter *et al.* studied how users engage with onion services [49]. A gist is that Tor users have a hard time discovering onion addresses and verifying them as authentic. Common discovery mechanisms that are associated with human-meaningful identifiers include personal communication, webpage links, onion-location redirects [33], third-party lists [34], and search engines like DuckDuckGo. Prior work has also focused on enumerating onion addresses without any associated identity, e.g., through CT-logged certificates with `.onion` addresses [27] and crawling [6, 31]. Sauteed onions enhance onion location by making the claimed associations transparent in CT, and facilitate third-party solutions with less blind trust and without assumptions about TLS sites not becoming blocked in the future.

Several ideas were proposed that mitigate or bypass the problem of random-looking onion addresses. Some sites generate vanity addresses that, e.g., start with a prefix and have other memorable traits [1]. Fink sketched out how to map onion addresses to a set of words [16]. Kadianakis *et al.* defined a common API to hook into alternative naming systems that give onion addresses pet names [18]. SecureDrop Onion Names is one such example that is, however, implemented directly in Tor Browser as an HTTPS Everywhere ruleset for selected news sites. Other alternative naming systems include Namecoin [2] and OnioNS [48]. Sauteed onions is also an alternative naming system, but one that relies on CAs and CT logs. It may be possible to construct sauteed onions via DNSSEC, but then relying on the DNS hierarchy without transparency logging. Scaife *et al.* [40] proposed the `.o` TLD as an onionsite with DNSSEC.

Nordberg connected transparency logs and the consensus mechanism that Tor uses [29]. Dahlberg *et al.* proposed CT in Tor for all certificate validations [11]. We only check signatures of embedded SCTs in relation to onion location, and our search engine is a simple application of CT monitoring. There is a large body of orthogonal work that improve CAs and CT. For example, multi-path domain-validation makes it harder to hijack onion associations [7], and deployment of gossip would harden our CT log assumptions [8, 30].

5 Conclusion

Sauteed onions declare unidirectional associations from domain names to onion addresses. These onion associations are established in CA-issued and CT-logged TLS certificates, thereby making them public for everyone to see. We propose two immediate applications: certificate-based onion location and more automated verifiable search. Both applications are opt-in for domain owners, and rely on similar assumptions as today's onion location. The added benefit is more transparency, which facilitates a higher degree of consistency between found onion associations as well as more censorship-resistance for TLS sites after setup. Configuration of sauteed onions requires one more DNS record and a domain-validated certificate from any CA (such as Let's Encrypt). In the future, the additional DNS record may be replaced by an X.509v3 extension. We leave it as a fun exercise to find the onion address of a TLS site that is intentionally being censored by us: `blocked.sauteed-onions.org`.

Acknowledgments

We would like to thank Kushal Das, Daniel Kahn Gillmor, Silvio Rhatto, and Tobias Pulls for helpful discussions and comments. Rasmus Dahlberg was supported by the Swedish Foundation for Strategic Research. Paul Syverson was supported by ONR.

References

[1] mkp224o—vanity address generator for ed25519 onion services. https://github.com/cathugger/mkp224o, accessed 2022-08-01.

[2] Namecoin. https://www.namecoin.org/, accessed 2022-08-01.

[3] Sauteed onion certificate. https://crt.sh/?id=5957691193, accessed 2022-08-01.

[4] Paper artifact. https://gitlab.torproject.org/tpo/onion-services/sauteed-onions, 2022.

[5] Josh Aas, Richard Barnes, Benton Case, Zakir Durumeric, Peter Eckersley, Alan Flores-López, J. Alex Halderman, Jacob Hoffman-Andrews, James Kasten, Eric Rescorla, Seth D. Schoen, and Brad Warren. Let's Encrypt: An automated certificate authority to encrypt the entire web. In *CCS*, 2019.

[6] Ahmia. Indexing and crawling. https://ahmia.fi/documentation/indexing/, accessed 2022-08-01.

[7] Henry Birge-Lee, Liang Wang, Daniel McCarney, Roland Shoemaker, Jennifer Rexford, and Prateek Mittal. Experiences deploying multi-vantage-point domain validation at Let's Encrypt. In *USENIX Security*, 2021.

[8] Laurent Chuat, Pawel Szalachowski, Adrian Perrig, Ben Laurie, and Eran Messeri. Efficient gossip protocols for verifying the consistency of certificate logs. In *CNS*, 2015.

[9] Rasmus Dahlberg and Tobias Pulls. Verifiable light-weight monitoring for Certificate Transparency logs. In *NordSec*, 2018.

[10] Rasmus Dahlberg, Tobias Pulls, and Roel Peeters. Efficient sparse Merkle trees - caching strategies and secure (non-)membership proofs. In *NordSec*, 2016.

[11] Rasmus Dahlberg, Tobias Pulls, Tom Ritter, and Paul Syverson. Privacy-preserving & incrementally-deployable support for Certificate Transparency in Tor. *PETS*, 2021(2).

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

[13] Benjamin Dowling, Felix Günther, Udyani Herath, and Douglas Stebila. Secure logging schemes and Certificate Transparency. In *ESORICS*, 2016.

[14] Changing a certificate's domain. https://eff-certbot.readthedocs.io/en/stable/using.html#changing-a-certificate-s-domains, accessed 2022-08-01.

[15] Adam Eijdenberg, Ben Laurie, and Al Cutter. Verifiable data structures. https://github.com/google/trillian/blob/111e9369ab032e493a2f19f9be6d16c4f78ccca5/docs/papers/VerifiableDataStructures.pdf, accessed 2022-08-01.

[16] Alex Fink. Mnemonic .onion URLs. https://gitweb.torproject.org/torspec.git/tree/proposals/194-mnemonic-urls.txt, accessed 2022-08-01.

[17] Daniel Kahn Gillmor. Dangerous Labels in DNS and E-mail. Internet-Draft draft-dkg-intarea-dangerous-labels-01, IETF, 2022.

[18] George Kadianakis, Yawning Angel, and David Goulet. A name system API for Tor onion services. https://gitweb.torproject.org/torspec.git/tree/proposals/279-naming-layer-api.txt, accessed 2022-08-01.

[19] Google LLC. https://github.com/google/certificate-transparency-community-site/blob/master/docs/google/known-logs.md, accessed 2022-08-01.

[20] Ben Laurie. Re: [Trans] Mozilla's basic take on binary transparency. https://mailarchive.ietf.org/arch/msg/trans/1FxzTkn4LVxU6KN2P3YfbVsKpho/, accessed 2022-08-01.

[21] Ben Laurie. Certificate transparency. *CACM*, 57(10), 2014.

[22] Ben Laurie, Adam Langley, and Emilia Kasper. Certificate Transparency. RFC 6962, IETF, 2013.

[23] Bingyu Li, Jingqiang Lin, Fengjun Li, Qiongxiao Wang, Qi Li, Jiwu Jing, and Congli Wang. Certificate Transparency in the wild: Exploring the reliability of monitors. In *CCS*, 2019.

[24] Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Edward W. Felten, and Michael J. Freedman. CONIKS: Bringing key transparency to end users. In *USENIX Security*, 2015.

[25] Simon Migliano and Samuel Woodhams. Websites blocked in Russia since Ukraine invasion. https://www.top10vpn.com/research/websites-blocked-in-russia/, accessed 2022-08-01.

[26] Mozilla. Security/binary transparency. https://wiki.mozilla.org/Security/Binary_Transparency, accessed 2022-08-01.

[27] Alec Muffett. Onion Certificate Transparency log. https://github.com/alecmuffett/real-world-onion-sites, accessed 2022-08-01.

[28] Alec Muffett. Same origin onion certificates. https://crt.sh/?id=6819596552, accessed 2022-08-01.

[29] Linus Nordberg. Tor consensus transparency. https://gitweb.torproject.org/torspec.git/tree/proposals/267-tor-consensus-transparency.txt, accessed 2022-08-01.

[30] Linus Nordberg, Daniel Kahn Gillmor, and Tom Ritter. Gossiping in CT. Internet-draft draft-ietf-trans-gossip-05, IETF, 2018.

[31] Juha Nurmi. *Understanding the Usage of Anonymous Onion Services*. PhD thesis, Tampere University, Finland, 2019.

[32] nusenu. HAROI: Human readable authenticated relay operator identifier. https://lists.torproject.org/pipermail/tor-dev/2021-December/014688.html, accessed 2022-08-01.

[33] Tor Project. Onion-location. https://community.torproject.org/onion-services/advanced/onion-location/, accessed 2022-08-01.

[34] Tor Project. Onion services. https://community.torproject.org/onion-services/, accessed 2022-08-01.

[35] CA/Browser Forum. Ballot 144 – validation rules for .onion names. https://cabforum.org/2015/02/18/ballot-144-validation-rules-dot-onion-names/, accessed 2022-08-01.

[36] CA/Browser Forum. Ballot sc27v3: Version 3 onion certificates. https://cabforum.org/2020/02/20/ballot-sc27v3-version-3-onion-certificates/, accessed 2022-08-01.

[37] DigiCert Inc. Ordering a .onion certificate from DigiCert. https://www.digicert.com/blog/ordering-a-onion-certificate-from-digicert, accessed 2022-08-01.

[38] Harica. DV certificates for onion websites. https://news.harica.gr/article/onion_announcement/, accessed 2022-08-01.

[39] Silvio Rhatto. Sauteed week API backend. https://gitlab.torproject.org/rhatto/sauteed-week/-/blob/main/docs/api.md, accessed 2022-08-01.

[40] Nolen Scaife, Henry Carter, and Patrick Traynor. OnionDNS: A seizure-resistant top-level domain. In *CNS*, 2015.

[41] SecureDrop. Getting an onion name for your SecureDrop. https://securedrop.org/faq/getting-onion-name-your-securedrop/, accessed 2022-08-01.

[42] Emily Stark, Joe DeBlasio, Devon O'Brien, Davide Balzarotti, William Enck, Samuel King, and Angelos Stavrou. Certificate Transparency in Google Chrome: Past, present, and future. *IEEE Secur. Priv.*, 19(6), 2021.

[43] Emily Stark, Ryan Sleevi, Rijad Muminovic, Devon O'Brien, Eran Messeri, Adrienne Porter Felt, Brendan McMillion, and Parisa Tabriz. Does Certificate Transparency break the web? Measuring adoption and error rate. In *IEEE S&P*, 2019.

[44] Paul Syverson. The once and future onion. In *ESORICS*, 2017.

[45] Paul Syverson, Matt Finkel, Saba Eskandarian, and Dan Boneh. Attacks on onion discovery and remedies via self-authenticating traditional addresses. In *WPES*, 2021.

[46] Paul Syverson and Matt Traudt. Self-authenticating traditional domain names. In *SecDev*, 2019.

[47] Filippo Valsorda. How Plex is doing HTTPS for all its users. https://words.filippo.io/how-plex-is-doing-https-for-all-its-users/, accessed 2022-08-01.

[48] Jesse Victors, Ming Li, and Xinwen Fu. The onion name system. *PETS*, 2017(1).

[49] Philipp Winter, Anne Edmundson, Laura M. Roberts, Agnieszka Dutkowska-Zuk, Marshini Chetty, and Nick Feamster. How do Tor users interact with onion services? In *USENIX Security*, 2018.

A Onion Association Search Examples

We host the search engine described in Section 3.3.2 on a Debian VM with 1GB RAM, 20GB SSD, and a single vCPU. It is available at `api.sauteed-onions.org` as well as `zpadxxmoi42k45iifrzuktwqktihf5didbaec3xo4dhvlw2hj54doiqd.onion`. Please note that we operate this prototype on a best-effort level until December, 2022.

An example for the `search` endpoint is provided in Figure 3, followed by extracting additional certificate information using the `get` endpoint in Figure 4. There are many CT-logged certificates for the same onion association because certificates are renewed periodically and typically submitted to multiple CT logs.

B Configuration Example

We used `certbot` to set up sauteed onions using Let's Encrypt and `apache` on a Debian system. The difference when compared to the usual `certbot` instructions is that the `-d` flag must be specified to enumerate all SANs as a comma-separated list [14]. The domain name with an associated onion address as a subdomain also needs to be reachable via DNS for Let's Encrypt to perform domain validation. Therefore, an appropriate A/AAAA or CNAME record is required.

A sanity-check for `www.sauteed-onions.org` would be to verify that `dig qvrbktnwsztjnbga6yyjbwzsdjw7u5a6vsyzv6hkj75clog4pdvy4cydonion.www.sauteed-onions.org` returns the same IP address as `dig www.sauteed-onions.org` before running `certbot -apache -d www.sauteed-onions.org,qvrbktnwsztjnbga6yyjbwzsdjw7u5a6vsyzv6hkj75clog4pdvy4cydonion.www.sauteed-onions.org`. See `crt.sh` for an example certificate [3].

```
$ curl -s https://api.sauteed-onions.org/search?in=www.sauteed-onions.org | json_pp
[
   {
      "identifiers" : [
         "2",
         "3",
         "24",
         "25",
         "28",
         "29",
         "37"
      ],
      "onion_addr" : "qvrbktnwsztjnbga6yyjbwzsdjw7u5a6vsyzv6hkj75clog4pdvy4cyd.onion",
      "domain_name" : "www.sauteed-onions.org"
   }
]
```

Figure 3: Find onion associations for `www.sauteed-onions.org`.

```
$ curl -s https://api.sauteed-onions.org/get?id=2 | json_pp
{
   "onion_addr" : "qvrbktnwsztjnbga6yyjbwzsdjw7u5a6vsyzv6hkj75clog4pdvy4cyd.onion",
   "domain_name" : "www.sauteed-onions.org",
   "log_id" : "b1N2rDHwMRnYmQCkURX/dxUcEdkCwQApBo2yCJo32RM=",
   "log_index" : 582362461,
   "cert_path" : "db/logs/Mammoth/582362461.pem"
}
$ curl -L https://api.sauteed-onions.org/db/logs/Mammoth/582362461.pem | \
  openssl x509 -text -noout
...
```

Figure 4: Get further information relating to the certificate with identifier “2”.

Paper V

Website Fingerprinting with Website Oracles

Reprinted from

PETS (2020)

Website Fingerprinting with Website Oracles

Tobias Pulls and Rasmus Dahlberg

Abstract

Website Fingerprinting (WF) attacks are a subset of traffic analysis attacks where a local passive attacker attempts to infer which websites a target victim is visiting over an encrypted tunnel, such as the anonymity network Tor. We introduce the security notion of a *Website Oracle* (WO) that gives a WF attacker the capability to determine whether a particular monitored website was among the websites visited by Tor clients at the time of a victim's trace. Our simulations show that combining a WO with a WF attack—which we refer to as a WF+WO attack—significantly reduces false positives for about half of all website visits and for the vast majority of websites visited over Tor. The measured false positive rate is on the order one false positive per million classified website trace for websites around Alexa rank 10,000. Less popular monitored websites show orders of magnitude lower false positive rates.

We argue that WOs are inherent to the setting of anonymity networks and should be an assumed capability of attackers when assessing WF attacks and defenses. Sources of WOs are abundant and available to a wide range of realistic attackers, e.g., due to the use of DNS, OCSP, and real-time bidding for online advertisement on the Internet, as well as the abundance of middleboxes and access logs. Access to a WO indicates that the evaluation of WF defenses in the open world should focus on the highest possible recall an attacker can achieve. Our simulations show that augmenting the Deep Fingerprinting WF attack by Sirinam *et al.* [75] with access to a WO significantly improves the attack against five state-of-the-art WF defenses, rendering some of them largely ineffective in this new WF+WO setting.

1 Introduction

A Website Fingerprinting (WF) attack is a type of traffic analysis attack where an attacker attempts to learn which websites are visited through encrypted network tunnels—such as the low-latency anonymity network Tor [22] or Virtual Private Networks (VPNs)—by analysing the encrypted network traffic [12, 29, 30, 41, 61, 77]. The analysis considers only the size and timing of encrypted packets sent over the network to and from a target client. This makes it possible for attackers that only have the limited *capability* of observing the encrypted network traffic (sometimes referred to as a *local eavesdropper*) to perform WF attacks. Sources of such capabilities include ISPs, routers, network interface cards, WiFi hotspots, and guard relays in the Tor network, among others. Access to encrypted network traffic is typically not well-protected over the Internet because it is already in a form that is considered safe to expose to attackers due to the use of encryption.

The last decade has seen significant work on improved WF attacks (e.g., [9, 28, 75, 85]) and defenses (e.g, [7, 8, 37, 47]) accompanied by an ongoing debate on the real-world impact of these attacks justifying the deployment of defenses or not, in particular surrounding Tor (e.g., [36, 62, 86]). There are significant real-world challenges for an attacker to successfully perform WF attacks, such as the sheer size of the web (about 200 million active websites [57]), detecting the beginning of website loads in encrypted network traces, background traffic, maintaining a realistic and fresh training data set, and dealing with false positives.

Compared to most VPN implementations, Tor has some basic but rather ineffective defenses in place against WF attacks, such as padding packets to a constant size and randomized HTTP request pipelining [9, 22, 85]. Furthermore, Tor recently started implementing a framework for circuit padding machines to make it easier to implement traffic analysis defenses [51] based on adaptive padding [37, 74]. However, the unclear real-world impact of WF attacks makes deployment of proposed effective (and often prohibitively costly in terms of bandwidth and/or latency overheads) WF defenses a complicated topic for researchers to reach consensus on and the Tor Project to decide upon.

1.1 Introducing Website Oracles

In this paper, we introduce the security notion of a *Website Oracle* (WO) that can be used by attackers to augment any WF attack. A WO answers "yes" or "no" to the question "was a particular website visited over Tor at this point in time?". We show through simulation that such a *capability*—access to a WO—greatly reduces the false positive rate for an attacker attempting to fingerprint the majority of websites and website visits through the Tor network. The reduction is to such a great extent that our simulations suggest that false positives are no longer a significant reason for why WF attacks lack real-world impact. This is in particular the case for onion services where the estimated number of websites is a fraction compared to the "regular" web [34].

Our simulations are based on the privacy-preserving network measurement results of the live Tor network in early 2018 by Mani *et al.* [49]. Besides simulating WOs we also identify a significant number of potential sources of WOs that are available to a wide range of attackers, such as nation state actors, advertisement networks (including their customers), and operators of Tor relays. Some particularly practical sources—due to DNS and how onion services are accessed—can be used by anyone with modest computing resources.

We argue that sources of WOs are inherent in Tor due to its design goal of providing *anonymous* and not *unobservable* communication: observable anonymity sets are inherent for anonymity [38, 63, 68], and a WO can be viewed as simply being able to query for membership in the destination/recipient anonymity set (the potential websites visited by a Tor client). The solution to the effectiveness of WF + WO attacks is therefore not to eliminate all sources—that would be impossible without unobservable communication [38, 63, 68]—but to assume that an attacker has WO access when evaluating the effectiveness of WF attacks and defenses, even for weak attackers like local (passive) eavesdroppers.

The introduction of a WO in the setting of WF attacks is similar to how encryption schemes are constructed to be secure in the presence of an attacker with access to *encryption* and *decryption* oracles (chosen plaintext and ciphertext attacks, respectively) [25, 55, 67]. This is motivated by the real-world prevalence of such oracles, and the high impact on security when paired with other weaknesses of the encryption schemes: e.g., Bleichenbacher [5] padding oracle attacks remain an issue in modern cryptosystems today despite being discovered about twenty years ago [52, 71].

1.2 Contributions and Structure

Further background on anonymity, Tor, and WF are presented in Section 2. Section 3 defines a WO and describes two generic constructions for combining a WO with *any* WF attack. Our generic constructions are a type of Classify-Verify method by Stolerman *et al.* [76], first used in the context of WF attacks by Juarez *et al.* [36] and later by Greschbach *et al.* [26]. Section 4 presents a number of sources of WOs that can be used by a wide range of attackers. We focus on practical sources based on DNS and onion service directories in Tor, offering *probabilistic* WOs that anyone can use with modest resources. We describe how we simulate access to a WO throughout the rest of the paper in Section 5, based on Tor network measurement data from Mani *et al.* [49].

Section 6 experimentally evaluates the performance of augmenting the state-of-the-art WF attack Deep Fingerprinting (DF) by Sirinam *et al.* [75] with WO access using one of our generic constructions. We show significantly improved classification performance against unprotected Tor as well as against traces defended with the WF defenses WTF-PAD by Juarez *et al.* [37] and Walkie-Talkie by Wang and Goldberg [87], concluding that the defenses are ineffective in this new setting where an attacker has access to a WO. Further, we also evaluate DF with WO access against Wang *et al.*'s dataset [85] with simulated traces for the constant-rate WF defenses CS-BuFLO and Tamaraw by Cai et al. [7, 8]. Our results show that constant-rate defenses are overall effective defenses but not efficient due to the significant induced overheads. We then evaluate two configurations of the WF defense DynaFlow by Lu *et al.* [47], observing similar effectiveness as CS-BuFLO but at lower overheads approaching that of WTF-PAD and Walkie-Talkie.

In Section 7 we discuss our results, focusing on the impact on false positives with WO access, how imperfect sources for WOs impact WF+WO attacks, limitations of our work, and possible mitigations. Our simulations indicate that WF defenses should be evaluated against WF attacks based on how they minimise *recall*. We present related work in Section 8, including how WF+WO attacks relate to traffic correlation and confirmation attacks. Section 9 briefly concludes this paper.

2 Background

Here we present background on terminology, the anonymity network Tor, and WF attacks and defenses.

2.1 Anonymity and Unobservability

Anonymity is the state of a subject not being identifiable from an attackers perspective within the *anonymity set* of possible subjects that performed an action such as sending or receiving a message [63]. For an anonymity network, an attacker may not be able to determine who sent a message into the network—providing a sender anonymity set of all possible senders—and conversely, not be able to determine the recipient of a message from the network out of all possible recipients in the recipient anonymity set. Inherent for anonymity is that the subjects in an anonymity set change based on what the attacker observes, e.g., when some subjects send or receive messages [38, 68]. In gist, anonymity is concerned with hiding the *relationship* between a sender and recipient, not its existence.

Unobservability is a strictly stronger notion than anonymity [38, 63, 68]. In addition to anonymity of the relationship between a sender and recipient, unobservability also requires that an attacker (not acting as either the sender or recipient) cannot sufficiently distinguish if there is a sender or recipient or not [63]. Perfect unobservability is therefore the state of an attacker being unable to determine if a sender/recipient should be part of the anonymity set or not.

2.2 Tor

Tor is a low-latency anonymity network for anonymising TCP streams with about eight million daily users, primarily used for anonymous browsing, censorship circumvention, and providing anonymous (onion) services [22, 49]. Because Tor is designed to be usable for low-latency tasks such as web browsing, its threat model and design does not consider powerful attackers, e.g., global passive adversaries that can observe all network traffic on the Internet [20, 22]. However, less powerful attackers such as ISPs and ASes that observe a fraction of network traffic on the Internet are in scope.

Users typically use Tor Browser—a customised version of Mozilla Firefox (bundled with a local relay)—as a client that sends traffic through three *relays* when browsing a website on the regular Internet: a guard, middle, and exit

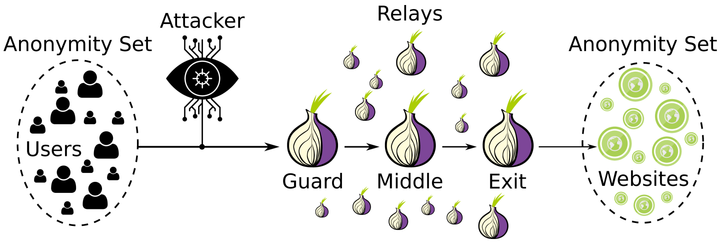


Figure 1: Using Tor to browse to a website, where an attacker observes the encrypted traffic into the Tor network for a target user, attempting to determine the website the user is visiting.

relay. Traffic from the client to the exit is encrypted in multiple layers as part of fixed-size cells such that only the guard relay knows the IP-address of the client and only the exit relay knows the destination website. There are about 7000 public relays at the time of writing, all available in the *consensus* generated periodically by the network. The consensus is public and therefore anyone can trivially determine if traffic is coming from the Tor network by checking if the IP-address is in the consensus. Note that the encrypted network traffic in Tor is exposed to network adversaries as well as relays as it traverses the Internet. Figure 1 depicts the setting just described, highlighting the anonymity sets of users of Tor Browser and the possible destination websites.

2.3 Website Fingerprinting

As mentioned in the introduction, attacks that analyse the encrypted network traffic (a trace) between a Tor client and a guard relay with the goal to detect the website a client is visiting are referred to as *website fingerprinting* (WF) attacks. Figure 1 shows the typical location of the attacker, who can also be the guard itself. WF attacks are evaluated in either the *closed* or the *open* world. In the closed world, an attacker *monitors* a number of websites and it is the goal of the attacker to determine which website out of all the possible monitored websites a target is visiting. The open world is like the closed world with one significant change: the target user may also visit *unmonitored* websites. This means that in the open world the attacker may also classify a trace as unmonitored in addition to monitored, posing a significantly greater challenge for the attacker in a more realistic setting than the closed world. The ratio between monitored and unmonitored traces in a dataset is further a significant challenge for WF attacks when assessing their real-world significance for Tor [36]. Typically, WF attacks are evaluated on the frontpages of web*sites*: web*page* fingerprinting is presumably much more challenging due to the orders of magnitude of more webpages than websites. Unless otherwise stated, we only consider the frontpages of websites in this paper.

2.3.1 Website Fingerprinting Attacks

Prior to WF attacks being considered for use on Tor, they were used against HTTPS [12], web proxies [30, 77], SSH tunnels [41], and VPNs [29]. For Tor, WF attacks are typically based on machine learning and can be categorized based on if they use deep learning or not.

Traditional WF attacks in the literature use manually engineered features extracted from both the size and timing of packets (and/or cells) sent by Tor. State of the art attacks with manually engineered features are Wang-kNN [85], CUMUL [59], and k-FP [28]. For reference, Wang-kNN has 1225 features, CUMUL 104 features, and k-FP 125 features. In terms of accuracy, k-FP appears to have a slight edge over the other two, but all three report over 90% accuracy against significantly sized datasets. As traditional WF attacks progressed, the features more than the type of machine learning method have shown to be vital for the success of attacks, with an emerging consensus on what are important features (e.g., coarse features like number of incoming and outgoing packets) [13, 28, 59].

Deep learning was first used for WF attacks by Abe and Goto in 2016 [2]. Relatively quickly, Rimmer *et al.* reached parity with traditional WF attacks, lending credence to the emerging consensus that the research community had found the most important features for WF [70]. However, recently Sirinam et al. [75] with Deep Fingerprinting (DF) significantly improved on other WF attacks, also on the WTF-PAD and Walkie-Talkie defenses, and is at the time of writing considered state-of-the-art. DF is based on a Convolutional Neural Network (CNN) with a customized architecture for WF. Each packet trace as input to DF is simply a constant size (5000) list of cells (or packets) and their direction (positive for outgoing, negative for incoming), ignoring size and timings. Based on the input, the CNN learns features on its own: we do not know what they are, other than preliminary work indicating that the CNN gives more weight to input early in the trace [50].

The last layer of the CNN-based architecture of DF is a *softmax* function: it assigns (relative) probabilities to each class as the output of classification. These probabilities allow a threshold to be defined for the final classification in the open world, requiring that the probability of the most likely class is above the threshold to classify as a monitored website.

2.3.2 Website Fingerprinting Defenses

WF defenses for Tor modify the timing and number of (fixed-size) cells sent over Tor when a website is visited. The modifications are done by injecting dummy traffic and introducing artificial delays. Defenses can typically be classified as either based on constant-rate traffic or not, where constant rate defenses force all traffic to fit a pre-determined structure, forming *collision sets* for websites where their traffic traces appear identical to an attacker. Non-constant rate defenses simply more-or-less randomly inject dummy traffic and/or artificial delays with the hope of obfuscating the resulting network traces. WF defenses are typically compared in terms of their induced *bandwidth* (BOH) and *time*

(TOH) overheads compared to no defense. Further, different WF defenses make more or less realistic and/or practical assumptions, making comparing overheads necessary but not nearly sufficient for reaching conclusions.

We briefly describe WF defenses that we later use to evaluate the impact of attackers performing enhanced WF attacks with access to WOs:

Walkie-Talkie by Wang and Goldberg [87] puts Tor Browser into half duplex mode and pads traffic such that different websites result in the same cell sequences. This creates a collision set between a visited website and a target *decoy website* which results in the same cell sequence with the defense. Their evaluation shows 31% BOH and 34% TOH. Collision sets grow beyond size two at the cost of BOH.

WTF-PAD by Juarez *et al.* [37] is based on the idea of *adaptive padding* [74] where fake padding is injected only when there is no real traffic to send. The defense is simulated on collected packet traces and its design is the foundation of the circuit padding framework recently implemented in Tor. The simulations report 50-60% BOH and 0% TOH.

CS-BuFLO by Cai *et al.* [7] is a *constant rate* defense where traffic is always sent at a constant rate between a sender and receiver, improving on prior work by Dyer et al. [23]. Their evaluation shows 220-270% BOH and 270-340% TOH.

Tamaraw by Cai *et al.* [8] is another constant rate defense that further improves on CS-BuFLO. In the evaluation by Wang and Goldberg, they report 103% BOH and 140% TOH for Tamaraw [87].

DynaFlow by Lu *et al.* [47] is a *dynamic* constant-rate defense that allows for the defense to adjust its parameters (notably the "inter-packet interval") based on configuration and on the observed traffic. The evaluation shows an overall improvement over Tamaraw when configured to use similar overheads.

The primary downside of defenses like Walkie-Talkie that depend on creating collision sets for websites is that they require up-to-date knowledge of the target website(s) to create collisions with (to know how to morph the traffic traces): this is a significant practical issue for deployment [58, 85, 87]. Constant rate defenses like CS-BuFLO and Tamaraw are easier to deploy but suffer from significant overheads [7, 8]. WTF-PAD is hard to implement both efficiently and effectively in practice due to only being simulated on packet traces as-is and also being vulnerable to attacks like Deep Fingerprinting [37, 75]. While DynaFlow shows great promise, but requires changes at the client (Tor Browser, local relay, or both) and at exit relays to *combine* packets with payloads smaller than Tor's cell size [47]. Without combined packets its advantage in terms of overhead compared to Tamaraw likely shrinks.

2.4 Challenges for WF Attacks in Practice

A number of practical challenges for an attacker performing WF attacks have been highlighted over the years, notably comprehensively so by Mike Perry of the Tor Project [62] and Juarez et al. [36]. Wang and Goldberg have showed that several of the highlighted challenges—such as maintaining a fresh data set and determining when websites are visited—are practical to overcome [86]. What remains are two notably significant challenges: distinguishing between different goals of the attacker and addressing false positives.

For attacker goals when performing WF attacks, an attacker may want to detect website visits with the goal of censoring access to it, to identify all users that visit particular websites, or to identify every single website visited by a target [62]. Clearly, these goals put different constraints on the attacker. For censorship, classification must happen before content is actually allowed to be completely transferred to the victim. For monitoring only a select number of websites the attacker has the most freedom, while detecting all website visits by a victim requires the attacker to have knowledge of all possible websites on the web.

For addressing false positives there are a number of aspects to take into account. First, the web has millions of websites that could be visited by a victim (not the case for onion services [34]), and each website has a significant number of webpages that are often dynamically generated and frequently changed [36, 62]. Secondly, how often victims potentially visit websites that are monitored by an attacker is unknown to the attacker, i.e., the *base rate* of victims are unknown. The base rate leads to even a small false positive rate of a WF attack overwhelming an attacker with orders of magnitude more false positives than true positives, leaving WF attacks impractical for most attacker goals in practice.

3 Website Oracles

We first define a WO and then present two generic constructions for use with WF attacks based on the kind of output the WF attack supports.

3.1 Defining Website Oracles

Definition 1. A website oracle answers true or false to the question "was a particular monitored website w visited over the Tor network at time t?".

A WO considers only web*sites* and not web*pages* for w, but note that even for webpage fingerprinting being able to narrow down the possible websites that webpages belong to through WO access is a significant advantage to an attacker. The time t refers to a *period of time* or *timeframe* during which a visit should have taken place. Notably, different sources of WOs may provide different *resolutions* for time, forcing an attacker to consider a timeframe in which a visit could have taken place. For example, timestamps in Apache or nginx access logs use regular Unix timestamps as default (i.e., seconds), while CDNs like Cloudflare maintain logs with Unix nanosecond precision. Further, there are inherent limitations in approximating t for the query when the attacker in addition to WO access can only directly observe traffic from the victim into Tor. We explore this later in Section 5.3.

One important limitation we place on the use of a WO with WF is that the attacker can only query the WO for *monitored* websites. The open world setting is intended to capture a more realistic setting for evaluating attacks, and inherent in this is that the attacker cannot train (or even enumerate) all possible websites on the web. Given the ability to enumerate and query all possible websites gives the adversary a capability in line with a global passive adversary performing correlation attacks, which is clearly outside of the threat model of Tor [22]. We further relate correlation and confirmation attacks to WF+WO attacks in Section 8.

Definition 1 defines the ideal WO: it never fails to observe a *monitored* website visit, it has no false positives, and it can answer for an arbitrary t. This is similar to how encryption and decryption oracles always encrypt and decrypt when modelling security for encryption schemes [25, 55, 67]. In practice, sources of all of these oracles may be more or less ideal and challenging for an attacker to use. Nevertheless, the prevalence of sources of these imperfect oracles motivate the assumption of an attacker with access to an ideal oracle. Similarly, for WOs, we motivate this assumption in Sections 4 and 5, in particular wrt. a timeframe on the order of (milli)seconds. Section 7 further considers non-ideal sources of WOs and the effect on WF+WO attacks, both when the WO can produce false positives and when the source only observes a fraction of visits to monitored websites.

3.2 Generic Website Fingerprinting Attacks with Website Oracles

As mentioned in Section 2.3, a WF attack is a classifier that is given as input a packet trace and provides as output a classification. The classification is either a monitored site or a class representing unmonitored (in the open world). Figure 2 shows the setting where an attacker capable of performing WF attacks also has access to a WO. We define a generic construction for WF+WO attacks that works with *any* WF attack in the open world in Definition 2:

Definition 2 (Binary verifier)**.** Given a website oracle o and WF classification c of a trace collected at time t, if c is a monitored class, query the oracle $o(c, t)$. Return c if the oracle returns true, otherwise return the unmonitored class.

Note that the WO is only queried when the WF *classification* is for a monitored website and that Definition 2 is a generalisation of the "high precision" DefecTor attack by Greschbach *et al.* [26]. In terms of *precision* and *false positives*, the above generic WF+WO construction is strictly superior to a WF attack without a WO. Assume that the WF classification incorrectly classified an unmonitored trace as monitored, then there is *only a probability* that a WO

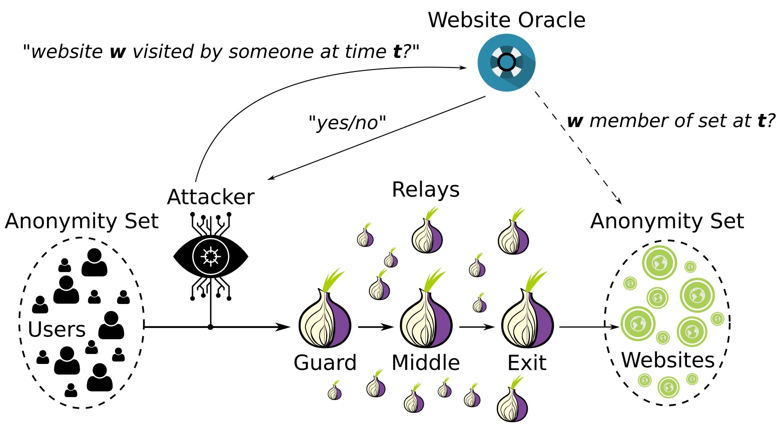


Figure 2: WF+WO attacks, where the WO infers membership of a particular website w in the website anonymity set of all possible websites visited over Tor during a particular timeframe t.

also returns true, depending on the probability that someone else visited the website in the same timeframe over Tor. If it does not, then a false positive is prevented. That is, a WF attack without WO access is identical to a WF attack with access to a useless WO that always returns true; any improvements beyond that will only help the attacker in ruling out false positives. We consider the impact on *recall* later.

We can further refine the use of WOs for the subset of WF attacks that support providing as output an ordered list of predictions in decreasing likelihood, optionally with probabilities, as shown in Definition 3:

Definition 3 (List verifier)**.** Given an ordered list of predictions in the open world and a website oracle:

```
for top prediction $p$ in list do
    if $p$ is unmonitored or oracle says $p$ visited then
        return list
    move $p$ to last in list and optionally update probabilities
```

First, we observe that if the WF attack thinks that it is most likely an unmonitored website, then we accept that because a WO can only teach us something new about monitored websites. Secondly, if the most likely prediction has been visited according to the WO then we also accept that classification result. Finally, all that is left to do is to consider this while repeatedly iterating over the top predictions: if the top classification is a monitored website that has not been visited according to the WO, then move it from the top of the list and optionally update probabilities (if applicable, then also set $p = 0.0$ before updating) and try again. Per definition, we will either hit the case of a monitored website that has been visited according to the WO or an unmonitored prediction. As mentioned in Section 2.3, WF output that has some sort of probability or threshold associated with classifications are useful for attackers with different requirements wrt. false positives and negatives.

One could consider a third approach based on repeatedly querying a WO to first determine if any monitored websites have been visited and then train an optimised classifier (discarding monitored websites that we know have not been visited). While this may give a minor improvement, our results later in this paper as well as earlier work show that confusing monitored websites is a minor issue compared to confusing an unmonitored website as monitored [26, 36, 85].

4 Sources of Website Oracles

There are a wide range of potential sources of WOs. Table 1 summarizes a selection of sources that are more thoroughly detailed in Appendix C. The table shows the availability of the source, i.e., if the attacker needs to query the source in near real-time as a website visit occurs or if it can be accessed retroactively, e.g., through a legal request. We also estimate qualitatively the false positive rate of the source, its coverage of websites it can monitor (or fraction of Tor network traffic, depending on source), as well as the estimated effort to access the source. Finally, the table gives an example of an actor with access to the source.

Next we focus on a number of sources of WOs that we find particularly relevant: several due to DNS in Section 4.1, the DHT of Tor onion directory services in Section 4.2, and real-time bidding platforms in Section 4.3.

4.1 DNS

Before a website visit the corresponding domain name must be resolved to an IP address. For a user that uses Tor browser, the exit relay of the current circuit resolves the domain name. If the DNS record of the domain name is already cached in the DNS cache of the exit relay, then the exit relay uses that record. Otherwise the domain name is resolved and subsequently cached using whichever DNS resolution mechanism that the exit relay has configured. Based on this process we present three sources of WOs that work for unpopular websites.

4.1.1 Shared Pending DNS Resolutions

If an exit relay is asked to resolve a domain name that is uncached it will create a list of pending connections waiting for the domain resolution to finish. If another connection asks that the same domain name be resolved, it is added to the list of pending connections. When a result is available all pending connections are informed. This is the basis of a WO: if a request to resolve a domain name returns a record *more quickly than previously measured by the attacker for uncached entries*, the entry was either pending resolution at the

Table 1: Comparison of a number of WO sources based on their *estimated* time of availability (when attacker likely has to collect data, i.e., retroactively or real-time), False Positive Rate (FPR), coverage of website/network visits, and primary entities with access.

Source	Availability	FPR	Coverage	Effort	Access
Dragnet surveillance programmes	retroactive	negl.	high	high	intelligence agencies
Content Delivery Networks	retroactive	negl.	high	high	operators
Real-time bidding	real-time (retroactive)	negl.	high	modest	customers (operator)
Webserver access logs	retroactive	negl.	high	medium	operators
Middleboxes	retroactive [1]	negl.	medium	medium	operators
OCSP	retroactive	low	high	medium	few CAs, plaintext
8.8.8.8 operator	retroactive	low [26]	16.8% of visits	high	Google, plaintext
1.1.1.1 operator	retroactive	low [26]	7.4% of visits	high	Cloudflare, plaintext
Exit relays	real-time	negl.	low	low	operators
Exit relays DNS cache	real-time	medium	high	medium	anyone
Query DNS resolvers	real-time	high	low	low	anyone
Onion v2 (v3)	real-time	negl.	high (low)	low (high)	anyone

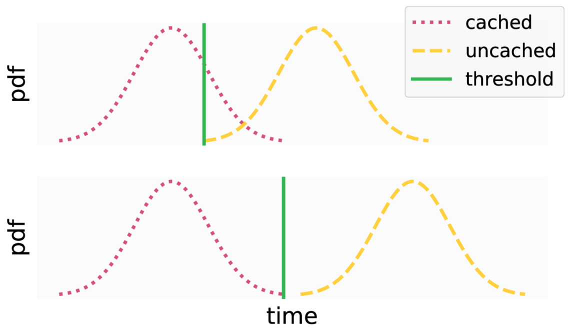


Figure 3: The two cases when deciding on a classifier's threshold.

time of the request or already cached. Notably this works regardless of if exit relays have DNS caches or not. However, the timing constraints of shared pending connections are significant and thus a practical hurdle to overcome.

4.1.2 Tor's DNS Cache at Exit Relays

If an unpopular website is visited by a user, the resolved domain name will likely be cached by a *single* relay. We performed 411 `exitmap` [88] measurements between April 1–10 (2019), collecting on average 3544 (un)cached data points for each exit using a domain under our control that is not used by anyone else.

Given a labelled data set of (un)cached times for each exit relay, we can construct distinct *per-relay* classifiers that predict whether a measured time corresponds to an (un)cached domain name. While there are many different approaches that could be used to build such a classifier, we decided to use a simple heuristic that should result in little or no false positives: output 'cached' iff no uncached query has *ever* been this fast before. Figure 3 shows the idea of this classifier in greater detail, namely create a *threshold* that is the minimum of the largest cached time and the smallest uncached time and then say cached iff the measured time is smaller than the threshold. Regardless of how well this heuristic performs (see below), it should be possible to construct other classifiers that exploit the trend of smaller resolve times and less standard deviation for cached queries (Figure 4). For example, 69.1% of all exit relays take at least 50 ms more time to resolve an uncached domain on average.

To estimate an *upper bound* on how effective the composite classifier of all per-relay classifiers could be *without any false positives* using our heuristic, we applied ten-fold cross-validation to simply exclude every exit relay that had false positives during any fold and then weighted the observed bandwidth for the remaining classifiers by the individual true positive rates. This gives us an estimate of how much bandwidth we could predict true positives for without having any false positives. By comparing it to the total exit bandwidth of the Tor network, we obtain an estimated upper bound true positive rate for the composite classifier of 17.3%.

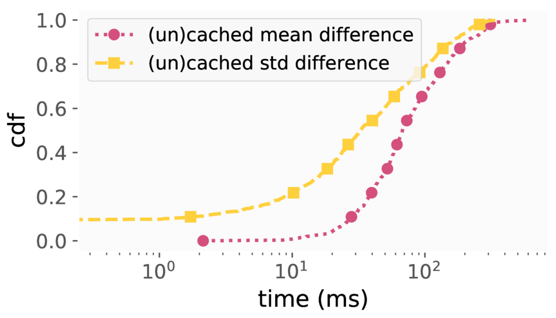


Figure 4: The difference between (un)cached standard deviation and mean times without any absolute values, i.e., a negative value implies that the uncached time is smaller than the cached time.

When an attacker measures if a domain is cached or not the domain will, after the measurement, be cached for up to an hour (current highest caching duration in Tor, independent of TTL) at every exit. However, if a an attacker can cause an exit to run low on memory, the entire DNS cache will be removed (instead of only parts of it) due to a bug in the out-of-memory manager of Tor. We have reported this to the Tor Project [65]. We further discuss in Section 7 how frequently an attacker on average can be expected to query a WO.

4.1.3 Caching at Recursive DNS Resolvers

For a website that is unpopular enough, there is a high chance that nobody on the web visited the website within a given timeframe. This is the basis of our next idea for a WO which is *not mutually exclusive* to the Tor network: wait a couple of seconds after observing a connection, then probe all recursive DNS resolvers of Tor exits that can be accessed to determine whether any monitored website was cached approximately at the time of observing the connection by inspecting TTLs.

In 2016 Greschbach *et al.* [26] showed that remote DNS resolvers like Google's `8.8.8.8` receive a large portion of all DNS traffic that exit relays generate. To better understand how the DNS resolver landscape looks today, we repeated their RIPE Atlas experiment setup for 35 hours in February 17–18 (2019), measuring every 30 minutes. Our results show that Google (16.8%) and Cloudflare (7.4%) are both popular. Many exits use a same-AS resolver which is presumably the ISP (42.3%), while other exits resolve themselves (15.2%) or use a remote DNS resolver that we did not identify (18.2%). Further, we note that there are at least one RIPE Atlas network measurement probe in the same AS as 53.3% of all exits, providing access to many of the same DNS resolvers as used by exits from a similar network vantage point.

Instead of using RIPE Atlas nodes we opted for a different approach which is *strictly worse*: query Google's and Cloudflare's DNS resolvers from VMs in

16 Amazon EC2 regions. With a simple experiment of first visiting a unique domain (once again under our control and only used by us) using `torify curl` and then querying the DNS resolvers from each Amazon VM to observe TTLs, we got true positive rates of 2.9% and 0.9% for Google and Cloudflare with 1000 repetitions. While this may seem low, the cost for an attacker is at the time of writing about 2 USD per day using on-demand pricing. Using an identical setup we were also able to find a subset of monitored websites that yield alarmingly high true positive rates: 61.4% (Google) and 8.0% (Cloudflare). Presumably this was due to the cached entries being shared over a wider geographical area for some reason (however, not globally). Regardless, coupled with the fact that anyone can *globally purge* the DNS caches of Google [43] and Cloudflare [33] for arbitrary domain names, this is a noteworthy WO source.

4.2 Onion Service Directories in Tor

To access an onion service a user first obtains the service's *descriptor* from a Distributed Hash Table (DHT) maintained by *onion service directories*. From the descriptor the user learns of *introduction points* selected by the host of the onion service in the Tor network that are used to establish a connection to the onion service in a couple of more steps [79, 80] that are irrelevant here. Observing a request for the descriptor of a monitored onion service is a source for a WO. To observe visits for a target (known) onion service in the DHT, a relay first has to be selected as one out of six or eight (depending on version) relays to host the descriptor in the DHT, and then the victim has to select that relay to retrieve the descriptor. For v2 of onion services, the location in the DHT is deterministic [79] and an attacker can position its relays in such a way to always be selected for hosting target descriptors. Version 3 of onion services addresses this issue by randomising the process every 24 hours [80], forcing an attacker to host a significant number of relays to get a WO for onion services with high coverage. At the time of writing, there are about 3,500 relays operating as onion service directories.

4.3 Real-Time Bidding

Real-Time Bidding (RTB) is an approach towards online advertisement that allows a publisher to auction ad space to advertisers on a *per-visit* basis in real time [83]. Google's Display Network includes more than two million websites that reach 90% of all Internet users [42], and an advertiser that uses RTB must respond to submitted bid requests containing information such as the three first network bytes of an IPv4 address, the second-level domain name of the visited website, and the user agent string within ≈100 ms [45]. While the exact information available to the bidder depends on the ad platform and the publisher's advertisement settings, anonymous modes provide less revenue [46]. Combined with many flavours of pre-targeting such as IP and location filtering [82], it is likely that the bidder knows whether a user used Tor while accessing a monitored website. Vines et al. [82] further note that

"35% of the DSPs also allow arbitrary IP white-and blacklisting (Admedo, AdWords, Bing, BluAgile, Criteo, Centro, Choozle, Go2Mobi, Simpli.fi)". Finally, observe that an attacker need not win a bid to use RTB as a WO.

5 Simulating Website Oracles

To be able to *simulate* access to a WO for *arbitrary monitored websites* we need to simulate the entire website anonymity set of Tor, because the anonymity set is what a WO queries for membership. We opt for simulation for ethical reasons. The simulation has three key parts: how those visits are distributed, the number of visits to websites over Tor, and the timeframe (resolution) of the oracle source. Note that the first two parts are easy for an attacker to estimate by simply observing traffic from live Tor exit relays, something we cannot trivially do as researchers adhering to Tor's research safety guidelines [64]. Another option available to an attacker is to repeatedly query a WO to learn about the popularity of its monitored websites and based on those figures infer the utility of the WO. We opted to not perform such measurements ourselves, despite access to several WOs, due to fears of inadvertently harming Tor users. Instead we base our simulations on results from the privacy-preserving measurements of the Tor network in early 2018 by Mani *et al.* [49].

5.1 How Website Visits are Distributed

Table 2 shows the average inferred website popularity from Mani *et al.* [49]. The average percentage does not add up to 100%, presumably due to the privacy-preserving measurement technique or rounding errors. Their results show that `torproject.org` is very popular (perhaps due to a bug in software using Tor), and beyond that focus on Alexa's [3] top one million most popular websites as bins. The "other" category is for websites identified not part of Alexa's top one million websites ranking. For the rest of the analysis (not simulation) in this paper we *exclude* `torproject.org`: for one, that Tor users visit that website is unlikely to be an interesting fact for an attacker to monitor, and its over-representation (perhaps due to a bug) will skew our analysis. Excluding `torproject.org`, about one third of all website visits go to Alexa (0,1k], one third to Alexa (1k,1m], and one third to other websites. The third column of Table 2 contains adjusted average percentages.

In our simulations for website visits we treat the entries in column two of Table 2 as bins of a histogram with the relative size indicated by the average website popularity. After randomly selecting a bin (weighted by popularity), in the case of an Alexa range we uniformly select a website within the range, and for the other category we uniformly select from one million other websites. This is a conservative choice given that there are hundreds of millions of active websites on the Internet. Uniformly selecting within a bin will make the more popular websites in the bin likely underrepresented while less popular websites in the bin get overrepresented. However, we typically simulate an attacker that monitors ≈100 websites and use the website popularity as the starting rank of

Table 2: Inferred average website popularity for the entire Tor network early 2018, from Mani *et al.* [49, Figure 2].

Website	Average primary domain (%)	Without torproject.org
torproject.org	40.1	
Alexa (0,10]	8.4	13.9
Alexa (10,100]	5.1	8.4
Alexa (100,1k]	6.2	10.3
Alexa (1k,10k]	4.3	7.1
Alexa (10k,100k]	7.7	12.7
Alexa (100k,1m]	7.0	11.6
other	21.7	35.9

the first monitored website. For the most popular websites, monitoring 100 websites covers the entire or significant portions of the bins (Alexa $\leq$1k), and for less popular websites (Alexa >1k), as our results later show, this does not matter.

5.2 The Number of Website Visits

Mani *et al.* also inferred with a 95% confidence interval that $(104 \pm 36) * 10^6$ *initial* streams are created during a 24 hour period in the entire Tor network [49]. Based on this, in our simulation we assume 140 million website visits per day that are distributed as described above and occur uniformly throughout the day. While assuming uniformity is naive, we selected the upper limit of the confidence interval to somewhat negate any unreasonable advantage to the attacker.

5.3 A Reasonable Timeframe

Wang and Goldberg show that it is realistic to assume that an attacker can determine the start of a webpage load even in the presence of background noise and multiple concurrent website visits [86]. An attacker can further determine if a circuit is used for onion services or not [40, 60]. Now, consider an attacker that observes traffic between a Tor client and its guard. The initial stream contains the first HTTP GET request for a typical website visit. The request will be the first outgoing packet as part of a website visit once a connection has been established. When the request arrives at the destination is the point in time when an oracle, e.g., instantiated by access logs would record this time as the time of visit. Clearly, the exact time is between the request and the response packets and the attacker observes the timing of those packets. So what is a realistic timeframe for the attacker to use when it queries a WO?

Between January 22–30 (2019) we performed Round-Trip Time (RTT) measurements using four Amazon EC2 instances that ran *their own* nginx

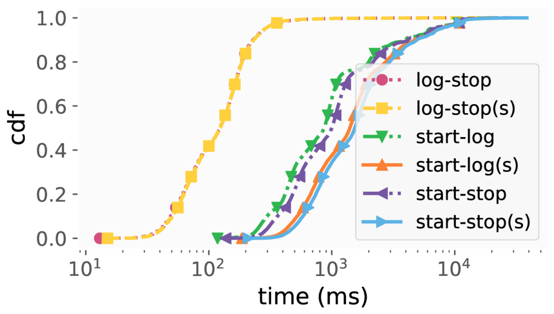


Figure 5: Time differences between start, log, and stop events when visiting a website over HTTP(S) using Tor.

HTTP(S) servers to visit *themselves* over Tor (with `torify curl`) using a fresh circuit for each visit. This allowed us easy access to start and stop times for the RTT measurement, as well as the time a request appeared in the nginx access log (without any clock-drift). In total we collected 21,497 HTTP traces and 21,492 HTTPS traces, where each trace contains start, log, and stop timestamps. Our results are shown in Figure 5. It is clear that that log-to-stop times are independent of HTTP(S). More than half of all log-to-stop times (54.5%) are within a 100 ms window (see 40–140 ms), and nearly all log-to-stop times are less than 1000 ms.

Based on our experiment results we consider three timeframes relevant: 10 ms, 100 ms, and 1000 ms. First, 10 ms is relevant as close to optimal for any attacker. On average, there are only 17 website visits during a 10 ms window in the entire Tor network. 100 ms is our default for the WF experiments we perform: we consider it realistic for many sources of WOs (e.g., Cloudflare logs and real-time bidding). We also consider a 1000 ms timeframe relevant due to the prevalence of sources of WOs with a resolution in seconds (e.g., due to Unix timestamps or TTLs for DNS). Based on our simulations and the different timeframes, Appendix A contains an analysis of the utility of WOs using Bayes' law. Appendix B presents some key lessons from the simulation, in particular that while the resolution and resulting timeframe is an important metric in our simulation, it is minor in comparison to the overall website popularity in Tor of the monitored websites.

6 Deep Fingerprinting with Website Oracles

We first describe how we augment the Deep Fingerprinting (DF) attack by Sirinam *et al.* [75] with WO access. Next we evaluate the augmented classifier on three different datasets with five different WF defenses. Source code and datasets for simulating WF+WO attacks as well as steps to reproduce all of the following results using DF are available at https://github.com/pylls/wfwo.

6.1 The Augmented Classifier

As covered in the background (Section 2.3), DF is a CNN where the last layer is a softmax. The output is an array of probabilities for each possible class. Compared to the implementation of DF used by Sirinam *et al.*, we changed DF to not first use binary classification in the open world to determine if it is an unmonitored trace or not, but rather such that there is one class for each monitored website and one for unmonitored. Conceptually, this slightly lowers the performance of DF in our analysis, but our metrics show that mistaking one monitored website for another is insignificant for the datasets used in the analysis of this paper. The principal source of false positives is mistaking an unmonitored website for a monitored.

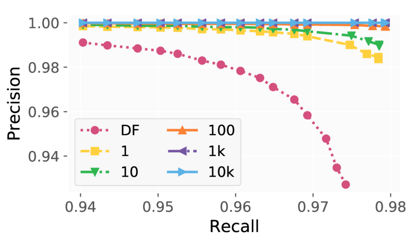
Given the probability of each possible class as output of DF, we used the second generic construction (Definition 3) from Section 3.2 to combine DF with a WO. To update the remaining probabilities after removing a (monitored) prediction with the help of the WO, we use a softmax again. However, due to how the softmax function is defined, it emphasizes differences in values above one and actually de-emphasizes values between zero and one [16]. This is problematic for us because all values we send through the softmax are probabilities that per definition are between zero and one. To account for this, we first divide each probability with the maximum probability and multiply with a constant before performing the softmax. Through trial-and-error, a constant of five gave us a reasonable threshold in probabilities. Note that this does not in any way affect the order of likely classes from DF, it simply puts the probabilities in a span that makes it easier for us to retain a threshold value between zero and one after multiple calls to the softmax function.

6.2 WTF-PAD and Walkie-Talkie

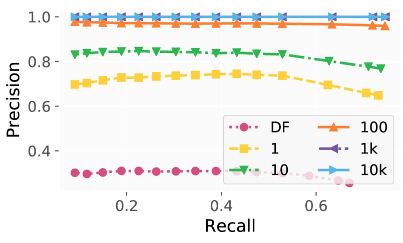
We use the original dataset of Sirinam *et al.* [75] that consists of 95 monitored websites with 1,000 instances each as well as 20,000 unmonitored websites (95x1k+20k). The dataset is split 8:1:1 for training, validation, and testing, respectively. Given the dataset and our changes to DF to not do binary classification means that our testing dataset is unbalanced in terms of instances per class. Therefore we show precision-recall curves generated by alternating the threshold for DF with and without WO access.

Figure 6 shows the results of DF and DF+WO with a simulated WO on Sirinam *et al.*'s dataset with no defense (Figure 6a), Walkie-Talkie (Figure 6b), and WTF-PAD (Figure 6c). For the WO we use a 100 ms timeframe and plot the results for different starting Alexa ranks of the 95 monitored websites. Regardless of defense or not, we observe that for Alexa ranks 1k and less popular websites the precision is perfect (1.0) regardless of threshold. This indicates that—for an attacker monitoring frontpages of websites—a 100 ms WO significantly reduces false positives for two-thirds of all website visits made over Tor, for the vast majority of potentially monitored frontpages of websites. Recall is also slightly improved.

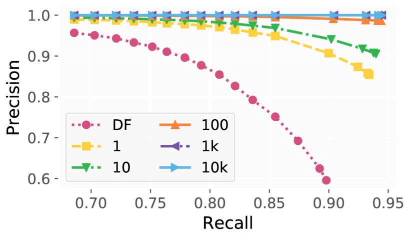
For Walkie-Talkie we observe a significant improvement in precision due to



(a) No defense.



(b) Walkie-Talkie [87].



(c) WTF-PAD [37].

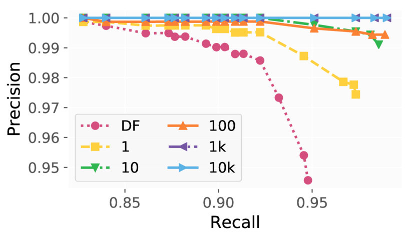
Figure 6: Attack simulation for Deep Fingerprinting (DF) with website oracles (100 ms timeframe) on Sirinam *et al.*'s dataset [75]. The lines in each sub-figure show DF with and without website oracle access for different starting Alexa ranks for monitored websites.

WO access. Wang and Goldberg note that the use of popular websites as decoy (non-sensitive) websites protects less-popular sensitive websites due to the base rate: an attacker claiming that the user visited the less-popular website is (per definition) likely wrong, given that the attacker is able to detect both potential website visits [87]. Access to a WO flips this observation on its head: if a WO detects the sensitive less-popular website, the base rate works in reverse. The probability of an unpopular website being both miss-classified and visited in the timeframe is small for all but the most popular websites. The key question becomes one of belief in the base rate of the network and that of the target user, as analysed in Appendix A.

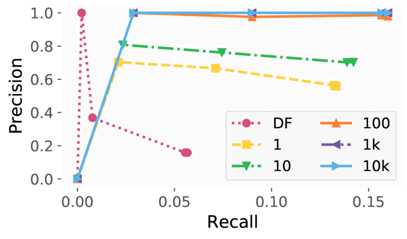
Further, WO access improves both recall and precision for all monitored websites against WTF-PAD. WTF-PAD only provides a three percentage points decrease in recall compared to no defense for monitored websites with Alexa ranks 1k and above.

6.3 CS-BuFLO and Tamaraw

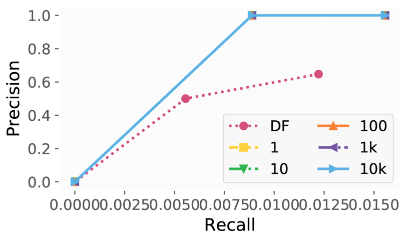
To evaluate the constant-rate defenses CS-BuFLO and Tamaraw by Cai et al. [7, 8] we use Wang *et al.*'s dataset in the open world [85]. The dataset consists of 100 monitored websites with 90 instances each and 9000 unmonitored sites (100x90+9k), that we randomly split (stratified) into 8:1:1 for training,



(a) No defense.



(b) CS-BuFLO [7], with reported 67.2% BOW and 575.6% TOH [13].



(c) Tamaraw [8], with reported 256.7% BOW and 341.4% TOH [13].

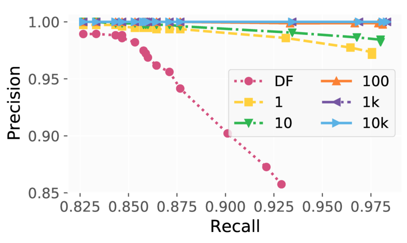
Figure 7: Attack simulation for Deep Fingerprinting (DF) [75] with website oracles (100 ms timeframe) on Wang *et al.*'s dataset [85]. The lines in each subfigure show DF with and without website oracle access for different starting Alexa ranks for monitored websites.

validation, and testing. We had to increase the length of the input to DF for this dataset, from 5000 to 25000, to ensure that we capture most of the dataset. To get defended traces for CS-BuFLO and Tamaraw we use the slightly modified implementations as part of Cherubin's framework [13].

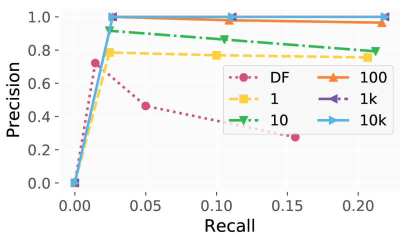
Figure 7 shows the results of our simulations. DF alone is also highly effective against the original Wang dataset—as expected—and our attack simulation shows that we can further improve it with access to website oracles. Most importantly, both CS-BuFLO and Tamaraw offer protection against DF with and without oracle access by *significantly lowering recall*. Tamaraw offers an order of magnitude better defense in terms of recall. As implemented in the framework by Cherubin, CS-BuFLO and Tamaraw reportedly has BOH 67.2% and 256.7%, and TOH 575.6% and 341.4%, respectively. This kind of overhead is likely prohibitively large for real-world deployment in Tor [7, 8, 37, 75, 87].

6.4 DynaFlow

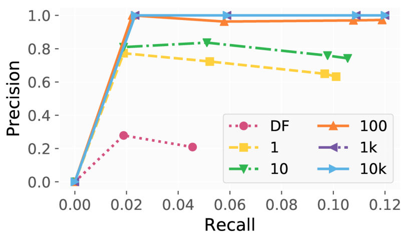
DynaFlow is a *dynamic* constant-rate defense by Lu *et al.* [47] with two configurations that result in different overheads and levels of protection. Lu *et al.* gathered their own dataset of 100 monitored websites with 90 instances each



(a) No defense.



(b) DynaFlow [47] config 1, with measured 59% BOH and 24% TOH.



(c) DynaFlow [47] config 2, with measured 109% BOH and 30% TOH.

Figure 8: Attack simulation for Deep Fingerprinting (DF) [75] with website oracles (100 ms timeframe) on Lu *et al.*'s dataset [47]. The lines in each subfigure show DF with and without website oracle access for different starting Alexa ranks for monitored websites.

and 9000 unmonitored websites (100x90+9k, same as Wang *et al.*'s [85]) to be able to combine smaller packets, as discussed briefly in Section 2.3. As for CS-BuFLO and Tamaraw, we had to increase the length of the input to DF for this dataset to 25000 to ensure that we capture most of the dataset.

Figure 8 shows the results of our simulations for no defense as well as the two configurations of DynaFlow. As for Wang *et al.*'s dataset [85], we see as expected that DF is highly effective and WO access further improves the attack. Further, both configurations of DynaFlow are effective defenses, comparable to CS-BuFLO with significantly lower overheads at first glance. However, note that the comparison is problematic due to DynaFlow combining smaller packets. The extra overhead for config 2 over 1 is not wasted: recall is significantly reduced, more than halved for regular DF and slightly less than half with a WO.

7 Discussion

For defenses that are based on the idea of creating collision sets between packet traces generated by websites, oracle access is equivalent to being able to perform set intersection between the set of websites in a collision set and monitored websites visited at the time of fingerprinting. As the results show from Section 6, some defenses can significantly reduce the recall of WF attacks with WOs, but not the precision for the majority of websites and website visits in Tor.

Next, in Section 7.1 we further cement that our simulations show that WOs significantly reduces false positives, highlighting that a WF+WO attacker surprisingly infrequently have to query a WO when classifying unmonitored traces. Section 7.2 discusses the impact of imperfect WO sources with limited observability and false positives on the joint WF+WO attack. Finally, Section 7.3 covers limitations of our work, and Section 7.4 discusses possible mitigations.

7.1 A Large Unmonitored Dataset

We look at the number of false positives for a large test dataset consisting of only unmonitored websites (representing a target user(s) with base rate 0, i.e., that never visits any monitored website). Using the dataset of Greschbach et al. [26], we trained DF on 100x100 monitored and 10k unmonitored websites (8:2 stratified split for validation), resulting in about 80% validation accuracy after 30 epochs (so DF is clearly successful also on this dataset). We then tested the trained classifier on *only* 100k *unmonitored* traces, with and without oracle access (100ms resolution) for different assumptions of the popularity of the *monitored* websites. With ten repetitions of the above experiment, we observed a false positive rate in the order of 10^{-6} for monitored websites with Alexa popularity 10k. Excluding `torproject.org`, this indicates that an attacker would have close to no false positives for about half of all website visits in Tor, according to the distribution of Mani *et al.* [49] (see Section 5.1). Without access to a WO, DF had a false positive rate in the order of 10^{-3} to 10^{-4}, depending on the threshold used by the attacker.

Recall how WOs are used as part of WF+WO attacks in Section 3.2: the WO is only used if the WF attack classifies a trace as monitored.[1] This means that, in the example above, the WO is used on average every 10^3 to 10^4 trace, to (potentially) rule out a false positive. Clearly, this means that WO sources that can only be used infrequently, e.g., due to caching as in DNS, are still valuable for an attacker.

7.2 Imperfect Website Oracle Sources

Our analysis considered an ideal source of a WO that observes all visits to targeted *monitored* websites of the attacker and that produces no false positives. Next, using the same dataset and setup as for Figure 6a with an Alexa starting rank of 10^4, we simulate the impact on recall and the False-Negative-to-Positive-rate[2] (FNP) of the *joint* WF+WO attack for five false positive rates *of the WO*

[1]For WF attacks like DF that produces a list of probabilities (Definition 3), just assume that the attacker picks the threshold and checks if the probability is above as part of the if-statement before using the WO.

[2]For a classifier with multiple monitored classes and an unmonitored class (as for our modified DF, see Section 6.1), FNP captures the case when the classifier classifies an unmonitored testing

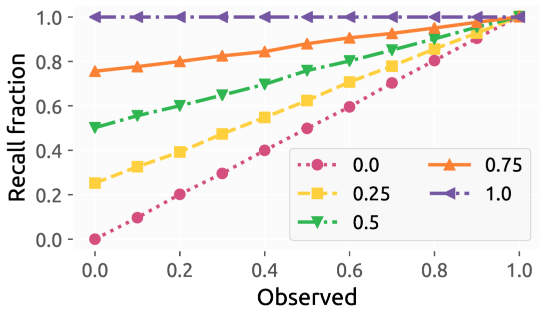


Figure 9: How limited WO observability effects the final recall of a WF+WO attack for five different WO false positive rates.

and a fraction of observed website visits.

Figure 9 shows the impact on the joint recall in the above setting. We see that recall is directly proportional to the fraction of observed visits, as per the results of Greschbach *et al.* [26]. Further, false positives for the WO have a positive impact on the fraction of recall, counteracting visits missed due to limited observability. For the same reason, a larger timeframe or monitoring more popular websites would also improve recall.

Figure 10 shows the impact on the joint FNP. Note that lower FNP is better for an attacker. We see that limited observability has no impact on FNP. This makes sense, because a WO cannot confirm anything it does not observe. The FNP fraction for the joint FNP is roughly proportional to the FP of the WO. We also see that the FNP fraction consistently is above the FP of the WO: this is because—beyond the simulated FP—there is a slight probability that someone else (in our simulation of the Tor network) visited the website for each classified trace. A larger timeframe or monitoring more popular websites would also increase FNP.

From above results, our simulations indicate that even with a deeply imperfect WO source an attacker can get significant advantage in terms of reduced false positives at a comparatively small cost of recall. For example, given a WO with 50% observability and false positives, the resulting WF+WO attack has about 75% of the recall of the WF attack and slightly more than half the false positives.

7.3 Limitations

As discussed in Section 2.3, there are a number of practical limitations in general for WF attacks. Regarding attacker goals, WOs are likely less useful for the purpose of censorship than for other goals. Many sources of WOs cannot be accessed in real-time, giving little utility for an attacker that needs to make a

trace as any monitored class. In addition to FNP, such a classifier can also confuse one monitored website for another. Both these cases are false positives [84].

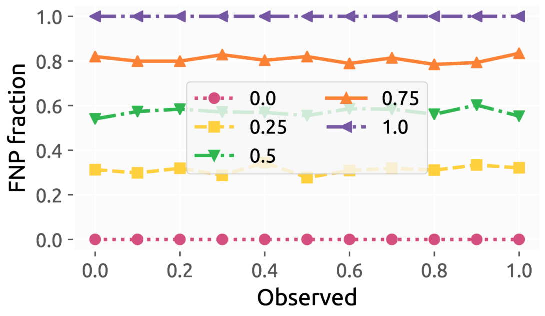


Figure 10: How limited WO observability effects the final False-Negative-to-Positive-rate (FNP) of a WF+WO attack for five different WO false positive rates. Lower is better.

near real-time censorship decision. An attacker that only wants to detect visits to a few selected monitored websites gains significant utility from WOs, as long as the detection does not have to be in real-time. It is also noteworthy that an attacker that wants to detect all possible website visits by a victim can use the WO to in essence "close the world" from all possible websites to only those visited over Tor while the victim is actively browsing. Granted, this requires a source for the WO that is slightly different from our definition, but some do offer this: e.g., an attacker that gains comprehensive control over the DNS resolvers used by Tor exits [26].

When it comes to false positives a significant limitation of our simulations is that we consider fingerprinting the frontpages of websites and not specific webpages. Several sources or WOs are not able to detect webpage visits. This is also true for subsequent webpage visits on the same website after first visiting the frontpage of a website (e.g., DNS and OCSP will be cached). An attacker with the goal of detecting each such page visit will thus suffer more false positives or fail at detecting them for some sources of WOs.

7.4 Mitigations

The best defense against WOs is WF defenses that significantly reduce the recall of WF attacks. In particular, if an attacker can significantly reduce the website anonymity set *after* accounting for information from the WO, then attacks are likelier to succeed. This implies that most websites need to (at least have the potential to) result in the same network traces, as we see with DynaFlow, Tamaraw, and CS-BuFLO.

For onion websites we note that the DHT source of a WO from Section 4 is inherent to the design of onion services in Tor. Defenses that try to make it harder to distinguish between regular website visits and visits to onion websites should also consider this WO source as part of their analysis, in particular for v2 onion services.

Finally, some sources of WOs could be minimized. If you run a potentially sensitive website: do not use RTB ads, staple OCSP, have as few DNS entries as possible[3] with a high TTL, do not use CDNs, do not retain any access logs, and consider if your website, web server, or operating system have any information leaks that can be used as an oracle. If you run a Tor exit, consider not using Google or Cloudflare for your DNS but instead use your ISP's resolver if possible [26].

8 Related Work

The combination of a WF attack with a WO is a type of Classify-Verify method as proposed by Stolerman et al. [76], which in turn is a type of rejection function as described by Chow [14]. Such a method was first used in the context of WF by Juarez *et al.* [36] and later by Greschbach *et al.* [26] to augment WF attacks with inferences from observed DNS traffic. Note that the attack by Greschbach et al. can be seen as a probabilistic WO due to the attacker under their threat model only observing a fraction of DNS traffic from the Tor network. Our work builds upon and generalises their work where DNS traffic is just one of many possible sources to infer website visits from. Further, our DNS-based sources are usable by anyone instead of relatively strong network attackers (or Google or Cloudflare).

All anonymity networks produce anonymity sets (per definition) that change with observations by an attacker over time [68]. Modelling the behaviour of an anonymity system (as a mix), what the attacker observes, and how the anonymity sets change over time allows us to reason about how the attacker can perform traffic analysis and break the anonymity provided by the system [21, 38, 73]. Attacks along these lines are many with more-or-less consistent terminology, including intersection attacks, (statistical) disclosure attacks, and traffic confirmation attacks [4, 17, 18, 19, 39, 68, 69, 81].

WOs are nothing more than applying the notion of anonymity sets to the potential destination websites visited over an anonymity network like Tor and giving an attacker the ability to query this anonymity set for membership for a limited number of monitored websites. The way we use WOs in our generic attacks is *not to learn long-term statistically unlikely relationships* between senders and recipients in a network. Rather, the WO is only used to learn *part of the anonymity set at the time of the attack*. That an attacker can observe anonymity sets is not novel, what is novel in our work is how we apply it to the WF domain and argue for its inclusion as a core attacker capability when modelling WF attacks and defenses.

Murdoch and Danezis showed how to use observed latency in Tor as an oracle to perform traffic analysis attacks [54]. Chakravarty *et al.* detailed similar attacks but based on bandwidth estimation [11] and Mittal *et al.* using throughput estimation [53]. Attackers in these cases do not need to be directly

[3]As noted by Greschbach *et al.* [26], websites may have several unique domain names. Each of those could be used independently to query several sources (e.g., DNS) of WOs.

in control of significant fractions Tor, but rather use network measurements to infer the state of the network and create an oracle that an attacker can utilize, similar to WOs.

Correlation of input and output flows is at the core of many attacks on anonymity networks like Tor [6, 35, 78]. Flow correlation attacks correlate traffic on the network layer, considering packet sizes and timing of sent traffic. The RAPTOR attack by Sun et al. [78] needs about 100MB of data sent over five minutes to correlate flows with high accuracy. The recent state-of-the-art attack DeepCorr by Nasr *et al.* [56]—based on deep learning like Deep Fingerprinting by Sirinam *et al.* [75]—needs only about 900KB of data (900 packets) for comparable accuracy to RAPTOR. While flow correlation attacks like RAPTOR and DeepCorr operate on the network layer, WF+WO attacks can be viewed as *application layer* correlation attacks. WF attacks extract the application-layer data (the website) while WOs reconstruct parts of the anonymity set of possible monitored websites visited. WF attacks need to observe most of the traffic generated when visiting a website that goes into the anonymity network. While a WO does not have to directly view any of the output flows of the network, it needs to be able to infer if a particular website was visited during a period of time, as shown in Section 4.

9 Conclusions

WF+WO attacks use the base rate of all users of the network against victims, significantly reducing false positives in the case of all but the most popular websites visited over Tor. This is troubling in many ways, in part because presumably many sensitive website visits are to unpopular websites used only by local communities in regions of the world where the potential consequences of identification are the worst.

The threat model of Tor explicitly states that Tor does not consider attackers that can observe both incoming and outgoing traffic [22]. Clearly, a WO gives the capability to infer what the outgoing traffic of the network encodes on the application layer (the website visits). This is in a sense a violation of Tor's threat model when combined with a WF attacker that also observes incoming traffic. However, we argue that because of the plethora of possible ways for an attacker to infer membership in the anonymity sets of Tor, WOs should be considered within scope simply because Tor asserts that it is an anonymity network.

While the real-world impact of WF attacks on Tor users remains an open question, our simulations show that false positives can be signficantly reduced by many attackers with little extra effort for some WO sources. Depending on WO source, this comes at a trade-off of less recall. For many attackers and attacker goals, however, this is a worthwhile trade. To us, the threat of WF attacks appears more real than ever, especially when also considering recent advances by deep learning based attacks like DF [75] and DeepCorr [56].

Acknowledgements

We would like to thank Jari Appelgren, Roger Dingledine, Nicholas Hopper, Marc Juarez, George Kadianakis, Linus Nordberg, Mike Perry, Erik Wästlund, and the PETS reviewers for their valuable feedback. Simulations were performed using the Swedish National Infrastructure for Computing (SNIC) at High Performance Computing Center North (HPC2N) This research was funded by the Swedish Internet Foundation and the Knowledge Foundation of Sweden.

References

[1] Chaabane Abdelberi, Terence Chen, Mathieu Cunche, Emiliano De Cristofaro, Arik Friedman, and Mohamed Ali Kâafar. Censorship in the wild: Analyzing internet filtering in Syria. In *IMC*, 2014.

[2] Kota Abe and Shigeki Goto. Fingerprinting attack on Tor anonymity using deep learning. *Proceedings of the Asia-Pacific Advanced Network*, 42, 2016.

[3] Amazon. The top 500 sites on the web. https://www.alexa.com/topsites, accessed 2019-02-13.

[4] Oliver Berthold, Andreas Pfitzmann, and Ronny Standtke. The disadvantages of free MIX routes and how to overcome them. In *International Workshop on Design Issues in Anonymity and Unobservability*, 2000.

[5] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In *CRYPTO*, 1998.

[6] Nikita Borisov, George Danezis, Prateek Mittal, and Parisa Tabriz. Denial of service or denial of security? In *CCS*, 2007.

[7] Xiang Cai, Rishab Nithyanand, and Rob Johnson. CS-BuFLO: A congestion sensitive website fingerprinting defense. In *WPES*, 2014.

[8] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. A systematic approach to developing and evaluating website fingerprinting defenses. In *CCS*, 2014.

[9] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. Touching from a distance: website fingerprinting attacks and defenses. In *CCS*, 2012.

[10] Yue Cao, Zhiyun Qian, Zhongjie Wang, Tuan Dao, Srikanth V. Krishnamurthy, and Lisa M. Marvel. Off-path TCP exploits of the challenge ACK global rate limit. *IEEE/ACM Trans. Netw.*, 26(2), 2018.

[11] Sambuddho Chakravarty, Angelos Stavrou, and Angelos D. Keromytis. Traffic analysis against low-latency anonymity networks using available bandwidth estimation. In *ESORICS*, 2010.

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

[13] Giovanni Cherubin. Bayes, not naïve: Security bounds on website fingerprinting defenses. *PETS*, 2017(4).

[14] C Chow. On optimum recognition error and reject tradeoff. *IEEE Trans. Inf. Theory*, 16(1), 1970.

[15] Taejoong Chung, Jay Lok, Balakrishnan Chandrasekaran, David R. Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, John P. Rula, Nick Sullivan, and Christo Wilson. Is the web ready for OCSP must-staple? In *IMC*, 2018.

[16] Wikipedia contributors. Softmax function—Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Softmax_function&oldid=883834589, accessed 2019-02-17.

[17] George Danezis. Statistical disclosure attacks. In *IFIP SEC*, 2003.

[18] George Danezis. The traffic analysis of continuous-time mixes. In *PETS*, 2004.

[19] George Danezis and Andrei Serjantov. Statistical disclosure or intersection attacks on anonymity systems. In *IH*, 2004.

[20] 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.

[21] Claudia Díaz, Stefaan Seys, Joris Claessens, and Bart Preneel. Towards measuring anonymity. In *PETS*, 2002.

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

[23] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. Peek-a-boo, I still see you: Why efficient traffic analysis countermeasures fail. In *IEEE S&P*, 2012.

[24] Roya Ensafi, Jong Chun Park, Deepak Kapur, and Jedidiah R. Crandall. Idle port scanning and non-interference analysis of network protocol stacks using model checking. In *USENIX Security*, 2010.

[25] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *JCSS*, 28(2), 1984.

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

[27] The Guardian. NSA stores metadata of millions of web users for up to a year, secret files show. https://www.theguardian.com/world/2013/sep/30/nsa-americans-metadata-year-documents, accessed 2019-02-27.

[28] Jamie Hayes and George Danezis. k-fingerprinting: A robust scalable website fingerprinting technique. In *USENIX Security*, 2016.

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

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

[31] Cloudflare Inc. FAQs. https://web.archive.org/web/20190227165850/https://developers.cloudflare.com/logs/faq/.

[32] Cloudflare Inc. Helping build a better internet. https://web.archive.org/web/20190227165133/https://www.cloudflare.com/.

[33] Cloudflare Inc. Purge cache. https://web.archive.org/web/20190228150344/https://1.1.1.1/purge-cache/.

[34] Rob Jansen, Marc Juárez, Rafa Galvez, Tariq Elahi, and Claudia Díaz. Inside job: Applying traffic analysis to measure Tor from within. In *NDSS*, 2018.

[35] 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.

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

[37] Marc Juárez, Mohsen Imani, Mike Perry, Claudia Díaz, and Matthew Wright. Toward an efficient website fingerprinting defense. In *ESORICS*, 2016.

[38] Dogan Kesdogan, Dakshi Agrawal, and Stefan Penz. Limits of anonymity in open environments. In *IH*, 2002.

[39] Dogan Kesdogan and Lexi Pimenidis. The hitting set attack on anonymity protocols. In *IH*, 2004.

[40] Albert Kwon, Mashael AlSabah, David Lazar, Marc Dacier, and Srinivas Devadas. Circuit fingerprinting attacks: Passive deanonymization of Tor hidden services. In *USENIX Security*, 2015.

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

[42] Google LLC. About targeting for display network campaigns. https://web.archive.org/web/20190228122431/https://support.google.com/google-ads/answer/2404191?hl=en&ref_topic=3121944%5C.

[43] Google LLC. Flush cache. https://web.archive.org/web/20190228150306/https://developers.google.com/speed/public-dns/cache.

[44] Google LLC. How Google retains data we collect. https://web.archive.org/web/20190227170903/https://policies.google.com/technologies/retention.

[45] Google LLC. Real-time bidding protocol buffer v.161. https://web.archive.org/web/20190228122615/https://developers.google.com/authorized-buyers/rtb/downloads/realtime-bidding-proto.

[46] Google LLC. Set your desktop and mobile web inventory to anonymous, branded, or semi-transparent in AdX. https://web.archive.org/web/20190228123602/https://support.google.com/admanager/answer/2913411?hl=en&ref_topic=2912022.

[47] David Lu, Sanjit Bhat, Albert Kwon, and Srinivas Devadas. Dynaflow: An efficient website fingerprinting defense based on dynamically-adjusting flows. In *WPES*, 2018.

[48] David Lyon. Surveillance, Snowden, and big data: Capacities, consequences, critique. *Big Data & Society*, 1(2), 2014.

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

[50] Nate Mathews, Payap Sirinam, and Matthew Wright. Understanding feature discovery in website fingerprinting attacks. In *WNYISPW*, 2018.

[51] Nick Mathewson. New release: Tor 0.4.0.1-alpha. https://blog.torproject.org/new-release-tor-0401-alpha, accessed 2019-02-08.

[52] Robert Merget, Juraj Somorovsky, Nimrod Aviram, Craig Young, Janis Fliegenschmidt, Jörg Schwenk, and Yuval Shavitt. Scalable scanning and automatic classification of TLS padding oracle vulnerabilities. In *USENIX Security*, 2019. to appear.

[53] Prateek Mittal, Ahmed Khurshid, Joshua Juen, Matthew Caesar, and Nikita Borisov. Stealthy traffic analysis of low-latency anonymous communication using throughput fingerprinting. In *CCS*, 2011.

[54] Steven J. Murdoch and George Danezis. Low-cost traffic analysis of Tor. In *IEEE S&P*, 2005.

[55] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proc. Annu. ACM Symp. Theory Comput.*, 1990.

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

[57] Netcraft. January 2019 web server survey. https://web.archive.org/web/20190208081915/https://news.netcraft.com/archives/category/web-server-survey/.

[58] Rishab Nithyanand, Xiang Cai, and Rob Johnson. Glove: A bespoke website fingerprinting defense. In *WPES*, 2014.

[59] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. Website fingerprinting at internet scale. In *NDSS*, 2016.

[60] Andriy Panchenko, Asya Mitseva, Martin Henze, Fabian Lanze, Klaus Wehrle, and Thomas Engel. Analysis of fingerprinting techniques for Tor hidden services. In *WPES*, 2017.

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

[62] Mike Perry. A critique of website traffic fingerprinting attacks. https://blog.torproject.org/critique-website-traffic-fingerprinting-attacks, accessed 2019-02-08.

[63] Andreas Pfitzmann and Marit Hansen. A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management, 2010.

[64] Tor Project. Tor research safety board. https://research.torproject.org/safetyboard.html, accessed 2019-02-13.

[65] Tobias Pulls and Rasmus Dahlberg. OOM manger wipes entire DNS cache. https://trac.torproject.org/projects/tor/ticket/29617, 2020.

[66] Zhiyun Qian, Zhuoqing Morley Mao, and Yinglian Xie. Collaborative TCP sequence number inference attack: how to crack sequence number under a second. In *CCS*, 2012.

[67] Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *CRYPTO*, 1991.

[68] Jean-François Raymond. Traffic analysis: Protocols, attacks, design issues, and open problems. In *International Workshop on Design Issues in Anonymity and Unobservability*, 2000.

[69] Michael G. Reed, Paul F. Syverson, and David M. Goldschlag. Anonymous connections and onion routing. *JSAC*, 16(4), 1998.

[70] Vera Rimmer, Davy Preuveneers, Marc Juárez, Tom van Goethem, and Wouter Joosen. Automated website fingerprinting through deep learning. In *NDSS*, 2018.

[71] Eyal Ronen, Robert Gillham, Daniel Genkin, Adi Shamir, David Wong, and Yuval Yarom. The 9 lives of Bleichenbacher's CAT: new cache attacks on TLS implementations. *IACR Cryptology ePrint Archive*, 2018.

[72] Quirin Scheitle, Oliver Hohlfeld, Julien Gamba, Jonas Jelten, Torsten Zimmermann, Stephen D. Strowes, and Narseo Vallina-Rodriguez. A long way to the top: Significance, structure, and stability of internet top lists. In *IMC*, 2018.

[73] Andrei Serjantov and George Danezis. Towards an information theoretic metric for anonymity. In *PETS*, 2002.

[74] Vitaly Shmatikov and Ming-Hsiu Wang. Timing analysis in low-latency mix networks: Attacks and defenses. In *ESORICS*, 2006.

[75] Payap Sirinam, Mohsen Imani, Marc Juárez, and Matthew Wright. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *CCS*, 2018.

[76] Ariel Stolerman, Rebekah Overdorf, Sadia Afroz, and Rachel Greenstadt. Classify, but verify: Breaking the closed-world assumption in stylometric authorship attribution. In *IFIP Working Group*, volume 11, 2013.

[77] 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.

[78] Yixin Sun, Anne Edmundson, Laurent Vanbever, Oscar Li, Jennifer Rexford, Mung Chiang, and Prateek Mittal. RAPTOR: routing attacks on privacy in Tor. In *USENIX Security*, 2015.

[79] Tor Project. Tor rendezvous specification - version 2. https://gitweb.torproject.org/torspec.git/tree/rend-spec-v2.txt, accessed 2019-02-13.

[80] Tor Project. Tor rendezvous specification - version 3. https://gitweb.torproject.org/torspec.git/tree/rend-spec-v3.txt, accessed 2019-02-13.

[81] Carmela Troncoso, Benedikt Gierlichs, Bart Preneel, and Ingrid Verbauwhede. Perfect matching disclosure attacks. In *PETS*, 2008.

[82] Paul Vines, Franziska Roesner, and Tadayoshi Kohno. Exploring ADINT: using ad targeting for surveillance on a budget - or - how alice can buy ads to track bob. In *WPES*, 2017.

[83] Jun Wang, Weinan Zhang, and Shuai Yuan. Display advertising with real-time bidding (RTB) and behavioural targeting. *Foundations and Trends in Information Retrieval*, 2017.

[84] Tao Wang. *Website Fingerprinting: Attacks and Defenses*. PhD thesis, University of Waterloo, 2015.

[85] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. Effective attacks and provable defenses for website fingerprinting. In *USENIX Security*, 2014.

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

[87] Tao Wang and Ian Goldberg. Walkie-talkie: An efficient defense against passive website fingerprinting attacks. In *USENIX Security*, 2017.

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

A Bayes' Law for Estimating Utility of Website Oracles

To reason about the advantage to an attacker of having access to a WO, we estimate the conditional probability of a target user visiting a monitored website. For conditional probability we know that:

$$P(C_0 \cap C_1) = P(C_0|C_1)P(C_1) \tag{1}$$

For a hypothesis H given conditional evidence E, Bayes' theorem states that:

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)} \tag{2}$$

Assume that E = $E_0 \cap E_1$, then:

$$P(H|E_0 \cap E_1) = \frac{P(E_0 \cap E_1|H)P(H)}{P(E_0 \cap E_1)} \tag{3}$$

Substituting $P(E_0 \cap E_1)$ with (1) we get:

$$P(H|E_0 \cap E_1) = \frac{P(E_0 \cap E_1|H)P(H)}{P(E_0|E_1)P(E_1)} \tag{4}$$

For a timeframe t, we define

H the probability that target user(s) visited website w over Tor in t

E_0 the probability that target user(s) visited a website over Tor in t

E_1 the probability that someone visited website w over Tor in t

We see that $P(E_0 \cap E_1|H) = 1$ by definition and get:

$$P(H|E_0 \cap E_1) = \frac{P(H)}{P(E_0|E_1)P(E_1)} \tag{5}$$

Consider $P(E_0|E_1)$: while the conditional E_1 may have some minor affect on user behaviour (in particular for overall popular websites), we assume that the popularity of using Tor to visit a particular website (by any of the users of Tor) has negligible impact on E_0 and treat E_0 and E_1 as independent:

$$P(H|E_0 \cap E_1) = \frac{P(H)}{P(E_0)P(E_1)} \tag{6}$$

We can further refine $P(H)$ as being composed of at least:

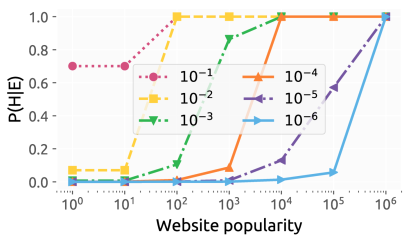
$$P(H) = P(E_0) \cap P(B_w) = P(E_0|B_w)P(B_w) \tag{7}$$

Where $P(B_w)$ is the base rate (prior) of the user(s) visiting website w out of all possible websites they visit ($P(E_0)$). We again assume (perhaps naively) that E_0 is also independent of B_w, which gives us:

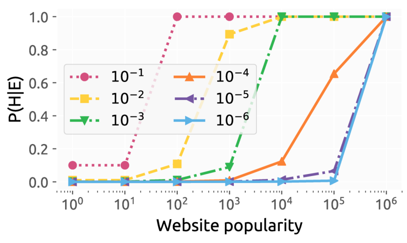
$$P(H|E_0 \cap E_1) = \frac{P(E_0)P(B_w)}{P(E_0)P(E_1)} = \frac{P(B_w)}{P(E_1)} \tag{8}$$

In other words, if an attacker learns that target user(s) visited a website (E_0) over Tor and that website w was also visited over Tor by some user (E_1), then we can estimate the probability that it was target user(s) that visited website w (H) as the ratio between the base rate (prior) for visiting w of target user(s) (B_w) and the probability that someone visited the website over Tor (E_1), all within a timeframe t.

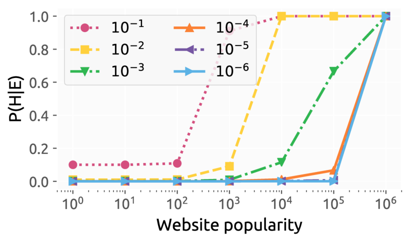
Figure 11 shows the results for simulating the probability $P(H|E_0 \cap E_1)$ for different website popularities of w, base rates, and timeframes. We see that with a realistic timeframe of 100 ms, for all base-rates but 10^{-6} there is non-zero conditional probability (and therefore utility of WO access) for Alexa top 100k or less popular websites, which covers about half of all website visits over Tor (excluding `torproject.org`).



(a) 10 ms



(b) 100 ms



(c) 1000 ms

Figure 11: The conditional probability as a function of user base rate and website popularity (Alexa) for three different timeframes.

B Lessons from Simulation

With the ability to simulate access to WOs we can now simulate the entire website anonymity set for Tor. To get a better understanding of why WOs are so useful for an attacker performing WF attacks, we look at two results from the simulation below.

B.1 Time Until Website Visited over Tor

Figure 12 shows the time until there is a 50% probability that a website has been visited over Tor depending on website popularity (Alexa, as discussed in Section 5.1). Within ten seconds, we expect that most of Alexa top 1k has been visited. Recall that this represents about one third of all website visits over Tor. The less popular websites on Alexa top one-million represent another third of all visits, quickly approaching hundreds of seconds between visits. For the remaining third of all website visits we expect them to be even less frequent.

B.2 Visits Until First False Positive

Assume that target user(s) have a base rate of 0, i.e., they never visit the attacker's monitored websites. With WO access, we can determine how many (naively assumed independent) website visits it *at least* takes until there is a 50% chance that the attacker's classifier gets a false positive. This is because

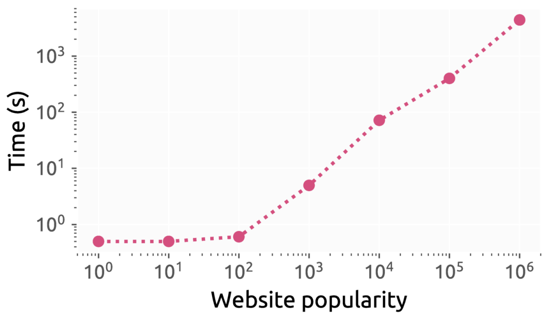


Figure 12: The simulated time until there is a 50% probability that a website for different Alexa ranks has been visited over Tor.

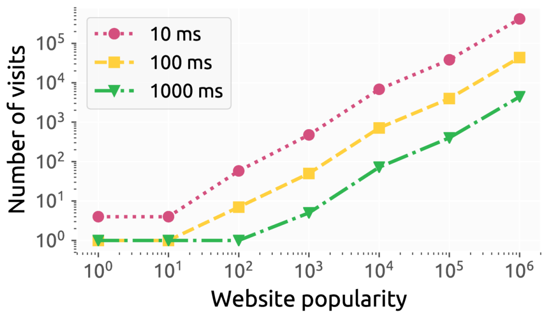


Figure 13: The number of website visits until there is a 50% probability that a website oracle would contribute to a false positive.

if the attacker's website classifier without oracle access always returns a false positive, then the false positive rate by the WF+WO attack will be determined by when the WO says that the—incorrectly classified as monitored—website has been visited. Figure 13 shows the expected number of visits *by the victim(s)* for different timeframes based on the popularity of the monitored websites. Note that the attacker per definition chooses which websites are monitored and can therefore take the probability of false positives into account.

C Sources of Website Oracles

There are a wide number of possible sources to instantiate WOs. Here we present some details on a selection of sources, far from exhaustive.

C.1 Surveillance Programmes

Intelligence agencies operate surveillance programmes that perform bulk collection and retention of communications metadata, including web-browsing [48].

For example, the Snowden revelations included *Marina*:

> Of the more distinguishing features, Marina has the ability to look back on the last 365 days' worth of DNI (Digital Network Intelligence) metadata seen by the Sigint collection system, *regardless* whether or not it was tasked for collection [27].

Another example is the prevalence of nation states to monitor Internet traffic that crosses geographic borders. For example, China operates the Great Firewall of China that is also used for censorship purposes. Due to the nature of Tor and how exits are selected, visits to websites that are not operated by world-wide reaching hosting providers are highly likely to cross multiple nation borders as traffic goes from an exit to the website. It is also worth to highlight that any sensitive website hosted from within a country where a state actor is interested in identifying visitors are likely to capture traffic to that website due to the Tor traffic crossing its borders more often than not.

C.2 Content Delivery Networks

Content Delivery Networks (CDNs), such as Akamai, Google, and Amazon host different types of content for a significant fraction of all websites on the Internet [72]. Inherently, all requests for these resources are easily identified as coming from Tor exits, and depending on content, things like unique identifiers and HTTP referrer headers enable the CDN provider to infer the website the content is hosted on.

C.3 Internet Giants

Internet giants like Google, Apple, Facebook, Amazon, Microsoft, and Cloudflare make up a large fraction the web as we know it. For example the use of Google Analytics is wide-spread, so is hosting in clouds provided by several of these giants, and Cloudflare with its "cloud network platform" hosts over 13 million domains [32]. While some of them may do what is in their power to protect the valuable data they process and retain, they are still subject to many legal frameworks across the world that might not offer the best of protections for, say, access logs pertaining to "anonymous" users of Tor when requested by authorities of nation states. As another example, Cloudflare offers a nice API for their customers to get their access logs with Unix nanosecond precision. The logs are retained for up to seven days [31], giving ample time for legal requests.

C.4 Access Logs of Web Servers

The vast majority of web servers retain access logs by default. Typically, they provide unix timestamps with seconds as the resolution (the case for Apache and nginx). Further, the access logs may be shipped to centralised security information and event management (SIEM) systems for analysis, with varying retention times and rigour in storage. For example, it is common to “anonymize” logs by removing parts of the IP-addresses and then retaining them indefinitely, as is the case for Google who removes part of IP addresses in logs after nine months [44].

C.5 Middleboxes

Network middleboxes that observe, analyse, and potentially retain network traffic abound. Especially in more oppressive countries, middleboxes are often used for censorship or dragnet surveillance, e.g., as seen with Blue Coat in Syria [1].

C.6 OCSP Responders

Chung *et al.* [15] found in a recent study that 95.4% of all certificates support the Online Certificate Status Protocol (OCSP), which allows a client to query the responsible CA in real-time for a certificate’s revocation status via HTTP. As such, the browsed website will be exposed to the CA in question. From a privacy-standpoint this could be solved if the server *stapled* a recently fetched OCSP response with the served certificate. Unfortunately, only 35% of Alexa’s top-one-million uses OCSP stapling [15].

Unless an OCSP response is stapled while visiting a website in a default configuration of the Tor browser, the status of a certificate is checked in real-time using OCSP. As such, any CA that issued a certificate for a website without OCSP stapling could instantiate a WO with an RTT-based resolution. Similarly, any actor that observes most OCSP traffic (which is in plaintext due to HTTP) gets the same capability. To better understand who could instantiate a WO based on OCSP we performed preliminary traceroute measurements[4] on the RIPE Atlas network towards four OCSP responders that are hosted by particularly large CAs: Let’s Encrypt, Sectigo, DigiCert, and GoDaddy. Let’s Encrypt and Sectigo are fronted by a variety of actors (mainly due to CDN caching), while DigiCert is fronted by a single CDN. Requests towards GoDaddy’s OCSP responder always end-up in an AS hosted by GoDaddy.

C.7 Tor Exit Relays

Anyone can run a Tor exit relay and have it be used by all Tor users. Obviously, the operator of the exit relay can observe when its relay is used and the destination websites. At the time of writing, the consumed exit bandwidth of the entire Tor network is around 50 Gbit/s. This makes the necessary investment for an attacker that wishes to get a decent chunk of exit bandwidth more a question of stealthily deploying new exit relays than prohibitively large monetary costs.

[4]Every RIPE Atlas probe used its configured DNS resolver(s). In total we requested 2048 WW-probes for one-off measurements.

C.8 Information Leaks

More sophisticated attackers can look for information leaks at the application, network, and operating system levels that allow them to infer that websites have been visited. Application level information leaks are particularly of concern for onion services: any observable state that can be tied to a new visitor is a WO for an onion visit (this is not the case for "regular" websites). Such state can include online status or the number of online users of a service, any observable activity with timestamps, a predictable caching structure, and so on. Similar information leaks can also occur on the network and operating system level [10, 24, 66].

Paper VI

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

Reprinted from

USENIX Security (2023)

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

Rasmus Dahlberg and Tobias Pulls

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, 34, 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 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

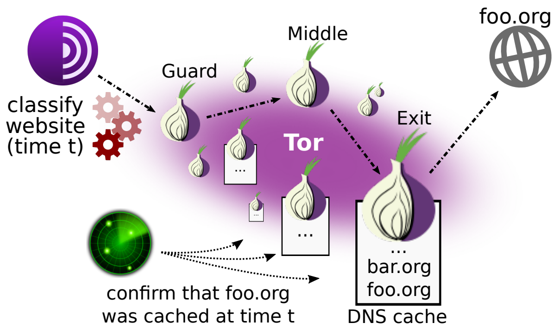


Figure 1: WF with an attacker that rules out false positives by checking that the expected DNS records were cached at the right time by conducting timeless timing attacks against exits.

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, 34, 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 ≤ 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 (≈ 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 by volunteers on the premises of DFRI[1] from May 2 until September 3, 2022. One exit was configured in its exit policy with *web* ports.[2] The other relay was configured with *permissive* ports to also allow non-web traffic.[3] 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.

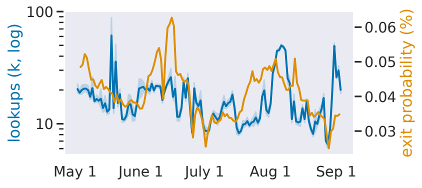
num_cache_entries Number of entries in Tor's DNS cache.

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.

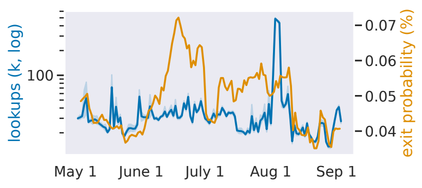
[1]More information about DFRI can be found at their website: `https://www.dfri.se`.

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

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

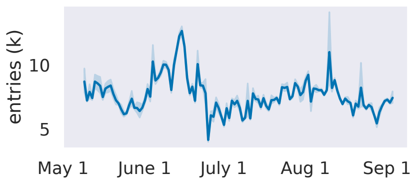


(a) web

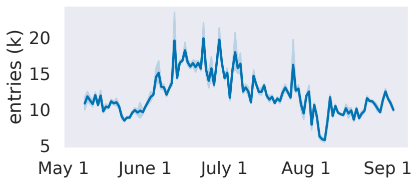


(b) permissive

Figure 2: Lookups every 15 minutes and exit probability.



(a) web



(b) permissive

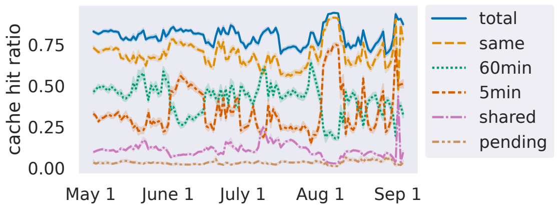
Figure 3: Cache entries every 15 minutes.

3.3 Metrics

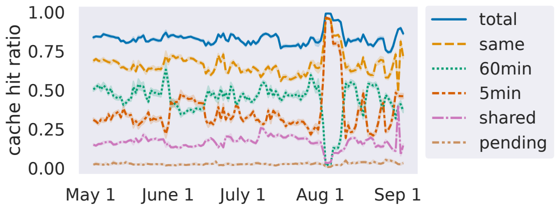
Regarding lookups per 15 minutes, the web exit processed a mean of 17,530 and median of 13,393 lookups (Figure 2a), and the permissive exit processed a mean of 41,100 and median of 26,940 lookups (Figure 2b). The permissive exit policy results in significantly more lookups. Around August 1, our exits 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).

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 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



(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.

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 only requires 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.

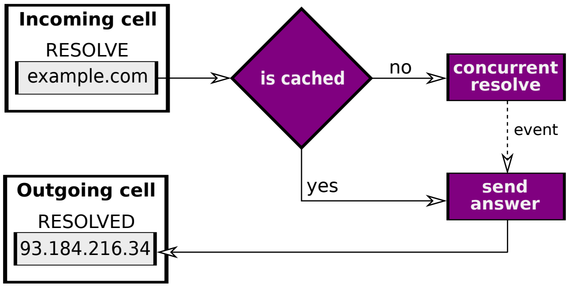


Figure 5: Processing of an incoming RESOLVE cell at an exit relay. Answers of concurrent resolves are triggered by events.

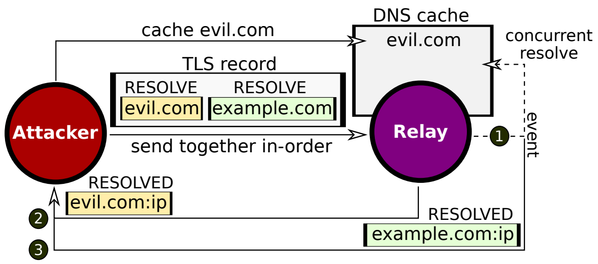
4.1 Detailed Description

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].

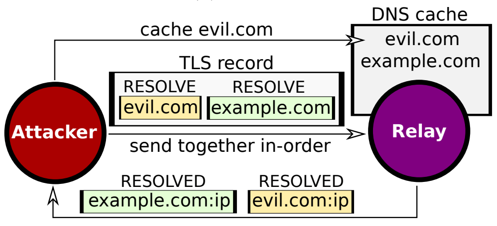
4.1.1 Attack Outline

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 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



(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.

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

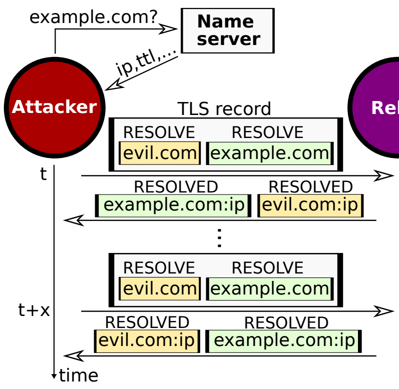


Figure 7: Repeated timeless timing attack to infer the exact time that a domain was cached by someone at an exit relay. For example, if the expected clip is 300s (ttl $\leq$ 300s), the attack is repeated every second, and the observed x is 40s, then caching of `example.com` happened at time $\approx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

(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.

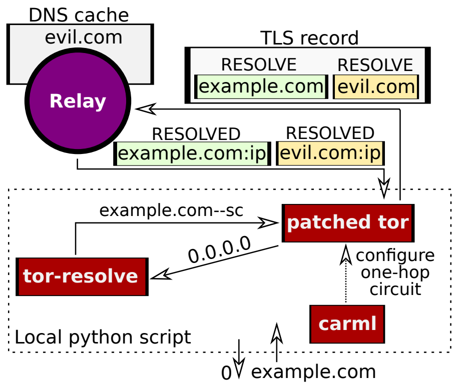


Figure 8: Local attack setup consisting of `carml` to build one-hop circuits, `tor-resolve` to inject queries, and a patched tor process that transforms them into timeless timing attacks.

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 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 specify-

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

ing 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.

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 $\mathsf{clipped\_ttl}$ 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 $[m, \mathsf{clipped\_ttl}]$, 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 $[t + x - \mathsf{clipped\_ttl}, t - m]$. Note that if $m = \mathsf{clipped\_ttl}$, 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 $x \leftarrow \mathsf{max}(x_1, \ldots, x_k)$. 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, 33]. 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 con-

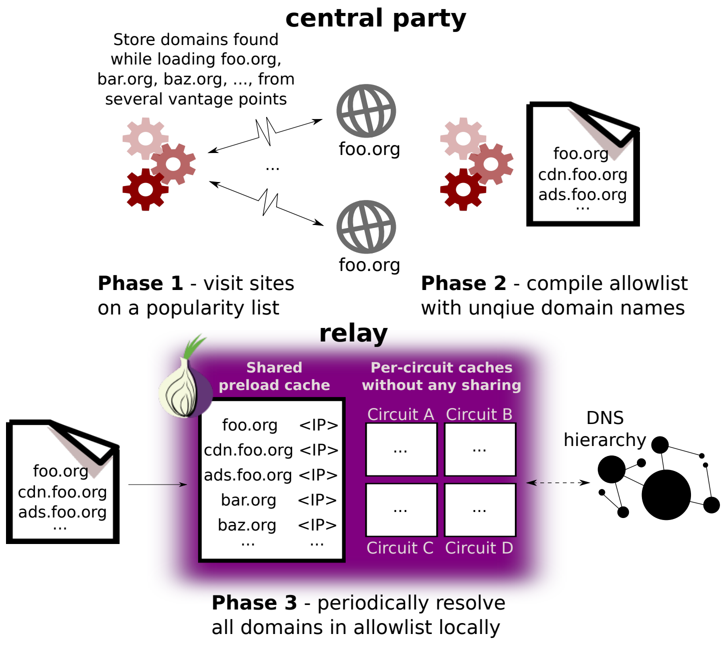

central party
Store domains found while loading foo.org, bar.org, baz.org, ..., from several vantage points
foo.org
...
foo.org
Phase 1 - visit sites on a popularity list
foo.org
cdn.foo.org
ads.foo.org
...
Phase 2 - compile allowlist with unqiue domain names
relay
Shared preload cache
foo.org <IP>
cdn.foo.org <IP>
ads.foo.org <IP>
bar.org <IP>
baz.org <IP>
Per-circuit caches without any sharing
Circuit A
Circuit B
Circuit C
Circuit D
DNS hierarchy
Phase 3 - periodically resolve all domains in allowlist locally

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.

current connections by referring to the `network.http.max-persistent-connections-per-proxy` option and/or enabling debug logging,[4] 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.

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

[4]Enable debug logging in Tor Browser: https://gitlab.torproject.org/tpo/applications/tor-browser/-/wikis/Hacking#debugging-the-tor-browser

(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 website 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`). H-owever, 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 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).

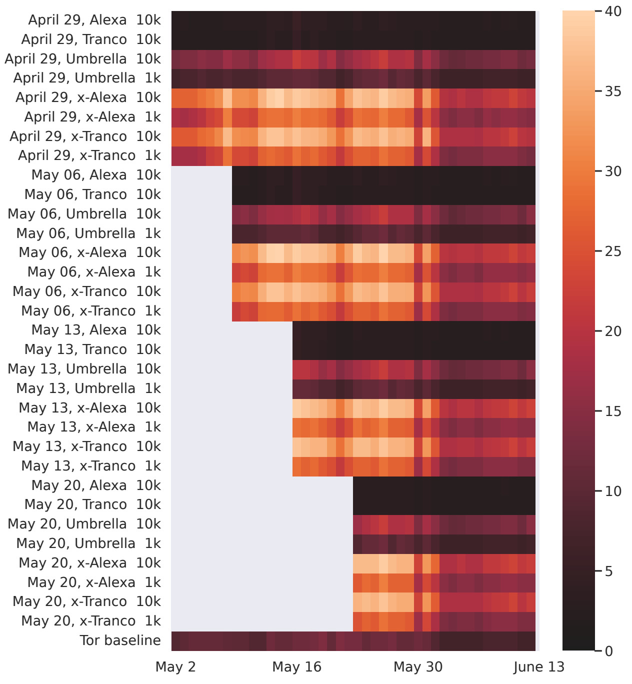
6.5.1 Results: Preload Lists

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].

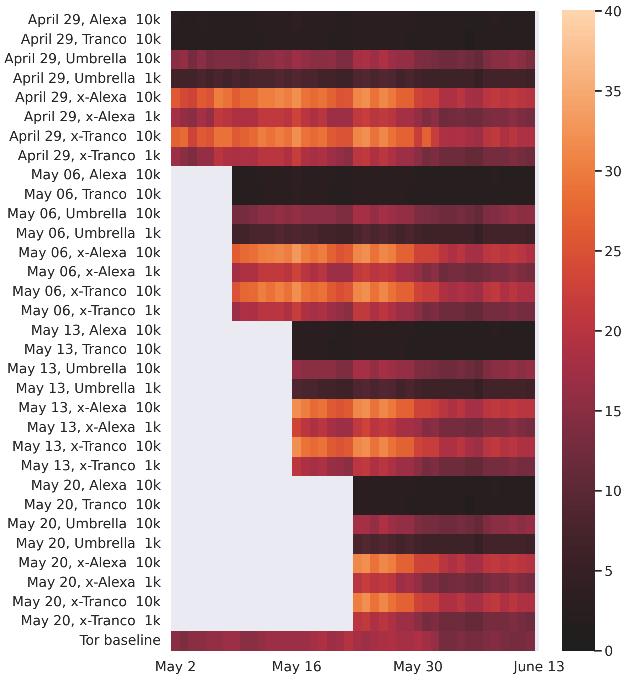
Figure 11 shows the observed *cross-circuit* cache-hit ratios for eight different extended Tranco lists, Umbrella top-10k, and Tor baseline. We used lists from the end of April because they have the most data. As a baseline, Tor's current DNS cache has a mean cache-hit ratio of 11% for web and 17% for 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

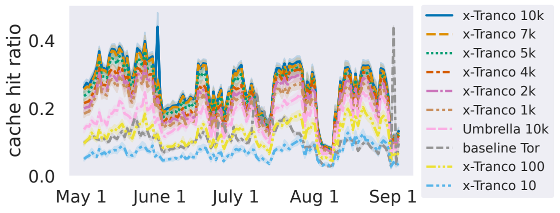


(a) web

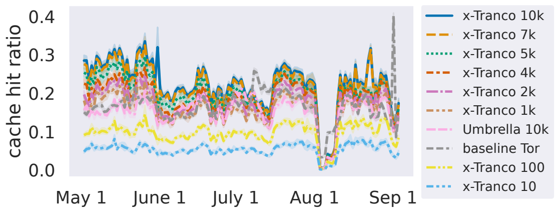


(b) permissive

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

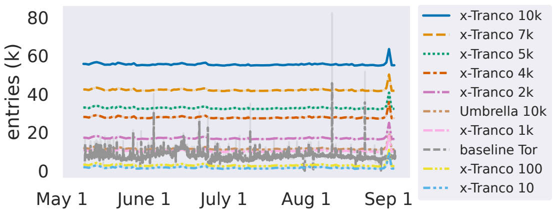
Figure 11: Shared cross-circuit cache-hit ratios for eight different extended Tranco lists, Umbrella top-10k, and Tor baseline during four months in 2022.

encouraging: time seems to have only a slight detrimental impact on cache hits. After four months the larger extended lists show a noticable performance 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.

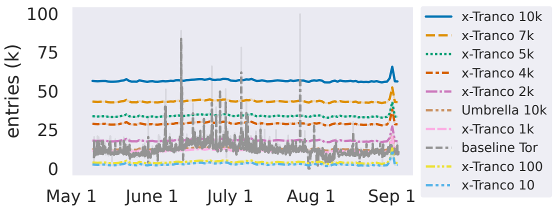
6.5.2 Results: Cache Entries

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



(a) web



(b) permissive

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

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.

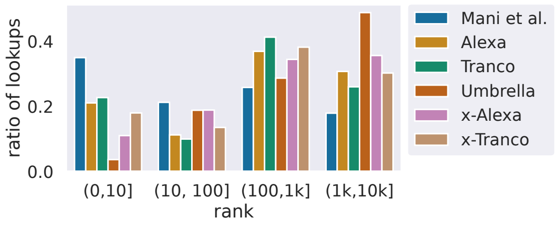
Greschbach *et al.* [13] also measured DNS requests from an exit for both web and a more permissive exit policy in 2016. The collection was done by observing DNS requests to the exit's resolver and aggregating results into five-minute buckets. Similarly, we aggregate over time in 15-minute buckets and 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.

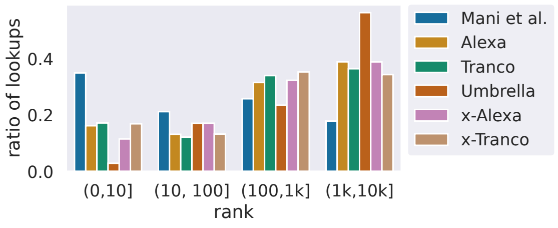
Sonntag collected hourly data from the resolver of an exit during five months in 2018 [45, 46]. In terms of frequency, they noted about 18.5 requests per second, with a peak of 291,472 requests in an hour. The average is higher than ours (3.9 and 7.8 requests per second) while the peak significantly 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



(a) web



(b) permissive

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

6.3 billion lookups, which corresponds to the number of streams.[5] This suggests 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

[5]Streams either do a lookup with RELAY_BEGIN or are closed after a RELAY_RESOLVE cell. Timeout and retries are possible on resolver failure, but the way we measure hides those extra lookups.

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.

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 to locate the above.

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/, accessed 2022-04-29.

[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, accessed 2022-04-29.

[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/, accessed 2022-06-01, 2021.

[15] Shuai Hao and Haining Wang. Exploring domain name based features on the effectiveness of DNS caching. *CCR*, 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, IETF, 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, accessed 2022-06-01.

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

[33] Mike Perry. Congestion control arrives in Tor 0.4.7-stable! https://blog.torproject.org/congestion-contrl-047/, accessed 2022-06-01.

[34] Mike Perry. A critique of website traffic fingerprinting attacks. https://blog.torproject.org/critique-website-traffic-fingerprinting-attacks, accessed 2022-06-01.

[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/, accessed 2022-06-01.

[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/.

[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, accessed 2022-06-01.

[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, accessed 2022-06-01.

[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, accessed 2022-06-01.

[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, accessed 2022-06-01.

[42] Tobias Pulls and Rasmus Dahlberg. Website fingerprinting with website oracles. *PETS*, 2020(1).

[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/`, accessed 2022-06-01.

[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).

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