TorPolice: Towards Enforcing Service-Defined Access Policies for Anonymous Communication in the Tor Network

Zhuotao Liu*, Yushan Liu†, Philipp Winter†, Prateek Mittal†, Yih-Chun Hu*
*University of Illinois at Urbana-Champaign, †Princeton University
*{zliu48, yihchun}@illinois.edu, †{yushan, pwinter, pmittal}@princeton.edu

***Abstract*—Tor is the most widely used anonymity network, currently serving millions of users each day. However, there is no access control in place for all these users, leaving the network vulnerable to botnet abuse and attacks. For example, criminals frequently use exit relays as stepping stones for attacks, causing service providers to serve CAPTCHAs to exit relay IP addresses or blacklisting them altogether, which leads to severe usability issues for legitimate Tor users. To address this problem, we propose TorPolice, the first privacy-preserving access control framework for Tor. TorPolice enables abuse-plagued service providers such as Yelp to enforce access rules to police and throttle malicious requests coming from Tor while still providing service to legitimate Tor users. Further, TorPolice equips Tor with global access control for relays, enhancing Tor's resilience to botnet abuse. We show that TorPolice preserves the privacy of Tor users, implement a prototype of TorPolice, and perform extensive evaluations to validate our design goals.**

I. Introduction

Counting almost two million daily users, the Tor network is among the most popular digital privacy tools. As of May 2017, the network consists of over 7,000 volunteer-run relays, carrying over 100 Gbps of traffic [2]. Tor clients build a path (also known as Tor circuit) consisting of three relays (guard, middle and exit) to reach service providers such as Yelp or WikiLeaks. Tor is used by law enforcement, intelligence agencies, political dissidents, journalists, whistle-blowers, and ordinary citizens to enhance their online privacy [3].

Today's Tor network does not implement any access control mechanism, meaning that anyone with a Tor client can use the network without limitation. While the lack of access control fosters network growth, it has also caused various problems, most importantly botnet abuse [15]. In practice, botnets use Tor to attack third-party services, spam comment sections on websites, scrape content, and scan services for vulnerabilities [27]. In response, many service providers and content delivery networks (CDNs) have started to treat Tor users as "second-class" Web citizens [19] by either forcing Tor users to solve numerous CAPTCHAs or blocking Tor exit relay IP addresses altogether.

Another type of botnet-related abuse of Tor arises from command and control (C&C) servers run as Tor hidden services [12, 15]. In the past, such events caused a rapid spike in the number of Tor clients [15]. Besides the reputational issue of Tor "hosting" botnet infrastructure, the massive number of circuit creation requests from botnets is a heavy burden on Tor relays, causing significant performance degradation for legitimate Tor users (*e.g.,* frequent Tor circuit failures). Other types of botnet abuse include paralyzing Tor relays via relay flooding attacks [4, 5] and performing large-scale traffic analysis via throughput or congestion fingerprinting [23, 25].

978-1-5090-6501-1/17/$31.00 2017 IEEE

Contributions. In this paper, we present TorPolice, the first privacy-preserving access control framework for the Tor network. Leveraging cryptographically computed *network capabilities*, TorPolice enables service providers to define access policies for Tor connections, allowing them to throttle Tor-emitted abuse while still serving legitimate Tor users. Thus, TorPolice offers a more viable alternative to abuse-plagued service providers than simply blocking all Tor connections. Further, TorPolice improves the Tor network's resilience to various botnet abuses by enabling global access control for Tor relays. Crucially, TorPolice achieves these benefits while still retaining Tor's anonymity guarantees.

TorPolice's design introduces a set of fully distributed and partially trusted *access authorities* (AAs) to manage and certify capabilities. To request capabilities from AAs, Tor clients must first obtain anonymous capability seeds which are types of resources that are costly to scale. Both service providers and the Tor network provide differentiated service to Tor clients that possess valid capabilities so to enforce self-defined access rules. The AAs generate capabilities using blind signatures [6] to break the linkability between capability requesting and capability spending. We conduct a rigorous security analysis to prove that TorPolice does not weaken privacy guarantees offered by the current Tor network.

We implement a prototype of TorPolice to demonstrate its practicality and evaluate the prototype extensively on our testbed, in the Shadow simulator [17] and via simulations. Our results show that TorPolice can effectively enforce service-selected access policies and mitigate large-scale botnet abuses against Tor at the cost of negligible overhead.

II. Problem Formulation

A. Tor Background

Tor clients anonymously connect to service providers (*e.g.,* WikiLeaks) by building three-hop circuits consisting of a guard, middle, and exit relay. Tor's use of layered encryption ensures that each relay only knows the identities of its direct neighbors (*i.e.,* the previous and next hop in the circuit). Clients randomly select these relays, weighted by the relays' bandwidth and their positions on the circuit. A list of all

Tor relays—the network consensus—is published hourly by a set of nine globally-distributed directory authorities that are run by volunteers trusted by the Tor Project. While the directive authorities and guard relays learn a Tor client's network identity (*i.e.,* her IP address), they cannot observe the client's online activity. Exit relays, however, can monitor the client's activity, but do not know her identity. Tor's anonymity stems from unlinking network identity from activity.

Besides client-side anonymity, Tor allows servers to host their service anonymously over Tor hidden services (HS). Once a HS is set up, it creates circuits to at least three relays serving as its *introduction points* (IPs). Then, the HS publishes its *descriptor*—which contains the IPs—to a distributed hash table that consists of a subset of all Tor relays. To connect to the HS, a Tor client first fetches the HS's descriptor using its *onion address*, and then builds two circuits: one to an IP and another one to a randomly-selected relay called the *rendezvous point* (RP). The client instructs the IP to send the identity of the RP to the HS, which then creates a circuit to the RP to be able to finally communicate with the client.

B. Design Goals

TorPolice adds access control to the anonymous communication in Tor, benefiting both service providers and the Tor network. Different from prior capability based schemes [20, 21, 26, 31], TorPolice's design needs to address a unique combination of the following three challenges: (i) preserving Tor's anonymity guarantees, (ii) avoiding central points of control, and (iii) being incrementally deployable.

Service-defined Access Policies. Project Honey Pot lists nearly 70% of all Tor exit relays as comment spammers [27], causing many service providers and CDNs to block and filter traffic originating from the Tor network. To reduce this tension between Tor users and service providers, TorPolice must allow service providers to define and enforce access rules for Tor connections, allowing them to effectively throttle Tor-emitted abuse while still serving legitimate Tor users.

Mitigate Botnet Abuse Against Tor. Being a service provider itself, the Tor network is also subject to botnet abuse, such C&C servers hosted as hidden services and (D)DoS attacks against (selected) relays. TorPolice allows the Tor network to control the network usage of Tor clients, making it possible to throttle the abuse. In contrast to local rate limiting by each relay, TorPolice's access control mechanism is *global*, meaning that an adversary cannot circumvent our defense by simply connecting to all Tor relays.

Preserving Tor User Privacy. TorPolice must not degrade Tor's anonymity guarantees. While we add a new layer of functionality to Tor (access control), this layer—like Tor itself—unlinks a client's identity from her activity, and therefore preserves Tor's anonymity.

Fully Distributed and Partially Trusted Authorities. In accordance with Tor's design philosophy of distributing trust, TorPolice relies on a set of fully distributed and partially-trusted access authorities (AAs) to manage capabilities. An AA is operated either by the Tor Project, a service provider, or a trusted third party. Since Tor clients are free to choose any AA to request capabilities using their capability seeds, no single AA has a global view on all Tor clients. Further, each AA is only partially trusted and a service provider can blacklist any misbehaving or compromised AA.

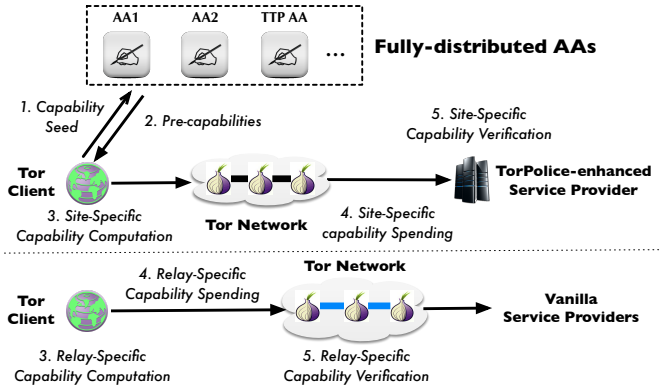

Fig. 1: The architecture of TorPolice. A Tor client (step 1) anonymously sends its capability seed to an randomly selected AA to request pre-capabilities (step 2), based on which the client computes site(relay)-specific capabilities (step 3). The client then spends capabilities on either service access or Tor circuit creation (step 4). The capability recipients validate capabilities before allowing services (step 5).

Incrementally Deployable. TorPolice must be incrementally deployable. Up-to-date Tor clients, relays, and service providers can benefit from a partially-deployed TorPolice immediately while outdated entities can continue their operations.

Elided Design Goals. Various attacks seek to break Tor's unlinkability. For instance, an AS-level adversary may de-anonymize a Tor user's Internet activities if the adversary can monitor both ingress and egress traffic [30]. *TorPolice is not designed to mitigate attacks on unlinkability*. Instead, we preserve the unlinkability currently offered by Tor.

C. Adversary Model and Assumptions

We consider a Byzantine adversary that deviates from our protocol and abuses Tor in arbitrary ways. The adversary can use Tor to abuse third-party services, *e.g.,* by scraping content, spamming comments, and scanning for vulnerabilities. The adversary may also abuse the Tor network directly, *e.g.,* by using Tor HSes as C&C servers, performing traffic analysis, or launching (D)DoS attacks against Tor relays. The adversary may control many bots, and hence a significant amount of resources. The bots can act passively (*e.g.,* monitor Tor traffic) or actively (*e.g.,* spoof and manipulate packets). We assume that the AAs are well-connected to the Internet backbone so that volumetric DDoS attacks against the whole set of AAs can be mitigated. Tor's existing directory authorities are subject to the same assumption. In practice, one way to assure this assumption is relying on DDoS prevention vendors [21].

III. Design Overview

In a nutshell, TorPolice is a generic access control framework based on *capabilities*. TorPolice enables both service providers and the Tor network itself to enforce access control

on Tor clients to mitigate various types of botnet abuse caused by the lack of access control. To this end, we consider two types of capabilities: *site-specific capabilities* for accessing TorPolice-enhanced service providers through Tor, and *relay-specific capabilities* for creating TorPolice-enhanced Tor circuits. Both types of capability are signed by a set of fully-distributed Access Authorities (AAs) that are deployed either by the Tor Project, service providers, or trusted third parties. To request capabilities from a particular AA, a Tor client is required to possess a *capability seed*—basically a costly-to-scale resource—accepted by the AA. Each AA accepts only a single type of capability seed. Since Tor clients are free to choose their AAs, no single AA has a global view on all Tor clients. TorPolice employs blind signatures [6] to unlink the requesting and spending of capabilities. When requesting capabilities from an AA, Tor clients express what kind of capability they request because the issuing process for two capability types differs. An AA maintains separate signing keys and rate limiters for two capability types.

Figure 1 illustrates the capability requesting and spending process. While both capability types have in common step one and two, the subsequent steps differ. A site-specific capability can only be spent at the service provider specified in the capability to request service while a relay-specific capability is spent at a specific Tor relay to build a TorPolice-enhanced circuit through the relay. A Tor client can use both capability types simultaneously by visiting a TorPolice-enhanced site via a TorPolice-enhanced Tor circuit. In Figure 1, we intensionally separate two capability use cases for clear presentation.

IV. The Access Authorities

TorPolice relies on fully distributed and partially trusted access authorities (AAs) to manage capabilities. We assume AAs to be honest-but-curious, meaning that they follow protocol, but seek to derive additional information about Tor clients. An AA can be deployed by the Tor Project, service providers (*e.g.,* large CDNs like Cloudflare), or third parties. Each AA is a conceptually centralized entity, but it can *distribute* its operations among multiple servers to achieve high availability.

A. Capability Seeds

AAs expect valid *capability seeds* from Tor clients to issue *pre-capabilities*, which are the basis for deriving spendable capabilities. For flexibility, we intentionally keep the definition of capability seeds broad: any resource that is readily available to Tor users, but costly to scale, can be adopted as capability seeds. Reasonable choices include proof-of-work schemes (*e.g.,* solutions to CAPTCHAs or computational puzzles) and anonymous monetary resources. *TorPolice does not assume that capability seeds can distinguish bots from humans.* Rather, botnets can still obtain more capability seeds than legitimate Tor users. Instead, TorPolice employs capability seeds as a form of anonymous identities that enable both service providers and the Tor network to control access by each client.

In this paper, we elaborate on two types of capability seeds (*i.e.,* solutions to CAPTCHAs and computational puzzles) and further discuss how TorPolice can incorporate more types of seeds in § IV-D. One key challenge of using anonymous capability seeds is to ensure that clients do not have to solve endless challenges while browsing the web and meanwhile ensure their activities are unlinkable. TorPolice proposes a capability renewal protocol to address this challenge (§ V-A).

Although CAPTCHAs are readily deployable using publicly available libraries like Google's reCAPTCHAs, TorPolice needs additional components to support computational puzzles. At a very high level, TorPolice's puzzle system design is similar to Portcullis [26]. However, TorPolice's puzzle system does make a great improvement over Portcullis: it can explicitly bound the percentage of CPU cycles that any client can spend on solving puzzles. Thus, the puzzle system can bring all bots down to the percentage that normal clients prefer to use for puzzle computation, which significantly reduces the computation disparity between the normal clients and bots. For more details, please refer to our technical report [22].

B. Per-seed Rate Limiting

Each AA accepts only one type of capability seed. The rate at which a seed can request pre-capabilities is limited. In particular, an AA publishes two rate limiters: one determines the maximum rate at which a capability seed can request pre-capabilities used for accessing TorPolice-enhanced service providers and the other one determines the maximum rate at which a seed can request pre-capabilities used for TorPolice-enhanced circuit creation. Based on these per-seed rate limiters published by all AAs, both service providers and Tor can configure rules to fulfill their access policies. This paper presents two concrete examples. In § V-C, we elaborate on a design that enables a site to bound an adversary's achievable service request rate through Tor using self-defined parameters. In § VI-A, we present a design that allows Tor to prevent botnets from creating numerous Tor circuits to conduct various abuses. To improve readability, detailed settings of these rate limiters will be discussed when presenting these access policies.

C. Key Management

Each AA maintains two pairs of keys for signing pre-capabilities, and each of them is dedicated for one capability type. Each AA must publish the public key of both key pairs, for instance, via Tor network consensuses, to ensure other entities (*e.g.,* Tor clients, relays and service providers) can verify the AA's signatures. An AA can periodically renew its keys, but at any time only two key pairs from the AA are valid. After receiving signed pre-capabilities from an AA, Tor clients must verify that the AA uses proper keys before using the pre-capabilities for accessing service providers or Tor. This prevents a malicious AA from using more keys simultaneously to partition the anonymous set. Finally, each AA is associated with a long-term fingerprint to uniquely identity the AA, similar to the fingerprint of a Tor relay.

D. Extending the Access Authorities

Besides Tor, service providers (*e.g.,* Cloudflare or Akamai) also have direct incentives to deploy and control their own

set of access authorities to mitigate Tor-emitted abuses while serving anonymous connections. In fact, Cloudflare is working on an independent implementation of a system whose design goals are similar to our AAs [8], although they focus on addressing the usability issues for Tor users when visiting Cloudflare-powered websites.

Finally, semi-trusted third parties such as social network operators (*e.g.,* Google, Twitter and Facebook), may also run access authorities (shown as *TTP AA* in Figure 1) based on pre-agreed terms. To prevent account information leakage to Tor and service providers, Tor users only authenticate themselves to the social network operators. Service providers or Tor only learn a single bit of information: whether a Tor client has a valid account (*i.e.,* capability seed) or not.

V. TorPolice-Enhanced Site Access

We now elaborate on the capability design for accessing TorPolice-enhanced service providers such as websites. To mitigate the tension between service providers and Tor users, our key observation is that service providers should not treat all connections from one Tor exit relay equally since each exit relay is shared by many Tor users. Instead, accountability should be enforced at the granularity of Tor clients so that each service provider can throttle malicious Tor clients without blocking legitimate Tor users. To this end, TorPolice designs site-specific capabilities that allows a service provider to enforce self-selected access rules on anonymous Tor connections.

A. Pre-capability Design

Before visiting a TorPolice-enhanced site, a Tor client must first request pre-capabilities from an AA. The client is free to choose any AA based on what capability seed the client prefers to give. To request a pre-capability, the client (i) provides a valid capability seed to its selected AA and (ii) provides blinded information for the AA to compute pre-capabilities. The client can hide its network identity from the AA, for instance, by using Tor.

Capability Seed Validation. Depending on the accepted type of capability seed, an AA performs corresponding seed verification. For instance, if an AA accepts proof-of-work schemes, it needs to verify that solutions to the presented challenge are correct. Further, an AA needs to ensure that the pre-capability request rates by any capability seed does not exceed the two rate limiters discussed in § IV-B. Since each AA maintains separate rate limiters and signing keys for two pre-capability types (*i.e.,* either for TorPolice-enhanced service access or for TorPolice-enhanced Tor circuit creation), Tor clients must specify the pre-capability type in their requests (in this section, it is for accessing service providers). In § V-C, we will explain how a site defines its access policies based on those pre-capability release rate limiters published by all AAs.

Information Required to Compute Pre-capabilities. To request pre-capabilities, the client provides its selected AA the following set of information $\{\mathbb{S}, n, \mathcal{T}_s, \mathcal{F}\}$, where $\mathbb{S}$ is the domain name of site that the client is going to visit, n is a 128-bit cryptographic nonce generated by the client, $\mathcal{T}_s$ is a universally agreed timestamp to indicate the freshness of the information and $\mathcal{F}$ is fingerprint of the selected AA. All information is blinded [6] by the client to avoid information leakage to the selected AA.

The set of information is designed to prevent abuse. In particular, $\mathbb{S}$ is used to make the capability site-specific to prevent capability double-spending at different sites. The nonce n is added to ensure the uniqueness of each pre-capability, which in turn ensures the uniqueness of each capability. The $\mathcal{T}_s$ indicates the freshness of pre-capabilities so that expired ones are nullified automatically. The client is required to use Tor's daily generated fresh random number [13] as $\mathcal{T}_s$ such that at any time all valid capabilities have the exactly same timestamp. This design eliminates the possibility of information leakage cased by timestamp abuse. $\mathcal{F}$ is added to allow other entities (*i.e.,* clients, Tor relays and service providers) to use correct public keys to verify signatures.

Computation. Upon validation of the client's pre-capability request, the AA computes pre-capabilities using the blinded information provided by the client. Pre-capabilities computed by an AA $\mathcal{A}_i$ are denoted by $\mathcal{P}_{\mathcal{A}_i}$. Then we have

$$\mathcal{P}_{\mathcal{A}_i} = \{\mathbb{S} \mid n \mid \mathcal{T}_s \mid \mathcal{F}_{\mathcal{A}_i}\}^b \mid \mathcal{S}^b_{\mathcal{A}_i}, \tag{1}$$

where $\mathcal{F}_{\mathcal{A}_i}$ is $\mathcal{A}_i$'s fingerprint, $\mathcal{S}^b_{\mathcal{A}_i}$ is $\mathcal{A}_i$'s blind signature over the set of blinded information $\{\mathbb{S} \mid n \mid \mathcal{T}_s \mid \mathcal{F}_{\mathcal{A}_i}\}^b$, and $\mid$ represents concatenation throughout the paper.

Pre-capability Renewal. One key design challenge for pre-capabilities is to ensure that Tor clients do not have to repeatedly solve challenges when browsing the web. A strawman design is that an AA can issue many (*i.e.,* a few hundred) pre-capabilities for each solved challenge. However, this strawman design has at least two shortcomings: (i) it breaks the site-specific pre-capability design since the client may not be able to forecast the sites that it is going to visit so as to provide these blinded information immediately after solving challenges; (ii) the design makes it easier for automated bots to accumulate pre-capabilities, weakening the entire system.

To combat these problems, we propose a pre-capability renewal protocol. In particular, when a client first presents its challenge solution (*i.e.,* capability seed) to an AA, the AA issues the client a unforgeable *pseudonym* $\mathcal{I} = \{r \mid \phi\}$ where r is a random 128-bit nonce and ϕ is the AA's signature over r. Later on, the client presents $\mathcal{I}$ as a proof of validation when requesting new pre-capabilities from the AA, allowing the client to bypass future challenges. With this design, not only the site-specific pre-capability design holds, but also the AA can account each pre-capability request on a specific solved challenge (*i.e.,* capability seed) to enforce the per-seed rate limiting described in § IV-B. Each pseudonym has a validation period determined by the AA. Clients with expired pseudonyms are required to solve new challenges to obtain new pseudonyms that are unlinkable to previous ones.

Impact of the Pseudonym on Anonymity. Different from prior pseudonym-based anonymous blacklisting systems [7], in which a user interacts with a service provider using a

persistent pseudonym, the pseudonym in our pre-capability renewal protocol is transient and never presented to service providers and Tor relays. The pseudonym in our protocol is only linked with a specific challenge solution served as an anonymous capability seed. Since a Tor client anonymously sends its pseudonym to an AA using Tor, the AA cannot link the pseudonym with the client. Further, since all site-related information sent to the AA is blinded, the AA cannot link the pseudonym with any site access as well. Thus, using pseudonym in our protocol has no impact on user anonymity.

B. Site-Specific Capability Design

After obtaining $\mathcal{P}_{\mathcal{A}_i}$, the Tor client *unblinds* the signature using its secret blind factor to produce the unblinded version of the pre-capability, which is the capability spendable at a specific site. In particular, $\mathcal{C} = \mathbb{S} \mid n \mid \mathcal{T}_s \mid \mathcal{F}_{\mathcal{A}_i} \mid \mathcal{S}_{\mathcal{A}_i}$. The capability $\mathcal{C}$ contains a set of unblinded information that allows the site $\mathbb{S}$ to perform capability verification when the client presents $\mathcal{C}$ to access the site, as detailed in § V-C.

Employing blind signature is the key to ensure that TorPolice preserves Tor's privacy guarantee. First, signatures from the AAs prevent unauthorized entities from issuing capabilities. Second, using blind signature avoids disclosing any site-related information to the AAs since the blinded information sent to the AAs is unlinkable with the "plain" information produced by the client. Such unlinkability further ensures the unlinkability between the client and its capability spending even if the AAs could collude with the site, which preserves Tor users' privacy. We provide a formal security proof in § VII.

C. Site-Specific Capability Spending

Capability Validation. Tor clients spend site-specific capabilities at TorPolice-enhanced sites to request services. Upon receiving capabilities, a TorPolice-enhanced site first validates them before subsequent processing. A site-specific capability is valid if (i) it encloses an authentic signature from an AA; (ii) it encloses a domain name that is consistent with the site; (iii) the capability is not expired (*i.e.,* $\mathcal{T}_s$ is the fresh random number released by Tor); and (iv) the capability is not nullified by the site. If any of these conditions does not hold, the site rejects this capability to deny access. If a CDN provider (*e.g.,* Cloudflare) processes capabilities on behalf of its powered sites, the second rule is passed as long as the enclosed domain is owned by one of the CDN provider's customers. In the fourth rule, whether a capability is nullified or not is decided by the site's access policies, as detailed below.

Site-Defined Access Policies. Once a site-specific capability is validated, the site accepts the Tor client's service request. Since the major form of Tor abuse is that automated bots use Tor to conduct various malicious activities against the site [27] (*e.g.,* content scraping, vulnerability scanning, comment spamming and so forth), the site needs to further control the number of service requests (*e.g.,* HTTP requests) allowed by each capability. We clarify that each site can have its own definition of service requests. Once a Tor client's service request count exceeds a threshold, the site nullifies the current capability and requires a new site-specific capability for subsequent service requests. Recall that the pre-capability request rate by each client is limited by the AAs through the per-seed rate limiting design in § IV-B. Thus, together with these rate limiters, it is possible for the site to design access policies so as to bound a strategic adversary's service request rate using self-selected parameters, as detailed below.

Policy Definition. Assume the following set of access authorities $\{\mathcal{A}_0, \mathcal{A}_1, ..., \mathcal{A}_n\}$ are deployed, and each authority accepts one type of capability seed. In this context, the site defines its access policy as $\{w_0, w_1, ..., w_n\}$ where w_i is the number of service requests allowed by one valid site-specific capability issued by the access authority $\mathcal{A}_i$.

We now formulate $\{w_0, w_1, ..., w_n\}$ mathematically. We denote the set of capability seeds by $\{s_0, s_1, ..., s_n\}$ and authority $\mathcal{A}_j$ accepts seed s_j. Let c_j denote the cost of obtaining a capability seed s_j. We denote the cost of obtaining a network identity (*i.e.,* IP address) by λ. Let r_j denote the maximum rate at which a seed s_j can request pre-capabilities (for accessing service providers) from authority $\mathcal{A}_j$. Assume that for any client connecting to the site directly without using Tor, the site allows a maximum service request rate $\widetilde{\mathcal{O}}$ before either blocking the client or forcing the client to solve challenges. Then to bound a strategic adversary's service request rate by using Tor, the site derives $\{w_0, w_1, ..., w_n\}$ to ensure that the following condition is satisfied for any set of parameters $[\alpha_0, \alpha_1, ..., \alpha_n]$ where $\alpha_i \in [0, 1]$ and $\sum_{i=0}^{n} \alpha_i = 1$.

$$\sum_{i=0}^{n} \frac{\alpha_i \cdot \lambda}{c_i} \cdot r_i \cdot w_i \leq \epsilon \cdot \widetilde{\mathcal{O}}, \tag{2}$$

where ϵ is a site-defined parameter.

Policy Correctness. The parameters $[\alpha_0, \alpha_1, ..., \alpha_n]$ represent the adversary's strategy of purchasing various types of capability seeds. Thus, if formula (2) holds for any strategy, the site can guarantee that the maximum Tor-emitted service request rate achieved by an adversary when spending λ on purchasing capability seeds is no greater than $\epsilon \cdot \widetilde{\mathcal{O}}$. Thus, if an adversary that spends a certain amount of resources on obtaining network identities can access the site with rate $\mathcal{O}$ without using Tor, then the maximum rate that the adversary can request service from the site by using Tor is no greater than $\epsilon \cdot \mathcal{O}$, given that the adversary spends the same amount of resources on acquiring capability seeds. Equivalently, in order to achieve the same service request rate, the adversary has to spend $1/\epsilon$ times as many resources when launching attacks through Tor as it spends when launching attacks natively without using Tor. To ensure that formula (2) holds for any attacker strategy, we choose

$$w_i \leq \epsilon \cdot \frac{c_i \cdot \widetilde{\mathcal{O}}}{\lambda \cdot r_i}, \ \forall i \in [0, n] \tag{3}$$

Policy Enforcement. If $w_i = 1$, then each capability is usable for exactly one service request. The site can enforce this by suppressing service requests with duplicate capabilities, for example, through the use of a Bloom filter. If $w_i > 1$, then

statistically more than one service request should be allowed for each capability. To enforce this, the site stops accepting a capability with probability $1/w_i$, and then adds the capability to the duplicate suppressor. However, multiple service requests carrying the same capability can trivially be linked by the site. We discuss how to address this issue through system parameterization below. Finally, if $w_i < 1$, then each capability is accepted with probability w_i, and exactly one service request is allowed for each accepted capability.

Policy Parameterization. We now discuss the parameterization of w_i. First, to compute w_i, the site does not need to exactly know c_i. Instead, the site simply assigns specific weights to all types of capability seeds based on its policies. Further, with an ideal parameterization, w_i should be exactly one since (i) no capability is spendable on more than one service request to ensure unlinkability and (ii) no additional capabilities are required for a single service request to avoid extra overhead. However, it is difficult to reach the ideal parameterization since r_i is chosen by the authority $\mathcal{A}_i$ that is unaware of the site's configurations ϵ and $\widetilde{\mathcal{O}}$. In addition, configurations may vary among different sites so that an ideal parameterization for one site could be undesirable for others.

To address the above problem, TorPolice sets r_i such that (with high probability) a Tor client can obtain enough capabilities so that it is feasible for the client to present a unique capability for each TCP connection to the site. This ensures that the client can achieve the highest level of unlinkability offered by Tor, *i.e.,* service providers only see TCP connections from Tor exit relays. We clarify that it is the client who determines how to spend its capabilities across TCP connections (as described below). The above parameterization is adopted only to ensure that spending a unique capability for each TCP connection is a feasible strategy for the client. A reasonable setting of r_i can be estimated based on the live Tor measurement in [18], which finds that during a 10-minute interval, each Tor client opens about 24 web streams. In practice, the authority $\mathcal{A}_i$ should enforce r_i over a longer period of time (*e.g.,* few hours) to accommodate usage burst.

Note that when an AA $\mathcal{A}_k$ is deployed by the site itself, system parameterization for $\mathcal{A}_k$ is easier since the site determines the rate limiters for issuing pre-capabilities.

Capability Spending by Tor Clients. Given r_i, some sites may end up with rules $w_i > 1$, *i.e.,* one capability is allowed for multiple service requests. In this case, the site needs to send a response to indicate whether a capability is nullified or not. Tor clients are free to determine their capability spending strategies. For instance, a Tor client can send w_i service requests using the same capability within a single TCP connection (due to HTTP keep-alive), which still ensures the highest level of unlinkability. Or the client may choose to spend one capability across multiple TCP connections to allow trans-TCP linkability. We note that if a Tor client uses the default setting of the Tor Browser, it already allows trans-TCP linkability since the Tor Browser uses session cookies. For a site that has w_i less than 1, it can enforce such policies by accepting one capability with probability w_i and for each accepted capability, the site allows only one service request.

VI. TorPolice-Enhanced Tor Access

In this section, we detail the capability design for creating TorPolice-enhanced Tor circuits. The current Tor network suffers from a variety of botnet abuses such as large scale C&C abuse [12, 15], relay flooding attacks [4, 5] and traffic analysis [23, 25]. These abuses lead to various bad results, including poor system performance for legitimate Tor users, de-anonymization threats and bad reputation for Tor. The root cause of these attacks is that botnets can create an arbitrary number of Tor circuits without any limitation. Enforcing local rate limiting for circuit creation at each relay is unlikely to stop these attacks since a strategic botnet can instruct each bot to enumerate all relays to circumvent the local rate limiting.

With TorPolice, Tor can globally control circuit creations by any client. In particular, when TorPolice is activated, clients are required to possess valid capabilities in order to create TorPolice-enhanced circuits (to be incrementally deployable, circuit creation requests without valid capabilities are de-prioritized in case of congestion). Then by controlling the rate at which a client can obtain capabilities, the Tor network can control the client's circuit creation rate, as described below.

A. Relay-Specific Capability Design

To create a three-hop TorPolice-enhanced circuit, a Tor client $\mathbb{U}$ needs to obtain three capabilities, each of them being specific to a relay on the circuit. The design of relay-specific capabilities is similar to that of site-specific capabilities, except for the following. (i) During pre-capability requesting, the client specifies the proper pre-capability type, *i.e.,* it is for Tor-enhanced circuit creations. Further, to request a pre-capability specific to a relay $\mathbb{R}$, the client encloses the fingerprint of relay $\mathbb{R}$ (rather than any site domain) in the blinded information sent to its selected AA. (ii) Relay-specific capabilities are spendable at TorPolice-enhanced relays (not at any sites) for creating Tor-enhanced circuits through the relays. The relays first validate received capabilities (based on a set of rules similar to those defined in § V-C) before extending circuits.

We clarify that to request pre-capabilities, Tor clients do not have to use TorPolice-enhanced circuits to reach the AAs. Thus, there is no deadlock for bootstrapping TorPolice. Another alternative is pre-installing few relay-specific capabilities on Tor clients so that using TorPolice-enhanced circuits to bootstrap the system is viable.

Policy Definition. Relay-specific capabilities enable Tor to enforce access rules for its relays. In this paper, we propose to use capabilities to control the circuit creation rate by any Tor client so as to mitigate those aforementioned botnet abuses against Tor. In particular, assume the following set of AAs $\{\mathcal{A}_0, \mathcal{A}_1, ..., \mathcal{A}_n\}$ are deployed and authority $\mathcal{A}_i$ accepts a type of capability seed s_i. In this context, Tor defines its access policies as $\{q_0, q_1, ..., q_n\}$, where q_i is the maximum rate at which a capability seed s_i can request pre-capabilities (for creating Tor-enhanced circuits) from authority $\mathcal{A}_i$. Then in order to bound a Tor client's circuit creation rate, $\{q_0, q_1, ..., q_n\}$

should satisfy the following condition for any attacker strategy $[\alpha_0, \alpha_1, ..., \alpha_n]$ where $\alpha_i \in [0, 1]$ and $\sum_{i=0}^{n} \alpha_i = 1$.

$$\sum_{i=0}^{n} \frac{\alpha_i \cdot \lambda \cdot q_i}{3 \cdot c_i} \leq \mathcal{T}, \tag{4}$$

where λ is the cost of getting one network identity, c_i is the cost for obtaining one capability seed s_i and $\mathcal{T}$ is the maximum circuit creation rate allowed for a client, which is a parameter controlled by Tor. We note that the constant 3 appears in above formula since a standard Tor circuit contains 3 relays and each of them consumes a relay-specific capability.

To ensure the correctness of formula (4) for any attacker strategy, we choose $q_i \leq \frac{3 \cdot c_i \cdot \mathcal{T}}{\lambda}, \ \forall i \in [0, n]$.

Parameterization. To compute q_i, the Tor project assigns certain weight to each type of capability seed. Further, Tor can properly configure $\mathcal{T}$ based on the live Tor measurements in [18]. In particular, during an 10-minute interval, PrivCount [18] estimates that a Tor client opens about 4 Tor circuits. Thus, the maximum rate $\mathcal{T}$ at which one Tor client can create circuits should be around 4 per 10 minutes. In practice, each AA should enforce q_i over a longer period of time (*e.g.,* few hours) to accommodate usage bursts and relay churn.

Please refer to our technical report [22] to the *capability exchange protocol* designed for Tor hidden services.

VII. Security Analysis

In this section, we perform formal security analysis for the impact of TorPolice on Tor users' anonymity. Let $\mathbb{N}_T$ denote the set of Tor clients that request pre-capabilities from the AAs, and subsequently present capabilities to access service providers or Tor relays. We first present two useful lemmas.

A. Lemmas

Lemma 1. *Consider any client $\mathbb{U} \in \mathbb{N}_T$. By colluding with each other, both the AAs and a service provider $\mathbb{W}$ gain only negligible advantage over random guessing when trying to link a specific Tor-emitted site access with the client $\mathbb{U}$.*

Proof. We first specify the notations used in the proof. Let $\mathbb{V}$ denote a Tor-emitted site access to $\mathbb{W}$ initiated by the Tor client $\mathbb{U}$. Note that the definition of a site access is decided by $\mathbb{W}$. Let $\mathcal{C}$ denote the service-specific capability that $\mathbb{U}$ sends to $\mathbb{W}$ to support the site access $\mathbb{V}$. Let $\mathcal{P}$ denote the pre-capability used by $\mathbb{U}$ to compute $\mathcal{C}$.

Since the client $\mathbb{U}$ can use Tor to connect to the AAs when requesting the pre-capability $\mathcal{P}$, in the ideal case, $\mathbb{U}$ is unlinkable with $\mathcal{P}$. However, to ensure that our lemma still holds in the worst case when Tor's unlinkability is broken by adversaries, we assume the AA $\tilde{\mathcal{A}}$ that issues $\mathcal{P}$ can link $\mathcal{P}$ with the client $\mathbb{U}$. Thus, the service provider $\mathbb{W}$ and other AAs can have such linkability as well by colluding with $\tilde{\mathcal{A}}$.

Next, we prove the lemma by contradiction. Assume that the AAs and $\mathbb{W}$ can design an algorithm $\mathcal{K}$ that enables the AAs and $\mathbb{W}$ to link the site access $\mathbb{V}$ with the client $\mathbb{U}$. Since the site access $\mathbb{V}$ is linkable with the capability $\mathcal{C}$ (as $\mathcal{C}$ is presented to the site to support the access $\mathbb{V}$) and the client $\mathbb{U}$ is linkable with the pre-capability $\mathcal{P}$ (based on the above worst-case assumption), designing the algorithm $\mathcal{K}$ is equivalent to designing another algorithm $\mathcal{K}'$ that enables the AAs and $\mathbb{W}$ to link the capability $\mathcal{C}$ with the pre-capability $\mathcal{P}$.

In TorPolice's design, $\mathcal{P}$ is the blinded message signed by the AA $\tilde{\mathcal{A}}$ (*i.e.,* the blind-signer), and $\mathcal{C}$ is the unblinded version of $\mathcal{P}$ produced by the client $\mathbb{U}$ using a secret factor unknown to the blind-signer. Thus, the problem of designing $\mathcal{K}'$ to link $\mathcal{P}$ with $\mathcal{C}$ is the same as designing another algorithm $\mathcal{K}''$ that allows a blind-signer to link the blinded message it signs to the unblinded message without knowing the secret factor, which is impossible in a blind signature [6]. This contradiction proves that the hypothetical algorithm $\mathcal{K}$ does not exist, indicating both AAs and $\mathbb{W}$ gain only negligible advantages of linking the site access $\mathbb{V}$ with client $\mathbb{U}$. □

Similarly, we can prove the following lemma.

Lemma 2. *Consider any client $\mathbb{U} \in \mathbb{N}_T$. By colluding with each other, both the AAs and Tor relays gain only negligible advantage over random guessing when trying to link a specific relay access (*i.e., *TorPolice-enhanced circuit creation) with $\mathbb{U}$.*

B. Information Leakage Analysis

Given the above two lemmas, we now analyze the impact of TorPolice on Tor user anonymity. We measure the possible information leakage to an arbitrary service provider $\mathbb{W}$ based on *degree of anonymity* [9, 28]. Our analysis uses information-theoretic entropy [29] as the measure of information contained in a probability distribution. Recall $\mathbb{N}_T$ denote the set of TorPolice-upgraded Tor clients. Given an arbitrary capability-enhanced site access (*i.e.,* an access supported by a valid capability), $\mathbb{W}$ believes that with probability p_i, the access originates from client i in $\mathbb{N}_T$. Thus, $\mathbb{W}$ maintains a probability distribution I for all anonymous accesses. Then, the entropy (*i.e.,* the information contained in the distribution I) is defined as $H_{\mathbb{W}} = -\sum_{i \in \mathbb{N}_T} p_i \cdot \log_2(p_i)$.

Based on the unlinkability proven in Lemma 1, we have $p_i = \frac{1}{N_T}$, where N_T is the size of the anonymity set $\mathbb{N}_T$. Thus, the entropy after introducing TorPolice is $H_{\mathbb{W}} = \log_2 N_T$.

Next, we analyze the entropy before introducing TorPolice. Let $\mathbb{N}$ denote the entire set of anonymous Tor clients. Notice that the current Tor network protects $\mathbb{W}$ from linking a (native) site access with a specific Tor client. Thus, given the entire anonymous client set $\mathbb{N}$, the maximum entropy H_M is $H_M = \log_2 N$, where N is the size of the anonymity set $\mathbb{N}$. Then, based on the definition in [9, 28], the degree of anonymity d after introducing TorPolice is $d = 1 - \frac{H_M - H_{\mathbb{W}}}{H_M} = \frac{\log_2 N_T}{\log_2 N}$.

Anonymous Set Analysis. Given the above anonymity degree, information leakage is affected by the size of the anonymous client set before and after TorPolice is introduced. Thus, once all Tor clients are upgraded to support TorPolice, there is no information leakage at all. Thus, eventually, TorPolice completely preserves the privacy guarantee offered by the Tor network. To mitigate the one-time privacy issue during the early deployment phase of TorPolice, the Tor project can

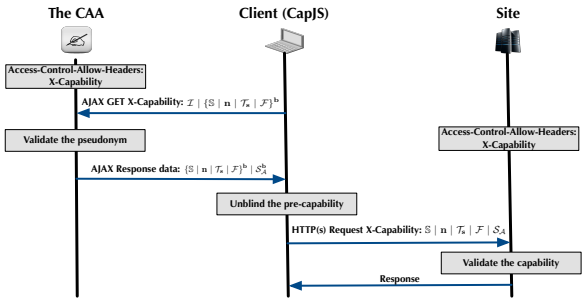

Fig. 2: Site access by a client with CapJS installed.

require mandatory client upgrades from a certain time point to "force" all active clients to serve as TorPolice initiators.

VIII. Implementation

For better readability, we cover high-level implementation of TorPolice in this section and defer details in [22].

Capability Implementation. We implement capability-related computation using C, Python and JavaScript to consider various usage scenarios. For instance, the capability design can be directly built into the Tor software written in C, or it can be implemented as a plugin for the Tor browser, which executes capability-related computation in JavaScript. Websites may compute capabilities using any language. Thus, we use Python as an example due to its popularity in web applications. We use the RSA algorithm to perform capability-related cryptographic operations such as blind signing.

AA Implementation. For an AA accepts CAPTCHAs as capability seeds (referred to as CAA), we implement it as a web server that deploys Google's reCAPTCHA service. For an AA accepts computational puzzles (referred to as PAA), it accepts puzzle solutions over HTTP requests. These AA servers define a customized HTTP header (X-Capability) to carry TorPolice-related cryptographic tokens such as pseudonyms and pre-capabilities. To make the implementation transparent to clients, the AA servers add X-Capability in the Access-Control-Allow-Headers HTTP header option. Although the PAA can be accessed using native HTTP libraries, the CAA needs to be accessed using browsers. Thus, we implement a Firefox add-on (referred to as CapJS) to execute TorPolice-related cryptographic operations in browsers. Detailed design of CapJS is presented [22].

TorPolice-enhanced Site Access. To use our capability scheme, the deployment required at websites is lightweight: a site only needs to add X-Capability in its Access-Control-Allow-Headers HTTP header option to allow CapJS to pass site-specific capabilities in the header. Upon receiving capabilities, the site verifies them using the rules defined in §V-C to fulfill its access policies. Figure 2 depicts a site access by a client with CapJS installed on its browser.

TorPolice-enhanced Tor Circuit Creation. We modify the Tor software source code to directly integrate our capability design into Tor circuit creation. In particular, besides these original cells sent for circuit creation, the modified onion proxy (OP) on a Tor client further sends a valid relay-specific capability to each hop. Each relay first verifies the received capability before processing the onionskin carried in the remaining payload. To validate our implementation, we test the modified Tor source code in Shadow [17], a safe development environment to run real Tor source code in a private Tor network. Via log analysis, our test experiments show that our implementation properly embeds relay-specific capabilities into the workflow of Tor circuit creation.

TABLE I: The computational time for capability-related cryptographic operations.

Operation	Language	Mean (μs)	Median (μs)	Std. Dev. (μs)
Generation	C	232.0	232.0	0.1
	Python	253.7	253.6	0.3
	JavaScript	27,320.0	27,240.0	245.5
Verification	C	25.6	25.6	0.0
	Python	32.0	32.0	0.1
	JavaScript	355.5	354.3	5.3
Blinding	C	3.5	3.5	0.0
	Python	46.3	46.3	0.1
	JavaScript	18.1	18.1	0.3
Unblinding	C	2.4	2.4	0.0
	Python	7.0	7.0	0.0
	JavaScript	64.8	64.7	6.8

IX. Evaluation

A. Capability Computation Overhead

This section benchmarks the overhead of capability-related computation. All results are obtained using a single 3.30GHz Intel i3-3120 core. We perform 10 thousand runs to learn the mean, median, and standard deviation of the computation times for a single capability generation, verification, information blinding and unblinding. Results shown in Table I are obtained when the RSA key length is 1024. The overall computational overhead is small. For instance, in C, it takes $\sim$230 µs to compute a pre-capability and $\sim$25 µs to verify a capability. A blinding and an unblinding operation can be finished in $\sim$3 µs and $\sim$2 µs, respectively, in C. The implementations in C and Python have comparable performance. Although it is more expensive to perform signing and verifying in JavaScript, the overhead of blinding and unblinding operations (performed by Tor clients) in JavaScript is comparable with other languages. The AAs, relays and service providers can adopt more efficient languages (*e.g.,* C) to perform signing and verifying.

B. Enforcing Site-Defined Policies

In this section, we show that TorPolice enables a site to enforce site-defined access policies.

Access Policies. For evaluation purpose, we assume that the site assigns equal weights to both types of capability seeds, *i.e.,* c_0 (for CAPTCHAs) and c_1 (for puzzles) in Equation (3) are the same. However, the actual costs, denoted by c_0' and c_1', can be different from c_0 and c_1. Further, base on the measurements in [10, 24], we assume c_0' is close to λ (the cost for obtaining one network identity).

We evaluate three strategies that a site may use in its access policies. The first strategy (referred to as *basic strategy*) is

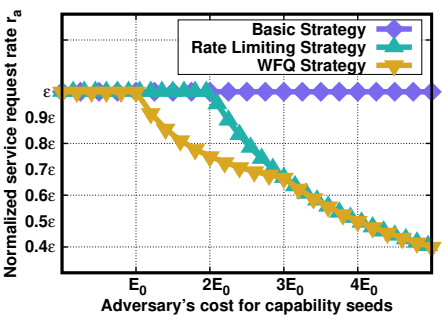

Fig. 3: With TorPolice, a site can bound an adversary's service request rate by $\Theta(\epsilon)$. E_0 is the adversary's cost at the point of diminishing returns.

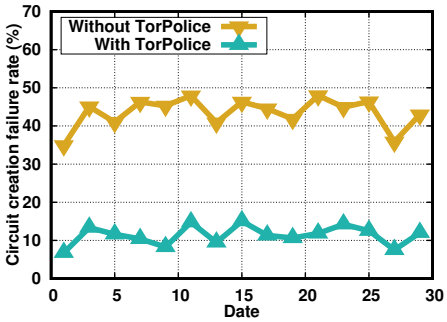

Fig. 4: Circuit creation failure rates when Tor faced a multi-million node botnet C&C abuse. TorPolice can reduce the average circuit creation failure rate by $\sim$74%.

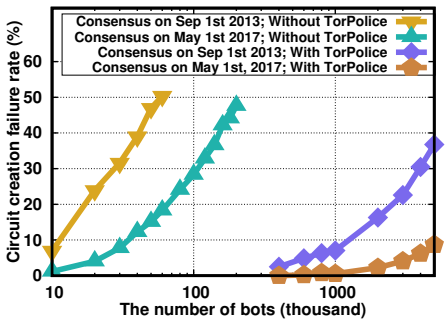

Fig. 5: Without TorPolice, an adversary can paralyze Tor by cell flooding attacks. TorPolice can effectively mitigate this vulnerability.

that the site accepts all Tor-emitted service requests with valid capabilities. In the second strategy (referred to as *rate limiting strategy*), the site enforces a maximum service request rate $r_{\max}$ for *all* valid requests. In the third strategy, besides rate limiting, the site further performs weighted fair queuing (WFQ) to serve requests: rather than serving all valid requests in one FIFO queue, requests with capabilities obtained using CAPTCHA solutions and puzzle solutions are served in two separate FIFO queues weighted equally.The third strategy (referred to as *WFQ strategy*) prevents one type of seed from overwhelming the other one.

Policy Enforcement. We study an adversary's service request rate through Tor when it invests a certain amount of money on acquiring capability seeds. Define $k = c_0'/c_1'$. We first present evaluation results for $k = 0.5$ in Figure 3 and then extend our discussion to arbitrary k. For any amount of investment, the adversary's service request rate through Tor (denoted by r_a) is normalized to the service request rate obtained when the adversary connects to the site directly without using Tor.

Since $c_0'<c_1'$ given k=0.5, the adversary's optimal strategy is spending all investment on solving CAPTCHA. Thus, we have r_a=ϵ, where ϵ is the site-configurable parameter defined in Equation (2). When the site adopts the basic strategy, r_a remains the same as the adversary's investment increases. However, for the other two strategies, r_a will reach a point of *diminishing returns* as the adversary's investment further increases (as shown in Figure 3). In particular, when the site adopts the rate limiting strategy, the point of diminishing returns is reached when the collective service request rate from the adversary and all legitimate Tor clients exceeds $r_{\max}$. In Figure 3, we denote the adversary's cost at this point by $2E_0$. After that, further increasing investment actually reduces r_a since no more Tor-emitted requests are allowed by the site.

When the WFQ strategy is adopted, r_a experiences two points of diminishing returns as the adversary's cost increases, as shown in Figure 3. The first one happens when the collective service request rate from all Tor clients using the optimal seed (CAPTCHAs in our evaluation) exceeds $\frac{r_{\max}}{2}$. After this point, the adversary has to use sub-optimal seeds in order to further get services. As a result, r_a starts to decline from the optimal rate ϵ. The second point of diminishing returns is reached when the collective Tor-emitted service request rate exceeds $r_{\max}$.

General Results. Our further analysis in [22] proves that for any k, $r_a \leq \epsilon$ if $k \leq 1$ and $r_a \leq k \cdot \epsilon$ if $k \geq 1$. Thus, regardless of the actual cost of obtaining capability seeds, the adversary's service request rate is bounded by $\Theta(\epsilon)$. This result holds no matter which strategy the site adopts and how many types of capability seeds are accepted.

C. Mitigating Botnet Abuse Against Tor

We now perform Tor-scale evaluations to demonstrate (i) TorPolice effectively mitigates large-scale botnet C&C abuse against Tor by reducing circuit failure ratios by $\sim$74% (§IX-C1) and (ii) TorPolice significantly increases Tor's resilience against cell flooding attacks (§IX-C2).

Tor-scale Simulator. We aim to show that TorPolice can mitigate the harm that a multi-million botnet can do on Tor. While we have a TorPolice prototype that runs on Shadow [17] (§VIII), we would run into scalability issues with simulating millions of Tor clients. Further, Shadow is unable to help us simulate the cryptographic overhead that botnets would impose on Tor relays [16]. Due to these shortcomings, we develop our own simulator. We faithfully implement Tor's path selection algorithm in our simulator and validate its correctness by comparing relays' selection probabilities with the ones published by Tor [1]. The computational capacity of relays are sampled from live Tor measurement results in [4].

1) Mitigating Botnet C&C Abuse: Based on the historical data from the botnet C&C abuse happened during Aug-Sep 2013, we show TorPolice can effectively mitigate the abuse.

We use the data collected by Tor to estimate the amounts of circuit creations initiated by the botnet during the C&C abuse. To improve readability, we defer detailed modeling in [22]. Due to the massive circuit creations by the botnet, compute resources of many relays are exhausted, resulting in very high circuit creation failure rates, as depicted in Figure 4. Such high failure rates are caused by the following vicious cycle. When the abuse starts, Tor relays begin to drop requests due to the lack of compute resources. These initial failures force the bot clients to keep sending requests until their circuits are successfully created, which further increases the network load. The resulting consequences are that the botnet still managed

to use Tor as its primary C&C channel after numerous trials whereas Tor is less usable for legitimate Tor users since it could take tens of trials to finally create a circuit.

The root cause of such high circuit creation failure ratios is that bot clients can request circuit creations without any limitation. With TorPolice, Tor can rely on the access policies discussed in §VI-A to counter this abuse. We plot the resulting circuit creation failure rates after enforcing the access rules in Figure 4: TorPolice reduces the average failure rate from $\sim$41% to $\sim$10%, a $\sim$74% reduction.

In response to the C&C abuse, Tor released a new version (0.2.4.17-rc) that prioritizes the processing of onionskins using the tnor [11] protocol since the bot clients used an older version without tnor support. Tor's countermeasure reduced the average circuit failure rate to $\sim$20% [15]. However, a strategic botnet could circumvent Tor's defense by changing adaptively (*e.g.,* upgrading software). On the contrary, TorPolice offers long-term countermeasures that can handle strategic botnets.

2) Mitigating Tor-targeted DDoS Attacks: Via cell flooding attacks [4], a botnet can easily paralyze Tor via excessive circuit creation requests. As shown in Figure 5, a moderate-sized botnet with hundreds of thousands of bots is enough to cause very high circuit failure rates via cell flooding attacks. When Tor is protected by TorPolice, however, even a multi-million node botnet can only cause very limited failure rates for the current Tor network (represented by the consensus published on May 1st 2017).

X. Related Work

We briefly discuss some closely related work.

Capabilities in the Internet. Capability schemes (*e.g.,* [20, 21, 31]) have been proposed to protect the Internet from DDoS attacks. In these approaches, capabilities specify certain traffic policing rules and meanwhile carry cryptographic signatures to ensure correctness. Victims (*e.g.,* servers or congested routers) police traffic based on received capabilities to stop attacks. Different from TorPolice, these designs do not consider privacy.

Anonymous Blacklisting Systems. Anonymous blacklisting systems [14] allow service providers to maintain a "blacklist" to block abusive users without breaking anonymity. These systems either offer pseudonymity instead of full anonymity or require a trusted or semi-trusted authority to provide anonymity. TorPolice is not designed to be a new anonymous blacklisting system. Rather, TorPolice is explicitly designed for Tor, focusing on proposing a capability-based access control framework that allows service providers and Tor to enforce access rules to throttle various botnet abuses while still serving legitimate Tor users properly.

XI. Conclusion

In this paper, we present TorPolice, the first privacy-preserving access control framework that allows service providers and Tor to enforce self-selectable access policies on anonymous Tor connections so as to throttle various botnet abuses while still providing service to legitimate Tor users. TorPolice leverages blindly signed network capabilities to preserve the privacy of Tor users. We implement a prototype of TorPolice, and perform extensive evaluations to validate TorPolice's design goals.

XII. Acknowledgements

We want to thank the anonymous reviewers for insightful comments. This research was partially supported by the Center for Information Technology Policy at Princeton University and NSF under Contract No. CNS-0953600, CNS-1423139, CNS-1553437, CNS-1540066 and CNS-1602399.

References

[1] Tor Atlas. https://atlas.torproject.org.
[2] Tor Metrics. https://metrics.torproject.org.
[3] Who Uses Tor? https://www.torproject.org/about/torusers.html.en.
[4] Barbera, M. V., Kemerlis, V. P., Pappas, V., and Keromytis, A. D. CellFlood: Attacking Tor onion routers on the cheap. In *ESORICS* (2013).
[5] Borisov, N., Danezis, G., Mittal, P., and Tabriz, P. Denial of service or denial of security? In *ACM CCS* (2007).
[6] Chaum, D. Blind signatures for untraceable payments. In *Advances in Cryptology* (1983), Springer.
[7] Chen, L. Access with pseudonyms. In *Cryptography: Policy and Algorithms* (1996).
[8] Cloudflare. Challenge bypass specification. https://github.com/cloudflare/challenge-bypass-specification, 2016.
[9] Díaz, C., Seys, S., Claessens, J., and Preneel, B. Towards measuring anonymity. In *International Workshop on Privacy Enhancing Technologies* (2002).
[10] Donohue, B. How Much Does A Botnet Cost? https://threatpost.com/how-much-does-botnet-cost-022813/77573/.
[11] Goldberg, I., Stebila, D., and Ustaoglu, B. Anonymity and one-way authentication in key exchange protocols. *Designs, Codes and Cryptography* (2013).
[12] Gottesman, G. RSA Uncovers New POS Malware Operation Stealing Payment Card & Personal Information. https://blogs.rsa.com/rsa-uncovers-new-pos-malware-operation-stealing-payment-card-personal-information/, 2014.
[13] Goulet, D., and Kadianakis, G. Random number generation during Tor voting. https://gitweb.torproject.org/torspec.git/tree/proposals/250-commit-reveal-consensus.txt, 2015.
[14] Henry, R., and Goldberg, I. Formalizing anonymous blacklisting systems. In *IEEE S&P* (2011).
[15] Hopper, N. Protecting Tor from botnet abuse in the long term. Tech. rep., Tech. Rep. 2013-11-001, The Tor Project, 2013.
[16] Jansen, R. Notes and FAQs of Shadow simulator. https://github.com/shadow/shadow/wiki/4-Notes-and-FAQs, accessed on 2017.
[17] Jansen, R., and Hopper, N. Shadow: Running Tor in a box for accurate and efficient experimentation. In *NDSS* (2012).
[18] Jansen, R., and Johnson, A. Safely measuring Tor. In *ACM CCS* (2016).
[19] Khattak, S., Fifield, D., Afroz, S., Javed, M., Sundaresan, S., Paxson, V., Murdoch, S. J., and McCoy, D. Do you see what I see? differential treatment of anonymous users. In *NDSS* (2016).
[20] Liu, X., Yang, X., and Xia, Y. NetFence: Preventing Internet denial of service from inside out. *ACM SIGCOMM CCR* (2011).
[21] Liu, Z., Hao, J., Hu, Y.-C., and Bailey, M. MiddlePolice: Toward enforcing destination-defined policies in the middle of the Internet. In *ACM CCS* (2016).
[22] Liu, Z., Liu, Y., Winter, P., Mittal, P., and Hu, Y.-C. TorPolice: Towards Enforcing Service-Defined Access Policies for Anonymous Systems. https://www.dropbox.com/s/bn9s3fl7ai2fohe/TorPoliceTechreport.pdf?dl=0, 2017.
[23] Mittal, P., Khurshid, A., Juen, J., Caesar, M., and Borisov, N. Stealthy traffic analysis of low-latency anonymous communication using throughput fingerprinting. In *ACM CCS* (2011).
[24] Motoyama, M., Levchenko, K., Kanich, C., McCoy, D., Voelker, G. M., and Savage, S. Re: CAPTCHAs-understanding CAPTCHA-solving services in an economic context. In *USENIX Security Symposium* (2010).
[25] Murdoch, S. J., and Danezis, G. Low-cost Traffic Analysis of Tor. In *IEEE S&P* (2005).
[26] Parno, B., Wendlandt, D., Shi, E., Perrig, A., Maggs, B., and Hu, Y.-C. Portcullis: Protecting connection setup from denial-of-capability attacks. In *ACM SIGCOMM* (2007).
[27] Prince, M. The trouble with Tor. https://blog.cloudflare.com/the-trouble-with-tor/, 2016.
[28] Serjantov, A., and Danezis, G. Towards an information theoretic metric for anonymity. In *PETs* (2003).
[29] Shannon, C. E. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review* (2001).
[30] Sun, Y., Edmundson, A., Vanbever, L., Li, O., Rexford, J., Chiang, M., and Mittal, P. RAPTOR: Routing attacks on privacy in Tor. In *USENIX Security Symposium* (2015).
[31] Yaar, A., Perrig, A., and Song, D. SIFF: A stateless Internet flow filter to mitigate DDoS flooding attacks. In *IEEE S&P* (2004).