Lennart Oldenburg*, Gunes Acar, and Claudia Diaz

From "Onion Not Found" to Guard Discovery

Abstract: We present a novel web-based attack that identifies a Tor user's guard in a matter of seconds. Our attack is low-cost, fast, and stealthy. It requires only a moderate amount of resources and can be deployed by website owners, third-party script providers, and malicious exits—if the website traffic is unencrypted. The attack works by injecting resources from non-existing onion service addresses into a webpage. Upon visiting the attack webpage with Tor Browser, the victim's Tor client creates many circuits to look up the non-existing addresses. This allows middle relays controlled by the adversary to detect the distinctive traffic pattern of the "404 Not Found" lookups and identify the victim's guard. We evaluate our attack with extensive simulations and live Tor network measurements, taking a range of victim machine, network, and geolocation configurations into account. We find that an adversary running a small number of HSDirs and providing 5 % of Tor's relay bandwidth needs 12.06 seconds to identify the guards of 50 % of the victims, while it takes 22.01 seconds to discover 90 % of the victims' guards. Finally, we evaluate a set of countermeasures against our attack including a defense that we develop based on a token bucket and the recently proposed Vanguards-lite defense in Tor.

Keywords: Tor, anonymous communications, guard discovery attack, onion services, web-based attack

DOI 10.2478/popets-2022-0026
Received 2021-05-31; revised 2021-09-15; accepted 2021-09-16.

1 Introduction

With an estimated eight million daily users [31], Tor [9] is the most popular anonymous communication network in use today. Tor enables users to browse privately, evade censorship, and access or run anonymous network services (*onion services*). In order to access the web via Tor, users install Tor Browser, a modified version of Firefox that routes all traffic through the Tor network. With its strong defenses against cross-site tracking and browser fingerprinting [39], Tor Browser also protects users from unwanted tracking at the application level.

When users visit a webpage with Tor Browser, their traffic is sent on a *circuit* routed through a sequence of Tor relays. When Tor Browser encounters a subresource (e.g., an image) located at an *onion address*, it first contacts a *hidden service directory* (HSDir) to resolve the address to its descriptor (`HS_DESC`). This resolution is made automatically over a newly created circuit. Using the descriptor, Tor Browser can privately connect to the onion service and access the content.

Every time a new circuit is created, Tor freshly selects new relays for all circuit positions—except the first (called *guard*), which Tor users keep fixed for many months. Guards play a critical role among all relays as they typically know the IP addresses of the Tor users connected to them. By picking one guard and keeping it for a long time, Tor users avoid repeatedly exposing themselves to selecting malicious relays as the first hop in their circuits, which would happen after a short time if users selected new first-hop relays every time [60]. The Tor network has put strong requirements in place for relays that want to become eligible as guards [8].

Attacks that discover the guard of a Tor user (victim) are thus an important subject of research. Identifying a victim's guard may help expose the victim's identity or enable other attacks, including user geolocation [18], traffic confirmation [25, 36], website fingerprinting [37, 44, 57, 58], and selective denial-of-service attacks to force the victim to choose a malicious guard [7, 13, 21, 23]. Depending on bilateral agreements and the guard's jurisdiction, an adversary may seize [56] or subpoena [6, 11] the guard or its network provider to access traffic logs. Adversaries with offensive capabilities may try to compromise the guard server.

We present an attack that allows an adversary to identify the guard of a Tor user visiting a target webpage, using a novel method that is *low-cost*, *fast*, and *stealthy*. The adversary controls (parts of) the target webpage and runs a limited number of HSDirs and Tor relays. When a victim visits the target webpage, the adversary launches the attack by injecting a large number of subresources (e.g., images) on non-existing onion

***Corresponding Author: Lennart Oldenburg:** imec-COSIC KU Leuven, E-mail: lennart.oldenburg@esat.kuleuven.be
Gunes Acar: Radboud University, E-mail: g.acar@cs.ru.nl
Claudia Diaz: imec-COSIC KU Leuven, E-mail: claudia.diaz@esat.kuleuven.be

service addresses. To obtain the embedded resources, the victim first connects to the HSDirs responsible for the addresses' descriptors, providing many opportunities for adversarial relays to be selected as second hops. Scanning her second-hop relays for the distinctive traffic pattern of an `HS_DESC` lookup for a non-existing address, the adversary obtains the victim's guard as the predecessor on the circuit that matched the target pattern.

A large class of adversaries is able to deploy our attack: website owners, embedded third parties such as advertisers, administrators of online marketplaces, users of blogging platforms, and malicious Tor exit relays when outbound connections are unencrypted or not protected against SSL stripping attacks [19]. While we assume the adversary injects the subresources using JavaScript, a scriptless version of our attack proves effective against Tor users who have disabled JavaScript.

We run our attack against a large number of our own clients on the live Tor network to obtain a set of realistic victim profiles. We find that Tor users on faster computers and network connections are more vulnerable to our attack, while geographical proximity to the majority of Tor relays also makes our attack more effective. Through extensive experimentation, we determine that our attack identifies the guards of most victims after they visit the attack webpage for a short amount of time. For instance, an adversary with a small number of HSDirs and 1 % share of Tor's relay bandwidth identifies half the victims' guards in less than 40 seconds. An adversary providing 5 % of Tor's relay bandwidth discovers more than 99 % of the victims' guards in under 50 seconds. In some instances, the adversary requires less than a second to learn a victim's guard.

Our attack is cheap and easy for the adversary to deploy. Of the Tor relays that the adversary requires for the attack, only a few HSDirs (e.g., ten) need to be launched at least 96 hours in advance of the attack. As our attack does not require to run relays in distinguished circuit positions (e.g., guards, exits), it attracts little attention to the adversary. An adversary controlling 5 % of Tor's relay bandwidth only needs a moderate budget of EUR 1,579.56 to run the attack for a full month. Precomputing onion service public keys that will be maintained by at least two adversarial HSDirs in an upcoming network status is perfectly feasible. An adversary with ten HSDirs in the network requires on average 10.3 minutes to generate the keys for a one-minute attack (52.3 minutes on average for a five-minute attack). The adversary has at least twelve hours for this precomputation before the first time a Tor client uses the new network status to download descriptors.

We make the following contributions:

- We present a novel, low-cost, fast, and stealthy guard discovery attack on Tor users. The attack exploits the unregulated creation of new circuits for `HS_DESC` lookups and the distinctive cell pattern of lookups for non-existing onion addresses.
- Through extensive simulations and experiments on the live Tor network, we find that the attack succeeds in identifying a victim's guard in seconds.
- We evaluate countermeasures: a token bucket per Tor Browser tab to limit the number of circuits, restricting the choice of second hops with the recent Vanguards-lite proposal, and a combination of both approaches. While Vanguards-lite prevents a majority of the attack attempts, over 18 % of Tor users remain vulnerable to our attack. We find the token bucket defense to be effective against our attack.

2 Background and Related Work

Tor. Tor [9] is an overlay network of volunteer-run servers (*relays*) that route user connections over multi-hop *circuits* to make them anonymous. While an open, generic network, Tor is primarily used to privately access the web with Tor Browser. This allows users to hide their IP addresses from web servers, and prevents linking of their visits by online trackers. When users visit public-Internet webpages with Tor Browser, their traffic is onion-encrypted and sent through a sequence (circuit) of three relays: *guard*, *middle*, *exit*. In order to limit linkability across different webpage visits, Tor Browser scopes these public-web circuits to the URL bar origin (*first party isolation*) [39]. The same third-party resource requested as part of visits to two distinct first-party webpages is sent across two distinct circuits.

The circuit's relays are chosen by the client according to Tor's *routing policy* which randomly selects a fresh middle and exit relay for every circuit. The choice of relay is weighted by its bandwidth while at the same time accounting for constraints such as destination filters on exits, relay family affiliation, and subnet overlap. Guards, however, are updated only once every 2–3 months [30] and are used on every circuit during that period. Guards were introduced to protect users from adversaries that compromise a subset of Tor relays and are able to fully deanonymize a circuit when they control its first and last hop [61]. Without guards, such adversaries typically have a low probability of compromise for each individual circuit created by a user unless

they control a very large fraction of the network. However, every new circuit is a new chance of adversarial success. If the user creates many circuits, it is only a matter of time until the adversary obtains the needed vantage position. The issue is mitigated by fixing the choice for the first relay (which becomes the *guard*). If the user has an honest guard, the adversary will have no chances to obtain the needed vantage position, no matter how many circuits are created.

When data is in transit, each hop of a circuit removes or adds one layer of encryption (*onion routing*), making input and output connections cryptographically unlinkable in terms of both cell headers and payloads. Users learn about all relays and their characteristics from the *consensus* document describing the network state once every hour. The unit of transmission on circuits is a *cell*, a single Tor packet with 509 bytes payload.

In addition to providing anonymous web browsing, Tor also enables services to hide their network identity (address and location) by running *onion services* (previously: *hidden services*). Onion services accept connections from within the Tor network via circuits of six hops (instead of three). Two versions of onion services currently exist, v2 [54] and v3 [55], with v2 nearing end-of-life due to its outdated cryptographic foundations [15, 16]. Onion services are accessed via Tor-internal domain names derived from the services' public keys. *Hidden service directories* (HSDirs) are Tor relays that collectively maintain the *hidden service descriptors* (`HS_DESC`s) of the onion services. An `HS_DESC` contains the information required by clients to establish private connections to the service. Each Tor relay carrying the `HSDir` flag serves a specific set of `HS_DESC`s to Tor clients.

`HS_DESC` lookups take place over dedicated four-hop circuits—which consist of the user's guard, two (instead of one) middles, and an HSDir that maintains the `HS_DESC` (instead of an exit relay). These `HS_DESC` lookup circuits are thus distinct from three-hop circuits used to browse the public web and from six-hop circuits used to connect to onion services. A fresh `HS_DESC` lookup circuit is created per lookup attempt.

Guard Discovery. Unlike the second and third hops on a Tor circuit, guards typically know the IP addresses of their users and remain their guards for a long time. Thus, discovering a victim's guard is a stepping stone towards full deanonymization. An adversary successfully mounting a guard discovery attack may be able to seize [56], subpoena [6, 11], or compromise a non-cooperating guard. This opens up a wide array of devastating deanonymization attacks on the guard's users. The adversary may monitor the guard and employ website fingerprinting techniques to learn which websites a victim is browsing [37, 44, 57, 58], uncover a victim's geolocation [18], or perform full traffic confirmation [25, 36]. Some approaches to guard discovery exploit side channels. Mittal et al. show that it is possible to identify the guard(s) of Tor users by fingerprinting the guard's throughput [32]. Rochet and Pereira [41] exploit the public relay load measurements to discover the guards of onion services.

Other approaches (including ours) rely on forcing the creation of many circuits, each of which increases the chances of adversarial compromise. Biryukov et al. [4] present an attack that discovers the guards of an onion service by forcing it to open many rendezvous connections to an adversarial rendezvous point, while a middle relay fingerprints the established circuits. They identify several guards in less than an hour. In a follow-up study, Biryukov et al. [5] propose a similar attack to deanonymize onion service users. The attack works by sending a unique traffic signature from adversary-controlled HSDirs during `HS_DESC` lookups and expecting the traffic to be routed through the adversary-controlled guards. Compared to our attack, Biryukov et al.'s attack can target all users of an onion service, but their attack misses all targets that do not use the adversarial guards. Their attack can also be easily detected based on the distinctive signature the HSDir sends and would not work against v3 onion services. Compared to guard discovery attacks in prior works, our attack converges much faster and does not require invasive techniques like congesting Tor relays.

Circuit Fingerprinting. Kwon et al. use decision trees to determine circuit purpose and relay position, which they use in turn to identify onion service circuits [29]. Addressing a similar problem, Jansen et al. instead opt for random forests to improve the robustness of the purpose and position classifier [22]. Through a series of simulations, Elahi et al. show that short-term guards make it easier to profile Tor users [12]. Motivated in part by Elahi et al.'s results, Dingledine et al. propose using one entry guard for nine months in Tor [10]. These studies informed the current design of Tor's guard selection method, which uses a single guard for 2–3 months for each user [30].

Attacks Using JavaScript. Evans et al. [13] exploit the then missing limit on path lengths to build long circuits (24 hops) to congest Tor relays selectively. Their attack requires taking latency measurements from the victim, which is achieved by injecting malicious JavaScript code on an exit controlled by the adversary.

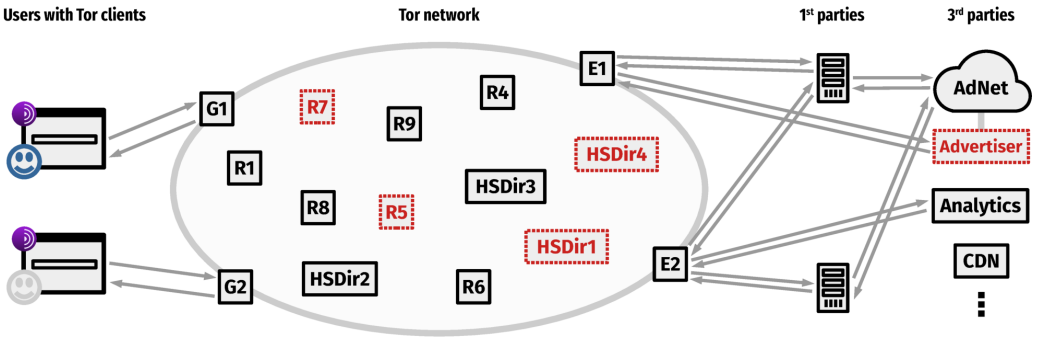

Fig. 1. Tor network model including adversarial nodes in red with dashed borders. As one possibility for how the adversary may become part of the website visited by victims, we show the adversary as a third-party advertiser here.

Evans et al.'s attack requires iterating over all potential Tor guards to find the ones that belong to the victim.

Abbott et al. present an end-to-end correlation attack based on injecting a distinctive signal using JavaScript from a malicious exit [1]. Their attack requires the victim to pick a malicious exit and malicious guard at the same time, which is made impractical with Tor's current guard rotation policy.

3 Guard Discovery Attack

In this section, we describe the resources required by an adversary to conduct our guard discovery attack. We provide a step-by-step overview of the attack and investigate crucial properties of the target cell pattern that we exploit to identify guards.

3.1 Adversarial Objectives and Capabilities

In order to identify a Tor user's guard, the adversary requires three capabilities. First, the adversary has the ability to run scripts on (or otherwise manipulate) the target webpage. Second, the adversary operates one or more HSDirs responding to `HS_DESC` lookups in the Tor network. Third, the adversary runs a set of middle relays in the Tor network. We now detail these capabilities.

Website Manipulation. The adversary needs to be able to inject subresources on non-existing onion services into a webpage visited by the victims using Tor Browser. Clearly, this is the case if the adversary controls the webpage entirely (first party). However, it suffices for the adversary to have the chance of becoming a third party on the target website. This might be the case if the adversary is an advertiser in an advertisement network that the target webpage uses. We illustrate this scenario in Figure 1. Other suitable settings include the adversary being part of a social network or marketplace that allows HTML as part of user-generated content (e.g., direct messages, posts, adverts). Going forward, we assume the adversary to be a third party with JavaScript capabilities on the target webpage. However, our attack also works when Tor Browser users have disabled JavaScript, which we discuss in Section 4.3.

Hidden Service Directories. When victims attempt to download the embedded resources at non-existing onion services, they first create a circuit to an HSDir to download the onion service's `HS_DESC`. Our attack relies on the adversary knowing if and when a specific target `HS_DESC` is requested. Thus, the adversary needs to operate at least one of the six responsible HSDirs for each attack onion address. Without HSDirs, the attack's success degrades significantly due to the high rate of false positives (see Section 5.1), as the adversary cannot observe the requested attack address and use this to distinguish victim lookups from noise.

Any relay is assigned the `HSDir` flag automatically if it requests it and meets a set of criteria. Specifically, the relay needs to be sufficiently reliable (`Stable` flag), capable of fast network operation (`Fast` flag), and have been running continuously for at least 96 hours [51]. In Section C in the Appendix, we show that operating the required relays including the HSDirs can be achieved at a low cost. Note that the Tor developers are discussing a change to make it more difficult to obtain the `HSDir` flag [46]. If implemented, this change will likely increase the attack costs for the adversary.

Middle Relays. The adversary needs to run middle relays that users may choose as second hops when constructing a new circuit (successors to guards). These middle relays are crucial for the attack as only middle re-

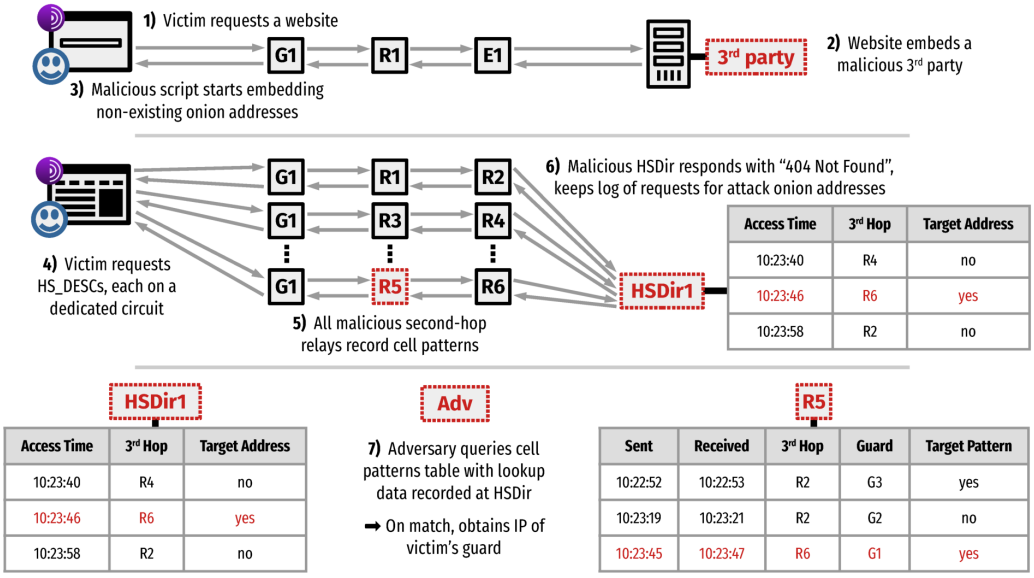

Fig. 2. Step-wise overview of our guard discovery attack, where a malicious third-party JavaScript causes the victim's Tor Browser to rapidly create a large number of new lookup circuits for non-existing HS_DESCs. Adversarial components in red with dashed borders.

lays are ultimately able to reveal the victims' guards to the adversary. As relays are selected according to their bandwidth share during circuit construction, the adversary may run more middle relays with high bandwidth to speed up the attack.

Relays that are not intended for first or last hops on circuits do not require any flags that are hard or slow to obtain. This benefits the adversary in terms of the time it takes to position all nodes for the attack, limits the attack's cost, and incurs less scrutiny compared to relays in more critical positions (e.g., guards, exits). All adversarial Tor nodes run tor binaries appropriately modified for the attack, i.e., exposing circuit metadata required for the adversary to learn the guards. Additionally, all nodes run a time synchronization protocol, as the attack requires comparing timestamps across machines.

3.2 Attack Overview

We now provide a step-by-step description of our web-based guard discovery attack, as shown in Figure 2.

First, a victim requests the target webpage using Tor Browser (step 1). If this is a webpage in the public Internet, the request will by default be relayed through a three-hop circuit to the webserver. We assume the adversary to be a part of this target webpage in the form of either a first or a third party. Once the victim connects to the adversary to download the embedded JavaScript, the attack can be launched by serving the malicious script (step 2).

The malicious JavaScript injects into the webpage's DOM a number of subresources (e.g., scripts, images, fonts) located on onion addresses (step 3). Crucially, the onion services referenced by the injected subresources do not exist. The adversary intentionally crafts attack onion addresses so that the requests for the onion services' HS_DESCs are (partly) handled by her HSDirs. The malicious JavaScript takes care of hiding the injected non-existing resources visually from the victim (e.g., by injecting scripts or setting image dimensions to zero) to evade detection.

For each non-existing onion address on the webpage, the victim's Tor Browser sequentially creates six four-hop circuits to download the HS_DESCs (called *victim lookups*), one per responsible HSDir (step 4). Each newly created circuit represents an opportunity for the adversary's middle relays to be selected as the hop succeeding the victim's guard. Currently, there is no limit to the number of circuits a webpage can cause to be created (a known issue since 2016 [2]). Through live Tor network measurements (Section 4) we find that the adversary can cause victims to perform a median of 12.4 HS_DESC lookups per second, while upwards of 23 lookups per second are also possible.

During the attack, adversarial middle relays continuously record cell metadata *circuit logs* (step 5). This metadata includes: predecessor and successor on the cir-

cuit, whether the circuit's cell pattern matches the target pattern, and if so, timestamp t_s of the cell carrying the HS_DESC lookup and timestamp t_r of the response cell. We discuss the target cell pattern in Section 3.3.

Adversarial HSDirs also record metadata (in *access logs*) when they respond to HS_DESC lookups (step 6). They keep track of the lookup's timestamp t_a, from which relay they received it, and whether it requested one of the specially crafted target addresses. Independent of the result of the latter check, malicious HSDirs always respond according to specification. That means that they respond with "404 Not Found" to lookups for which they do not have the descriptors, including to the ones for the attack addresses. This response always fits into the payload of a single cell, in contrast to the much larger responses that carry an HS_DESC for an existing resource. Upon receiving a negative lookup response, the victim's Tor client contacts the other HSDirs responsible for the address until all have been queried and the lookup is abandoned.

Our attack exploits the distinctiveness of HS_DESC lookups for non-existing onion addresses to identify victim circuits. The adversary compares entries of interest in the logs of the HSDirs and the middle relays (step 7). The adversary first selects those circuits from the middle relays that match the target cell pattern and those from the HSDir that correspond to the lookup of a target address. The adversary then compares the timestamps and third hops of the selected circuits. Two circuits match when they share the same third hop and the lookup timestamp t_a falls between t_s and t_r that the middle relay recorded. For each matching circuit the adversary records the identity of the relay at the first hop as candidate for being the victim's guard.

3.3 Characteristics of Target Cell Pattern

In order to determine the feasibility of our guard discovery attack, we investigate whether the cell pattern of an HS_DESC lookup with a "404 Not Found" response is deterministic and unique among a large number of Tor traffic patterns. Our attack relies on this target cell pattern being deterministic and distinctive enough so that adversarial middle relays can detect it among other unrelated Tor traffic. Evaluating the uniqueness of this pattern on the live Tor network would require recording the traffic of Tor users, which we avoid for ethical reasons. Instead, we opt for analysis in an isolated test network using Shadow [20] and TorNetTools [24, 43].

Shadow is a discrete-event network simulator that directly interfaces with a tor binary via a plugin. Shadow can run on a single machine of sufficient hardware capabilities and accepts a network model, traffic patterns among the network's nodes, and a custom tor binary. Shadow executes a simulation of the specified traffic patterns, producing logs for further analysis. TorNetTools allows us to generate a scaled-down, representative version of the Tor network based on official metrics data from the Tor Project.

We conduct two simulations, which we describe in Section A of the Appendix. In the first simulation, a single Tor client attempts to connect to a non-existing onion service multiple times over a minimal Tor network (Section A.1). From the logs produced by Shadow, we extract the expected number of the target "Onion Not Found" cell pattern as seen at the second hop in the lookup circuit. We find the cell pattern to be deterministic and visualize the involved cells and commands in Figure 9 in the Appendix.

Second, we conduct a much larger experiment where clients generate a diverse set of traffic patterns that *exclude* the target cell pattern (Section A.2). We run this experiment on a 2 %-scale Tor network generated with TorNetTools. We analyze the observed cell patterns to see if any of them match the target cell pattern, which would lead to false positives when used in this attack. However, among 1,866,782 circuits that clients generate as part of this simulation, we do not find the target cell pattern once in any circuit position. Based on the data we collect in these two simulations, we determine the cell pattern of an HS_DESC lookup for a non-existing onion address to be deterministic and unique, and thus usable for the adversary in our attack.

4 Tuning the Attack

The adversary aims to maximize the victim lookup rate as that speeds up discovery of the victim's guard. In this section, we perform two experiments to determine the parameters that trigger the highest number of victim lookups across different types of clients and thus optimize the effectiveness of our attack. We also present a version of the attack that works without JavaScript.

Injections / s	Onion service	Median	Min	Max
1.0	v2	5.216	4.076	5.681
	v3	5.267	3.737	5.674
2.0	v2	10.416	3.083	11.225
	v3	10.405	4.277	11.221
3.0	v2	14.377	7.790	16.620
	v3	14.334	6.877	16.600
4.0	v2	14.298	6.402	21.894
	v3	14.180	4.513	21.985
5.0	v2	12.906	0.170	22.091
	v3	12.555	6.535	22.607
6.0	v2	11.611	6.070	17.660
	v3	10.811	4.868	16.718

Table 1. Median, minimum, and maximum number of victim lookups per second, grouped by resource injection rate and onion service version. Experiment described in Section 4.1.

Injections / s	Machine and network setting	Median
1.0	Desktop, fiber, wired	5.293
	Fast notebook, fiber, wired	5.250
	Slow notebook, VDSL, wireless	5.194
2.0	Desktop, fiber, wired	10.520
	Fast notebook, fiber, wired	10.495
	Slow notebook, VDSL, wireless	10.200
3.0	Desktop, fiber, wired	15.680
	Fast notebook, fiber, wired	14.292
	Slow notebook, VDSL, wireless	12.763
4.0	Desktop, fiber, wired	19.063
	Fast notebook, fiber, wired	14.592
	Slow notebook, VDSL, wireless	11.275
5.0	Desktop, fiber, wired	16.342
	Fast notebook, fiber, wired	13.030
	Slow notebook, VDSL, wireless	9.434
6.0	Desktop, fiber, wired	13.783
	Fast notebook, fiber, wired	11.496
	Slow notebook, VDSL, wireless	8.585

Table 2. Median number of victim lookups per second, grouped by resource injection rate and victim setting. Experiment described in Section 4.1.

4.1 Attack Parameters

We investigate the impact of three attack parameters the adversary controls: type of the injected resource, onion service version of the injected address, and rate of resource injections per second. In addition, we study the influence of client machine and network characteristics. The adversary aims to adjust these parameters for triggering the highest rate of victim lookups per second in a broad range of clients. More lookups mean more opportunities for creating a circuit that includes the adversarial middle and HSDir relays and thus less time needed to compromise the victim.

We set up an attack webpage that we automatically visit with Tor Browser from victim clients under our control. We use the `tor-browser-selenium` [3] Python library to automate Tor Browser. We experiment with injecting six different resource types into the attack webpage: `script` elements (`js`), images, CSS stylesheets, fonts, `iframe` elements, and making asynchronous requests using JavaScript's `Fetch` API. Using `stem` [26], a Python library to interact with Tor, we collect Tor control port logs of victim lookups when visiting the attack webpage with a victim client. We use a desktop computer as victim Tor client that is configured to use a Tor relay we operate as its guard, and run the attack while varying onion version and injection rate for each resource type. In total, we obtain data from 242,004 `HS_DESC` lookups that result in a "404 Not Found" response. We find only very minor differences between resource types, and thus exclusively embed `script` elements going forward.

Next, we assess the impact of onion service version and injection rate on the achieved victim lookup rates over the live Tor network. To approximate Tor's distribution of users over guards, we sample 50 relays from all relays carrying the `Guard` flag in the most recent consensus, weighted by their respective `guard_probability`. For each injection rate in $\{1.0, 2.0, 3.0, 4.0, 5.0, 6.0\}$ resources per second and onion service version in {v2,v3}, we cycle three times through the guards list with three different clients and in each instance visit the attack webpage for 60 seconds.

In total, we collect traces for 3,532,291 victim lookups. In Table 1, we list the rate of `HS_DESC` lookups obtained in the experiments, grouped by resource injection rate and onion service version. Based on minimal differences in median values, we consider both the v2 and v3 onion services to be equally vulnerable to our attack. In the remaining live-network experiments we use v2 addresses for simplicity.

In our experiments, we use three different client configurations as shown in Table 2. The variance in results across these three client configurations indicates that Tor users with faster computers and better network connections are more vulnerable to our attack, as this results in higher lookup rates. In order to better understand these client-dependent variations and determine the optimal resource injection rate, we perform a larger

study taking into account various client machine and network settings, including a client's geolocation.

4.2 Maximizing the Victim Lookup Rate

We conduct a third live Tor network measurement to determine the injection rate that yields the highest rate of `HS_DESC` lookups across a diverse set of victim locations, machines, and network conditions. Based on results from the preceding experiments, we focus on covering the most relevant injection rates while fixing the onion service version to v2 and the embedded resource type to `script`. We define three victim client hardware and network profiles to represent three different types of Tor users. We deploy this experiment's clients across three geographically distributed vantage points of a public cloud provider in order to factor in the effects of physical distance along Tor circuits.

Our three victim profiles are characterized as follows. A *weak* Tor client has 2 virtual CPU cores, 2 GB of memory, and a *slow* network connection with 10 Mbit/s downstream and 1 Mbit/s upstream bandwidth, latency following a normal distribution with 30 ms mean and 100 ms variance, and 0.5% packet loss with 25% correlation with the prior drop decision. A *regular* client has 4 vCPUs, 8 GB RAM, and a *medium* connection (50 Mbit/s down, 10 Mbit/s up, latency following $\mathcal{N}(20, 25)$, 0.25% packet loss with 25% correlation). A *powerful* client has 8 vCPUs, 16 GB RAM, and a *fast* connection (150 Mbit/s down, 50 Mbit/s up, latency following $\mathcal{N}(10, 1)$, no packet loss). The network speeds, latency, and packet loss profiles are modeled with some abstraction after the FCC's annual consumer broadband measurements in the U.S. [14]. To also provide a realistic upper limit, we include our *Desktop* office machine (8 vCPUs, 64 GB RAM) connected with *no caps* to our university's high-speed network in Europe.

We sample a fresh list of 100 relays from all relays carrying the `Guard` flag in the most recent consensus, again weighted by their respective `guard_probability`. For each victim client, we visit the attack webpage three times per relay in the list of 100 guards, each time configuring the relay as the victim's guard for the duration of the attack. As before, we use `stem` to collect metadata (e.g., lookup start and end times) on 9,751,286 victim lookups from 17,163 completed out of 18,000 total runs. Failed runs are mainly due to the guard being unavailable during the experiment.

We find that injecting three resources per second yields the highest median lookup rate (12.4 per second) across all victim locations, machine, and network settings. Thus, three resource injections per second is the best choice for the adversary to be effective against the most victim configurations. Note that the adversary may further optimize the attack by first measuring the network latency of a victim [45] and adjusting the resource injection rate accordingly. We refer to Section B in the Appendix for further discussion of the impact of geolocation, machine characteristics, and network conditions on the adversarial injection rate.

4.3 Scriptless Attack

In the attack described so far, the adversary relies on JavaScript to control the timing and injection rate of non-existing resources into the webpage viewed by the victim. While JavaScript is enabled by default in Tor Browser, users with high security requirements may disable it by setting Tor Browser's security level to *high*. Such users can still be targeted by the *scriptless* version of our attack. Instead of relying on JavaScript to inject resources at regular intervals, the scriptless attack embeds all non-existing resources in a `noscript` HTML tag [34]. This causes the resources to be inserted into the page only if JavaScript is disabled.

The main challenge we have to overcome in this version of the attack is to prevent the page being blocked from loading, which would cause the loading indicator to spin for a long time—potentially raising suspicion. Experimenting with different techniques to delay the loading of DOM elements beyond initial page load, we find that embedding image elements with the `loading="lazy"` attribute causes Tor Browser to load images in the background, after the initial page load is completed. This effectively gives the victim the impression that the page has loaded successfully.

To test the effectiveness of our attack in this setting, we modify the attack webpage to include a `noscript` tag with 180 image elements having non-existing onions service domains (`src`). We pick 180 as it corresponds to the number of resources injected by a rate of three per second over 60 seconds. Using this attack webpage, we repeat the experiment described in Section 4.2 to measure the rate of victim lookups. We only take measurements using the university desktop and the same sample of 100 guards used in Section 4.2. We visit the attack page while having the security slider in Tor Browser automatically set to *high*, using `tor-browser-selenium`. We repeat the visit three times with each guard, resulting in 300 visits—of which one third failed due to offline

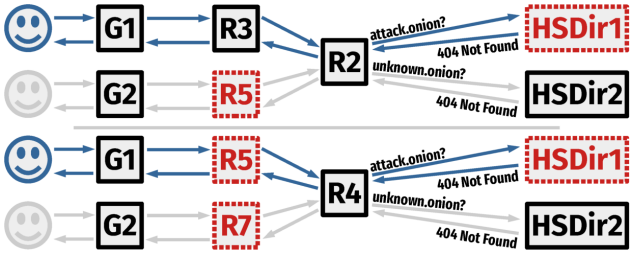

Fig. 3. Lookups for non-existing, non-attack onion addresses (noise lookups) may result in false positives. Second-hop adversarial relays (e.g., R5, R7) see the target cell pattern, but cannot distinguish whether it is due to a victim or noise lookup.

Onion service	HTTP code	Message	Response cells	Count (%)
v2	200	Found	multiple	47,331 (5.35)
	400	Decoding Failed	single	0 (0)
	400	Invalid Descriptor	single	10 (0)
	404	Not Encrypted	single*	0 (0)
	404	Not Found	single	446,826 (50.53)
v3	200	Found	multiple	31,564 (3.57)
	404	Decoding Failed	single	0 (0)
	404	Not Found	single	358,421 (40.53)
	503	Reject Single Hop	single*	108 (0.01)
			multiple	**78,895 (8.92)**
			single	**805,257 (91.07)**

Table 3. Counts and percentages of responses to v2 and v3 `HS_DESC` lookups, observed over 31 days on a single HSDir. Adversarial second-hop relays might mistake single-cell responses from non-attack `HS_DESC` lookups as false positives (except for responses marked with an asterisk).

guards. Based on the remaining 200 visits to the scriptless attack webpage, we obtain a median rate of 13.6 `HS_DESC` lookups per second. This shows that the attack is feasible without JavaScript, albeit slightly slower.

5 Attack Evaluation

We now evaluate how accurately and quickly our attack identifies the guards of victims. In order to account for false positives that could impair the accuracy of our attack, we first establish the rate of background noise lookups in Tor. We then present simulations to compute the expected time and accuracy of identifying the victims' guards through our attack. Next, we compute the time it takes to generate a fresh set of attack onion service public keys. Finally, we discuss the monetary cost of the attack, considering various adversarial capabilities.

5.1 Estimating Tor's Noise Lookup Rate

In Section 3.3, we show that the cell pattern of a victim lookup is unique among a large sample of other Tor traffic. However, as prior research [31] shows, Tor users still make `HS_DESC` requests that result in "404 Not Found" responses independently from our attack. We call these *noise lookups*. Noise lookups occur when Tor users try to connect to an offline onion service or mistype an onion address. Noise lookups may cause false positives when they go through a middle relay controlled by the adversary. Below we discuss in detail how we measure and address these false positives to prevent them from reducing the efficacy of our attack.

We show in Figure 3 two scenarios where a pattern match may be due to a false positive. In the first case (top figure), the victim lookup (blue arrows) uses an honest second hop (R3) and the noise lookup (grey arrows) uses an adversarial second hop (R5), while victim and noise lookups share the same (honest) third hop (R2). In this case, the adversary may mistakenly identify the noise lookup guard (G2) as the victim's guard (G1). In the second case (lower figure), the adversary detects two simultaneous attack cell patterns and can only deduce that one of the two candidate guards (G1, G2) belongs to the victim, but not which one.

In order to quantify the rate (N) of noise lookups naturally occurring in the Tor network, we take measurements on the Tor relay that we are running at our university, which already carries the `HSDir` flag. We modify the tor binary (version `0.4.4.5`) to count the number of times our relay responds to `HS_DESC` lookups with a specific response (HTTP code and message). We log these counts binned by hour for 31 days (744 hours) and we make sure to log nothing else. Prior to data collection, we discussed our plans with the Tor Research Safety Board and integrated their feedback (with an overall *no objections* conclusion from both reviewers). Data from these measurements are shown in Table 3, noting additionally whether a particular response requires a single or multiple cells. Responses marked with an asterisk may not confuse adversarial middle relays, because they either do not take place over four-hop circuits or are not encrypted (and thus distinguishable).

Of 884,260 responses over 31 days, more than 91 % were single-cell responses. Of those 805,365 single-cell responses, almost all are due to lookups that resulted in "404 Not Found", which is in line with prior results [31].

We take the number of single-cell responses (without asterisk) that the adversary might mistake for the target cell pattern as the basis for the noise lookup rate. Extrapolating this number from our single HSDir to all ca. 3,500 relays carrying the `HSDir` flag as of the time of writing, we obtain:

$$\mathrm{N} = \frac{805{,}257 \cdot 3{,}500}{744 \cdot 60 \cdot 60} = 1{,}052.27$$

as an estimate for the rate of background noise lookups per second across the entire Tor network.

5.2 Experimental Setup

We use simulations to evaluate the efficacy of our attack taking into account factors such as noise lookups, Tor's path selection algorithm, and a varying range of adversarial capabilities. The simulations have three main components as explained below. In order to realistically model these components, we use empirical data collected in the previous sections.

Victim. We simulate a single victim that visits the attack webpage for 300 seconds. For the first 60 seconds, we use the actual victim lookup timings that we record in our live-network experiments (see Section 4.2). Limiting ourselves to experiments with the optimal rate of three injections per second, we extract the start and end times of victim lookups from 3,000 experiments (10 machines × 100 guards × 3 visits), excluding 133 live-network experiments where tor failed to launch. Extracting the lookup timings allows us to *replay* the victim lookups from different client configurations. To prolong each 60-second experiment to 300 seconds, we generate cover victim lookups at the average lookup rate of the experiment's last 30 seconds. We use the average from the last 30 seconds to prevent a potential initial burst of lookups from skewing the stable-state rate.

We run each experiment 100 times to account for the variance introduced by Tor's path selection process. We simulate a range of adversarial capabilities: adversarial fraction $b = \{0.01, 0.02, 0.05\}$ of Tor's relay bandwidth and fraction $h = \{\frac{1}{6}, \frac{1}{3}, 1\}$ of responsible HSDirs for each attack address that are adversarial (i.e., either one, two, or all six responsible HSDirs). For v3 onion services, $h = \frac{1}{3}$ is the attack success lower bound when the adversary spends a few minutes every day (see Section 5.4) to compute a set of attack onion service keys that each map to *at least two* of her HSDirs. Similarly, $h = \frac{1}{6}$ is the attack success lower bound when she uses attack onion service keys that each map to *at least one* of her HSDirs. Setting $h = 1$ requires future HSDir hashring placement to be predictable, which is the case for v2 onion services but not v3. As v2 is deprecated and will soon stop being supported [15, 16], we consider $h = \frac{1}{3}$ as the most representative baseline for the attack impact.

For each victim lookup, we simulate building a four-hop circuit, taking into account relay weights, adversarial relay bandwidth share, and adversarial fraction of responsible HSDirs per attack address. First, we assign the same guard from the live-network experiment to the victim. Next, to select the second hop, we represent all adversarial relays as a single logical relay that we select with probability b. Accordingly, we represent all benign relays as a single logical relay with selection probability $1 - b$. This abstraction does not affect the results of our evaluation, as the adversary is assumed to coordinate among her set of middle relays. Then, we pick a third hop according to relay probabilities from the same consensus we used in Section 4.2. Finally, depending on h, we pick an HSDir as follows: if $h = 1$, every victim lookup is sent to the adversary's HSDirs. If $h = \frac{1}{3}$, two out of six victim lookups are routed to the adversary's HSDirs on average. With $h = \frac{1}{6}$, on average only one of six victim lookups reaches an adversarial HSDir.

Noise Lookups. A realistic model of the noise lookups is crucial for our evaluation, as the overwhelming majority of `HS_DESC` lookups in the Tor network exhibits the target cell pattern that the adversary's middle nodes scan for. Using the measurements from Section 4.2, we simulate noise lookups at the rate we extrapolate for the Tor network from our HSDir measurements (Section 5.1, N = 1052.27 noise lookups per second).

We build circuits for noise lookups following a similar method as for victim lookups. We assume each noise lookup comes from a different user. Thus, we sample a new guard for each noise lookup circuit from all relays marked as guards in the consensus obtained during Section 4.2, weighted by their `guard_probability`. The HSDirs for noise lookups are picked randomly from the ca. 3,500 HSDirs present in the Tor network.

Adversary. The simulated adversary follows the steps described in Figure 2 to identify the guard of the victim. The adversary runs middle relays with combined bandwidth share b and controls fraction h of responsible HSDirs. As the target cell pattern can be reliably detected by adversarial middle relays (see Section 3.3), we assume that the simulated malicious second hop detects all "404 Not Found" lookup circuits that it relays.

The adversary compares the lookup logs from her HSDirs with pattern matches recorded on her adversarial middle. The adversary sees a match when the HSDir and the middle are connected to the same third hop, and

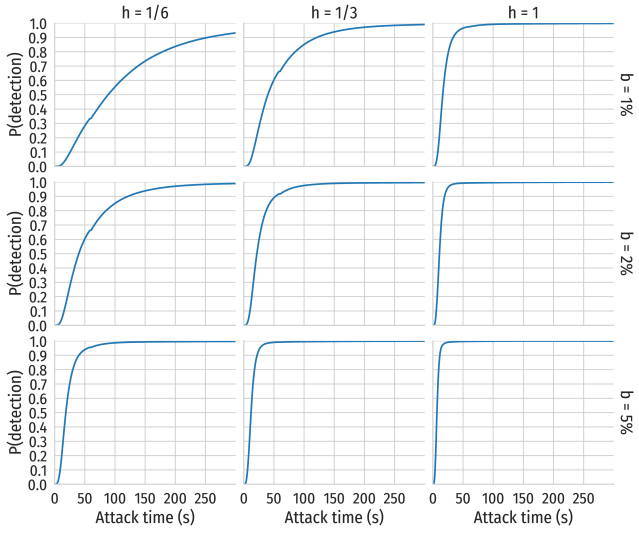

Fig. 4. Probability of identifying the victim's guard by the attack time in seconds based on 286,700 simulations per setting. The probability increases with time spent on the attack webpage, adversarial fraction of responsible HSDirs for attack onion addresses (h), and adversarial relay bandwidth share (b).

lookup timestamp t_a is between t_s and t_r that the middle relay recorded (Section 3.2). The adversary keeps track of the number of observed matches per guard. Once any of these counters reaches two (*double match*), the adversary concludes that the corresponding guard belongs to the victim.

5.3 Attack Success

We run 100 simulations for each of the 2,867 distinct experiments and nine different adversarial settings, obtaining a total of 2,580,300 simulation instances. To give an indication of the scale, we generate more than 814.8 billion noise circuits during the simulations, considering 300 seconds of simulation per experiment and 1,052.27 noise lookups per second. For each simulation instance, we record whether the guard detected by the adversary is correct or not, the time it takes to make the detection, and the adversarial parameters used in the simulation.

We use the circuits we constructed as ground truth to identify false positives in guard detection. We manually verify by checking a sample of false positives that they indeed correspond to the scenarios presented in Figure 3. Due to the high background rate of noise lookups, we observe that false positives for single matches range from 19.41 % to 20.22 % for different adversarial settings. While a true positive rate of ca. 80 % is already substantially better than randomly guessing a victim's guard from among the ca. 3,600 Tor guards, the attack's false positive rate drops almost by factor 100 when requiring a double (instead of a single) match. With double matching, the attack's false positive rate decreases to between 0.08 % and 0.37 % (Table 4). We use guard decisions from double matching in attack accuracy and duration results in Table 4 and Figure 4, as well as other metrics reported throughout the paper.

In some simulation instances the adversary does not observe enough matches for any guard to determine the victim's guard. This occurs more often for adversaries with a lower share of controlled HSDirs and relay bandwidth. Such cases fall under the "No Call" column in Table 4. The weakest adversary ($h = \frac{1}{6}$, $b = 1\,\%$) is unable to make a decision 6.82 % of the time even after five minutes of attack. However, the rate of "No Call" attacks drops to 0.09 % when the adversary has $b = 5\,\%$ of Tor's relay bandwidth and generates attack addresses for which she controls $h = \frac{1}{3}$ of the responsible HSDirs. Due to "No Call" outcomes, some CDF subplots in Figure 4 do not reach 1.0 within the simulation time.

As shown in Figure 4, the probability of identifying a victim's guard increases with 1) the time spent on the attack page, 2) the adversarial bandwidth share (b), and 3) the fraction of adversarial HSDirs for the attack addresses (h). We particularly note the effect h has on the attack's speed. The adversary may substantially reduce the attack duration by spending a small amount of time each day (see Section 5.4) to precompute attack public keys that map to two of the six responsible HSDirs ($h = \frac{1}{3}$) instead of one ($h = \frac{1}{6}$).

The strongest adversary in our evaluation (providing $b = 5\,\%$ of Tor's relay bandwidth and controlling $h = \frac{1}{3}$ of the responsible HSDirs per attack address) needs 8.7 seconds for a quarter of the victims, 12.06 seconds for half of them, and 48.56 seconds for 99 % of the victims (false positive rate: 0.13 %). If this adversary controls only one of the six responsible HSDirs per v3 attack onion address ($h = \frac{1}{6}$), the attack times increase to 12.76 (P25), 18.47 (P50), and 112.92 (P99) seconds (false positive rate: 0.09 %). A 1 % adversary ($h = \frac{1}{3}$) needs less than 40 seconds for 50 % and 74.32 seconds for 75 % of the victims (false positive rate: 0.15 %). Even the weakest adversary ($h = \frac{1}{6}, b = 1\,\%$) discovers the guards of more than 50 % of the victims within 90 seconds of the webpage visit (false positive rate: 0.08 %).

For v2 onion services, an adversary is able to generate multiple HSDir relay public keys offline that are placed in sequence on the HSDir hashring once they obtain the `HSDir` flag. Through offline precomputation, the

Setting (h, b)	Attack Duration (Percentiles) Min	P25	P50	P75	P90	P99	Accuracy TP	FP	"No Call"
$\frac{1}{6}$, 1%	1.56	45.09	88.62	155.95	254.07	⌊300.0⌋	93.10%	0.08%	6.82%
$\frac{1}{6}$, 2%	1.23	23.90	39.98	74.29	121.31	⌊300.0⌋	98.90%	0.08%	1.02%
$\frac{1}{6}$, 5%	0.59	12.76	18.47	27.13	40.03	112.92	99.63%	0.09%	0.28%
$\frac{1}{3}$, 1%	1.07	23.75	39.89	74.32	121.15	⌊300.0⌋	98.85%	0.15%	1.00%
$\frac{1}{3}$, 2%	0.71	14.59	21.69	33.13	52.49	144.84	99.50%	0.14%	0.37%
$\frac{1}{3}$, 5%	0.34	8.70	12.06	16.42	22.01	48.56	99.78%	0.13%	0.09%
1, 1%	0.76	11.52	16.37	23.49	33.59	91.49	99.42%	0.37%	0.21%
1, 2%	0.37	7.87	10.90	14.67	19.35	40.20	99.63%	0.30%	0.06%
1, 5%	0.42	4.91	6.71	8.98	11.52	24.07	99.75%	0.24%	0.01%

Table 4. Time in seconds it takes an adversary to identify a victim's guard (via double matching) based on 286,700 simulations per setting. For a $b = 5\,\%$ adversary controlling two of the six responsible HSDirs per v3 attack onion address ($h = \frac{1}{3}$), the attack takes 8.70 seconds for 25 % of the victims and 50 seconds for more than 99 % of the victims. Even a moderate $b = 1\,\%$ adversary ($h = \frac{1}{3}$) needs less than 40 seconds to discover 50 % of the guards. We include $h = 1$ for the end-of-life v2 onion services where HSDirs could be placed. The percentiles where our attack does not converge during 300 seconds of simulation ("No Call") are indicated as ⌊300.0⌋.

adversary can then obtain v2 attack onion service keys that map exclusively to her HSDirs ($h = 1$). As shown in Figure 4 and Table 4, this leads to extremely short attack times. The strongest v2 adversary ($h = 1, b = 5\,\%$), learns the guards of 50 % of the victims in just 6.71 seconds, and in 24.07 seconds for 99 % of them. We include the results on v2 just for the record however, as v2 onion services are nearing end-of-life [15, 16].

We note that the small notch in some CDF plots around 60 seconds is an artifact of how we generate cover lookups to extend 60-second experiments to 300-second simulations. To avoid inflating the victim lookup rates we start the cover lookups after the 60 second mark, but never before. An actual attack deployed against real Tor users does not experience this issue and will thus be slightly more advantageous to the adversary.

5.4 Time to Generate Attack Keys

Figure 4 shows that the attack is much faster for $h = \frac{1}{3}$, i.e., when the adversary precomputes v3 attack onion addresses that map to two of her adversarial HSDirs. A hashring constructed from data about each HSDir in the network and public parameters determines which HSDir serves which part of the v3 onion services address space. To prevent HSDir placement on the hashring from being predictable too far into the future, the process involves a shared random value from Tor's consensus that changes daily [55]. A relay wanting to become an HSDir needs to be active in the network for at least 96 hours before that happens, so that the shared random value has changed multiple times when the relay joins the hashring.

We implement the algorithm to obtain a set of v3 attack onion service keys that map to *at least two* adversarial HSDirs for a set of public parameters (shared random value, current time period) from an actual Tor network state. We target a set size of 900 keys, which corresponds to five minutes of attack at three injections per second (see Section 4.2). We are interested in the average time it takes an adversary to generate keys that map to at least two adversarial HSDirs, given that she controls a number of HSDirs in the network. We detail the generation process in Section D in the Appendix.

We run the key generation script ten times for each number of adversarial HSDirs to account for variance in their selection. We parallelize the script to run across 20 processes (using ten hyper-threaded cores). The average time to obtain 900 v3 attack onion service keys that map to at least two adversarial HSDirs is shown in Figure 5. On average, an adversary running 25 HSDirs needs between 1.68 minutes (180 keys) and 8.1 minutes (900 keys). As the adversary has at least twelve hours between knowing the subsequent shared random value and this value's first use by Tor clients, it is easily feasible to precompute attack onion service keys to obtain $h = \frac{1}{3}$. For an adversary with ten HSDirs, this takes between 10.3 minutes (180 keys) and 52.3 minutes (900 keys), on average. Thus, the adversary makes a trade-off between running more HSDirs to minimize the daily precomputation and running fewer HSDirs that require longer—but feasible—daily precomputation efforts.

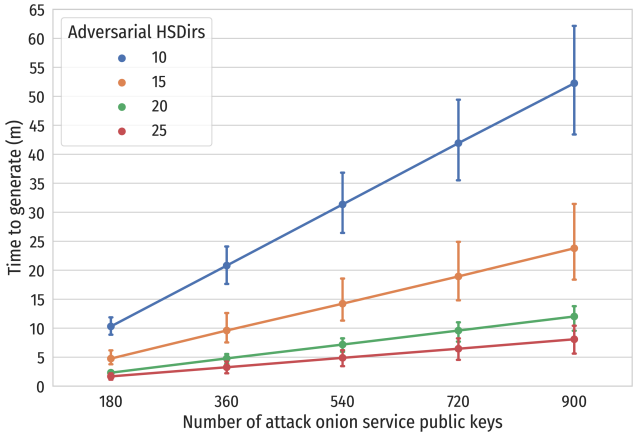

Fig. 5. Average time in minutes it takes an adversary running 10, 15, 20, or 25 HSDirs to generate a set of up to 900 onion service public keys that map to at least two adversarial HSDirs. Due to daily rotation of the responsible HSDirs per v3 onion service, an adversary needs to regenerate these keys every day. Error bars mark the 95 % confidence interval from ten repetitions.

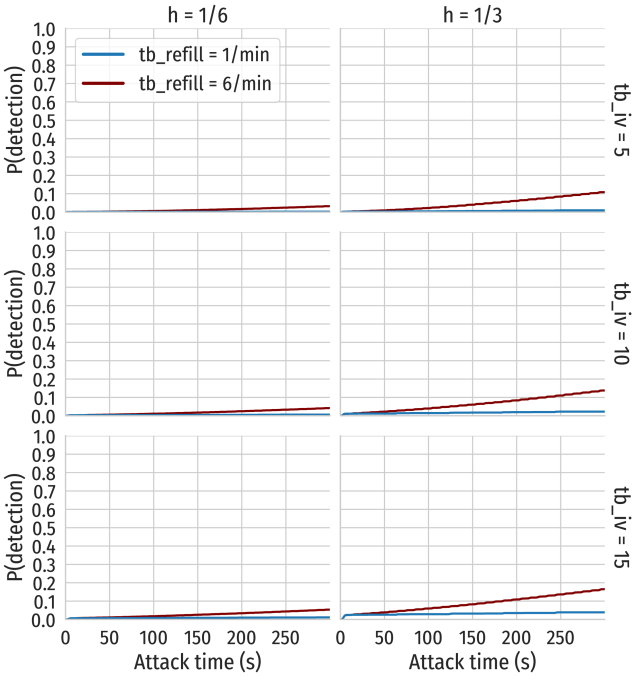

Fig. 6. Probability of detection for a 5 % adversary, considering a token bucket with `tb_iv` initial tokens and refill rate `tb_refill`.

6 Countermeasures

We evaluate three countermeasures against our guard discovery attack: a **token bucket** that rate-limits the number of lookup circuits created by a Tor Browser tab, deploying **Vanguards-lite** [27] to introduce second-hop guards, and a **combination** of the two approaches. The token bucket restricts the number of lookup circuits a Tor Browser tab can create and thus reduces the adversary's chances to observe the lookup pattern. Vanguards-lite restricts the set of eligible second-hop relays to four per user (called *L2 guards*) and thus limits our attack to users with adversarial relays among their L2 guards. Combining Vanguards-lite with a specially configured token bucket allows to limit attack success in the presence of adversarial L2 guards.

We evaluate each countermeasure by integrating it into our attack simulation from Section 5.3. Each individual configuration below is based on 50 simulations for each of our 2,867 live-network lookup profiles from Section 4.2, leading to 143,350 simulations per displayed CDF. We focus on v3 onion services and a $b = 5\,\%$ adversary, and thus exclude the adversarial settings of $h = 1$ and $b = \{\ 1\,\%, 2\,\%\ \}$.

Token Bucket. The token bucket regulates the initial (and burst) number `tb_iv` of lookup circuits that each tab in Tor Browser can cause the Tor client to create, as well as the rate `tb_refill` with which tokens are replenished as long as less than `tb_iv` are available. When `tb_iv` and `tb_refill` are chosen appropriately, for most scenarios the adversary does not get enough chances to be selected as second hop before the rate limiting sets in. On the other hand, selecting too low `tb_iv` and `tb_refill` impairs the user experience of benign websites that embed more onion service resources than these limits allow. A special case represents a `tb_refill` of zero, where `tb_iv` acts as an upper limit on the total number of circuits a tab can create.

In order to estimate which `tb_ivs` do not impair most benign websites, we crawl 115 onion websites from a list of high-profile onion websites [35] with `tor-browser-selenium`. Using tor and Tor Button logs, we find three to be the maximum number of distinct onion addresses embedded on the crawled websites. We pick `tb_ivs` of five, ten, and fifteen to account for onion websites with a few more embedded onion addresses and cases where not all lookups are successful on first try.

We evaluate each `tb_iv` in { 5, 10, 15 } with each `tb_refill` rate in { 1/min, 6/min } and show the time it takes the adversary to compromise a victim's guard in Figure 6. Against an $h = \frac{1}{3}$ adversary, a refill rate of 6/min still leads to compromise probabilities above 10 % after five minutes, independently of the chosen `tb_iv`. Thus, we do not further consider `tb_refill` = 6/min. However, with a refill rate of 1/min, we see ca. 1 % (`tb_iv` = 5), 2 % (`tb_iv` = 10), and 4 %

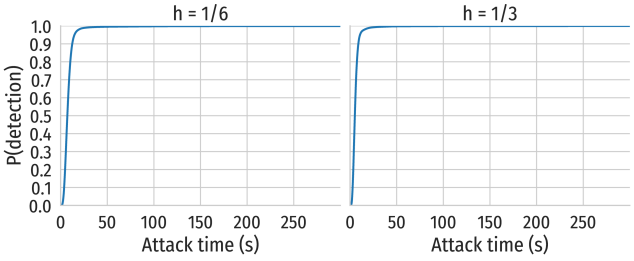

Fig. 7. (Entry) Guard detection probability for a 5 % adversary when at least one of the victim's four L2 guards is adversarial.

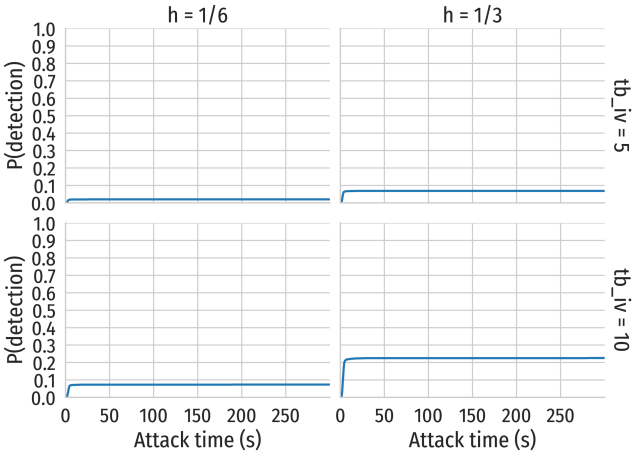

Fig. 8. Guard detection probability for a 5 % adversary when at least one of the four second-hop guards on onion circuits (Vanguards-lite) chosen by the victim is adversarial and a token bucket with `tb_iv` initial tokens and `tb_refill` = 0 is used.

(`tb_iv` = 15) after five minutes against the $h = \frac{1}{3}$ adversary. Thus, at `tb_refill` = 1/min, even with `tb_iv` = 15 (five times the observed maximum of three distinct onion addresses) a victim's chance of compromise is reduced to ca. 2.5 % after 60 seconds on the attack webpage and ca. 4 % after 300 seconds.

Vanguards-lite. With the recent Vanguards-lite [27] proposal enabled, Tor clients and onion services sample four relays from the list of all active relays weighted by bandwidth and use them as second-hop guards (L2 guards) on every onion circuit. Instead of picking a second hop for a new onion circuit from all middle relays, this hop is now chosen uniformly at random from the four L2 guards. In contrast to the existing Vanguards Tor add-on [28, 40], Vanguards-lite rotates L2 guards sooner (average retention: seven days), and does not write them to disk. Thus, a new set of four L2 guards is picked when the user restarts tor or requests a new identity [49]. Moreover, unlike Vanguards, Vanguards-lite does not change the default path lengths by adding an additional layer of (L3) guards.

Considering four L2 guards and a $b = 5\,\%$ adversary, on average about 18.5 % of Tor users will have at least one adversarial L2 guard. As our attack requires the adversary to control the second and fourth hop of a victim lookup circuit, Vanguards-lite reduces the share of vulnerable Tor users, from close to 100 % to about 18.5 % on average. The alternative of three L2 guards considered by the proposal authors further reduces the average share of vulnerable Tor users to about 14.3 %.

However, once an adversarial relay is part of a victim's L2 guards, our attack needs a much shorter time to complete. We implement Vanguards-lite in our attack simulation and evaluate the case of a Vanguards-lite user with at least one adversarial L2 guard against a $b = 5\,\%$ adversary with h in $\{ \frac{1}{3}, \frac{1}{6} \}$. The simulation results (Figure 7) show that it takes 21.31 seconds for an $h = \frac{1}{3}$ adversary (or 26.76 seconds for $h = \frac{1}{6}$) to discover the entry guards of 99 % of the victims. Comparing the results to Table 4, the median attack time drops by more than half: from 12.06 to 5.13 seconds.

Vanguards-lite with Token Bucket. To address the vulnerability of Vanguards-lite users with an adversarial L2, we combine the two approaches presented above by applying a token bucket per Tor Browser tab on top of Vanguards-lite. Due to limited usefulness of low refill rates yet steadily increasing compromise probabilities over longer attack times, we set `tb_refill` = 0 for this defense. Thus, `tb_iv` effectively acts as an upper bound on the total number of lookup circuits any Tor Browser tab can create. We show a victim's guard discovery probability with at least one of four adversarial L2 guards and a token bucket with `tb_iv` in { 5, 10 } in Figure 8. For the stronger adversary ($h = \frac{1}{3}$), compromise probability is at 7 % (`tb_iv` = 5) and 22.5 % (`tb_iv` = 10), respectively. Combined with applying to only about 18.5 % of Tor users to begin with, we obtain on average about 1.3 % (`tb_iv` = 5) and 4.2 % (`tb_iv` = 10) compromised Tor users after five minutes of attack, down from almost 100 % without any defenses.

Discussion. Based on our simulations, a token bucket per Tor Browser tab with a relatively high `tb_iv` of 15 and a `tb_refill` of one (or lower) appears to be a simple and effective countermeasure to limit the risk from our guard discovery attack. However, it is a specific defense that only targets `HS_DESC` lookup circuits from Tor Browser tabs and may lead to user experience (UX) issues once a tab has exhausted its tokens. If a webpage embeds many subresources on non-existing onion addresses, the token bucket defense blocks all `HS_DESC`

lookups once it is empty, preventing the webpage from loading fully or functioning properly. A challenging UX decision then is to decide whether and how to communicate the status *"no tokens left"* to users. On onion websites, Tor Browser displays different icons in the address bar to warn users about potential security issues such as "The Onion Service is served with a mixed form over an insecure URL" [50]. We show the existing set of onion icons in Section E in the Appendix. We believe adding a new icon that is displayed when no tokens are left to be the most effective and least intrusive solution. Further, research on token replenishment strategies based on user actions (e.g., clicks) may be an interesting avenue for future work to avoid unwanted token depletion.

Vanguards-lite is a more general-purpose countermeasure that does not impair website rendering. It is informed by and benefits from the Tor community's experience with the guard concept and is derived from the already used and tested Vanguards add-on. However, a substantial number of Tor users will have at least one adversarial L2 guard (ca. 18.5 % for $b = 5\,\%$) and thus remain vulnerable to our attack. Adding a cap on the number of lookup circuits to Vanguards-lite has a limited effect: there are only four choices for the second hop, so it does not take many circuits to select an adversarial second hop twice, allowing the adversary to succeed.

7 Discussion

Fundamental Problem and Countermeasures. The fundamental issue that enables our attack is the same that led Tor to introduce guards in the first place: a small probability of circuit compromise becomes problematic if the user can be tricked into creating many circuits (i.e., taking many fresh chances), as the adversary only needs to get lucky once to succeed. Our attack uncovers a new vector exploitable by adversaries who target users visiting a certain webpage. Known countermeasures aim to reduce the chances of compromise, either by *capping the number of attempts* (e.g., token bucket) or by *reusing previously made choices* (e.g., Vanguards-lite and guards themselves).

Capping attempts can have a usability impact, while reusing choices introduces linkability and thus may be exploitable for deanonymization. We find that capping circuit creations is a more effective protection in our attack scenario. Note however, that while a cap on the number of `HS_DESC` lookup circuits per Tor Browser tab is possible with a small impact on user experience, capping is not a viable solution in other contexts. For example, the "capping" alternative to reusing the first-hop relay (entry guard) on circuits would imply limiting the amount of pages browsed with Tor, completely defeating its purpose. The best solution (capping attempts vs. reusing choices) for each manifestation of the problem is therefore highly context-dependent.

Recurring attacks against Tor guards show that a useful future research direction may be to systematize the concept of guards, including attacks and defenses against them. Such research may attempt to consolidate the existing specialized defenses into a framework and inform future countermeasures about the fundamental problems and trade-offs.

Attack Impact and Feasibility. Typically, guard discovery serves as a stepping stone for further attacks such as traffic confirmation [25], website fingerprinting [37, 57], and selective denial of service [7]. Upon identifying a victim's guard, actors with offensive capabilities may go on to compromise, coerce, or subpoena the server. Past web-based guard discovery attacks [47] with a similar impact were flagged as *high priority* by Tor developers, while being roughly 60× slower than our attack. Furthermore, our attack only needs a modest budget and can thus be carried out by adversaries with limited budgets (Section C, Appendix). The Tor relays that the adversary operates are of the types that receive the least scrutiny. The biggest constraint for the adversary is that she needs to run her designated HSDirs at least 96 hours in advance of the attack so that each obtains the `HSDir` flag.

By forcing a Tor client to create many circuits in a short time, our attack enables other attacks that can be deployed by an adversary with a limited view of the network (e.g., with a small bandwidth share). For instance, if an adversary deploys our attack while also running malicious guards, she may fully deanonymize Tor users instead of just identifying their guards.

Our attack can be deployed by a wide range of adversaries, including website owners and embedded third parties. If the adversary cannot get access to a website visited by the target, she may lure the victim into visiting a malicious website via a spear phishing email, text message, or direct message on an online platform (e.g., a dark-web marketplace). Moreover, a malicious exit may inject the attack code into websites visited by Tor users over unencrypted HTTP connections [59].

Ethical Considerations. We collected data with the potential to impact other Tor network participants on two occasions. First, when we performed our geographically distributed victim crawls (Section 4). Sec-

ond, when we counted the number of times our HSDir responded with a specific code to HS_DESC requests (Section 5.1). For both experiments, we first discussed our plans with Tor's Research Safety Board and modified details based on the valuable feedback we received. We took great care of not logging any privacy-sensitive information about the requests we saw on our HSDir. We made sure not to overload guards by spreading our experimental traffic over time.

Limitations. Tor tries to use a single primary guard across all circuits of a user [30]. However, if for example exit and first primary guard are part of the same relay family or /16 subnetwork, Tor needs to use one of the other two primary guards as the first hop [38]. In such unlikely cases, our attack may identify the second or third instead of the first primary guard of a victim.

Our attack works best when deployed against a single victim or small number of victims in a targeted manner. The adversary may rate-limit the attack by serving the malicious script selectively. When run against a large number of victims concurrently, our attack will output the set of guards used by all victims, without a mapping between an individual victim and its guard. However, due to the short time it takes to obtain a guard using our attack, the adversary may run it one-by-one against a large group of victims in rapid succession.

Our estimate of the Tor-wide noise lookup rate is extrapolated from data collected on a single HSDir. We collected the data over 31 days to account for variance and approximate our numbers to the average HSDir, but our view of the HS_DESC space was still limited. While fluctuations in the noise lookup rate may affect the efficacy of our attack, the adversary can use her HSDirs to continuously monitor the noise lookup rate and adjust her relay bandwidth share accordingly.

We demonstrate determinism and uniqueness of the target cell pattern with Shadow simulations. These simulations make assumptions about the network conditions they work in, which may not account for all possible traffic patterns occurring on the live Tor network and thus should be considered an abstraction.

Tor users may detect that they are being targeted by our attack, e.g., by checking their HTTP requests with Tor Browser's Network Monitor [33] and seeing the suspiciously high number of onion service requests.

Feedback from Tor Developers. We shared our findings with the Tor developers and have been in discussion with them about potential ways to address our attack. We provided them early results from our countermeasures evaluation, including on the recent Vanguards-lite proposal [27]. We made sure the evaluation we presented in this paper represents Vanguards-lite fairly and accurately. In addition, they provided very useful feedback on the potential side effects of countermeasures on performance and reliability, as well as their implementation difficulty. For instance, the current design of our token bucket defense has benefited from their feedback that countermeasures that only require client-side changes are easier to implement than those affecting relays. Concurrent to our work, the Tor developers integrated Vanguards-lite [27, 48] into Tor 0.4.7.

8 Conclusions

We present a fast, stealthy, and low-cost attack that reveals a Tor user's guard through a web-based attack. Our attack exploits the fact that webpages can cause victim Tor clients to create an unlimited number of circuits to look up onion service descriptors as well as the distinctiveness of the cell pattern if the lookup is unsuccessful. The attack can be deployed for a few hundred euros per month by a diverse set of adversaries, including third-party script providers, website administrators, and malicious exits (when unencrypted HTTP is used). Through extensive simulations and live-network experiments, we evaluate our attack against Tor users with different geolocations, computer speeds, and network conditions. We find that a 5 % adversary controlling two HSDirs per attack onion address identifies a victim's guard in 12.06 seconds at the median case and under 50 seconds for 99 % of the users. The better a victim's network connection and computer, the more effective is our attack. While our attack works best when JavaScript is enabled, we develop a scriptless version that works against Tor users with stricter security settings. We show that a token bucket countermeasure is effective in mitigating our attack and that the use of Vanguards-lite can also reduce its impact.

9 Acknowledgements

We thank George Kadianakis and Mike Perry for valuable discussions, particularly about the countermeasures. We also thank the anonymous reviewers for their helpful feedback that improved this work. Lennart Oldenburg is funded by a PhD fellowship of the Fund for Scientific Research - Flanders (FWO). Gunes Acar was supported by a FWO postdoctoral fellowship, and was affiliated with imec-COSIC KU Leuven during the

study. This research is partially supported by the Research Council KU Leuven under the grant C24/18/049, by CyberSecurity Research Flanders with reference number VR20192203, and by DARPA FA8750-19-C-0502. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of any of the funders.

References

[1] T. G. Abbott, K. J. Lai, M. R. Lieberman, and E. C. Price. Browser-Based Attacks on Tor. In *International Workshop on Privacy Enhancing Technologies*, pages 184–199. Springer, 2007.

[2] G. Acar. Tor can be forced to open too many circuits by embedding .onion resources (#20212) · Issues · The Tor Project / Core / Tor, 9 2016. URL https://gitlab.torproject.org/tpo/core/tor/-/issues/20212.

[3] G. Acar, M. Juarez, and individual contributors. GitHub: tor-browser-selenium - Tor Browser automation with Selenium, 2020. URL https://github.com/webfp/tor-browser-selenium. [Online, accessed 2021/09/14].

[4] A. Biryukov, I. Pustogarov, and R.-P. Weinmann. Trawling for Tor Hidden Services: Detection, Measurement, Deanonymization. In *2013 IEEE Symposium on Security and Privacy*. IEEE, 2013.

[5] A. Biryukov, I. Pustogarov, F. Thill, and R.-P. Weinmann. Content and Popularity Analysis of Tor Hidden Services. In *2014 IEEE 34th International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE, 2014.

[6] Boing Boing. What happened when we got subpoenaed over our Tor exit node, 8 2015. URL https://boingboing.net/2015/08/04/what-happened-when-the-fbi-sub.html. [Online, accessed 2021/09/14].

[7] N. Borisov, G. Danezis, P. Mittal, and P. Tabriz. Denial of Service or Denial of Security? In *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 2007.

[8] R. Dingledine. The lifecycle of a new relay, 9 2013. URL https://blog.torproject.org/lifecycle-new-relay. [Online, accessed 2021/09/14].

[9] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *13th USENIX Security Symposium*. USENIX Association, 2004.

[10] R. Dingledine, N. Hopper, G. Kadianakis, and N. Mathewson. One Fast Guard for Life (or 9 months). In *7th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs 2014)*, 2014.

[11] T. Durden. [tor-relays] Subpoena received, 4 2015. URL https://lists.torproject.org/pipermail/tor-relays/2015-April/006804.html. [Online, accessed 2021/09/14].

[12] T. Elahi, K. Bauer, M. AlSabah, R. Dingledine, and I. Goldberg. Changing of the Guards: A Framework for Understanding and Improving Entry Guard Selection in Tor. In *Proceedings of the 2012 ACM Workshop on Privacy in the Electronic Society*. ACM, 2012.

[13] N. S. Evans, R. Dingledine, and C. Grothoff. A Practical Congestion Attack on Tor Using Long Paths. In *USENIX Security Symposium*, 2009.

[14] Federal Communications Commission, Office of Engineering and Technology. Tenth Measuring Broadband America Fixed Broadband Report, 1 2021. URL https://www.fcc.gov/reports-research/reports/measuring-broadband-america/measuring-fixed-broadband-tenth-report. [Online, accessed 2021/09/14].

[15] D. Goulet. Onion Service version 2 deprecation timeline, 7 2020. URL https://blog.torproject.org/v2-deprecation-timeline. [Online, accessed 2021/09/14].

[16] D. Goulet. [tor-dev] Onion Service v2 Deprecation Timeline, 6 2020. URL https://lists.torproject.org/pipermail/tor-dev/2020-June/014365.html. [Online, accessed 2021/09/14].

[17] Hetzner Online GmbH. Hetzner Cloud: Pricing, n.d. URL https://www.hetzner.com/cloud?country=gb#pricing. [Online, accessed 2021/09/14].

[18] N. Hopper, E. Y. Vasserman, and E. Chan-Tin. How Much Anonymity Does Network Latency Leak? *ACM Transactions on Information and System Security (TISSEC)*, 13(2), 2010.

[19] isabela. Tor security advisory: exit relays running sslstrip in May and June 2020 | Tor Blog, 8 2020. URL https://blog.torproject.org/bad-exit-relays-may-june-2020. [Online, accessed 2021/09/14].

[20] R. Jansen and N. Hopper. Shadow: Running Tor in a Box for Accurate and Efficient Experimentation. In *Network and Distributed Systems Security (NDSS) Symposium 2012*. Internet Society, 2 2012.

[21] R. Jansen, F. Tschorsch, A. Johnson, and B. Scheuermann. The Sniper Attack: Anonymously Deanonymizing and Disabling the Tor Network. In *Network and Distributed Systems Security (NDSS) Symposium 2014*. Internet Society, 2 2014.

[22] R. Jansen, M. Juarez, R. Galvez, T. Elahi, and C. Diaz. Inside Job: Applying Traffic Analysis to Measure Tor from Within. In *Network and Distributed Systems Security (NDSS) Symposium 2018*. Internet Society, 2018.

[23] R. Jansen, T. Vaidya, and M. Sherr. Point Break: A Study of Bandwidth Denial-of-Service Attacks against Tor. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*. USENIX Association, 2019.

[24] R. Jansen, J. Tracey, and I. Goldberg. Once is Never Enough: Foundations for Sound Statistical Inference in Tor Network Experimentation. In *30th USENIX Security Symposium*. USENIX Association, 2021.

[25] A. Johnson, C. Wacek, R. Jansen, M. Sherr, and P. Syverson. Users Get Routed: Traffic Correlation on Tor by Realistic Adversaries. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013.

[26] D. Johnson. Stem: a Python controller library for Tor, n.d. URL https://stem.torproject.org/. [Online, accessed 2021/09/14].

[27] G. Kadianakis. [tor-dev] [RFC] Proposal 332: Vanguards lite, 6 2021. URL https://lists.torproject.org/pipermail/tor-dev/2021-June/014569.html. [Online, accessed 2021/09/14].

[28] G. Kadianakis and M. Perry. Mesh-based vanguards, 05 2018. URL https://gitweb.torproject.org/torspec.git/tree/proposals/292-mesh-vanguards.txt. [Online, accessed

2021/09/14].

[29] A. Kwon, M. AlSabah, D. Lazar, M. Dacier, and S. Devadas. Circuit Fingerprinting Attacks: Passive Deanonymization of Tor Hidden Services. In *24th USENIX Security Symposium*, pages 287–302. USENIX Association, 8 2015.

[30] I. Lovecruft, G. Kadianakis, O. Bini, and N. Mathewson. Tor Guard Specification, n.d. URL https://gitweb.torproject.org/torspec.git/tree/guard-spec.txt. [Online, accessed 2021/09/14].

[31] A. Mani, T. Wilson-Brown, R. Jansen, A. Johnson, and M. Sherr. Understanding Tor Usage with Privacy-Preserving Measurement. In *Proceedings of the Internet Measurement Conference 2018*. ACM, 2018.

[32] P. Mittal, A. Khurshid, J. Juen, M. Caesar, and N. Borisov. Stealthy Traffic Analysis of Low-Latency Anonymous Communication Using Throughput Fingerprinting. In *Proceedings of the 18th ACM conference on Computer and Communications Security*. ACM, 2011.

[33] Mozilla and individual contributors. Network Monitor - Firefox Developer Tools | MDN, 2 2021. URL https://developer.mozilla.org/en-US/docs/Tools/Network_Monitor. [Online, accessed 2021/09/14].

[34] Mozilla and individual contributors. <noscript> - HTML: HyperText Markup Language | MDN, 2 2021. URL https://developer.mozilla.org/en-US/docs/Web/HTML/Element/noscript. [Online, accessed 2021/09/14].

[35] A. Muffett. GitHub: Real-World Onion Sites, n.d. URL https://github.com/alecmuffett/real-world-onion-sites. [Online, accessed 2021/09/14].

[36] M. Nasr, A. Bahramali, and A. Houmansadr. DeepCorr: Strong Flow Correlation Attacks on Tor Using Deep Learning. In *CCS '18*. ACM, 2018.

[37] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle. Website Fingerprinting at Internet Scale. In *Network and Distributed Systems Security (NDSS) Symposium 2016*. Internet Society, 2016.

[38] M. Perry. The move to two guard nodes, 3 2018. URL https://gitweb.torproject.org/torspec.git/tree/proposals/291-two-guard-nodes.txt. [Online, accessed 2021/09/14].

[39] M. Perry, E. Clark, S. Murdoch, and G. Koppen. The Design and Implementation of the Tor Browser [DRAFT], 6 2018. URL https://www.torproject.org/projects/torbrowser/design/. [Online, accessed 2021/09/14].

[40] M. Perry et al. GitHub: The Vanguards Onion Service Addon, n.d. URL https://github.com/mikeperry-tor/vanguards. [Online, accessed 2021/09/14].

[41] F. Rochet and O. Pereira. Dropping on the Edge: Flexibility and Traffic Confirmation in Onion Routing Protocols. *Proceedings on Privacy Enhancing Technologies*, 2018(2), 2018.

[42] Shadow Team. GitHub: TGen, n.d. URL https://github.com/shadow/tgen. [Online, accessed 2021/09/14].

[43] Shadow Team. GitHub: tornettools, n.d. URL https://github.com/shadow/tornettools. [Online, accessed 2021/09/14].

[44] P. Sirinam, M. Imani, M. Juarez, and M. Wright. Deep Fingerprinting: Undermining Website Fingerprinting Defenses with Deep Learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018.

[45] P. Tellis. Analyzing Network Characteristics Using JavaScript And The DOM, Part 1, 11 2011. URL https://www.smashingmagazine.com/2011/11/analyzing-network-characteristics-using-javascript-and-the-dom-part-1. [Online, accessed 2021/09/14].

[46] The Tor Project. Make it even harder to become HSDir (#19162) · Issues · The Tor Project / Core / Tor, 5 2016. URL https://gitlab.torproject.org/tpo/core/tor/-/issues/19162. [Online, accessed 2021/09/14].

[47] The Tor Project. #9063 enables Guard discovery in about an hour by websites (#9072) · Issues · Legacy / Trac, 2 2021. URL https://gitlab.torproject.org/legacy/trac/-/issues/9072. [Online, accessed 2021/09/14].

[48] The Tor Project. Build vanguards lite into little-t-tor (#40363) · Issues · The Tor Project / Core / Tor, 4 2021. URL https://gitlab.torproject.org/tpo/core/tor/-/issues/40363. [Online, accessed 2021/09/14].

[49] The Tor Project. Introduce vanguards-lite (!408) · Merge requests · The Tor Project / Core / Tor, 7 2021. URL https://gitlab.torproject.org/tpo/core/tor/-/merge_requests/408/diffs#fb72b9489ffeb300a7d2e454d0407f5947ecfdc4_3933_4144. [Online, accessed 2021/09/14].

[50] The Tor Project. Onion Services | Tor Project | Support, n.d. URL https://support.torproject.org/onionservices/#onionservices-5. [Online, accessed 2021/09/14].

[51] The Tor Project. Tor directory protocol, version 3, n.d. URL https://gitweb.torproject.org/torspec.git/tree/dir-spec.txt. [Online, accessed 2021/09/14].

[52] The Tor Project. Tor Metrics: Traffic, n.d. URL https://metrics.torproject.org/bandwidth-flags.html. [Online, accessed 2021/09/14].

[53] The Tor Project. Relay requirements, n.d. URL https://community.torproject.org/relay/relays-requirements/. [Online, accessed 2021/09/14].

[54] The Tor Project. Tor Rendezvous Specification, n.d. URL https://gitweb.torproject.org/torspec.git/tree/rend-spec-v2.txt. [Online, accessed 2021/09/14].

[55] The Tor Project. Tor Rendezvous Specification - Version 3, n.d. URL https://gitweb.torproject.org/torspec.git/tree/rend-spec-v3.txt. [Online, accessed 2021/09/14].

[56] torservers.net. Coordinated raids of Zwiebelfreunde at various locations in Germany, 7 2018. URL https://blog.torservers.net/20180704/coordinated-raids-of-zwiebelfreunde-at-various-locations-in-germany.html. [Online, accessed 2021/09/14].

[57] T. Wang and I. Goldberg. On Realistically Attacking Tor with Website Fingerprinting. *Proceedings on Privacy Enhancing Technologies*, 2016(4), 2016.

[58] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg. Effective Attacks and Provable Defenses for Website Fingerprinting. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*. USENIX Association, 2014.

[59] P. Winter, R. Köwer, M. Mulazzani, M. Huber, S. Schrittwieser, S. Lindskog, and E. Weippl. Spoiled Onions: Exposing Malicious Tor Exit Relays. In *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, 2014.

[60] M. Wright, M. Adler, B. N. Levine, and C. Shields. Defending Anonymous Communications Against Passive Logging Attacks. In *2003 Symposium on Security and Privacy*, pages

28–41. IEEE, 2003.

[61] M. K. Wright, M. Adler, B. N. Levine, and C. Shields. The Predecessor Attack: An Analysis of a Threat to Anonymous Communications Systems. *ACM Transactions on Information and System Security (TISSEC)*, 7(4):489–522, 2004.

A Adversarial Cell Pattern Characteristics with Shadow

For our cell pattern investigations in Section 3.3, we conduct two experiments that analyze the target "404 Not Found" cell pattern using Shadow [20] and TorNetTools [24, 43]. In all experiments, we run a tor binary patched at release `0.4.4.5` to export metadata about routed cells. We log a timestamp, the cell's channel-circuit assignment and purpose of the channel, whether cells are incoming or outgoing of the relay, their direction (away from or towards the circuit origin), and their command (`CREATE`, `CREATED`, `RELAY`, and `DESTROY` cells). For cells with command `RELAY`, we additionally extract the relayed command, if visible to the node.

A.1 Is the Target Cell Pattern Deterministic?

For our first experiment we construct a minimal Tor network in Shadow, consisting of five Tor nodes: a client that connects via a guard, a fixed second-hop relay, and a fixed third-hop relay to an HSDir. The client uses the circuit to request the `HS_DESC` of a non-existing onion address. We use this simple setup to extract the second hop's view on the cell pattern exhibited by the victim lookups. As we are interested primarily in the cell pattern, we configure a low, fixed latency of 10 ms per edge and no packet loss or jitter. We note that this is an abstraction from network conditions in the live Tor network that may require adaptations to the matching routine when deployed on live networks. The experiment simulates one hour of operation, during which the client attempts to connect to the non-existing onion address a fixed number of times (leading to a fixed number of lookups). Among other things, the long simulation time allows us to account for cells that are sent due to inactivity and are still part of the target cell pattern.

Analyzing the cell traces and metadata collected for the second-hop relay in the simulation, we find that the adversarial cell pattern is indeed deterministic. Figure 9 shows how the pattern is viewed at each circuit hop. Specifically, we are interested in the circuit's second-hop position, which is next to a victim's guard (relay "R5" in Figure 9). The pattern always features three parts, possibly completed by a final closure part. First, the telescoping circuit setup iterates from guard "G1" to middle "R5" to middle "R1" and finally "HSDir1". Next, the client opens a directory connection and requests the (non-existing) `HS_DESC`. Third, the HSDir sends back confirmation of directory connection establishment, the "404 Not Found" response, and a cell ending the stream. We do not depict the circuit teardown behavior after a certain period has elapsed.

The target cell pattern being deterministic is advantageous for our attack. The adversary is able to distinguish whether an adversarial middle is positioned at the second or third hop of the victim lookup circuit. This allows to exclude all circuits where an adversarial middle relay has only been chosen into the third circuit position and does not have visibility over the guard. Further, the "404 Not Found" response payload is the same across both onion service versions, v2 and v3, and fits into a single cell in both scenarios. Finally, a deterministic cell pattern removes the need for more elaborate classification techniques such as machine learning.

A.2 Is the Target Cell Pattern Unique?

In order for the attack to work, the `HS_DESC` lookup cell pattern also needs to be unique: in case the cell pattern is otherwise frequently occurring in all sorts of unrelated circuits, its observation likely corresponds to a false positive. We conduct a second experiment using Shadow to investigate whether the target pattern is spuriously observed. In contrast to the preceding experiment, here we generate a larger, diverse, and more realistic network using TorNetTools.

Based on data published by the Tor Metrics team for May 2020, we have TorNetTools produce a Tor test network that is representative of a 2%-scale sampled version of the live Tor network. Note that downscaling does not have a significant impact on the experiment, since it is concerned with cell patterns that appear within a circuit, which are insensitive to total network size. The simulated network includes 133 Tor relays, 16 `tgen` [42] clients instructed to measure download performance, 158 `tgen` clients modelled to emulate actual user behavior, and 16 `tgen` servers. The entities are mapped to locations in a network model that aims to realistically capture geographical distribution and net-

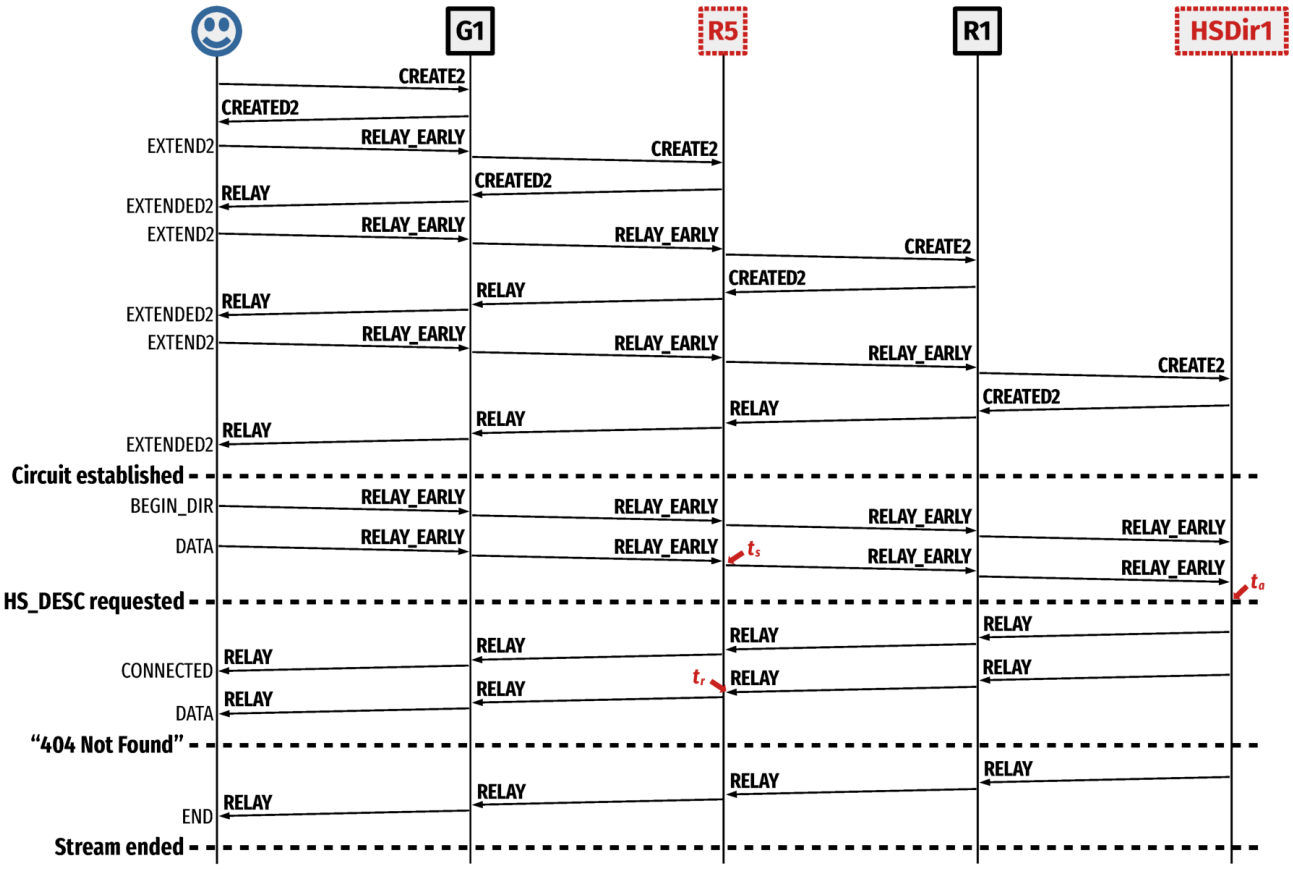

Fig. 9. Target cell pattern of a victim lookup that the adversarial middle relays (e.g., R5) scan for. Timestamps t_s, t_a, and t_r correspond to columns `Sent` (R5), `Access Time` (HSDir1), and `Received` (R5) in the tables in Figure 2, respectively.

work conditions of the Tor network. Thus, upstream and downstream bandwidth as well as latency are adjusted according to the nodes' geographical locations.

In order to have `HS_DESC` lookups (of *existing* addresses) as part of the results, we configure four of the 16 `tgen` servers to operate as onion services, two as v2 and two as v3 onion services. The remaining TorNetTools parameters stay at their defaults (`process_scale = 0.01`, `server_scale = 0.1`, `torperf_scale = 0.001`, and `load_scale = 1.0`). This means that each `tgen` client process emulates 100 actual Tor users. The total simulated time is set to one hour, of which the initial five minutes are dedicated to bootstrapping. Once bootstrapping is completed, `tgen` traffic generation begins and runs until the end of the experiment.

In total, clients construct and use 1,866,782 circuits over the course of the experiment. For each hop on each circuit, we check whether any of the observed cell patterns matches the attack cell pattern, as observed by R5 in Figure 9. We do not find a single occurrence of the adversarial pattern. Based on our observations from these two simulations using Shadow and TorNetTools, we find the cell pattern of an `HS_DESC` lookup with a "404 Not Found" response to be deterministic and unique, and thus exploitable as part of our attack.

B Establishing the Most Effective Set of Attack Parameters

Regarding our experiments in Section 4.2 that determine the most effective attack parameters for the adversary on the live Tor network, we detail how injection rates and geolocation affect victim lookup rates in Figure 10. Each boxplot shows the distribution of the average number of lookups per second over 60 seconds of attack, per victim setting and configured injection rate. We observe that geographical distance between the victim and the region where most Tor relays are located (Europe) appears to have an effect on lookup rates. Lookup rates for victims in Frankfurt are highest

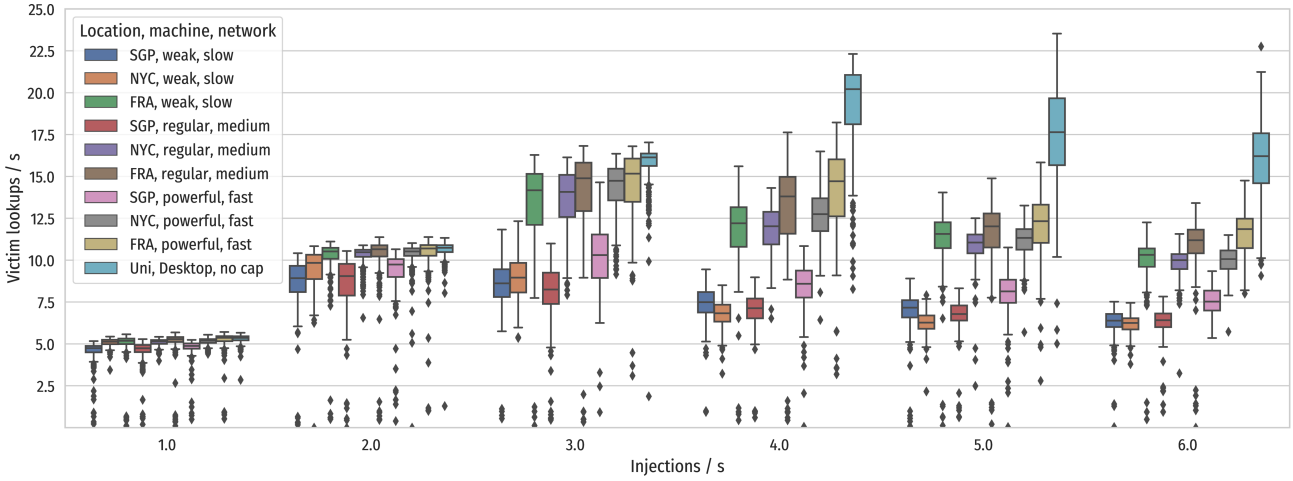

Fig. 10. Victim lookups per second across different rates of injected non-existing resources and victim settings. Experiment and victim settings described in Section 4.2. The four geographical locations represent: Singapore (SGP), New York City (NYC), Frankfurt (FRA), and our university (Uni).

in each machine-network class, followed by victims in New York City, and finally victims in Singapore (exceptions are the weakest and slowest clients at four, five, and six injections per second).

At injection rates higher than three per second, results begin to diverge for more and less powerful clients. While more powerful machines generate more lookups with a higher injection rate, less powerful machines become "saturated" and generate fewer lookups than if only three resources per second are injected. Our office Desktop, which we include as an upper limit, reaches a peak rate of 23.5 lookups per second at five injected resources per second.

C Monetary Cost of the Attack

We give an exemplary cost calculation of our attack to demonstrate its affordability using prices from Hetzner, a low-cost cloud provider [17]. While the adversary's expenses may also include the cost of manipulating the webpage (e.g., as an advertising third party), we focus on HSDir and relay operation costs, as these likely represent the bulk of the budget.

According to Tor Metrics [52], the total advertised relay bandwidth is around 600 Gbit/s. However, Tor users actually consume only about half of it. Thus, the adversary may advertise twice as much bandwidth as officially available with each instance. Hetzner includes 20 TB traffic per month per instance, thus we advertise $2 \cdot \frac{160{,}000}{30 \cdot 24 \cdot 60 \cdot 60} \approx 0.123$ Gbit/s per middle relay. To cumulatively offer 5 % of Tor's relay bandwidth, the adversary needs to advertise $\frac{1}{19} \cdot 600 \approx 31.58$ Gbit/s and thus requires $\frac{31.58}{0.123} < 257$ instances. For 2 % cumulative relay bandwidth, she needs to run $\frac{12.25}{0.123} < 100$ instances, while she only requires $\frac{6.06}{0.123} < 50$ instances for 1 %.

Based on the Tor Project's information on relay requirements [53], Hetzner's `CX21` instances (2 vCPUs, 4 GB RAM, 5.88 €/month) appear adequate. Assuming ten HSDirs, the adversary needs to spend EUR 1,579.56 (5 %), EUR 656.40 (2 %), and EUR 362.40 (1 %) for one month of attack. We deem these costs to be affordable even for adversaries with limited financial resources.

D Attack Key Precomputation

Our attack key precomputation script works as follows. We load the current shared random value from a Tor consensus file and calculate a time period involving the `valid-after` field. We load the list of identity public keys of all relays with the `HSDir` flag in the server descriptors document downloaded with the consensus file and build the sorted list of each relay's hash of key information and public parameters (i.e., the hashring). For each number of adversarial HSDirs in { 10, 15, 20, 25 }, we sample this many unique HSDirs uniformly at random from the list of all HSDirs as the current set of adversarial HSDirs. Now we take the time it takes to generate fresh onion service key pairs, blind them according to v3 specification [55], calculate their hashring indices and respective sets of responsible HSDirs, and

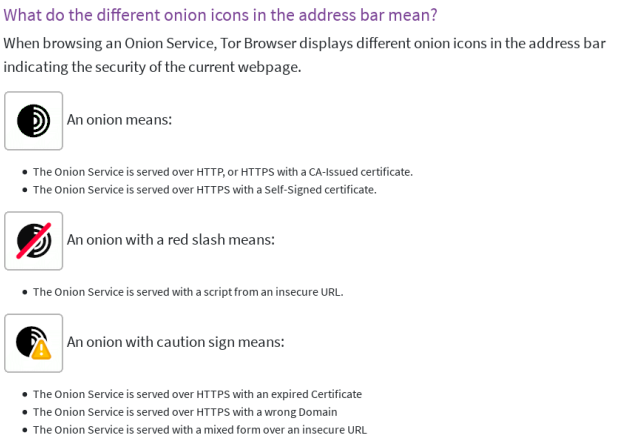

Fig. 11. The existing set of onion icons used by Tor Browser to convey the security of an onion website.

select the ones that map to at least two of the sampled adversarial HSDirs until the list of selected attack onion service keys has size 900. Reaching intermediate sizes of multiples of 180 is marked as dedicated timestamps.

E Onion Security Icons

We show the existing set of onion icons used by Tor Browser in Figure 11.