Inside Job: Applying Traffic Analysis to Measure Tor from Within

Rob Jansen[†*], Marc Juarez[‡*], Rafa Gálvez[‡], Tariq Elahi[‡] and Claudia Diaz[‡]
[†]U.S. Naval Research Laboratory, rob.g.jansen@nrl.navy.mil
[‡]imec-COSIC KU Leuven, {marc.juarez, rafa, tariq}@kuleuven.be, {claudia.diaz}@esat.kuleuven.be

***Abstract*—In this paper, we explore traffic analysis attacks on Tor that are conducted solely with middle relays rather than with relays from the entry or exit positions. We create a methodology to apply novel Tor circuit and website fingerprinting from middle relays to detect onion service usage; that is, we are able to identify websites with hidden network addresses by their traffic patterns. We also carry out the first privacy-preserving popularity measurement of a single social networking website hosted as an onion service by deploying our novel circuit and website fingerprinting techniques in the wild. Our results show: (i) that the middle position enables wide-scale monitoring and measurement not possible from a comparable resource deployment in other relay positions, (ii) that traffic fingerprinting techniques are as effective from the middle relay position as prior works show from a guard relay, and (iii) that an adversary can use our fingerprinting methodology to discover the popularity of onion services, or as a filter to target specific nodes in the network, such as particular guard relays.**

I. Introduction

Tor [8] network entry and exit points have received considerable focus over the years through continuous research activity because the entry and exit points directly connect to the end-user and destination, respectively. However, potential threats from *middle relay* positions have received far less attention. We believe that this is because there is a common misconception that malicious or compromised middle relays are not a significant threat to end-users' privacy since they do not directly connect to either the end-user or the destination and thus are unable to link both parties together. While middle relays are not privy to the network addresses of the client and destination, they can still passively yet directly observe a plethora of other information including service access times, transfer volumes and data flow directions, and the preceding and succeeding relays chosen for connections. This *information leakage* could be analyzed to discover client usage and network patterns in the Tor network and thus yield potential attack vectors, and we believe that such threats present a wide gamut of possible avenues for research.

In this work we focus on the novel application of different traffic analysis techniques such as circuit fingerprinting [22] and website fingerprinting (WF) [4], [14], [26], [28], [29], [30] performed from middle relays as opposed to the usual guard relays. We design the first circuit and website fingerprinting algorithms specifically for use at middle relays, and we are the first to apply machine learning to detect from which of many possible circuit positions a relay is serving. Using our novel *circuit purpose*, *relay position*, and *website fingerprinting* algorithms, we produce a classification pipeline (*i.e.*, a python library) that can identify which onion service websites accessible through the Tor network are visited by Tor users. In selecting this specific scope we are keen to focus on sensitive usages of Tor (*i.e.*, onion services) where assumption failures may lead to high-stakes consequences. We are also specific enough to yield a concrete methodology, tangible code, and data artifacts, as well as empirical results that were thoroughly analyzed with well understood limitations. We provide our viable framework of tools[1] so that future work may use them as foundations for the study of attacks and their mitigation.

In the interest of real-world applicability, we deploy our classification pipeline in the first real-world measurement study using WF with real Tor users. For the sake of ethical research and limiting the impact on user privacy, our pipeline is augmented with PrivCount [19], a privacy-preserving statistics collection suite designed to provide differentially-private [9] results, hiding individual user activity. We use PrivCount to measure onion services while focusing on the popularity of only a single well-known social networking platform (SNS) to further limit the impact on user privacy: the site we measure is accessible through a single-hop onion service indicating that the service itself does not require anonymity. Our measurement not only yields useful information about this particular onion service, but it also serves as a proof-of-concept that our classification pipeline can be used at middle relays as a vector for information leakage. We note that while we as ethical researchers are constrained by the differentially-private (*i.e.*, noisy) results, a malicious actor is not and would be able to produce measurements of higher accuracy. Therefore, our results represent a lower bound on the information leakage potential of WF at middle relays.

We highlight that in this work we treat website fingerprinting techniques as general tools to identify websites. In contrast, previous work on Tor website fingerprinting assumes a malicious guard relay or a monitored node on the client-to-guard network path; under this previous model, it is assumed that an adversary already knows or can easily discover a

∗ Equally credited authors.

[1] https://github.com/onionpop

Network and Distributed Systems Security (NDSS) Symposium 2018
18-21 February 2018, San Diego, CA, USA
ISBN 1-891562-49-5
http://dx.doi.org/10.14722/ndss.2018.23261
www.ndss-symposium.org

target client's IP address and all that remains in order to mount a *linking attack* is to identify the website that the client visits. In that setting it constitutes an attack to merely perform WF successfully, whereas in this work we use WF to detect revealing information about the Tor network and its usage. One could then leverage that information, *e.g.*, to perform wide-scale monitoring of onion site usage or to mount an attack linking client to destination (see Section IX-B).

Our results include: (i) our *circuit purpose* and *relay position* classifiers achieve $92.41\% \pm 0.07$ and $98.48\% \pm 0.01$ accuracy respectively; (ii) our WF at the middle classifier achieves more than 60% accuracy for a closed world of 1,000 onion services, which is competitive with classical WF applications; (iii) our one-class classifier for real-world deployment is bounded to 0.6% false positive rate at a cost of decreasing the true positive rate to 40%; and (iv) our real-world deployment shows that the social networking website's onion service accounts for 0.52% of all onion service circuits created. These results are compelling and provide evidence that WF is viable at the middle relay position, that we can effectively target onion service traffic, and that real-world deployments can yield actionable results.

We make the following contributions in this paper:

1) we design the first classifier to detect relay position, and the first classifier to detect circuit purpose from a middle relay position using a novel feature set that utilizes internal Tor protocol meta-data;
2) we are the first to show that traffic fingerprinting techniques are effective from a middle relay position for both closed-world and one-class open-world problems;
3) we produce a classification pipeline that combines circuit purpose, relay position, and WF classifiers for real-world deployment; and
4) we perform the first measurement study that applies traffic fingerprinting to discover Tor onion service popularity, done ethically with privacy-preserving statistics collection methods.

II. Background

A. Tor

Clients use Tor by first telescoping a long-lived *circuit* through a series of three volunteer *relays*: the client chooses a persistent *entry guard* relay as its first relay (using this same guard for three months before rotating to a new one), chooses a new random *middle* relay, and chooses a last *exit* relay that will allow it to connect to the intended Internet service external to the Tor network. The relays forward traffic in both directions through the circuit to facilitate communication between the client and its communicating peer.

Tor's popularity is partly due to the flexibility provided by its design: the external peer need not run Tor or even be aware that the client is connecting through the Tor network. However, clients who connect to external peers must still rely on the existing DNS and SSL/TLS certificate authentication systems, and the external peers themselves are not anonymous. To mitigate these issues, Tor also develops and maintains an *onion service* protocol, a communication mode in which both the user and its peer run Tor and build circuits that are connected together. All communication is internal to the Tor network, and therefore the user and its peer both enjoy anonymity and end-to-end encryption without relying on external insecure name and certificate authentication systems.

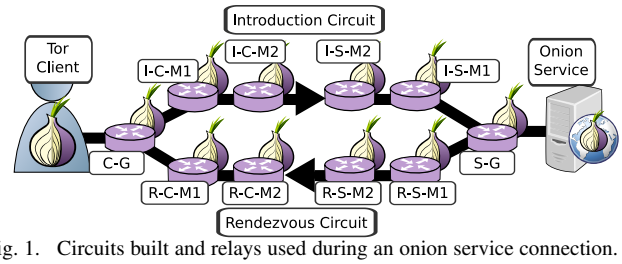

Fig. 1. Circuits built and relays used during an onion service connection. The I-S-M2 relay serves as the *introduction point*, and the R-C-M2 relay serves as the *rendezvous point*.

B. Onion Service Protocol

The protocol that Tor clients and onion services use to establish a connection is as follows. In order to advertise themselves and be reachable by clients, onion services maintain a few long-term circuits that last at least one hour. Relays at the end of these long-term circuits play the role of *Introduction Points* (IP). The addresses of IPs along with other onion service information such as their public keys are stored as *descriptors* in a distributed database formed by Tor relays that have been assigned the `HSDir` flag. To create an IP circuit, the onion service must create a regular three-hop circuit and send an `establish_intro` cell to the last relay. The last relay replies with an `intro_established` cell if it agrees to act as the IP on this circuit.

To establish a connection with the onion service, the client first selects a middle relay to serve as a *Rendezvous Point* (RP) and builds a circuit ending at that relay. The client then sends an `establish_rendezvous` cell to the RP which replies with a `redezvous_established` cell. The client must then inform the onion service of this RP using the service's IP. To learn the IP, the client builds a circuit to an HSDir, sends the service's .onion address that was communicated out-of-band, and receives the onion service's descriptor (including the addresses of the service's current IPs). The client then builds a circuit to one of the IPs and sends it an `introduce` cell which contains the RP's address and a one-time secret, all encrypted with the onion service's public key. The IP relays the cell to the onion service which acknowledges the receipt by sending a `introduce_ack` back to the client.

The onion service decrypts the RP address, builds a circuit to it, and sends it a `rendezvous` cell that contains the one-time secret provided by the client for authentication. The RP relays this cell to the client who will verify the one-time secret and acknowledge receipt to the onion service. At this point, a six-hop circuit exists between the client and the onion service and can be used for application communication. Figure 1 depicts the four main types of circuits that have been created (excluding the HSDir circuit): an onion service to IP, a client to IP, a client to RP and an onion service to RP. Note that the circuits created for this process are dedicated to the onion service protocol and cannot be reused for other communications.

C. Stream Isolation

Applications tunnel peer connections, called *streams*, through Tor circuits. The Tor software decides if a new stream should be assigned to an existing used circuit, an existing unused circuit, or if a new circuit should be built to handle the stream. Tor would like to provide unlinkability of *unrelated traffic* in order to reduce the exposure to honest-but-curious exit nodes that may track unrelated visits by the same user. However, completely isolating each stream to its own circuit

would significantly degrade Tor's performance while allowing malicious servers to cause a client to create an arbitrary number of circuits, which would increase the probability that a client selects a compromised node for at least one of them. For this reason, Tor prefers to isolate groups of streams to their own circuit. In TorBrowser, a hardened fork of Firefox and the recommended web browser to use with Tor, streams are grouped by the first-party domain that appears in the URL bar (across different tabs). This means that almost all streams generated during a page download will go through the same circuit, including requests to third parties. Note that, although unlikely, Tor may create a new circuit while fetching a web page; for example, a restrictive exit policy of a circuit may cause Tor to create a new circuit with an exit that supports the fetch of a particular resource (*e.g.*, transitions from HTTP to HTTPS and vice versa).

However, the rules for handling onion service traffic are different. Since onion service circuits do not exit the Tor network and require that a *Rendezvous Point* is agreed upon between the client and the onion service, there is currently no stream grouping by the first-party domain of the onion URL in the address bar. Therefore, a user visiting an onion service with mixed first-party onion service and third-party onion or non-onion service content may create multiple circuits to fetch the content; it will create a circuit to fetch all of the first party content, a circuit to fetch all non-onion service third party content (even if each third party is served from a different domain), and a circuit for each third-party embedded resource hosted from a unique onion service.

These peculiarities of the onion service protocol limit the visibility of an adversary monitoring the traffic at the middle: while an adversary at the entry or client-to-entry link is able to capture all the traffic that a user generates during a visit to an onion service, the adversary we consider would only be able to record the traffic for the first-party content.

D. Traffic Fingerprinting

The traffic analysis techniques that we study in this paper are based on applying supervised learning methods on the encrypted and anonymized traffic traces that are captured from middle relays we control. We study traffic fingerprinting attacks such as circuit and website fingerprinting that use side-channel information leaking from encrypted network packet timing and lengths to discover patterns in the communication. In particular, circuit fingerprinting allows an attacker to distinguish between visits to onion services from regular sites and website fingerprinting enables one to identify the website being accessed. To the best of our knowledge, all previous website fingerprinting studies in Tor have been conducted either at the entry guard or somewhere on the network path between the client and the guard.

Most studies evaluate WF in a *closed world* model in which it is assumed that the classifier could be trained on data for *all* of the sites that the user was possibly going to visit. This assumption is unrealistic because there are potentially billions of websites and the resources necessary to collect, store, and process the data for all such sites would be overwhelming. A more realistic evaluation method uses an *open world* model in which only a small fraction of the sites are available to the adversary for training. However, the closed world *has* been considered a realistic scenario *if* the adversary aims at detecting only onion services [6]. It has been shown that a local and passive adversary can effectively first detect onion service visits using circuit fingerprinting, and then apply website fingerprinting methods to infer to which website they belong [22], [27].

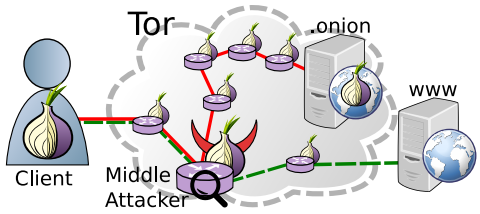

Fig. 2. The adversary runs a middle relay and monitors the Tor messages that it relays. The adversary will observe circuits carrying traffic for onion services internal to Tor and regular web servers external to Tor.

In this paper, we evaluate the effectiveness of traffic fingerprinting from Tor middle relays under both open and closed world models while focusing on onion services. WF is particularly threatening for onion services for two reasons [6], [25]: (i) there are fewer onion services than regular sites and, since the adversary can filter out visits to regular sites, it needs less resources to fingerprint onion services effectively; and (ii) onion services hide their location on the network so that it is difficult to censor them and may host sensitive content, and therefore visitors of those sites are especially vulnerable if the WF attack is successful.

III. Requirements and Ethical Research

We now describe the capabilities required to fingerprint onion service websites from the middle relay position and discuss ethical considerations.

A. Requirements

To apply the techniques described in the following sections, we do not depend on the ability to break the encryption of Tor but do depend on the ability to eavesdrop on all network traffic to and from relays we control. We can obtain a traffic *trace* or *sample* of both the encrypted network packets and the Tor protocol messages (*i.e.*, *cells*). We are able to observe, decrypt, and read the headers and payloads of Tor cells that are destined for the middle relays we control, but we can only observe and read the headers of cells intended for another destination and forwarded through our relay (the payloads of such cells are encrypted).

We also need to deploy at least one middle relay that contributes bandwidth to Tor. Our attacks become more statistically sound as we observe more circuits, and the fraction of circuits that we will observe roughly correlates with the amount of bandwidth that we contribute. Note that it is quite affordable to run Tor relays in dedicated servers or as virtual instances on the various low-cost cloud platforms available.

Furthermore, we only require *local* visibility of the network because we can only observe circuits from clients that pick our relays and cannot observe other activity. Figure 2 depicts the position of the relay from which we perform the website fingerprinting that is the focus of this paper.

We desire to be able to utilize the machine learning techniques we propose on common desktop hardware. This means that along with the ability to collect data (described in more detail in Section VI-D), we can also perform the pre-processing and data cleansing, the training, and finally the classification tasks all on hardware commonly found on desktop computers of today.

B. Ethical Considerations

We have contacted the Tor Research Safety Board[2] for advice on the ethical implications of this research and have followed their recommendations. We also contacted the SNS that we have taken as use case for this study for responsible disclosure but did not receive any response from them. Since we investigate a number of settings with varying levels of risk for real users, we provide additional details about ethical research throughout the paper.

IV. Advantages of the Middle of the Path

In this section, we discuss the benefits of running middle relays to analyze onion service traffic as compared to relays that are at the ingress and egress points of the network.

A. Exit

Exit relays make connections outside of the Tor network, and an exit relay will never be chosen by Tor's default path selection algorithm in non-exit positions or in onion service positions due to exit relay bandwidth scarcity. An exit relay will thus not be useful for our purposes, given that it will not route any onion-service visit.

B. Guard

Each client chooses a relay from the set of all relays with the guard flag and uses it as its first hop entry into the Tor network for all circuits it builds. A guard relay may serve in both the first-hop guard position and in the middle relay positions, and its bandwidth is split among these two positions. In order to be eligible to serve as a guard, a relay is required to be stable and have high up-time relative to other relays. Additionally, a guard relay will not be fully utilized when it first becomes a guard, because clients only drop their current guard and rotate to new ones after two to three months (there exists a proposal to increase this time to nine months [7]). As a result, it will take several months to reach steady state, and during that time the relay will observe less traffic than in other positions. A guard will observe many circuits, including onion service circuits, from a smaller slowly churning set of clients.

C. Middle

A middle relay can be used for any circuit, and may potentially observe the traffic of *any* Tor user in the network, given that enough circuits are made over time. This is in contrast to guard relays that can only observe the traffic of users that have picked them as their first hop.

We are particularly interested in regularly visited onion services. The following equation shows that the probability of a particular middle relay observing a client's circuit increases as the client builds more and more circuits over time, where l is the likelihood of picking that middle for a single circuit and c is the number of circuits that the client has made so far.

$$P(observed) = 1 - (1 - l)^c$$

We investigate how the frequency of visits to an onion site, $f = \frac{c}{t}$ where f is the fraction of the number of visits c in a given unit of time t, affects the probability of being observed by a middle relay. Let's assume that a user visits onion services just once every unit of time, for instance once per day. From the line labeled $f = 1$ in the left plot of Figure 3 we see that this client will have an almost 80% chance of making *at least* one circuit through the malicious middle relay after 1,000 days, or two years and nine months. In contrast, a user that visits onion services ten times per day has the same chances in just a little over three months. As point of reference a similarly provisioned guard relay, shown as the rightmost line labeled $f = 10, guard$, reaches similar levels of probability only after two orders of magnitude of time later, 10,000 days.

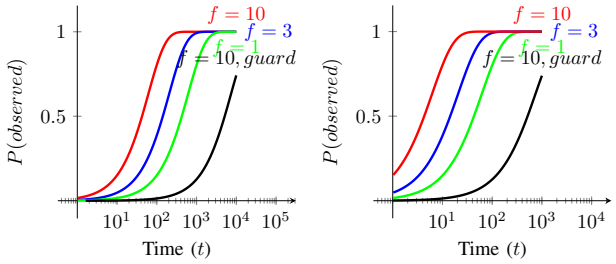

Fig. 3. Probability of a client making a circuit using a malicious node with 0.16% (2 MB/s) and 1.6% (20 MB/s), left and right plots respectively, of the middle bandwidth. Plots are shown for various frequencies of visit within a fixed time interval.

This time may be shortened, for instance, by operating ten identical relays—which is not an onerous burden on resources nor difficult to practically achieve—and then there is an 80% chance that a user who connects to an onion service once per day would take about 100 days and a user that connects ten times per day will now only take about 10 days to create at least one circuit through our middle relay (see lines labeled $f = 1$ and $f = 10$ in the right plot of Figure 3). As reference, a similarly provisioned guard in this setting, the rightmost line labeled $f = 10, guard$, has the same probability of observing that client only after two orders of magnitude later, 1,000 days.

The preceding illustrates that middle nodes can enumerate more clients in a shorter time frame. We want to clarify that both the guard relay and middle relay observe the same number of *circuits*, but a different set of *clients*. The guard relay will only observe circuits from a somewhat static subset of Tor clients, but the guard observes all circuits from that subset. In contrast, the middle relay will observe circuits from the set of all clients but only some of the circuits built by those clients (in the same time frame). However, when dealing with a frequently visiting user, the middle relay will be able to obtain a representative sample of these accesses, which provides it qualitatively the same information as the guard. In this way middles have a better overview of the entire onion service and Tor userbase activity, albeit sampled at a known rate.

V. Circuit Purpose and Position Fingerprinting

We have argued that the middle relay position is advantageous for obtaining a statistical sampling of client activities across the Tor network. In this section, we show how machine learning classification techniques can be used by the middle relay to determine which middle position and which circuit purpose (*i.e.*, onion service or general) it is serving, enabling it further analyze only circuits that carry onion service traffic.

A. Methodology

There are multiple middle onion service circuit positions in which a middle relay could serve, and a middle relay will also serve in non-onion service, general purpose circuits (see Figure 1). To understand how a middle relay can detect its position in a circuit and the circuit purpose, we first generate a

[2] https://research.torproject.org/safetyboard.html

large data set of circuits that were built using Tor. We modified a Tor middle relay[3] to log messages containing the information necessary to perform the classification, and we incorporated a new signaling mechanism that enables the client to send ground truth to the relay for evaluation purposes. We run our modified Tor code under simulation in Shadow [18] as well as on the live Tor network. For the latter, we need to be sure that we do not capture any information from circuits that are not under our control in order to protect real Tor users.

1) Circuit Signaling: In order to perform classification, our middle relay requires timing and flow information from Tor circuits. A middle relay and client under our control will not be directly connected, however, and therefore we need a signaling mechanism with which our client can identify to our middle relay the circuits that we control and that are safe to analyze.

We added a new *signaling* cell to the Tor protocol and a mechanism to allow the client to pin a relay as its R-C-M1 middle on all circuits. The signaling cell is inserted by our Tor client into new circuits that it creates through our middle. The new cell is encrypted for and sent to our middle relay through the Tor circuit, and no other relay in the circuit can read it. The new cell identifies to our middle relay that the circuit on which the cell is sent should be labeled as our own circuit and therefore that it is safe to start tracing the circuit. The signal cell may include an optional payload so that the client can send ground truth information about the circuit (*i.e.*, the purpose and position of the relay), stream, and HTTP request (*i.e.*, the URL being fetched) to our middle relay.

2) Tracing Traffic Patterns at the Middle Relay: Once our middle relay has identified that a circuit is initiated at our client, it begins collecting information about the circuit and the cells transferred through it. This information is exported through the Tor control protocol, which provides a well-defined interface [1] for other applications to request and consume information about Tor (including information associated with periodic events). For every circuit on which we receive a signal from our client, the middle relay exports a unique circuit ID, circuit creation time, as well as the IP address, fingerprint, and relay flags of the previous and next hop relays. It also begins logging information about each cell it sends and receives on that circuit: it logs the direction of the cell (outbound or inbound), how the cell was transferred (sent or received), the time the cell was transferred, the unique ID of the circuit to which the cell belongs, and the cell type and command (which are used to instruct relays to, *e.g.*, create, extend, and destroy circuits or relay traffic).

3) Collecting Data with Shadow: We ran our customized Tor software under simulation in Shadow [18] in order to generate a large corpus of circuits suitable for analysis. Shadow is a discrete-event network simulator that directly executes Tor, and therefore faithfully executes all of the logic associated with building Tor circuits. Shadow allows us to construct and run a private Tor network, complete with directory authorities, relays, clients, servers, and onion services, and gives us full control over our network. We run Shadow experiments in a private, local environment free of privacy risks to real users.

Our primary goal is to collect a large sample of circuits from the perspective of a middle relay, which could be done either by running many smaller experiments in parallel or fewer larger experiments sequentially using the same amount of RAM. We didn't believe it was necessary to run a full-scale Tor network because the features used by our purpose and position classifiers are not sensitive to network congestion or scale (we dont use any time-based features). Running smaller networks means we can more effectively parallelize our experiments, sample more initial random seeds, and more quickly obtain results. Therefore, we generated a small private Tor network configuration with 50 relays, 128 web clients, 42 bulk clients, and 100 servers. We ran 83 experiments with distinct seeds for one simulated hour each (83 simulated hours in total) on a machine with a total of 40 Intel Xeon 3.20GHz CPU cores (80 hyper-threads) running the latest CentOS 7 version of Linux. We ran 4 multi-threaded experiments at a time, and each experiment took roughly 3.5 hours to complete.

During our experiments, the clients behave as follows. The bulk clients continuously download 5 MiB files. The web clients request data according to an HTTP model where the size of the first request and response, the number of embedded objects per page, the size per embedded object request and response, and the number of distinct domains per page are all sampled from the HTTP Archive.[4] Web clients pause for a time drawn uniformly from 1 to 30 seconds after each full web page has been downloaded before starting the download of another page. Because HTTP Archive data is constructed by downloading real websites from the Alexa top sites list, we believe that the number of Tor circuits that are created when clients use this model is representative of typical Tor circuit usage. Finally, 8 of the web clients and 2 of the bulk clients download their data from onion services, while the remainder of each download their data over regular 3-relay-hop circuits.

We configured one middle relay to act as a middle measurement relay. We enabled the signaling mechanism described above on each client in the network, so that our measurement middle relay would receive ground truth circuit information and collect cell traces for all circuits built through it. A significant advantage of using Shadow is that we are able to inspect all such circuits without risking user privacy. During our experiments our middle measurement relay collected tracing information for 1,855,617 total circuits, 813,113 of which were onion service circuits.

B. Feature Extraction

We extracted features from our large corpus of circuits based on the observation that cell meta-data, such as *cell type* and *relay command*, leaks information to a relay about its position in the circuit and the circuit type (see Figure 4). We also make the following observations: (i) a relay will send a different number of cells during the circuit construction process depending on its position in a circuit; (ii) different relay positions may receive different cell types and relay commands during circuit construction (*e.g.*, guards and middles will *extend* circuits while exits will not); (iii) relays may or may not connect to other known relays on either side of a circuit (iv) onion service introduction circuits transfer much less data than rendezvous circuits used to download web content; and (v) asymmetric web content downloads would result in more cells traveling toward the circuit originator on client-side rendezvous circuits but away from the circuit originator on service-side rendezvous circuits. To incorporate the previous observations, we use as features the counts of

[3] We branched Tor at version `0.2.7.6`

[4] http://httparchive.org

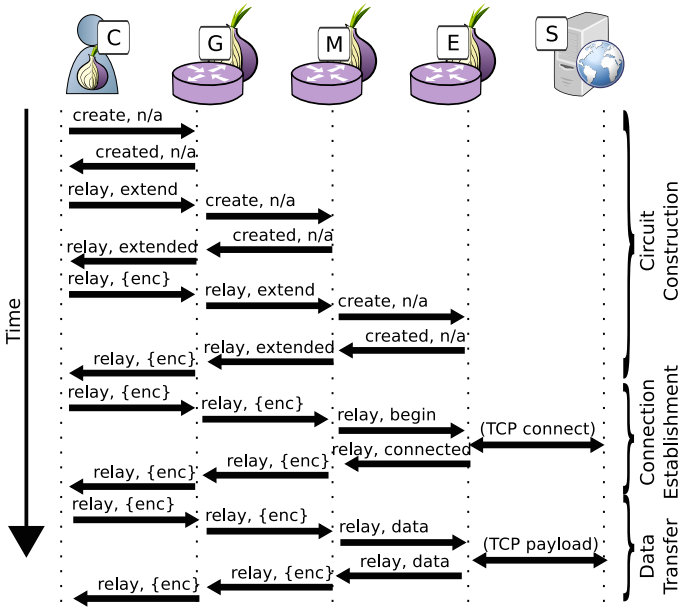

Fig. 4. Circuit establishment and data transfer involves various cell *types* and *commands*, which in this example are labeled on the arrows and are readable by the node to which the arrow points ({enc} denotes that the command is unreadable). Cell meta-data leaks information to a relay about its position in the circuit and the circuit type.

Fig. 5. A relay may send and receive a different number of cell types on the inside and outside of a circuit.

the number of each possible (cell type, relay command) pair that a relay handles. A relay may observe a cell on different sides of a circuit (see Figure 5), so we also include counts of the number of cells that a middle is sending and receiving on both the inbound initiator side of the circuit (inside) and the outbound extension side of the circuit (outside). Finally, we include the total number of cells that the relay is sending and receiving on either side of the circuit, and whether or not the previous and next hops may serve as guard or exit relays.

Note that although it was recommended in previous research [22], we do not include cell timing information in order to be more robust to Tor relay congestion. Interestingly, during the feature analysis we incorporated the circuit duration and cell sequence features that Kwon *et al.* have shown to be useful in distinguishing circuit purpose when classifying circuits from the position of the guard-to-client link [22]. However, we found that these features *reduced* the accuracy of our classifier, and therefore we ignore those features in the remainder of our analysis.

C. Training

The goal of our classifier is to predict when a relay is serving on a rendezvous *purpose* circuit and in the first middle relay *position* (*i.e.*, an R-C-M1 relay as shown in Figure 1). Previous work by Kwon *et al.* [22] provides a decision-tree driven classifier to perform *purpose* filtering but it was designed for use by relays in the guard position or an eavesdropper on the client-to-guard path and not from middle relays. We design a new random-forest driven classifier that performs with comparable accuracy but is tuned for relays in positions between the guards of the client and onion service. Random forests generalize better than simple decision trees, which tend to overfit the training data, thus random forests are more robust against small differences between training and testing settings. We followed prior work on model selection and tuning [14], [22] and, after a search of the parameter space, we found that 30 trees for the random forest provide the highest accuracy.

We trained separate random forest classifiers for circuit purpose and circuit position using the `pyborist` and `sklearn` python APIs on our Shadow-generated circuit dataset and the features we previously described. For both classifiers, we assume no prior knowledge about the circuit purpose or position, so the classifiers could be run independently of one another without affecting the accuracy. To ensure that the classifier does not overfit to our specific dataset, we used standard machine learning procedures such as balancing the dataset so that each class (*i.e.*, rendezvous vs. other purpose, and C-M1 vs. other position) has the same number of circuits.

We used k-fold cross validation ($k = 10$) to measure how well the classifiers generalize to unseen data. During this process, our original circuit sample is randomly partitioned into k equally-sized subsamples. There are k phases in total: in each phase, a distinct subsample is used as the *testing set* while the remaining $k-1$ subsamples are used as the *training set*. To train, we convert each circuit from the training set into a feature set labeled with the true class (the true purpose or position) and pass that into the classifier's training function. To test, we convert each circuit from the testing set into a feature set (without the ground truth class label) and pass it into the classifier's prediction function to predict the class label. We evaluate prediction performance by measuring true and false positives and negatives and computing standard related metrics such as accuracy and precision.

D. Results

The evaluation results are shown in Table I. As shown, the accuracy for the purpose classifier is over 92 percent with a standard deviation of 0.07 and the accuracy of the position classifier is over 98 percent with a standard deviation of 0.01. Table II shows the most important features as determined by our analysis, *i.e.*, the features that minimize the information gain in every branch of the random forest. Not surprisingly, cells associated with the circuit construction (create/created type cells and relay type cells with extend/extended commands) are often some of the top features for distinguishing both purpose and position, and the total number of cells sent and received are also useful for both classifiers.

Based on our results, we believe that our simple cell-based counters serve as effective features for position and purpose classification. They are simple and easy to compute and may potentially be useful in other contexts such as onion service fingerprinting when access to Tor cell meta-data is available.

VI. Onion Service Fingerprinting

In this section, we explore the extent to which middle relays can be effective at carrying out state-of-the art website fingerprinting techniques on onion sites. We describe how we modify and use the Tor and `tor-browser-crawler` software to gather data from a middle relay position, explain the fingerprinting techniques that we performed on these data, and how we evaluated their efficacy.

TABLE I. 10-FOLD CROSS-VALIDATED CIRCUIT CLASSIFICATION RESULTS

	Purpose (rendezvous vs other)	Position (C-M1 vs other)
Accuracy	$92.41 \pm 0.07\%$	$98.48 \pm 0.01\%$
Precision	$91.87 \pm 0.11\%$	$97.16 \pm 0.03\%$
Recall	$93.05 \pm 0.09\%$	$99.88 \pm 0.01\%$
F-1	$92.46 \pm 0.07\%$	$98.50 \pm 0.01\%$
True Positives	396,615 (91.77%)	821,478 (97.08%)
False Positives	35,576 (8.23%)	24,689 (2.92%)
False Negatives	30,056 (6.95%)	984 (0.12%)
True Negatives	402,135 (96.05%)	845,183 (99.88%)

TABLE II. MOST IMPORTANT CIRCUIT CLASSIFICATION FEATURES*

Purpose (rendezvous vs other)	Position (C-M1 vs other)
13.73% # (relay,{enc}) cells	23.22% next node is relay
11.11% # (create2,n/a) cells	11.23% # (relay,extended2) cells
09.10% # cells sent total	09.26% # (created2,n/a) cells
08.89% # cells received total	06.66% # cells sent total
08.31% # cells sent inside	06.61% # (create2,n/a) cells
07.78% next node is exit	06.12% # (relay_early,extended2) cells
07.66% # (relay_early,extended2) cells	05.75% # cells received outside
06.78% # cells received inside	05.32% # cells received total
05.83% # (relay_early,{enc}) cells	05.25% previous node is guard
04.26% # (created2,{enc}) cells	04.06% # (relay,{enc}) cells

* Shown are the top 10 of 22 total features used (both classifiers used the same features).

A. Evaluating Website Fingerprinting

The website fingerprinting techniques that we chose for this evaluation are the most scalable and successful known so far in the literature. These are:

1) Wang-kNN [29]: presented by Wang *et al.*, it achieves over 90% accuracy for 100 non-onion sites. Wang *et al.*'s features were actually families of features defined by a certain parameter—for instance, the number of outgoing packets in the first N (parameter) packets. By varying these parameters they generated more than 3,000 features. The underlying learning model they used was a k-Nearest Neighbors classifier (k-NN). k-NN classifies an instance by averaging over the k closest instances in the dataset according to a given distance metric. In their case, they used a weighted euclidean distance with a tuning mechanism that minimizes the distance among traffic samples that belong to the same site, a property that is especially suited for k-NN.

2) CUMUL [26]: presented by Panchenko *et al.*, it is based on an SVM with a Radial Basis Function (RBF) as a kernel. Their evaluations show that CUMUL achieves 93% accuracy for 100 non-onion sites. CUMUL's main feature is the cumulative sum of packet lengths. The cumulative sum of a traffic trace is represented as a vector with as many components as the number of packets in the trace. Recursively, the first coordinate value is the length of the first packet and the i-th coordinate is calculated by adding the length of packet i and the value of coordinate $i-1$. Since SVM expects fixed-size feature vectors and the cumulative sums have varying sizes, they interpolate 100 points for each cumulative sum.

3) k-Fingerprinting (k-FP) [14]: presented by Hayes and Danezis, it is the most recent website fingerprinting technique. It is based on Random Forests (RF) and k-NN and achieves similar accuracy to CUMUL. Their feature sets are formed by 175 features that, among others, include most of the features that have already been proposed in the website fingerprinting literature to date. Their feature representation is novel: instead of plugging the features directly into a classifier, they instead use the leaves in a trained RF as the representation for classifying with a k-NN with Hamming distance.

All of these attacks have also been evaluated in an *open world* of websites where they perform with high accuracy. The open world is a more realistic setting where the adversary cannot train on all sites that can be visited by the victim.

B. Methodology

We gather data that enables us to analyze the effectiveness of onion service website fingerprinting attacks from internal circuit positions. We do this by running our modified Tor software described in V-A1 and V-A2, crawling a set of known onion sites, and tracing our client's circuits from our own middle relay.

We have automated our crawls using a web crawler that visits a list of onion service URLs with the Tor Browser, called `tor-browser-crawler`.[5] We based our collection methodology on previous studies on website fingerprinting. As Wang and Goldberg proposed [30], we divided the crawls into batches. In each batch, we iterate over the whole list of URLs, visiting each URL several times. The rationale behind batched crawls is that visits to a page in different batches allows the capture of features that are invariant over time; and combining visits within a batch reduces the time-independent noise due to sporadic errors or per-visit variations such as advertisements. We also disable the `UseEntryGuards` Tor option so that we select a different entry guard for each circuit. As a result, we significantly reduce the probability that our testing and training instances are collected over a circuit with the same entry guard, which would unrealistically improve the accuracy of the attack [20].

To speed up the total crawling time, for every visit we create a new identity using Tor Browser's `torbutton` add-on, and then signal the Tor controller to select a random entry relay. This way we don't restart Tor on every visit. In addition, restarting the identity guarantees that we have a clean browser state for the visit, as previous studies have pointed out that keeping the browser state may create artificial dependencies between consecutive visits [30].

The client logs TCP packet headers with `tshark` during each visit to an onion page. We ignore the TCP payloads because they are encrypted and thus not useful. By sniffing network traffic, we can reproduce previous WF evaluation techniques that do not allow access to cell-level information. Because we are interested in cell-level information in this work, we also use OnionPerf[6] to collect Tor cell traces at the client. For debugging and error detection purposes, we take a screenshot of the page as rendered by the Tor Browser, intercept and dump HTTP requests and responses with a browser add-on that the crawler install on Tor, and dump the `index.html` source code. Recall that we apply these techniques only on our own circuits and not those of regular users.

As we described in V-A1, the client pins one of our middles as the R-C-M1 relay (see Figure 1). Our middle relay collects

[5] https://github.com/onionpop/tor-browser-crawler
[6] https://github.com/robgjansen/onionperf

the information from our custom signaling cells as described in V-A2 using OnionPerf in `monitor` mode. OnionPerf will produce a log file containing the data sent by the client as well as other standard Tor events (*e.g.*, bandwidth information). Each circuit that is created by our own crawler will be labeled as such in the OnionPerf log file. We later process these raw log files as necessary to apply the website fingerprinting technique.

In addition, our crawler also flags the start of a visit and sends a unique visit ID to the middle along with the ID of the circuit used to carry the first HTTP request. When we parse the logs, we discard all other circuits built to fetch that onion site. We need these IDs so that we can parse only the cells that go through that first circuit and discard cells to other circuits. (Recall from Section II-C that our middle relay would miss third party onion service circuits, whereas the entry relay will be able to record all clients' circuits.) In fact, due to Tor's stream isolation, the middle relay has the advantage that traffic to the first-party onion service will not blend with traffic to other sites, eliminating the need to use special parsing techniques [31]. Note that our custom signaling cells will be present in the client `tshark` logs, however, we filter out these artificially added cells before classifier training.

The list of URLs that we crawl has been obtained from the ahmia[7] Tor onion service search engine maintainers. Before starting the crawls, we used `torsocks` and `curl` to remove from the list onion sites that were down. We have removed all the screenshots after error detection to avoid keeping illegal data on our hard drives. In total we ran four middle relays to crawl 5,000 different onion websites in parallel. After removing failed visits and thresholding so that all websites had the same number of instances, the dataset ended up having 2,500 onion sites and 80 instances per site.

C. Ethics

1) Safety: We have tested our Tor source code modifications using Shadow [18]. However, Shadow does not run the Tor browser crawler that we require to crawl onion services in the website fingerprinting experiments. In order to capture the complexities of the Tor Browser, and also to evaluate our attacks under realistic background traffic and network congestion conditions, we conduct our experiments in the live Tor network.

The signaling mechanism described in V-A1 ensures that we only collect traffic generated by our crawler. Using OnionPerf, we log only the events associated with the traffic generated by our client. Thus, analysis and potential attacks will be applied only on traffic data generated by our own visits.

Following the principle of data minimization, our middle relays only collect traffic data attributes strictly necessary for applying traffic fingerprinting attacks. We ignore the payload of network packets captured at our client, as they are encrypted and are not useful for fingerprinting purposes. The HTML sources and screenshots are also removed after the error detection and outlier removal phases.

2) Benefits and Risks: Since we are not collecting any data of regular Tor users, there is no de-anonymization risk from our traffic datasets. There may be a small indirect risk of leaking user personal data in the screenshots and HTML sources, but they were deleted before publication, after being used for the integrity checks of our traffic dataset.

[7] https://ahmia.fi/

TABLE III. 10-FOLD CROSS-VALIDATED ACCURACIES FOR THE THREE STATE-OF-THE-ART ATTACKS ON OUR *client-side* TCP TRACES. THE EVALUATIONS ARE CLOSED WORLDS OF 10, 50 AND 100 ONION SITES.

Num sites	k-NN (%)	k-FP (%)	CUMUL (%)
10	95% ± 0.03	95% ± 0.06	92% ± 0.04
50	75% ± 0.02	85% ± 0.03	81% ± 0.02
100	67% ± 0.01	68% ± 0.03	64% ± 0.02

With respect to the impact of our experiments on Tor's performance, the volume of the traffic generated by our crawls is comparable to that from a regular user actively browsing the Web for a few hours. We do not expect a significant impact on network performance.

Our methodology allows us to explore one of the main research questions in this work: whether fingerprinting is effective in the middle position of our circuits. In addition, it will help us compare the effectiveness of these techniques at different layers of the network stack (*i.e.*, the application layer and the transport layer). Previous studies only applied WF on TCP packets and used heuristics to filter cell types that are not useful for fingerprinting (*e.g.*, `SENDME` cells). Our middle relays have access to cell information and thus can directly utilize or filter control cells that are not related to a website.

D. Results

1) Website Fingerprinting Effectiveness at the Middle: Here we explore the following research question: *how effective is website fingerprinting at the middle with respect to the client*? Specifically, we design an experiment to determine whether the accuracy of onion service fingerprinting is affected by the position in the circuit (*i.e.*, middle relay as compared to the entry link).

We follow the methodology outlined in the previous section to obtain two datasets: (i) TCP traces as collected between the client and the entry guard and (ii) cell traces as collected from the middle relay. Both sets of traces were collected at the same time to avoid confounding variables like changes of the website over time [20]. To evaluate the effectiveness of website fingerprinting at the middle, we apply the state-of-the-art techniques on both datasets and compare the success rates.

Table III shows the accuracy scores for three classifiers on the network traffic data collected at the client. The accuracy is defined as the number of True Positives—test instances that have been correctly classified—over the total, also known as True Positive Rate (TPR) or Recall.

As we see in the table, k-FP outperforms the other techniques by a small margin followed by CUMUL and, lastly, k-NN, the least accurate technique. These accuracies are consistent with existing evaluations of these techniques on onion service sites [6]. We also evaluated the techniques on existing datasets [29] to make sure that we are able to reproduce previous evaluations on regular sites and that we do not introduce errors that stem from our methodology; we did not find any major discrepancies from previous results.

On the other hand, Table IV shows the accuracies the techniques achieved when applied on the cell traces collected from our middle relay. In this table we see that the classifiers are ranked in the same order as in the client: k-FP being the most accurate and k-NN the least accurate. With respect to the accuracies obtained at the client for the same classifier,

TABLE IV. 10-FOLD CROSS-VALIDATED ACCURACIES FOR THE THREE STATE-OF-THE-ART ATTACKS ON OUR DATASET OF CELL TRACES COLLECTED AT THE *middle relay*

Num sites	k-NN (%)	k-FP (%)	CUMUL (%)
10	91% ± 0.03	100% ± 0.00	99% ± 0.03
50	73% ± 0.01	91% ± 0.01	86% ± 0.03
100	68% ± 0.01	76% ± 0.02	76% ± 0.02
500	64% ± 0.00	72% ± 0.01	66% ± 0.01
1,000	59% ± 0.00	56% ± 0.01*	63% ± 0.01

* Due to RAM constraints we were not able to evaluate k-FP using the optimal parameters for 1,000 sites which reduced classifier accuracy.

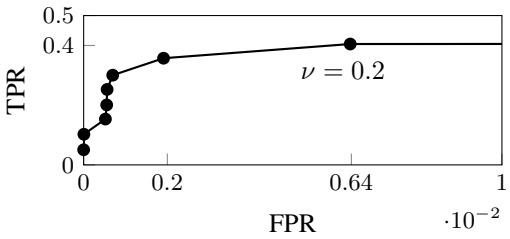

Fig. 6. ROC curve to optimize the ν parameter. We can see that $\nu = 0.2$ makes a reasonable trade-off between TPR and FPR. To deal with extreme base rates, it is possible to minimize the FPR at the expense of the TPR.

we see some interesting differences: while k-NN has a few percent points decrease in accuracy with respect to the entry link scenario, both k-FP and CUMUL perform a few percent points better when they are applied at the middle. This is plausible because each classifier uses different features that may be more or less robust to timing and order differences between both positions. The accuracy improvement can be explained by the fact that we used TCP traces at the client, whereas the middle dataset includes cell traces, conveying a higher amount of information and including less noise than TCP traces [30].

Another interesting observation is that discarding all third-party circuits for training and testing does not impact classifier accuracy. We attribute this result to the low prevalence of third party embedded content in onion services (it has been found on a large dataset of onion services to be less than 20% overall [6]).

2) Open world scenario: The preceding analysis and results are applicable to an idealized closed-world scenario where we try to identify known and trained-on onion websites. We now consider a more realistic and challenging open-world setting—one where unseen and unknown onion websites may be introduced at testing time. We now present an enhancement of our website fingerprinting techniques for this scenario.

In other recent open-world evaluations, the classifier is only trained on a small fraction of the web pages in the world. In this setting, the user may visit any page in the world, including pages of which the classifier is not aware. These works define the open world evaluation as a binary problem: the positive class is formed by a set of *monitored* pages that the classifier aims to identify and the negative class by the remaining *non-monitored* pages [4], [15], [20], [21], [26], [29]. During training, the classifier is shown examples of both monitored and non-monitored pages; however, the majority of the pages in the non-monitored set are not present in the training set.

In this paper we have approached the open world differently. There is no strong support to believe that the non-monitored samples used for training necessarily represent the whole world of non-monitored pages because the sample that is taken to train the classifier is small compared to the population, *i.e.*, all pages that could possibly be visited. This sample may bias the classifier toward a specific choice of non-monitored pages selected for training or not actually help the classifier discriminate monitored from non-monitored sites. Instead, we propose to model the open world as a *one-class* classification problem: the classifier only takes instances for the monitored class and draws a boundary between the monitored pages and *all other* pages.

In particular, we have taken the monitored set to be composed by one single web page—the best-case scenario for the adversary in such an open world. We have collected 5,000 instances of a popular social network service (SNS) that is deployed as an onion site and use 3,750 of the samples to draw the decision boundary and 1,250 for testing. We have used 200,000 instances of 2,500 different sites (80 instances per site) for testing the one-class classifier for onion service pages that are not the SNS.

For the one-class classifier we have used `sklearn`'s implementation of one-class SVM with a radial basis function. The one-class SVM is parameterized on ν which defines an upper bound on the fraction of training errors; ν can be used to adjust the trade-off between False Positives and True Positives. We plotted the ROC curve to find a value of ν that maximizes the number of True Positives while keeping a low False Positive Rate. In Figure 6 we can see that $\nu = 0.2$ achieves such a compromise, providing a FPR lower than 1% while the TPR is slightly higher than 40%.

We have chosen a subset of CUMUL's features because they are also used in an SVM for the closed world problem [26]. After analyzing different subsets, we found that a combination of the first and last interpolation points of the cumulative sum can separate SNS instances from the rest. In particular, we used the second interpolation point (CUMUL's 5th feature), describing the first region of packets in the original trace, and the 87th one (CUMUL's 90th feature), which described mid- and end-regions of the trace.

The results are presented in Figure 7 which shows a projection of the classification space on these two features. The orange plus marks are the SNS's training instances, the purple empty circles are the SNS's testing instances, and black filled dots are "Others" instances. The decision boundary learned by the classifier is depicted by the black line.

As we observe in Figure 7, there are many testing samples of SNS that fall outside the boundary. This is because we tuned the classifier to minimize the number of False Positives—instances of non-SNS pages that are classified as SNS. This way we achieve a False Positive rate below 1%, but this has a cost of a large number of False Negatives: the True Positive rate is 40%. The reason we have optimized for low FPR instead of TPR becomes is because we are interested in realistic deployments where the base rate of the positive class becomes relevant, as we discuss next in Section VI-E.

E. Precision is in the detail

The base rate is the probability that a site is visited, and can also be interpreted as site popularity. Previous work has discussed the importance of the effect of the base rate

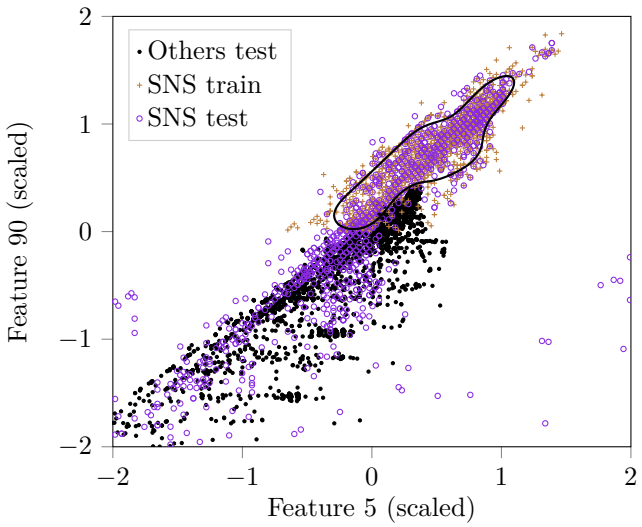

Fig. 7. Projection over two CUMUL features of the one-class classification instances. The plus sign marks are instances used for training the classifier, the circle marks are SNS instances used to test the positive class and the cross marks are instances that belong to non-SNS sites used to test the negative class. The black line shows the boundary that was learned by the classifier that minimizes False Positives.

of the positive class (*i.e.*, the SNS) on the Precision of the classifier [20], [21]. Precision is proportional to the number of samples that our classifier detected as SNS that are actually SNS. In other words, Precision is an estimate of the probability that the classifier was correct when it guessed SNS.

Prior work has pointed out that if the base rate of the positive class (*i.e.*, the SNS) is orders of magnitude lower than the negative class (*i.e.*, Others), the False Positive Rate (FPR) has to be negligible so that the classifier can perform with sufficient Precision [20]. Since we cannot estimate the base rate of the SNS's onion site directly for ethical and privacy reasons, we have evaluated the Precision of our one-class classifier for several hypothetical base rates.

In Figure 8, we show the Precision, the TPR, the FPR and training error (*i.e.*, ν). In the graph we see that all of the metrics are fixed for the whole range of considered base rates, while precision decreases exponentially when the base rate of the SNS tends to zero. The vertical dashed line indicates the base rate (1%) where Precision is 50%; the point where the classifier is correct only half of the time. These results are comparable to previous work that evaluated the precision of the CUMUL classifier and achieved similar results [26].

We further analyzed the sites that the classifier misclassified most often. The distribution of errors over the sites is shown in Figure 9. We observe that 80% of the errors are concentrated over 12 of the sites and only 3% of the total number of sites are responsible for 100% the misclassification. Based on this observation, we argue that it is possible that even for 1% FPR (see Figure 8), the classifier may have greater precision if those 12 sites that are responsible for most of the errors are less popular. Note that we assumed a uniform distribution of the sites that belong to the Others sites. Further, it may be possible to design dedicated classifiers that learn to distinguish between the SNS and each of these 12 sites individually in order to reduce the number of False Positives that the classifier incurs overall.

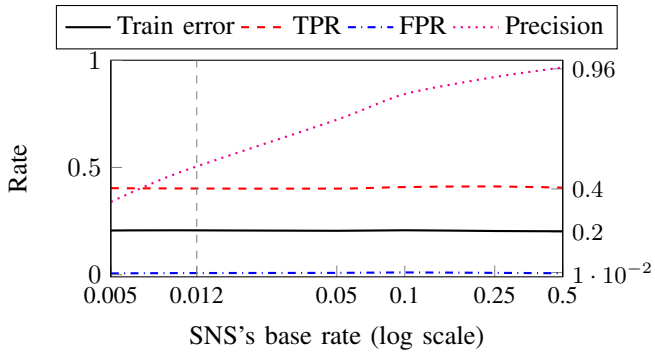

Fig. 8. Performance metrics for one-class open world for the SNS's base rates ranging from 0.5% to 50%. The vertical dashed line shows the point in which Precision is 50%.

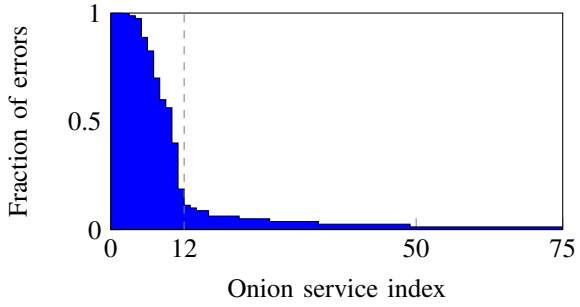

Fig. 9. Sites that were confused with the SNS at least once during the classification (2,443 sites had zero errors). Note that the distribution is heavily skewed and 80% of all the errors are concentrated in the 0.5% (12) of the sites (see vertical dashed line) and 3% of the sites (75) include all the errors.

We manually checked the 12 sites that were misclassified as the SNS some weeks after we crawled them. Five of the sites were offline and one of them had been seized by the German Federal Criminal Police. The remaining sites were up and are of a diverse nature: one is a personal homepage, two are movie streaming sites, another is a porn site, one is a hacking page and, surprisingly, the last one is the download page for the SecureDrop anonymous whistle-blowing software (`secrdrop5wyphb5x.onion`) run by the Freedom of the Press. We believe that they were confused with the SNS's onion service page due to similarities in page size.

VII. Onion Service Popularity Measurement

We showed in Section V how a relay can predict that it is serving as a middle (in the R-C-M1 position) and on a rendezvous onion service circuit with high confidence, and we showed in Section VI how website fingerprinting techniques can be used to accurately predict which onion service webpage is visited. In this section, we validate our previous results and show the practicality of the techniques that we developed through a privacy-preserving measurement of the popularity of a social networking site (SNS) accessible as an onion service.

A. Measurement Goals and Methodology

Tor circuit and website fingerprinting techniques have thus far been discussed in the literature in the context of client deanonymization attacks. The goal of the measurement study in this section is to show how to use the classification techniques presented in the previous sections not for client deanonymization, but to predict accesses to and safely measure

the popularity of an onion service SNS. In this study, we seek to: (i) develop a reusable framework for safe onion service popularity prediction and measurement; (ii) validate our classification techniques from the previous sections by running them in real time in a realistic public Tor network environment on live Tor traffic (something that has never been done before to the best of our knowledge); and (iii) show how our proof-of-concept measurement framework can be used to discover the popularity of an onion service in the open-world. Note that doing this measurement safely is a primary goal that is further discussed below in Section VII-D.

Achieving these goals involves several components. First, we run middle relays in the Tor network that provide resources to Tor users. Second, our relays must predict which circuits in which they are serving are onion service circuits (specifically, rendezvous circuits since those are used to access web content). Third, our relays must predict when they are in a middle position of circuits in which they are serving (specifically, the R-C-M1 position since our classifiers were trained for that position). Finally, our relays must predict which of the predicted rendezvous circuits in which they predict the R-C-M1 position are used to access the SNS. We next explain the tools that we built, modified, and used to realize these goals.

B. Measurement Tools

We enhance PrivCount [19], an existing privacy-preserving Tor measurement tool,[8] to use a new prediction library that we developed to allow us to make predictions in real time on real Tor relay traffic.

1) PrivCount Overview: PrivCount is a distributed measurement tool, based on the secret-sharing variant of PrivEx [11], that can be used to safely measure Tor. PrivCount works by extracting events from a Tor process and then counting the occurrence of those events across a set of relays. PrivCount consists of a tally server, several share keepers, and several data collectors (one for each relay in a set of measurement relays). The tally server acts as a centralized, but *untrusted*, entity that is primarily used as a proxy to facilitate data transfer between the other nodes and as a single aggregation point following a completed measurement phase. Each data collector initializes each counter that it is configured to count with *differentially-private noise*, and also with *random blinding values* that are secret shared to each share keeper. After initialization, each counter on each data collector and share keeper will appear to hold a random value. The data collectors each connect to their configured Tor relay, extract events, and increment the configured counters when appropriate. At the end of a measurement phase, the data collectors and share keepers send their counter values to the tally server for aggregation. After aggregation, the blinding values that were stored on the share keepers during the measurement will cancel out with the blinding values that were added at the data collectors, and the final output will be the sum of the true counter value and the differentially-private noise that was added by the data collectors.

The noise that is added to each counter protects users under differential privacy [9], and the blinding values provide for secure aggregation *across* measurement relays. No data collector can learn anything about the counters of other relays not also controlled by the same operator, and individual data collector contributions to the final aggregated counter values are hidden (by the random blinding values) as long as at least one share keeper is honest. Jansen and Johnson provide additional PrivCount details and proofs of PrivCount's security and privacy properties [19].

2) Enhancing PrivCount: PrivCount supports a wide range of statistics (*e.g.*, number of circuits, amount of data transferred, etc.), and we enhanced it to support counting the real-time predictions of a circuit's purpose, a relay's position in a circuit, and the onion page being accessed in a circuit. To enable these predictions, we utilize a version of Tor that has been modified to allow for circuit signaling as described in V-A1, and to export the same circuit and cell meta-data that we discussed in V-A2. We use circuit signaling to collect ground truth during the measurement while separating ground truth circuits that we created from regular circuits that we did not (see Section VII-E for more details).

We developed a new library for PrivCount called `onionpop`[9] that implements the classifiers needed for prediction. The onionpop library extracts the features we need for each of the three classifiers from Tor circuits and cells, and wraps the python `sklearn` and `pyarborist` APIs to train the classifiers and predict purpose, position, and webpage. We train our models to make binary predictions of when the purpose is rendezvous, position is R-C-M1, and webpage is the front page of our SNS of interest. We added new counters to PrivCount to record the results of the predictions.

Due to its sensitive nature, we do not log any information to disk from circuits that we did not originate ourselves. Therefore, a significant concern during the development of our prediction library is that PrivCount will need to process cell information from fast Tor relays in real time. For safety, we store circuit and cell meta-data in RAM only for the lifetime of the circuit; when the circuit ends, we run our predictions, increment counters to count the results, and then clear the corresponding circuit and cell meta-data from RAM. This is consistent with PrivCount's data storage model, however, it requires that we process and store a potentially large number of cells. To mitigate potential memory and computational resource bottlenecks, we implement a configurable hard upper limit on the number of cells that we store per circuit (well above the amount we need to distinguish the SNS) and only process a subset of the circuits on our relays by sampling circuits uniformly at random according to a configurable sample rate.

C. PrivCount Deployment

We set up a PrivCount deployment with 1 tally server, 3 share keepers, and 17 data collectors each connecting to a distinct Tor relay. These nodes were distributed among 3 operators and hosted in 3 countries (Canada, France, and the United States). Each of the relays ran our modified version of Tor, and each of the tally server, share keepers, and data collectors ran our modified version of PrivCount.[10]

1) Privacy: Our PrivCount deployment uses the parameters and privacy budget allocation techniques set out by Jansen and Johnson [19]. Specifically, we use differential privacy parameters $\epsilon = 0.3$ (which has also been used by Tor [13]), and $\delta = 10^{-3}$ (which is an upper bound on choosing a noise value that violates ϵ-differential privacy). Our deployment provides

[8] https://github.com/privcount

[9] https://github.com/onionpop/onionpop

[10] All framework components are available at https://github.com/onionpop.

TABLE V. DAILY ACTION BOUNDS FOR PRIVCOUNT DEPLOYMENT

Action	Bound
New general-purpose circuits	90
New rendezvous circuits	48
Client-side rendezvous circuits	24
Server-side rendezvous circuits	24
Client-side rendezvous circuits to SNS	2

TABLE VI. COMBINED POSITIONAL RELAY BANDWIDTH BY PERCENT FOR PRIVCOUNT DEPLOYMENT

Round	Guard	Middle	Exit	Intro.	Rend.
Measurement 1	1.15%	0.78%	3.52%	0.88%	0.78%
Measurement 2	1.30%	0.87%	3.11%	0.99%	0.87%
Measurement 3*	1.03%	0.68%	*0.0%	0.77%	0.68%

* To mitigate potential resource issues, exit relays were excluded from measurement 3 (the classification round) since they would not have contributed to middle relay onion service prediction counters.

privacy according to the daily action bounds shown in Table V, which are all based on circuit counts since that is what our deployment will measure; users whose actions stay below these bounds will be protected under differential privacy. We protect users who use 90 or fewer general-purpose circuits per day, which could be used to access one site every ten minutes for 8 hours plus 10 additional circuits. We protect users who use 48 or fewer rendezvous circuits per day when not distinguishing between client-side or server-side, and otherwise 24 or fewer each of client-side and server-side rendezvous circuits per day: 75 percent of the onion sites we crawled (Section VI) used 4 or fewer circuits, and so the number of circuits we protect could be used to access 1 onion site every 10 minutes for one hour.

2) Measurement Rounds: We ran three different 24-hour long measurement rounds. The first round of measurements was used to calibrate the noise added to our counters. We used previously published measurements of Tor activity [19] to allocate our privacy budget across the configured counters. We then measured general and onion service circuit usage from different relay positions to obtain updated estimates of circuit activity, which we used to adjust the allocation of our privacy budget in the subsequent rounds.

In the second measurement round, we focused on measuring the number of direct connections from the SNS of interest to our relays serving in the rendezvous position based on the IP address and autonomous system (AS) number of the SNS. This was possible because the SNS runs a *single onion service* that connects directly to the rendezvous point rather than a normal onion service which builds a three-hop circuit to connect. This set of measurements allow us to verify our circuit purpose and position classifiers.

In the third measurement round, we enabled our classifiers and focused on counting the results of the predictions. We also configured a crawler under our control to access the SNS in order to assert that our deployment was working properly and to cross check our prediction results (the prediction results for our crawler's circuits were kept separate from the results for other circuits). During round three only, we configured a circuit sample rate of 0.12 and excluded our exit relays from the measurement (since they would not contribute to the middle relay prediction counters) in order to prevent resource bottlenecks in our deployment pipeline. The percentage of Tor network bandwidth that the relays in our deployment controlled during each measurement round is shown in Table VI.

D. Research Ethics and User Safety

Our measurement study explicitly prioritizes user safety as a primary goal. We practice data minimization, limit measurement granularity, and provide additional security to the measurement process as described above. We have incorporated feedback from the Tor Research Safety Board[11] into our methodology: on suggestion of the board we created a website explaining our study[12] and linked our measurement relays to it, and we informed the SNS of our intentions to measure their site (although we did not receive a response from any of the employees of the SNS).

Because the main classification-based measurements are done from middle relay positions, onion-encryption technically prevents us from learning any client-identifying information. Although this protects users to some extent, we further protect users by utilizing the state-of-the-art in safe Tor measurement tools and techniques. Specifically, we use PrivCount and the techniques set out by Jansen and Johnson [19] and Elahi *et al.* [11] to provide differential privacy and securely aggregate measurements across all of our relay data collectors.

The PrivCount counters are initiated to noisy values to ensure differential privacy is maintained, and are then blinded and distributed across several share keepers to provide a secure aggregation process. At the end of the process, we learn only the value of these noisy counts aggregated across all data collectors, and nothing else about the information that was used during the measurement process. Specifically, we do not learn relay-specific inputs to the final counter value, and client usage of Tor during our measurement is protected under differential privacy.

Importantly, we chose to show our proof-of-concept by only predicting accesses to a single onion site that we believed had non-trivial usage and that already has implied that it does not require anonymity by running a non-anonymous single onion service. We explicitly chose *not* to measure additional regular onion sites because: (i) we did not believe it was necessary to show the effectiveness of our techniques; (ii) we wanted to avoid leaking more information than necessary about specific onion site usage; and (iii) running a hidden onion service would imply that anonymity is required or at least desired by the service.

E. Results

In addition to measuring the results of our classifiers, we also focused our PrivCount deployment on direct measurements that would allow us to validate our classification results.

1) Direct Measurements: The direct measurement results are shown in Table VII. We measured the number of observed circuits on our relays from the circuit entry, middle, and end (including various types of rendezvous circuits). Our measurements indicate a significantly lower number of onion service rendezvous circuits compared to non-onion service circuits, as expected. While we discuss how these measurements give us an idea of popularity below, here we note that there are more than an order of magnitude fewer rendezvous circuits compared to non-onion service circuits.

Because there are many circuits built in Tor over the period of a day, the relative accuracy of our direct measurements is quite high: most of the 95% confidence intervals lie between

[11] https://research.torproject.org/safetyboard.html
[12] https://onionpop.github.io

TABLE VII. RESULTS FOR DIRECT MEASUREMENT OF ONION SERVICE PROTOCOL

Circuit Count Description	Count ± 95% CI
Entry	20,351,667 ± 3.45%
Middle	16,212,157 ± 4.33%
End (Exit + Rendezvous + etc.)	18,904,815 ± 3.71%
Rendezvous (Client or Service)	272,180 ± 5.15%
Rendezvous Client	136,191 ± 5.15%
Rendezvous Service	136,874 ± 5.12%
Rendezvous Service to SNS ASN	718 ± 91.64%
Exit + Rendezvous Client	11,327,103 ± 6.19%
Exit + Rendezvous Service	11,394,600 ± 6.16%

TABLE VIII. RESULTS FOR MEASUREMENT OF ONION SERVICE CLASSIFIER DETECTION

Classifier	# Positives	# Negatives
Purpose is Rendezvous*	114,762±28.54%	2,444,166±79.82%
Ground Truth Tests**	645 (100%)	0 (0%)
Position is R-C-M1*	49,679±32.99%	68,022±48.15%
Ground Truth Tests**	623 (96.5%)	22 (3.4%)
Site is SNS*	10±1200%	45,376±36.12%
Ground Truth Tests**	374 (60.0%)	249 (40.0%)

* These values may appear lower than expected because we sampled circuits at a rate of 12% due to resource constraints.
** The ground truth tests were run with a crawler accessing the SNS during measurement, so these values represent true positives and false negatives.

3 and 6 percent. The one outlier is the direct measurement from the rendezvous node position of connections from the SNS ASN, which we can use to measure its popularity since this particular SNS runs a single onion service. The confidence interval is higher than expected (91.64%) which indicates that the SNS onion service is much less popular than expected, with potentially fewer than one hundred accesses through our relays during our measurement period.

2) Classifier Measurements: A primary purpose of our measurement is to test the ability of our classifiers to detect when a relay serves on a rendezvous circuit, in the R-C-M1 position, and if it can identify accesses to a site of interest (SNS in our case). To do this, we send circuit meta-data (including cell meta-data for cells transferred on the circuit) to our classifiers when the circuit ends and record the detection results. We also run a crawler that creates rendezvous circuits through our middle relays during our measurement. The circuits created by our crawler provide ground truth that we can use to evaluate the classifiers' true positive and false negative rates.

Our classifier detection measurement results (including our ground truth crawler tests) are shown in Table VIII. We again see a similar trend in that an order of magnitude fewer rendezvous circuits are detected compared to non-rendezvous circuits. With these measurements, there is a significant amount of noise associated with our measurements; this is primarily because we added the full amount of noise to provide differential privacy while at the same time sampling only 12% of circuits due to resource constraints. This has significantly increased the relative noise in our measurements. As in our direct measurements, the low number of SNS circuits has also caused our measure of the number of positive SNS detections to appear insignificant due to the large confidence interval associated with the noise that we added in order to protect privacy.

Our ground truth measurements show that the true positive rate for the purpose classifier was 100%, the true positive rate for the position classifier was 96.5% while the false negative rate was 3.4%, and the true positive and false negative rates for the SNS classifier were 60% and 40%, respectively. With these results, we are optimistic that our classifiers are functioning as intended. We assert that an adversary who is not concerned with privacy (and does not add noise) would be able to make much more precise measurements than we describe here.

TABLE IX. LIKELY ONION SERVICE POPULARITY BY FRACTIONS OF CIRCUITS OF VARIOUS TYPES

Description	Method	Popularity
Onion Service Popularity (as % of non-onion circuits)		
Rendezvous / Entry	Direct	1.34%
Rendezvous / End	Direct	1.45%
Rendezvous Client / Exit + Rendezvous Client	Direct	1.20%
Rendezvous Service / Exit + Rendezvous Service	Direct	1.20%
Purpose is Rendezvous / Total	Classified	4.48%
SNS Popularity (as % of onion circuits)		
Rend. Service to SNS ASN / Rend. Service	Direct	0.52%
Site is SNS / Total	Classified	0.02%

3) Popularity: We estimate the popularity of the onion service protocol by computing the fraction of middle relay circuits that are rendezvous circuits. Middle relays that do not serve as the rendezvous point on a circuit cannot determine with certainty whether or not the circuit is a rendezvous circuit, but they can predict it by running our circuit purpose classifier. As previously discussed, we also measure the popularity of the onion service protocol independently and directly when our relays do serve as rendezvous points, since in that case our relays can distinguish client-side and server-side rendezvous circuits from others.

Our popularity estimates are shown in Table IX. The entries in the table show several ways one could estimate popularity, with our classification-based estimates at the bottom of each section. The direct measurement approaches indicate that onion service popularity is between 1% and 1.5% based on circuit counts; for comparison, 0.9% of Tor traffic by volume (*i.e.*, bytes) is onion service traffic (900 Mbit/s onion[13] of 100 Gbit/s total[14]) according to Tor metrics. Our classification-based estimate is a bit higher at 4.48%, but we note this result includes noise and an unknown number of false positives. Similarly, our direct measurement of accesses to the SNS onion site front-page is 0.52% of rendezvous service circuits whereas our classification-based estimate is 0.02%.

F. Discussion

Our laboratory results from the previous sections show that WF at the middle relay position is just as effective w.r.t. recall and precision as has been shown from the guard position in previous works—both for closed- and open-world scenarios. On the other hand, our real-world results indicate that the base-rate of the site we chose (*i.e.*, the SNS) was too low for our classifier to provide high confidence for its counter. However, what we *can* learn with high confidence is that the popularity of the SNS as an onion service is almost negligible when comparing SNS onion service circuits to all other onion service circuits. This result was unexpected: our intuition for picking this particular SNS was that it is known to be one of the most popular websites in the world. Our results show that a much lower FPR—up to two orders of magnitude lower—is necessary for WF to be useful in measuring individual onion sites.

[13] https://metrics.torproject.org/hidserv-rend-relayed-cells.html
[14] https://metrics.torproject.org/bandwidth.html

VIII. Related Work

Although there are many studies that explore the extent to which traffic analysis can leak information on Tor [12], [23], [24], here we focus on website and onion site fingerprinting.

A. Tor Website Fingerprinting Attacks

The first WF attack on Tor was proposed and evaluated by Herrmann *et al.* and only achieved 3% success rate [15]. This research area has since seen great activity and the latest WF studies achieve more than 90% accuracy under specific conditions and scenarios [4], [14], [26], [28], [29], [30].

Recent work attempted to address WF on non-onion websites in an *open world* model and increases the scalability of evaluation approaches [26], but the *closed world* model has been considered realistic for the evaluation of WF on onion services [6] due to their limited number. It has been shown that a local and passive adversary can effectively detect onion service visits using circuit fingerprinting, and then apply website fingerprinting methods to infer to which website they belong [22]. Errors when classifying onion service websites have been explored in order to further improve WF techniques [25], but the practicality of monitoring a realistic number of sites even in the smaller onion service world is still in question [27].

To the best of our knowledge, we are the first to apply circuit, position, and WF techniques from middle relays, and we are the first to use our classification techniques on traffic initiated from real Tor users. While we apply our techniques for measurement purposes, recent work has shown how our techniques can be used to further target specific users [17].

B. Tor Website Fingerprinting Defenses

Several defenses have been designed to mitigate WF attacks. Most of these defenses are based on link padding [10], [21], [29], that is, adding dummy messages that are indistinguishable from real ones in order to make the features that WF exploit ineffective. Prior work assumes that the middle collaborates in the defense and removes the padding, in which case our techniques would not be affected. End-to-end padding that does not depend on collaborating Tor infrastructure [6] could disrupt the traffic analysis techniques we leverage, but would come at a prohibitively-high performance cost.

Restricted routing—*e.g.*, if middles were chosen and used long term as is currently the case with guards—would limit the number of users from which a middle could observe circuits. In that case, middles could lose much of the advantage over guards as preferential observation points.

C. Onion Site Enumeration

Existing HSDir lookup protocols have been shown to be vulnerable to attack by an adversary running low-bandwidth relays [3]. By exploiting the lookup protocols, an adversary running an HSDir can directly measure the popularity of the onion services whose addresses are assigned to it. Previous work has used this approach to better understand the popularity of content in the onion service ecosystem [2]. These attacks can be mitigated by changes in the HSDir protocol.

Tor is currently deploying next-generation onion services in order to limit the effectiveness of onion service enumeration attacks, but the planned defenses will not significantly change the flow of cells through a circuit (like padding does) and therefore we believe that they will not significantly affect the accuracy of our techniques.

IX. Conclusion

We have shown that a significant amount of information is leaked to middle relay positions, although the extent of this threat is often overlooked. We describe how the design of Tor admits to middle relays a wider visibility over all users of the network because clients pick new middle relays for every circuit that they build. We have shown through extensive data collection and experimentation that traffic analysis techniques are as effective from internal middle positions as they are from ingress and egress (guard and exit) positions. In particular, we have built a traffic analysis pipeline that can detect a relay's position in a circuit, the purpose of the circuit, and identifies the onion service being accessed through a circuit. We have then put the pipeline into practice to measure the popularity of a well-known social network onion service: we are the first to apply these traffic analysis techniques on real Tor user traffic to the best of our knowledge. Although our measurement results are constrained in scale and accuracy due to resource and ethical concerns (constraints not shared by malicious actors), our framework provides the means to study effective mitigation to potential threats and to gather additional measurements.

A. Lessons Learned

It is clear that more progress needs to be made and this present work provides positive first steps in that direction. We anticipate that classification techniques at middle relay positions will not deteriorate and point out some of the challenges to deploying them in the real-world. First, our pipeline was created in order to reduce the number of circuits that need to be processed by the WF classifier; we filter out real user circuits for training and non-onion service circuits with the circuit classifier during testing. This greatly reduced the overhead both in training and testing and improved our results. Therefore, careful filtering and data pre-processing are keys to successful real-world deployments. Second, our measurement was done in real-time: everything was kept in RAM, and we used a low circuit sampling rate of 0.12 due to computational and memory limitations. We found that real-world scale may overwhelm available resources and pragmatic compromises may need to be made. Third, we were very concerned with user safety in our real-world measurements and hence our results are noisy. Depending on the use-case (*e.g.*, a malicious actor), noise may not be necessary; removing this requirement would reduce the operational overhead of running the privacy-preserving apparatus and may allow higher sampling rates.

B. Future Work

Using our current WF classification pipeline, an adversary could target the guards that originate connections to websites of interest (*e.g.*, the SNS). We have shown that there are a small number of SNS circuits, and therefore the set of guards used to access the SNS would also be small. An adversary could reduce the time and cost of a targeting attack by focusing on only these guards rather than, *e.g.*, compromising guards at random and waiting until it is used to access a website of interest. Some related target attacks that depend on our techniques have recently been explored [17].

An alternative to compromising guards that route interesting connections is locating the originating client or destination onion service using middle relay network latency measurements. Hopper *et al.* [16] show the effectiveness of such attacks from malicious websites. Mapping latency between an adversarial middle and all Tor relays (or at least the

most popular) [5] would assist in narrowing the network and geographic location of circuit originators (*e.g.*, to a region or possibly a country).

An adversary could fingerprint protocols instead of websites to target a broader base of users. For example, a censoring regime may fingerprint Tor's pluggable transports (PT) from the middle relay positions. Fingerprinting PTs from the client-side—which is the current state-of-the-art—has a high FPR since PTs are designed to be confused with other protocols that the censor is reluctant to block. In contrast, fingerprinting PTs at a middle does not provide this same confusion since only Tor traffic is present in the Tor network and the protocols that the censor is reluctant to block (*e.g.*, HTTPS) are not present. Assuming that the censor already has the ability to identify users on the client-side, the censor could greatly reduce the incidence of false positives in detecting PT circuits. Furthermore, using timing correlations between client-side observations could also identify PT users, and an adversary could use our fingerprinting techniques to identify which websites PT users visit.

Acknowledgments

We thank the anonymous reviewers for their helpful feedback. We thank Roger Dingledine from the Tor Research Safety Board for providing feedback that helped to make our measurement safer and more transparent. We thank Tim Wilson-Brown for the suggestion to directly measure the SNS, for implementing much of the Tor and PrivCount code that we used during our measurement, and for the helpful suggestions and feedback on our approach. We thank Tim Wilson-Brown and Matt Traudt for operating Tor relays and PrivCount nodes as part of our PrivCount deployment.

This work has been partially supported by the Defense Advanced Research Project Agency (DARPA) and the National Science Foundation (NSF) under grant number CNS-1527401. This material is based upon work supported by the European Commission through KU Leuven BOF OT/13/070, H2020-DS-2014-653497 PANORAMIX, and H2020-ICT-2014-644371 WITDOM. Juarez is supported by a PhD fellowship of the Fund for Scientific Research - Flanders (FWO), and Elahi is supported by NSERC through a Postdoctoral Fellowship Award, the Research Council KU Leuven: C16/15/058. The views expressed in this work are strictly those of the authors and do not necessarily reflect the official policy or position of any employer or funding agency.

References

[1] "TC: A Tor control protocol (Version 1)," https://gitweb.torproject.org/torspec.git/tree/control-spec.txt.

[2] A. Biryukov, I. Pustogarov, F. Thill, and R.-P. Weinmann, "Content and popularity analysis of Tor hidden services," in *International Conference on Distributed Computing Systems Workshops*, 2014.

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

[4] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson, "Touching from a Distance: Website Fingerprinting Attacks and Defenses," in *Conference on Computer and Communications Security*, 2012.

[5] F. Cangialosi, D. Levin, and N. Spring, "Ting: Measuring and exploiting latencies between all tor nodes," in *Internet Measurement Conference*, 2015.

[6] G. Cherubin, J. Hayes, and M. Juarez, "Website Fingerprinting Defenses at the Application Layer," in *Proceedings on Privacy Enhancing Technologies*, 2017.

[7] R. Dingledine, N. Hopper, G. Kadianakis, and N. Mathewson, "One fast guard for life (or 9 months)," in *Workshop on Hot Topics in Privacy Enhancing Technologies*, 2014.

[8] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second-Generation Onion Router," in *USENIX Security Symposium*, 2004.

[9] C. Dwork, "Differential privacy," in *International Colloquium on Automata, Languages and Programming*, 2006.

[10] K. Dyer, S. Coull, T. Ristenpart, and T. Shrimpton, "Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail," in *Symposium on Security and Privacy*, 2012.

[11] T. Elahi, G. Danezis, and I. Goldberg, "Privex: Private collection of traffic statistics for anonymous communication networks," in *Conference on Computer and Communications Security*, 2014.

[12] N. S. Evans, R. Dingledine, and C. Grothoff, "A practical congestion attack on tor using long paths." in *USENIX Security Symposium*, 2009.

[13] D. Goulet, A. Johnson, G. Kadianakis, and K. Loesing, "Hidden-service statistics reported by relays," Tor Project, Tech. Rep., April 2015.

[14] J. Hayes and G. Danezis, "k-fingerprinting: a Robust Scalable Website Fingerprinting Technique," in *USENIX Security Symposium*, 2016.

[15] D. Herrmann, R. Wendolsky, and H. Federrath, "Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial Naïve-Bayes Classifier," in *Workshop on Cloud Computing Security*, 2009.

[16] N. Hopper, E. Y. Vasserman, and E. Chan-Tin, "How much anonymity does network latency leak?" *Transactions on Information and System Security*, vol. 13, no. 2, 2010.

[17] A. D. Jaggard and P. Syverson, "Onions in the Crosshairs: When The Man really is out to get you," in *Workshop on Privacy in the Electronic Society*, 2017.

[18] R. Jansen and N. Hopper, "Shadow: Running Tor in a box for accurate and efficient experimentation," in *Network and Distributed System Security Symposium*, 2012.

[19] R. Jansen and A. Johnson, "Safely Measuring Tor," in *Conference on Computer and Communications Security*, 2016.

[20] M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt, "A Critical Analysis of Website Fingerprinting Attacks," in *Conference on Computer and Communications Security*, 2014.

[21] M. Juarez, M. Imani, M. Perry, C. Diaz, and M. Wright, "Toward an efficient website fingerprinting defense," in *European Symposium on Research in Computer Security*, 2016.

[22] A. Kwon, M. AlSabah, D. Lazar, M. Dacier, and S. Devadas, "Circuit fingerprinting attacks: passive deanonymization of tor hidden services," in *USENIX Security Symposium*, 2015.

[23] P. Mittal, A. Khurshid, J. Juen, M. Caesar, and N. Borisov, "Stealthy traffic analysis of low-latency anonymous communication using throughput fingerprinting," in *Conference on Computer and Communications Security*, 2011.

[24] S. J. Murdoch and G. Danezis, "Low-cost traffic analysis of tor," in *Symposium on Security and Privacy*, 2005.

[25] R. Overdorf, M. Juarez, G. Acar, R. Greenstadt, and C. Diaz, "How Unique is Your. onion? An Analysis of the Fingerprintability of Tor Onion Services," in *Conference on Computer and Communications Security*, 2017.

[26] A. Panchenko, F. Lanze, A. Zinnen, M. Henze, J. Pennekamp, K. Wehrle, and T. Engel, "Website Fingerprinting at Internet Scale," in *Network and Distributed System Security Symposium*, 2016.

[27] A. Panchenko, A. Mitseva, M. Henze, F. Lanze, K. Wehrle, and T. Engel, "Analysis of Fingerprinting Techniques for Tor Hidden Services," in *Workshop on Privacy in the Electronic Society*, 2017.

[28] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, "Website Fingerprinting in Onion Routing Based Anonymization Networks," in *Workshop on Privacy in the Electronic Society*, 2011.

[29] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, "Effective Attacks and Provable Defenses for Website Fingerprinting," in *USENIX Security Symposium*, 2014.

[30] T. Wang and I. Goldberg, "Improved Website Fingerprinting on Tor," in *Workshop on Privacy in the Electronic Society*, 2013.

[31] ——, "On Realistically Attacking Tor with Website Fingerprinting," in *Proceedings on Privacy Enhancing Technologies*, 2016.