Repositioning Real-World Website Fingerprinting on Tor

Rob Jansen
U.S. Naval Research Laboratory
Washington, DC, USA

Ryan Wails
U.S. Naval Research Laboratory
Georgetown University
Washington, DC, USA

Aaron Johnson
U.S. Naval Research Laboratory
Washington, DC, USA

Abstract

Website fingerprinting (WF) is a potentially devastating attack against Tor because it can break anonymity by linking a Tor user to their purportedly unlinkable internet destinations. Previous work proposes that an adversary trains WF classifiers either on synthetic traces that are programmatically generated using automated tools, or on real-world traces collected from one or more Tor exit relays. However, no existing work accurately represents a real-world threat model in which a WF adversary's classifiers must be tested against real-world entry traces that are naturally created by real Tor users. In this paper we present Retracer, a novel method for producing labeled entry traces of genuine Tor traffic patterns. Retracer uses high-fidelity network simulation to accurately transform real-world exit traces into entry traces prior to training and testing WF classifiers. After first demonstrating that Retracer accurately transforms exit traces into entry traces, we then apply it to the recently released GTT23 dataset in a WF evaluation in which more than 3,500 classifiers are tested against, for the first time, labeled entry traces of real Tor traffic patterns. Our evaluation yields the best available estimate of the performance an adversary can achieve when directing WF attacks at real Tor users.

CCS Concepts

• **Networks** → **Network privacy and anonymity**.

Keywords

traffic analysis; website fingerprinting; anonymous communication

ACM Reference Format:
Rob Jansen, Ryan Wails, and Aaron Johnson. 2024. Repositioning Real-World Website Fingerprinting on Tor. In *Proceedings of the 23rd Workshop on Privacy in the Electronic Society (WPES '24), October 14, 2024, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3689943.3695047

1 Introduction

Website fingerprinting (WF) is a serious attack on Web privacy in which an adversary that can observe the encrypted traffic patterns of a user's internet connections can infer the website being accessed or the content being loaded through those connections [5, 14, 15, 30, 45]. WF has most often been studied as an attack against the Tor network [10]. To carry out a Tor WF attack, the adversary first trains machine learning (ML) classifiers to associate website labels with their corresponding *cell traces*, each of which encodes the directionality and timing of a sequence of onion-encrypted, application-layer Tor messages (called *cells*). From a vantage point on the entry-side of the Tor network, the adversary may then observe users' Tor traffic and employ the classifiers to predict the visited websites. Thus, a WF attack enables the adversary to deanonymize Tor users by linking them to their purportedly unlinkable internet destinations [4, 7–9, 13, 22, 33–36, 39–44, 48, 50, 51].

In using ML for WF, example cell traces and labels are required to train WF classifiers. Recent work by Cherubin et al. was the first to consider that an adversary can train its classifiers using the *real-world* cell traces and labels that are naturally created by real Tor users and observed by Tor exit relays [8] rather than using *synthetic* traces that are programmatically generated using automated tools [1]. The authors argue that the naturally occurring exit traces are more realistic than synthetic traces and thus improve our ability to accurately assess the real-world threat of WF. However, the proposed training strategy has a major limitation: a WF adversary may train classifiers on real traces from an *exit* relay, but must evaluate them during a WF attack on observations from their *entry*-side vantage point. The inconsistency between the training and testing positions was shown to reduce classifier accuracy by 5–18% [8, § 6.4], raising unanswered questions about the extent to which an adversary could benefit from real exit-relay traces in practice.

We examine the research question: *to what extent can a WF adversary effectively mitigate the classifier performance degradation resulting from training on real-world Tor exit traces?* Effective methods that are robust to variation in the cell trace capture position would increase the viability of a real-world exit training strategy but have thus far been unexplored. Although Cherubin et al. conducted a preliminary measurement of the amount of degradation, they did not evaluate any potential mitigation strategy [8, § 6.4]. Other methods have been proposed for augmenting traces to be robust to variation in Tor network conditions such as relay congestion [2, 22], but these approaches do not consider inconsistencies between training and testing positions; indeed, we evaluate NetAug [2] in §4 and find that it has a detrimental effect on performance compared to training directly on non-augmented exit traces.

In this paper we present *Retracer*, a novel method for *trace transduction*: for producing cell traces in a target relay position given traces in any other relay position. The key insight behind Retracer is that the directionality and timing information present in cell traces can be used to reproduce the original sequence of Tor cells in different network environments. From this insight we design Retracer to replay an existing set of cell traces inside of large-scale, high-fidelity Tor network simulations [18, 19], during which we measure the resulting traces that are observed by different simulated vantage points. Retracer was carefully designed to consider network latency and replay position while attempting to reproduce cell traces in simulation exactly as they were originally observed in the real world. We highlight that Retracer is not an attack itself,

WPES '24, October 14, 2024, Salt Lake City, UT, USA
2024. ACM ISBN 979-8-4007-1239-5/24/10
https://doi.org/10.1145/3689943.3695047

but a pre-training method that we will demonstrate can boost the real-world performance of WF classifiers trained on its output.

We evaluate Retracer's efficacy in a likely task of a real-world adversary: transducing real cell traces collected by an exit relay into entry traces prior to training classifiers that will later be deployed on an entry-side vantage point during a WF attack. Our evaluation uses three new WF datasets that we measured to provide accurately labeled traces from both entry and exit positions in the live Tor network. We use these datasets to evaluate Retracer considering a downstream WF task as a scoring metric. We find that WF classifier accuracy degrades by 13 percentage points when training on exit traces but testing on entry traces as proposed in previous work [8], and that 10 points of classifier accuracy is recovered when training using entry traces transduced from exit traces by Retracer. We also find that classifiers trained with Retracer are consistently 15 points more accurate than classifiers trained with the NetAug trace augmentation method [2] across multiple algorithm settings.

After having established Retracer's efficacy, we conduct an extensive real-world WF evaluation using GTT23, a recently published dataset of traces that were naturally created by real Tor users and measured by real-world Tor exit relays [23, 24]. GTT23 contains *exit* traces and thus suffers from the same limitation as the exit relay strategy of Cherubin et al. [8], but we use Retracer to overcome this limitation by transducing the exit traces to entry traces. We consider the fingerprintability of individual websites by training and testing more than 3,500 website classifiers using more than 3,500,000 real-world traces and comparing their performance distributions. We find that an overwhelming majority of our trained classifiers are unlikely to be viable in real-world WF attacks because they would be quickly overrun with false positives: fewer than 20% of our website classifiers achieve greater than a 0.75 precision score. In the first feature importance analysis on real-world exit traces, we find that the number of traces and trace variability are among the most important features predicting classifier performance. We also draw comparisons to previous WF methods while considering more accurate methods of estimating real-world WF performance.

Our contributions yield the best real-world WF threat estimate:

- The design of Retracer, a novel trace transducer that produces cell traces in a target position given traces in another position.
- An evaluation of Retracer using three new datasets containing accurately labeled traces from Tor entry and exit relays.
- An extensive evaluation of real-world WF that, for the first time, estimates the performance of classifiers by testing them on entry traces transduced from real-world exit traces.
- An evaluation methodology that considers individual website fingerprintability to estimate the real-world threat of WF.
- The first analysis of feature importance for real-world traces.

2 Background and Related Work

2.1 The Tor Network

Tor is a privacy-preserving network of proxy *relays* that protects its users' traffic using onion routing [10, 46]. A Tor client builds a *circuit* through three consecutive relays—the *entry*, *middle*, and *exit*—and then forwards its internet-bound TCP *streams* through the circuit. The exit relay proxies the client's requests from the circuit to a destination service and then forwards traffic in both directions. When using Tor Browser, all streams corresponding to the first-party domain are multiplexed onto the same circuit while other streams are isolated on separate circuits. We use *website* to refer to the first-party domain name of a web connection, which for such a connection coincides with the domain resolved by the exit relay when connecting a circuit's first stream to its destination, and *webpage* to refer to a page's full URL.

Tor packages messages sent by Tor clients and destinations into *cells*, each with a fixed size of 514 bytes, before forwarding them through a circuit. A *cell trace* C is a collection of N components $\langle (t_i, d_i) \text{ or } \perp \rangle_{i=1}^{N}$ where t_i is the time that the ith cell was observed relative to circuit creation, and $d_i \in \{-1, 1\}$ is the direction in which the ith cell was forwarded (1 indicates the cell was sent by the client toward the exit, and -1 indicates it was sent by the exit toward the client). $C[i] = \perp$ indicates that a trace is shorter than i cells. Any of a circuit's relays may observe and record a cell trace [8].

2.2 WF Threat Model

Fig. 1 summarizes our WF threat model. We consider a WF adversary who is interested in conducting targeted surveillance and censorship of a select *monitored set* of websites, and may choose to launch digital or physical harassment campaigns against Tor users visiting those sites. We focus on a non-targeting adversary who is interested in classifying all Tor traffic, no matter the traffic type, sources, or destinations involved. Defending against non-targeting adversaries is important in order to protect as many Tor users as possible against mass-deanonymization attacks and large-scale privacy violations. An adversary who targets specific Tor users using knowledge of their behavior or user-specific destinations of interest is also important, but the smaller scale of a targeted attack limits the overall harm to users, and we consider it out of scope for this paper.

We consider an adversary who controls an entry-side vantage point and uses it to collect unlabeled cell traces on connections into the Tor network. Cell traces are most directly observable by Tor relays, but previous work has shown that network path vantage points between a Tor user and Tor entry relay may also infer them [51]. Given an observed entry trace, the adversary's WF task is to accurately guess the trace's label, i.e., a destination website, domain name, or service being accessed by the user.

We assume that the adversary uses ML to train classifiers to associate cell traces with destination website labels. The WF classifier training process requires *accurately labeled traces* so that the classifiers can form correct associations. However, it is impossible for the adversary to gather labeled traces by passively monitoring its entry-side vantage point because Tor hides all destination-related information using onion routing [46]. Thus, the adversary must use a different method for collecting training traces.

We assume that the adversary is capable of employing two previously proposed methods of collecting labeled training traces. First, the adversary can use automated browser tools to programmatically fetch a set of webpages through Tor and collect the resulting cell traces [1]. Second, the adversary can run one or more Tor exit relays and use them to passively observe and measure the naturally occurring exit-side cell traces and first-party domain labels [8, 23]. The collected traces may be pre-processed (as we do with Retracer) or else directly used for WF classifier training. Note that cell traces

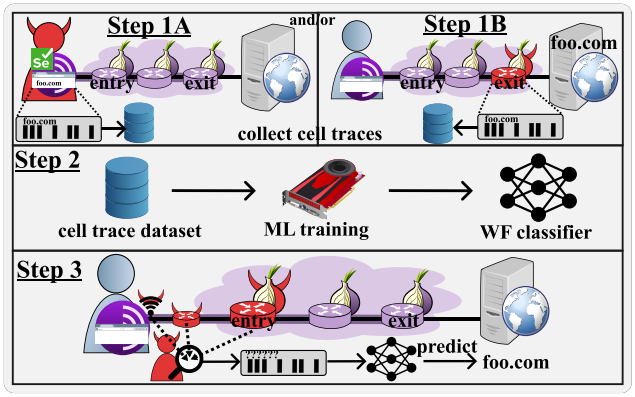

Figure 1: In the WF threat model, the adversary may collect cell traces using an automated browser (Step 1A) and/or an exit relay (Step 1B), train a WF classifier (Step 2), and then use one or more entry-side vantage points to observe users' Tor connections and infer their visited destinations (Step 3).

collected on an exit-side relay will be inconsistent with those an adversary would observe on the entry-side during a WF attack; our Retracer method is designed to overcome this limitation.

2.3 WF Attacks and Defenses

Techniques for WF have evolved along with advances in ML. Early methods used simple statistical techniques [5, 15, 45]. Subsequent works applied more sophisticated ML algorithms to specified features [7, 13, 14, 35, 36, 50]. More recently, deep-learning methods, which provide automatic feature extraction, have achieved SOTA performance [4, 8, 22, 33, 34, 39–44].

Throughout this paper, we evaluate WF using Deep Fingerprinting (DF) [43], a powerful classifier based on a deep neural network. DF does not require manual feature engineering: it is trained only on a sequence of cell directions. We consider WF with DF for two main reasons. First, DF is the primary algorithm underlying many recently published WF attacks, including Deep Fingerprinting [43], Tik-Tok [40], Triplet Fingerprinting [44], GANDaLF [33], and NetCLR [2]. The SOTA attacks continue to build on it, and thus evaluating it directly gives us a common baseline for comparison of many attacks. Second, DF consistently outperforms other WF attacks, including three others we tested in a preliminary experiment (CUMUL [35], *k*-Fingerprinting [13], and Tik-Tok [40]). Thus, DF is a performance benchmark against which new attacks compete.

Many WF defenses have been proposed, with key techniques that include adding "padding" traffic [16], delaying cells [12], and splitting traffic across separate connections [28]. A fundamental challenge in designing defenses is balancing their effectiveness against WF with their performance cost in latency or bandwidth [11]. Synthetic traffic is typically used to evaluate WF defenses, and Wang argues that a more-conservative "one-page" evaluation is more appropriate for evaluating defenses [49]. Mathews et al. provide a comprehensive and critical evaluation of WF defenses [32].

2.4 Real-World Considerations

Cherubin et al. [8] were the first to study WF attacks using "genuine" Tor traces collected from exit relays. They considered a streaming model where WF classifier training and testing is done online (that is, one trace at a time). However, many WF attacks based on classical models are not designed to be updated online which is required in the streaming model. Moreover, their work did not produce a dataset, limiting reproducibility and preventing a feature-importance analysis to help interpret their results. Finally, the authors evaluated real-world WF performance using exit-side rather than entry-side traces while acknowledging the inaccuracy of the approach and calling for future work to mitigate its impact. We evaluate the inaccuracy of their approach and our ability to mitigate it with Retracer throughout this paper.

Recently, Jansen et al. conducted a large-scale exit relay measurement of over thirteen million genuine Tor traces [23]. The resulting dataset, called GTT23, is the largest WF dataset currently available [24]. GTT23 contains cell traces and labels that were naturally produced by real Tor users and collected from real Tor exit relays over a period of 13 weeks in 2023. Thus, GTT23 is more realistic than any previous WF dataset and can be used to more accurately assess the real-world performance of WF attacks. We use Retracer to transduce the exit traces from GTT23 into entry traces, considering, for the first time, that WF classifiers must be tested against entry-side traces to accurately reflect the WF threat model.

Other pre-training and augmentation methods have been proposed to make fingerprinting more practical, but none of them transduce traces from one network position into traces from another as Retracer does. Oh et al. use Generative Adversarial Networks to expand the training set in data-limited scenarios [33], while Xie et al. augment TCP traces via a set of algorithms that simulate the effects of packet loss and data buffering [52]. Jansen and Wails show how to fetch Wikipedia pages inside of network simulations to improve control over dataset collection and quantify the effects of changing network conditions on WF performance [22]. The design of Retracer was inspired by their use of network simulation to, but our replay methodology is otherwise orthogonal.

Most closely related to Retracer is the NetAugment approach of Bahramali et al. [2]. NetAugment aims to make WF classifiers more robust by providing them with any number of randomly augmented cell traces that represent unobserved network conditions or settings. The key idea is that classifiers can train on different variations of cell traces that might be more representative of the traces they will be presented when deployed, and the approach was shown to produce classifiers that are more robust to changing network conditions or concept drift. Although not specifically designed for changes in the trace capture position, it is reasonable to expect that NetAugment might be robust to such changes. Thus, we compare it to Retracer in §4. One noteworthy limitation of NetAugment is that its augmented traces do not include timing information, which some classifiers are designed to utilize [13, 40]. See Appendix A and the work of Bahramali et al. [2, Alg. 1–4] for more details.

3 Trace Transduction with Retracer

In this section, we present the design of our Retracer method of obtaining entry cell traces given traces from other positions.

3.1 Problem Overview

An overwhelming majority of prior WF studies evaluate performance using cell traces programmatically collected using automated browser tools (Step 1A in Fig. 1) [23, Table 3]. While this method is convenient, it has been shown to produce unrealistic datasets due to the inherent difficulty in modeling the behavior of anonymous Tor users, the differences between browser versions and configuration, and the natural variation in Tor network conditions [2, 8, 23, 25]. More realistic datasets would allow us to produce more accurate performance estimates of real-world WF attacks.

The exit relay methodology proposed by Cherubin et al. is a new direction for the study of real-world WF (Step 1B in Fig. 1) [8]. The first-party domain labels and associated cell traces that an exit relay can observe and passively measure naturally exhibit the patterns associated with real user behavior, real browser configuration choices, and real network conditions. These real-world traces are beneficial in two ways. First, the adversary can incorporate them during *training* to produce classifiers that can better distinguish monitored websites from the real destinations visited by users. Second, they can be used during *testing* to better represent real-world data diversity which may present a more challenging WF problem. Thus, researchers using real-world traces can better model the WF problem and better estimate real-world WF performance.

Unfortunately, the exit relay WF methodology carries a critical limitation: exit relay measurement yields *exit*-side cell traces, whereas the adversary conducts WF attacks from an *entry*-side position (Step 3 in Fig. 1). The inconsistency between WF training and testing positions has been shown to reduce classifier accuracy by as much as 5–18% [8, § 6.4], raising questions about the viability of an exit training strategy. Retracer is designed to overcome this limitation by transducing (i.e., transforming) traces from one position into traces from another prior to training WF classifiers.

3.2 Cell Trace Augmentation vs. Transduction

We define two functionalities for manipulating cell traces: *augmentation* and *transduction*. An important distinction between these functionalities is that augmentation manipulates traces indiscriminantly with respect to relay position, whereas transduction manipulates traces with the specific objective of transforming it from a given relay position to a target position.

Augmentation is a process that maps an input cell trace into a set of augmented output cell traces. More precisely, a cell trace *augmenter* is a function $\mathcal{A}(I, M) \mapsto \langle O_i \rangle_{i=1}^{M}$ where I and O_i are cell traces of the form $\langle (t_i, d_i)$ or $\perp \rangle_{i=1}^{N}$ as defined in §2 and M is a multiplicative factor that controls the degree of data expansion. That is, an augmenter can expand a single input trace I into M output traces by repeatedly applying the augmentation algorithm. Augmenters may have various objectives; NetAugment is an augmenter with an objective of increasing the diversity of cell bursts to build robustness against unobserved network conditions [2] (see §2.4).

Transduction is a process that abstractly functions analogously to augmentation, except that it also takes as input a position from which the trace was measured and a target position into which it should be transformed. More precisely, a cell trace *transducer* is a function $\mathcal{T}(I, M, p_{in}, p_{out}) \mapsto \langle O_i \rangle_{i=1}^{M}$ where I, O_i, and M are as previously defined for augmentation, p_{in} is the position in which I was measured (e.g., exit relay) and p_{out} is the target position (e.g., entry relay). Retracer is a novel transducer that builds robustness against inconsistent training and testing positions, but transduction has never before been explored in the WF literature.

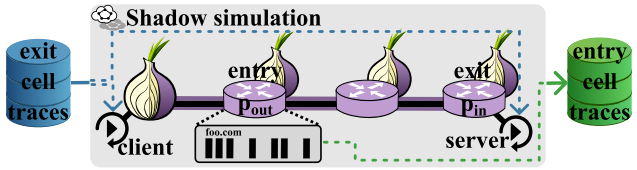

Figure 2: Retracer transduces cell traces from one position (here, p_{in}=exit) by replaying them in high-fidelity Shadow simulations of the Tor network [18, 19] while measuring the simulated traces from a target position (here, p_{out}=entry).

An adversary may use any combination of $\mathcal{A}(\cdot)$ and $\mathcal{T}(\cdot)$ functionalities to manipulate cell traces in a processing phase that is run prior to classifier training (i.e., in a new pre-training step immediately preceding Step 2 in Fig. 1). Pre-training can be considered an extension of the training process, but is not itself an attack.

3.3 Retracer: A Cell Trace Transducer

3.3.1 Overview. We present *Retracer*, a novel cell trace transducer that produces cell traces in a target position from traces originally collected in another position. Our key insight is that cell traces that are collected in the real Tor network already contain the cell metadata (including cell directionality and timing information [23, Listing 1]) that would be necessary to reproduce the original sequence and flow of Tor cells in different network environments. Moreover, recent advancements in Tor network modeling algorithms and tools enable us to accurately model a virtual Tor network with a high degree of fidelity using publicly available Tor data that describes the state of the network at any given time [20]. From these insights, we design Retracer to (1) construct detailed Tor network simulation models that incorporate the state of the network precisely as it existed during the input trace measurement period; (2) replay the input cell traces during a high-fidelity network simulation according to the input position p_{in} while measuring the corresponding output traces observed by the simulated nodes in the target position p_{out}; and (3) run multiple such simulations to meet the desired data expansion factor M for all input traces.

Retracer uses Shadow to replay traces in Tor network simulations (see Fig. 2). Shadow is a high-fidelity, discrete-event, packet-level network simulator that directly executes unmodified Tor binaries to obtain a high degree of control over the conditions in each simulated Tor network [18, 19]. Because Shadow directly executes Tor, we can expect that it will faithfully reproduce the end-to-end cell trace modifications that would naturally occur in the real world, including both directionality and timing modifications that result from changes in circuit position and network congestion. In using Shadow, Retracer draws inspiration from previous work that shows that the simulator's high degree of control improves the explainability of WF datasets and results [22].

3.3.2 Constructing a Simulation Model. To simulate a Tor network in Shadow, we first construct a simulation model that accurately

represents the real Tor network as it existed during the period of measurement using recently published Tor network modeling tools (i.e., tornettools [20]). The tornettools process uses publicly available Tor metrics data [47] and recent measurements of Tor network traffic [21] to construct representative Tor networks. The modeled characteristics include: network topology, latency, and bandwidth; Tor relay processes and relay selection weights; and traffic generation processes that use Markov models to generate realistic Tor traffic flows. All processes that compose a model may communicate over Shadow's simulated network, the accuracy of which has been extensively validated [17, 19–21].

Retracer extends the tornettools simulation model as follows. First, a custom Tor patch is used to support Retracer. The patch enables the simulated Tor relays to measure and store cell trace metadata records analogous to previous measurement methods [8, 23]. All relays in the p_{out} position are configured to store cell trace metadata records during the simulation. The patch also enables a Tor client to build a circuit to a relay, send a website label, and connect through the circuit to another process over the relay's localhost interface as will be required during trace replay. Second, Retracer adds new virtual hosts and replay processes to the model. Each node in the p_{in} position is configured to run a replay server process that listens for incoming connections from a replay client. Each new client host is configured to run a replay client process that makes connections to replay servers through Tor circuits that end at the p_{out} position. The replay hosts facilitate replaying a configurable set of input cell traces in the simulated Tor network (at a configurable level of parallelism) as described in §3.3.3 below.

3.3.3 Replaying Traces. A primary component in Retracer is a replay process that reproduces an input cell trace I in simulation exactly as it was observed in the live Tor network. Our replay algorithm requires cooperation between a replay client R_c that runs in simulation coincident with a simulated Tor client, and a replay server R_s that runs coincident with a simulated node that runs in the same position from that in which I was observed (i.e., p_{in}). To ease exposition, we assume R_s runs in the exit relay position and that both R_c and R_s have access to a database containing trace I.

Recall that I is a cell trace of the form $\langle (t_i, d_i) \text{ or } \perp \rangle_{i=1}^{N}$. I represents a transcript that R_c and R_s will follow to replay the trace. The directions d in I represent sending events: $d = 1$ and $d = -1$ indicates that R_c and R_s should send 498 bytes of application payload, respectively.[1] The timestamps t in I represent the times that the sending events should be executed. Because the timestamps were measured from the perspective of a relay, we consider R_s to be the *time anchor* for I. That is, the cells in I are considered to have occurred relative to R_s. Thus, R_s will execute a sending event $d_i = -1$ exactly at time t_i (relative to replay initialization). However, R_c will need to execute a sending event $d_j = 1$ prior to time t_j in order to ensure that R_s observes the cell at t_j. Thus, R_c computes the network latency l between R_c and R_s during replay initialization, and then will proceed to execute each sending event $d_j = 1$ at time $t_j - l$. The following summarizes the Retracer replay algorithm:

Handshake: R_c selects a trace I to replay, connects to R_s through the simulated Tor network, and sends a handshake message containing the *id* of I, the current time, and a proposed start time. R_s computes the latency l to R_c, and replies with the current time and a chosen start time τ that is consistent with l. Upon receiving the reply, R_c computes l and stores τ.

Replay: R_s and R_c begin iterating through the cell items in I at time τ and $\tau - l$, respectively. Until $\perp$ is reached: R_s sends 498 bytes at t_i only when $d_i = -1$ and then pauses until t_{i+1}, while R_c sends 498 bytes at $t_j - l$ only when $d_j = 1$ and then pauses until $t_{j+1} - l$.

Termination: The replay terminates when R_c and R_s have both sent and received the expected cells from I (i.e., they have both reached $\perp$), or upon error.

We implemented the above algorithm in 971 lines of Rust code; it is used in Retracer simulation models as described in §3.3.2.

3.3.4 Running Tor Network Simulations. Retracer uses Shadow to run a Tor network simulation according to the constructed simulation model and using the cell trace replay algorithm. Recall that, during the simulation, input cell traces are replayed according to the p_{in} position and output cell traces are measured from the p_{out} position. After the simulation, the recorded output cell traces are collated into a new dataset for subsequent WF analysis.

The Retracer method may be repeated multiple times in order to produce enough traces to meet the desired data expansion factor M. This may be useful, e.g., to parallelize the transduction of a very large number of traces. Additionally, Retracer by default sets the tornettools options network_scale=0.15 and load_scale=2.0 as in previous work [22], but these values can be adjusted to produce Tor network models of varying size or with varying levels of traffic load. See Appendix B for an analysis of the simulated Tor networks.

Key Takeaway: Retracer transduces traces of a given position into traces of a target position by replaying them in high-fidelity Tor network simulations. It can enhance classifier training, and help us more accurately estimate real-world WF attack performance.

4 Transduction Evaluation

In this section, we evaluate Retracer in a likely task of a real-world adversary: transducing *exit*-side cell traces prior to training WF classifiers that will be deployed on an *entry*-side vantage point.

4.1 Overview

Recall from the threat model in § 2 that the adversary's goal in conducting WF attacks is to use a client's traffic patterns observed on an *entry*-side vantage point to predict the visited destination website and thus deanonymize the client (see Fig. 1). The adversary would ideally train on the same type of real-world traces that it will face during an attack, but Tor encrypts traffic such that the genuine entry traces that are naturally observable by the adversary are unlabeled. To overcome this limitation, Cherubin et al. propose collecting real-world traces from exit relays that *can* observe *both* traces *and* labels, but their method is limited to the exit position and reduced classifier accuracy by as much as 5–18% [8, § 6.4].

Our goal is to evaluate the extent to which Retracer can mitigate the performance degradation caused by training on *exit*-side traces when a real-world adversary must test on *entry*-side traces. That is,

[1] Tor prepends a 16 byte cell header. GTT23 traces also contain cell metadata, allowing Retracer to reproduce stream creation and filter out prior to replay the control cells that Tor will naturally reproduce as data is forwarded [23].

we want to evaluate Retracer's ability to transduce exit traces into entry traces (i.e., $\mathcal{T}(I, M, exit, entry)$) prior to classifier training. Thus, we will train multiple classifiers with and without transduced traces, and then test them with labeled entry traces.

To properly evaluate transduction performance, we require both labeled entry and labeled exit traces collected from the Tor network. With both types of traces, we can compare the difference between transduced exit→entry traces produced by Retracer and true entry traces from the real network. Unfortunately, no existing WF dataset contains both types of traces. GTT23 is the best candidate with over 13 million real-world traces [23, 24], but it contains only exit traces. (We defer a real-world WF performance evaluation with GTT23 to §5, after we first establish Retracer's efficacy.) Because no existing dataset fulfills our requirements, we construct new datasets of our own cell traces for which we can provide ground-truth labels for both entry- and exit-side observations.

4.2 Dataset Measurement

We construct a set of URLs to use for measurement as follows. First, we use the Wikipedia random page generator[2] to fetch 1,000 random Wikipedia pages. On each random page, we scan all <a> tags for `href` attribute values that start with "http" and exclude "wikipedia" or "wikimedia"; we recorded 22,463 such URLs. Second, we bucket the recorded URLs by their domain name, sample 1,000 such names, and randomly choose one URL from the bucket corresponding to each sampled name. The resulting URLs point to webpages of news, sports, and other typical internet sites. Third, we fetch each of the 1,000 URLs once using tor-browser-selenium [1], record a screenshot of the browser window, and manually filter out URLs that resulted in error pages or otherwise apparently failed to properly load. Our final set contains 494 valid URLs.

We conduct three independent measurements through the live Tor network over the period of a week in 2023-08 using a client machine hosted in Chameleon Cloud [26] and a measurement relay running on a machine hosted in NY, USA. The client is configured to concurrently fetch the 494 URLs through Tor (using 10 worker processes) many times each while pinning our measurement relay as either the entry relay (in the entry_1 and entry_2 measurements) or the exit relay (in the exit measurement) for all built circuits. The client and relay run our patched version of Tor from §3.3.2, which enables the client to send a special control cell with the true website label and enables the relay to record labeled cell traces only for those circuits created by our client.

Following each of the three measurements, we collect and clean the data recorded by our relay. First, we discard any cell trace for which we observe a second trace to the same label within 30 minutes, which suggests at least one error/retry since our worker pool takes longer than an hour to fetch all URLs from our set. Second, we discard traces with a cell count that is less than 25 or less than Tukey's lower fence for the label (i.e., Q1 − 1.5 · IQR where Q1 and IQR are computed across all traces of a label). Finally, we use random sampling to balance the number of traces per label in the final datasets. The resulting datasets are summarized in the top half of Table 1, where "Tor" is a label that indicates the source of the fetched URLs. We discuss our ethical considerations in Appendix C.

[2]https://en.wikipedia.org/wiki/Special:Random

Table 1: Datasets used to Evaluate Trace Transduction

Dataset	URLs×Traces	Description
$\text{Tor}(\text{entry}_1)$	444×60	Live Tor, pinned entry relay
$\text{Tor}(\text{entry}_2)$	472×40	Live Tor, pinned entry relay
Tor(exit)	423×60	Live Tor, pinned exit relay
Retracer(M)	423×60×M	$\mathcal{T}(\text{Tor}(\text{exit}), M, exit, entry)$
NetAug(M)	423×60×M	$\mathcal{A}(\text{Tor}(\text{exit}), M)$

$\mathcal{T}$: transducer, $\mathcal{A}$: augmenter, M: data expansion factor (see §3.2)

4.3 Dataset Transformations

4.3.1 Transducing with Retracer. We apply the Retracer methodology described in §3.3 to transduce the exit traces from Tor(exit) into entry traces. In order to experiment with multiple values of M (the data expansion factor), we apply the Retracer methodology across 19 simulation models for which we vary the tornettools load_scale value. Previous work used a load_scale value of 2.0 as a baseline, and found that using a range of values improves trace diversity and classifier robustness [22]. Thus, we use load_scale values in the range [1.38, 2.9] (specifically, each load value $\ell \in \bigcup_{i=20}^{29}\{i/10,\ 40/i\}$). We replay each cell trace from Tor(exit) once in each of our 19 simulated Tor networks, and then we collate all entry-side cell traces for subsequent analysis. See Appendix B for details about the simulated network performance characteristics and simulation resource costs.

The Retracer transducer function $\mathcal{T}(I, M, exit, entry)$ is implemented using the simulated entry-side cell traces that are correlated with the unique identifier of the input trace I (we have 19 such entry-side traces per input trace). When $M = 1$, $\mathcal{T}(\cdot)$ returns the correlated entry-side cell trace from the baseline simulation. For $M > 1$, $\mathcal{T}(\cdot)$ returns the baseline correlated entry trace along with the entry trace from each of the $M - 1$ most loaded simulations. We prefer traces from the more loaded networks since previous work found them to produce more robust classifiers [22]. The above strategy yields the dataset Retracer(M) as shown in Table 1.

4.3.2 Augmenting with NetAugment. Our evaluation will also consider NetAugment as it is the most recent and most closely related method for trace augmentation. Recall that we describe NetAugment in §2.4. Bahramali et al. present its algorithms in more detail [2] and released an implementation on Github [3].

The NetAugment augmenter function $\mathcal{A}(I, M)$ is implemented as follows. First, NetAugment is initialized using Tor(exit), from which the initializer constructs a burst size distribution that can later be queried during augmentation. The NetAugment code exposes an augment(·) function that takes a cell trace as input and outputs a randomly augmented trace. We implement the augmenter $\mathcal{A}(I, M)$ by repeatedly calling augment(I) a total of M times for an input trace I (as with Retracer, we consider each integer value in the range $1 \le M \le 19$). We apply $\mathcal{A}(\cdot)$ to all traces in Tor(exit) and store the output dataset NetAug(M) as shown in Table 1.

4.4 Evaluation Methodology

We carry out an evaluation of the efficacy of Retracer as a transducer using the datasets summarized in Table 1. Our objective is to understand how well the entry traces produced by Retracer represent the

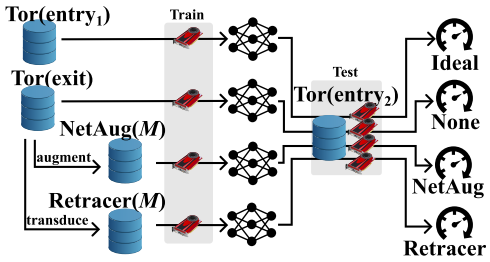

Figure 3: Cell trace transduction evaluation methodology.

entry traces that are naturally produced in the real Tor network for the same set of webpages. Thus, we take the exit traces from the real Tor network (Tor(exit)), transduce them into entry traces with Retracer (Retracer(M)), and then compare the transduced traces with real entry traces from the real Tor network (Tor(entry$_2$)).

The scoring metric we use to measure efficacy is the accuracy in performing a WF classification task. We considered using other metrics to compare the difference between Retracer entry traces and real Tor entry traces including Levenshtein (i.e., edit) distance [8], but we found such metrics are indirectly related to the feature importance of a trace. For example, the earlier part of a trace generally carries more information than the later part [29], but subsequence weighting is not possible with many distance functions and a valid weight assignment is largely unknown. Because we expect that Retracer output would be used in a downstream WF task, it is more direct and consistent to also use WF as a scoring metric.

We highlight that we do *not* consider this evaluation to accurately estimate the performance of real-world WF attacks against real Tor users: we do not expect that synthetic datasets capture the true diversity in the traces that would result from real users using the real network. However, this evaluation is informative in understanding Retracer's efficacy and for comparing trace transformation methods considering a common baseline. We defer a real-world WF evaluation to §5, after first establishing the effectiveness of the Retracer methodology below.

4.4.1 Training. We train two types of WF classifiers chosen as baseline representatives of deep learning WF approaches (see §2.3). Deep Fingerprinting (DF) [43] is based on a deep neural network that does not require manual feature engineering and requires only cell directions, and Tik-Tok [40] uses the identical method of DF but requires both cell direction and time. We exclude other classical WF methods because prior work has shown that they are not robust to natural changes in the network over time [22].

The classifiers are trained in a multiclass closed-world classification setting in which they have full knowledge of all possible class labels and learn how to associate examples of each class with its unique label. Prior to training, each training set is split such that 80% of the traces are used to fit the classifier models and 20% are used for model validation; the split is stratified by class label such that an equal number of traces of each class is present in both the training and validation sets. During training, we employ the published implementations (and optimized hyperparameters) of DF and Tik-Tok. However, we tune the batch size by training a classifier for every batch size $b \in \{64, 128, 256, 512\}$. Additionally, DF and Tik-Tok are trained for 100 epochs, but to avoid overfitting we use an early stopping strategy that stops training if the validation loss metric has not improved for five consecutive epochs. The classifiers are trained in the following scenarios as depicted in Fig. 3:

Ideal: Upper bound for reference; classifiers are trained on the Tor(entry$_1$) dataset (creation of which requires cooperation from clients since otherwise training labels are encrypted).

Retracer: Our Retracer transducer is applied to traces from Tor(exit); classifiers are trained on the entry traces from Retracer(19).

NetAug: The NetAug augmenter [2] is applied to Tor(exit); classifiers are trained on the augmented traces from NetAug(19).

None: No cell transformation is applied; classifiers are trained directly on traces from Tor(exit), as in the OnlineWF method [8].

Table 2: Classifier Accuracy in a Multiclass Closed World Classification Experiment when Tested on Tor(entry$_2$)

Method	Training set	DF	Tik-Tok
Ideal	Tor(entry$_1$)	89%	87%
Retracer	Retracer(19)	86% (↓ 3 pp)	85% (↓2 pp)
NetAug	NetAug(19)	70% (↓19 pp)	⊥
None	Tor(exit)	76% (↓13 pp)	79% (↓8 pp)
Classifier Properties →		Time-Oblivious	Time-Aware

⊥: Timing information required by classifier but unavailable in data.

4.4.2 Testing. The trained classifiers are tested in a multiclass closed-world classification experiment in which each classifier attempts to predict the website label associated with each test trace. Tor(entry$_2$) is used as a test set for every classifier; it was collected independently from the other datasets and its traces are never used during training. During testing, we only attempt to predict traces for which the class label was present during training. Because our test set is balanced across class labels, we use accuracy (the fraction of correct predictions made by the classifier) as a measure of performance. For each tested scenario, we report the results from the classifier with the batch size resulting in the highest test accuracy.

4.5 Results

The results of our experiment are shown in Table 2. We find that training on entry traces as in the ideal scenario yields the highest accuracy across all training sets as expected. The classifiers that are trained on transduced traces from Retracer outperform the other augmentation strategies: DF improves by 10 percentage points (pp) relative to training with no augmentation (76→86%) and achieves within 3 pp of ideal accuracy, while Tik-Tok improves by 6 pp (79→85%) and achieves within 2 pp of ideal accuracy. Comparatively, the DF classifiers that are trained on augmented traces from NetAugment perform the worst of those tested, losing 19 pp (89→70%) relative to the ideal scenario. Moreover, NetAugment performs 6 pp lower than training directly on exit traces with no augmentation (76→70%), suggesting that randomly splitting, merging, or modifying bursts may not be strongly representative of real-world end-to-end trace variation. Finally, training without augmentation directly on the exit traces as suggested by the OnlineWF method performs 8–13 pp lower than ideal and is consistent with previous findings [8, Fig. 11].

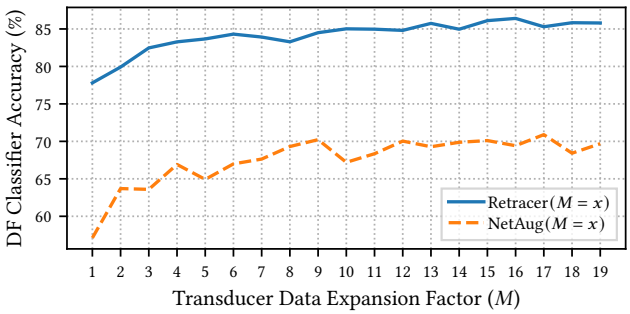

Figure 4: DF classifier accuracy in a multiclass closed-world experiment when training on datasets transduced with an increasing data expansion factor M and tested on $\text{Tor}(\text{entry}_2)$.

We further analyze the accuracy of classifiers trained using output from the augmenters for $M \in \bigcup_{i=1}^{19}\{i\}$ (which produces increasing amounts of training data). Here we limit our evaluation to the DF classifier, but otherwise use the same experimental setup as previously described. Fig. 4 shows slightly positive trends in accuracy for classifiers trained using both NetAug and Retracer training datasets. The classifier trained on Retracer(16) achieves the highest Retracer accuracy (86.4%) while the classifier trained on NetAug(17) achieves the highest NetAug accuracy (70.9%). Classifiers are consistently about 15 pp more accurate when trained using the Retracer than when trained using NetAugment.

Key Takeaway: Retracer is effective at transducing exit→entry traces, yielding performance estimates that are within 97% of the ideal scenario. Thus, Retracer is a viable method for obtaining realistic entry traces for estimating real-world WF performance.

5 Real-World WF Evaluation

In this section, we use *real-world* exit traces and Retracer to produce the best available estimate of real-world WF performance.

We highlight the advantages of including real-world exit traces in the WF training process. First, real exit traces will often naturally contain real examples of the websites in the adversary's monitored set (i.e., the positive class). These examples will form the most-similar representation of the monitored websites as they will be presented at testing time, and allowing a classifier to learn from them should improve its ability to differentiate them. Second, and perhaps more importantly, even if the positive class is not observed in the real-world data, the presence of examples of the unmonitored background websites (i.e., the negative class) will still be informative and help the classifier learn useful decision boundaries.

It is reasonable to expect that an adversary would want to utilize real-world exit traces given the significant advantages. Although one previous study has also considered that an adversary could train on real-world exit traces [8], they unrealistically tested against exit traces too which is inconsistent with the WF threat model. Our evaluation is the first to consider that exit-trained classifiers *must be tested against entry-side traces* to be realistic (see Fig. 1). We also consider than an adversary could use Retracer prior to training in order to improve WF performance (as we established in §4).

Table 3: Datasets used to Evaluate Real-World WF

Dataset	Description
GTT23(W)	GTT23 traces observed on all days of each week w in a set of weeks W: $\{w \mid w \in [1, 13]\}$ [23]
GoodEnough$(\cdot)$	GoodEnough traces from Jan or Feb, 2020 [38]
BigEnough$(\cdot)$	BigEnough traces partitioned by page using random assignment and a 60/40 split threshold [32]
Retracer(W, M)	$\mathcal{T}(\text{GTT23}(W), M, exit, entry)$ (see §3.2)

5.1 Methodology

5.1.1 Datasets. Our evaluation considers multiple input datasets as shown in Table 3, with a particular focus on real-world exit traces from the GTT23 dataset [23]. Recall from §2.4 that GTT23 is the largest published WF dataset with over 13 million cell traces measured over a period of 13 weeks in 2023. GTT23 is also the only WF dataset containing traces that were naturally produced by real Tor users and measured by real-world exit relays. GTT23 was very recently made available to WF researchers [24], and we are the first to use it to study real-world WF. The characteristics of GTT23 were previously evaluated in detail [23].

GTT23 contains only labeled *exit* traces. We are able to train classifiers (1) on entry traces that are transduced from the exit traces by Retracer as described in §3, and (2) directly on the exit traces as in the OnlineWF method of Cherubin et al. [8]. However, GTT23 does not contain labeled *entry* traces, which are needed in order to realistically evaluate the trained classifiers in the WF threat model. We address this limitation by separating GTT23 data into independent training and testing sets, and transducing the exit testing traces into entry testing traces using Retracer. We then test classifiers against entry traces from Retracer$(W, M = 1)$ to better approximate the problem facing a WF adversary while still maintaining the complexity and diversity found in the GTT23 traces.

The Retracer dataset is built following the Retracer methodology from §3 to transduce GTT23 exit traces into entry traces. In order to experiment with multiple values of M (the data expansion factor), we apply the Retracer methodology across simulation models for which we vary the tornettools load_scale to each value $\ell \in \bigcup_{i=0}^{7}\{2 + \frac{i}{10}\}$. We replay each GTT23 cell trace in a simulated Tor network for each value of ℓ. The Retracer transducer function $\mathcal{T}(I, M, exit, entry)$ is implemented using the simulated entry traces that are correlated with the unique identifier of the input trace I such that, when $M = m$, $\mathcal{T}(\cdot)$ returns the m correlated entry traces from the simulations configured with $\ell \in \bigcup_{i=0}^{m-1}\{2 + \frac{i}{10}\}$.

We also consider traces from two additional datasets, GoodEnough [38] and BigEnough [32]. GoodEnough and BigEnough are "synthetic" datasets in that they were programmatically collected with automated browsers, but they both include traces from many different webpages per website to help improve data diversity and thus are good candidates for comparison to GTT23 (see Appendix D for a full description).

5.1.2 Training. We train Deep Fingerprinting (DF) [43] classifiers (see §2.3) in a binary open-world setting in which the classification task is to associate examples of a selected website label with the positive class and the remaining examples of unselected labels with

the negative class. In the open-world setting, classifier training and testing sets are constructed such that some websites in the negative class only contribute traces to the testing set, which is consistent with the challenge faced by a real-world adversary.

Our evaluation considers a new perspective for real-world WF by forming a positive class around examples of *individual* websites. In a prior evaluation of WF using exit traces, Cherubin et al. considered the positive class to contain a number of different labels of a monitored set of websites to provide an *average* performance metric [8]. However, this method weakens our ability to explain why a *particular* classifier performs better or worse than another; a poorly performing website with a high base rate can reduce the average performance for the entire positive class, obfuscating the potentially high risk of WF against individual websites. In our evaluation, we train a unique classifier for each of a set of websites in order to isolate individual website performance and improve explainability in our subsequent feature importance analyses. Thus, our evaluation demonstrates the viability of an advantageous case for the adversary since classifying larger monitored sets should be more difficult. In practice, an adversary that is interested in multiple websites could combine single-website classifiers in an ensemble.

Prior to training, we construct training sets using a data selection function $Train(D, \omega) \mapsto \langle (w_i, C_i) \rangle$, where D is an input dataset of labeled cell traces, ω is a chosen positive class website label, and the output is a vector of labeled website (w) cell traces (C). $Train(\cdot)$ first selects from D traces with at least 1,000 cells (≈496 KB) to exclude erroneous and other short traces that are unlikely to carry full website transfers according to previous work [23]. To limit the amount of label imbalance among those composing the negative class, $Train(\cdot)$ then randomly samples traces by label such that each negative class website contributes no more than 10 example traces during training. The resulting vector is used to train a classifier on ω.

During training, the selected training set is split such that 80% of the traces are used to fit the classifier models and 20% are used for model validation; the split is stratified by class label such that an equal number of traces of each of the positive and negative classes is present in both the training and validation sets. The training set's positive class examples are then repeated such that the total number of traces in both classes are equal. We train each classifier using the optimized DF hyperparameters published by Sirinam et al., which include training for 30 epochs with a batch size of 128 [43, Table 1]. For each positive class ω, the final classifier after 30 training epochs is stored and used to evaluate performance during testing.

5.1.3 Testing. We construct testing sets using a data selection function $Test(D) \mapsto \langle (w_i, C_i) \rangle$ that is defined similarly to $Train(\cdot)$, including the 1,000 cell limit for rejecting short circuits but excluding the random sampling and filtering of traces by label. Note that the $Test(\cdot)$ selector does not differentiate based on a website label ω.

We measure test performance using precision and recall to account for the effects of low base rates on performance. Precision is $tp/(tp + fp)$ where tp and fp are the number of true and false positives, respectively. Recall is $tp/(tp + fn)$ where fn is the number of false negatives. We also use the F_1 score (the harmonic mean of precision and recall) as a single, overall performance metric. When reporting these metrics, we exclude results from the website classifiers for which we did not observe at least 30 positive examples in the testing set in order to ensure statistical strength in our estimates. We also exclude values for which a metric is undefined (because the denominator is zero). All metrics yield values between 0 and 1, with values closer to 1 indicating a better classifier. Testing is always performed with traces that were observed subsequent to the training traces, following best practices [37].

5.2 Evaluating Real-World WF

In this section, we evaluate the performance a WF adversary might expect when utilizing real-world traces that are naturally created by real Tor users and passively observed by exit relays (Step 1B in Fig. 1). Here, we consider that an adversary would want to be robust against the inconsistent training (exit) and testing (entry) positions, and so it uses Retracer to transduce the exit traces prior to training to mitigate this inconsistency. As previously discussed, testing is performed on entry traces to accurately represent the real-world threat model.

In our evaluation, we use Retracer traces replayed from GTT23's high-volume weeks ($\{1, 7, 13\}$) for training, and traces replayed from GTT23's low-volume weeks ($[2, 6] \cup [8, 12]$) for testing. More precisely, we define Experiment($W_{\text{Train}}, M, W_{\text{Test}}$) as follows:
(1) $\Omega \leftarrow$ labels from $Test(\text{GTT23}(W_{\text{Train}}))$ with ≥ 100 traces
(2) $\forall \omega \in \Omega$: train a model on $Train(\text{Retracer}(W_{\text{Train}}, M), \omega)$
(3) Test each trained model on $Test(\text{Retracer}(W_{\text{Test}}, 1))$
We conduct an experiment in which we train on traces from week 1 and test on traces from weeks 2–4 while varying Retracer's data expansion factor: Experiment($\{1\}, m, \{2, 3, 4\}$) $\forall m \in \{1, 4, 8\}$. We then repeat the experiment using traces from weeks 7–10: Experiment($\{7\}, m, \{8, 9, 10\}$) $\forall m \in \{1, 4, 8\}$. Across the six experiments (two for each value of m), we trained and tested 2,511 DF classifier models using more than 2,336,549 unique traces.

5.2.1 WF Results. Fig. 5 shows DF classifier performance for each of the three values of the data expansion factor M. We observed at least 30 test traces for 309 of 404 websites targeted for training in week 1, and for 343 of 433 websites targeted for training in week 7. Thus, each CDF plotted in Fig. 5 shows performance over 652 total individual website classifiers.

Overall, we find that WF performance varies greatly across websites. Although the classifiers for some websites perform well with precision or recall scores at or near 1.0, most classifiers perform poorly with fewer than 20% and 45% of the classifiers achieving greater than a 0.75 precision and recall score, respectively, and many achieving scores of 0.0. Moreover, the median F_1 score is at most 0.35 across all three values of M. Our results indicate that most of our trained classifiers are unlikely to be viable in a real-world WF attack since they would quickly be overrun with false positives.

Our results contradict those from prior WF studies considering less realistic threat models than we do. For example, Cherubin et al. study *average* performance across sets with different numbers of monitored websites and found that small monitored sets are fingerprintable while large sets are not [8]. However, our study of *individual* website fingerprintability reveals that even small monitored sets may not be fingerprintable if they are entirely composed of low-performing websites. We draw additional comparisons to prior work in §5.3 and §5.4.

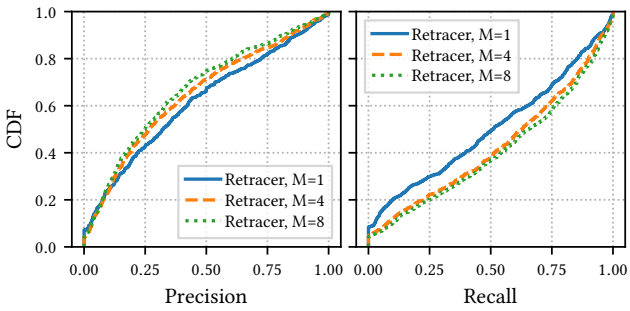

Figure 5: Distribution of precision and recall scores for DF classifiers trained and tested on entry traces from Retracer over three data expansion factors M.

5.2.2 Feature Importance. We now present the first analysis of the important features of real-world traces that most affect classification performance. Our analysis is novel because prior work did not have access to both real-world traces and their labels [8, § 6.3].

Because deep learning classification models do not use human-interpretable features, we conducted a meta-analysis using per-domain summary statistics computed from the traces of each domain. For each of the 652/837 websites that was observed while testing with $Test(\text{Retracer}(\{2,3,4\},1))$ and $Test(\text{Retracer}(\{8,9,10\},1))$ we trained a random forest regressor to predict each target classifier's overall achieved recall, precision, and F_1 score using the statistics of the websites' traces as features. The set of possible trace features we considered were drawn from prior work [13, 29]. The regressor's split points are used to identify important features [6].

Table 4 shows the top 5 features identified by our analysis predicting classifier performance. In the table, "num traces" means the total number of traces in the dataset, "count incoming" means the number of server-to-client[3] cells in the specified cell interval, and "CUMUL x" means the cumulative sum of cell direction values after the circuit is x% complete [35]. The value ρ shows the feature's Spearman rank correlation with classification performance, which indicates a feature's monotonic relationship with performance and takes on values in the range $[-1, 1]$. The correlation value ρ is positive if an increase in the feature's value improves performance and is negative if a decrease in the value degrades performance. The table shows that, for real-world traces, characteristics of the *front* of the trace (i.e., the first few cells) are highly predictive of overall performance. For example, increases in standard deviation of the number of server-to-client cells in the first 10 cells has a strong negative correlation with overall F_1 score. Our results also show that website popularity is an important dataset characteristic; more popular websites are classified with higher recall, possibly due to the larger number of traces that can be used during classifier training. In Appendix E, we give an expanded listing of feature importances in Table 5 and a description of all features we considered.

We visualize the effects of two important trace features on WF performance: the number of traces of the positive class during training, and the variation in the length of the positive class traces.

[3]Prior work considers both incoming *and* outgoing cell counts, but we use only incoming counts since outgoing counts are simply the numerical complement.

Table 4: Important per-website trace features. The value ρ shows the feature's rank correlation with performance.

	Rank	Feature	Importance	ρ
Recall	1	Num traces	.26	.49
	2	Avg of Count Incoming $[0, 10)$	.21	.50
	3	Stddev of Count Incoming $[0, 100)$	.14	−.44
	4	Stddev of Count Incoming $[0, 10)$	.03	−.51
	5	Stddev of Count Incoming $[0, 30)$	.03	−.51
Precision	1	Stddev of Count Incoming $[0, 10)$	.27	−.50
	2	Avg of Count Incoming $[0, 10)$	.10	.51
	3	Stddev of CUMUL 10	.04	−.22
	4	Max of Count Incoming $[0, 100)$	.04	−.18
	5	Median of Length of Longest Burst	.02	.06
F_1	1	Stddev of Count Incoming $[0, 10)$	.28	−.54
	2	Avg of Count Incoming $[0, 10)$	.11	.54
	3	Num traces	.05	.22
	4	Stddev of CUMUL 10	.05	−.25
	5	Median of Count Incoming $[0, 10)$	.04	.49

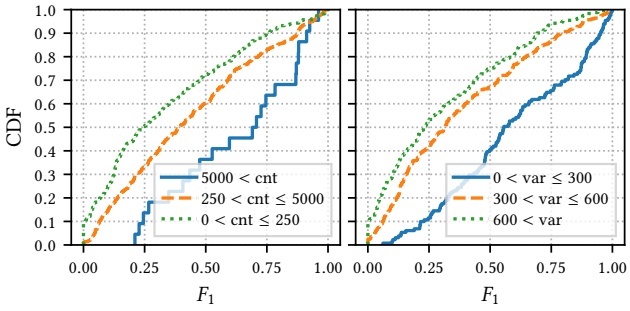

Figure 6: Distribution of the F_1 score when training and testing on entry traces obtained from Retracer with $M = 4$. Results are categorized by the trace count ("cnt") and relative variance of trace lengths ("var") per website.

Fig. 6 shows the distribution of performance for classifiers that are separated by the associated website feature values into three bins as shown in the legend. We find that the median F_1 score increases from 0.25 for websites with at most 250 example traces to 0.70 for websites with more than 5,000 traces. In other words, the median F_1 score increased by 0.45 when 20× as many traces were available during training. When grouped by the variation in trace length (computed here as the relative standard deviation), the median F_1 score increases from 0.23 for websites with lengths that vary by more than 600 cells to 0.56 for websites with lengths that vary by at most 300 cells. Lower trace length variation indicates that a website may have more consistent patterns that are easier to learn, generally resulting in classifiers that achieve higher F_1 scores.

5.2.3 Temporal Analysis. We next conduct a temporal analysis across the 13 weeks of GTT23 data to understand the effects of concept drift on WF classifier performance. Note that previous work considered a time span of at most one week [8].

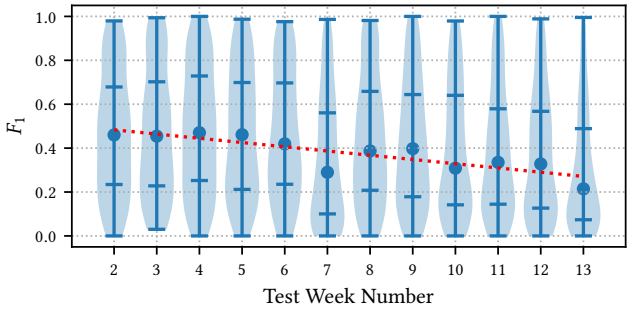

Figure 7: Temporal concept drift analysis: we train on traces from week 1 and test on traces from subsequent weeks. The circles and dashed red line show the weekly medians and trendline, respectively. The horizontal lines mark the min, Q1, Q3, and max while violin width estimates density.

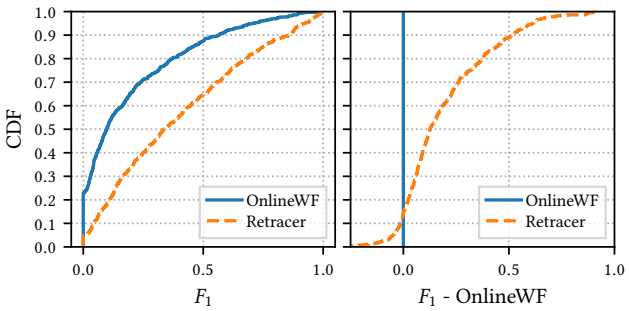

Figure 8: Classifier performance when training on exit traces as in OnlineWF [8] and training on entry traces transduced from the exit traces by Retracer.

Our evaluation considers the 404 classifiers trained using week 1 traces and tested on a weekly basis for each of the following 12 weeks. That is, we perform 12 experiments: Experiment($\{1\}, 4, \{w\}$) $\forall w \in [2, 13]$. We focus on the classifiers trained with traces from Retracer's $M = 4$ setting here and in the remainder of this section because it achieved the greatest mean F_1 score. Following our prior methodology, for each test week we report the results across the training websites for which we observed at least 30 examples of the positive and negative classes during that week; this process resulted in the removal of at least 63/404 websites (week 7) and at most 278/404 websites (week 10) for which we do not have a statistically rigorous estimate.

The results of our temporal analysis of F_1 scores are shown in Fig. 7, wherein the circles and dotted red line show the weekly medians and trendline, respectively. We observe that the density of the weekly distributions (represented visually by the width of the violins) shifts downward over time, especially during weeks 7–13. The trendline demonstrates a strong statistical correlation of the downward shift in the medians ($r = -0.99$), where the median F_1 score decreases from a maximum of 0.47 in week 4 to a minimum of 0.21 in week 13. Our analysis again demonstrates the challenging task of WF against real-world GTT23 data: few of the classifiers that perform well in week 2 still achieve an F_1 score above 0.5 in week 13. Note that it would be natural for an adversary to continuously collect exit traces and periodically retrain its classifiers in order to mitigate concept drift, but this does come at a cost.

Key Takeaway: We conduct the first WF evaluation that uses real-world GTT23 data while testing on entry-side traces, and present the best available estimate of real-world WF performance. We find low WF performance relative to previous work, even when using Retracer to build robustness to the trace capture position.

5.3 Comparison to Online WF

In the OnlineWF method of Cherubin et al., the adversary continuously trains WF classifiers on real-world *exit* traces in real time [8], and in the WF threat model those classifiers should be tested on *entry* traces. Thus, we evaluate the performance of classifiers trained in the OnlineWF scenario by training classifiers directly on the exit traces from GTT23 and testing them on entry traces as they would be in a real-world WF attack. More formally, we define the experiment OnlineWF($W_{\text{Train}}, W_{\text{Test}}$) as follows:
(1) $\Omega \leftarrow$ labels from $Test(\text{GTT23}(W_{\text{Train}}))$ with ≥ 100 traces
(2) $\forall \omega \in \Omega$: train a model on $Train(\text{GTT23}(W_{\text{Train}}), \omega)$
(3) Test each trained model on $Test(\text{Retracer}(W_{\text{Test}}, 1))$
We use OnlineWF($\{1\}, \{2, 3, 4\}$) and OnlineWF($\{7\}, \{8, 9, 10\}$) to evaluate the OnlineWF performance we expect can be achieved in the real world. Across the two experiments, we trained and tested 837 DF classifiers using more than 709,339 unique traces.

We compare the results from the two OnlineWF($\cdot$) experiments above to the results from Retracer in §5.2 (the two Experiment($\cdot$) runs with M=4) to show how the Retracer method improves upon the state-of-the-art method of training on real-world traces. Fig. 8 compares classifier performance across the 652 websites for which we have statistically rigorous estimates (i.e., at least 30 testing examples); larger values ($\rightarrow$ on the x-axis) indicate better performance. We plot the distribution over the absolute F_1 scores of the classifiers in the left subplot. We observe an appreciable improvement in F_1 scores when using Retracer: the median F_1 score is 0.34 for Retracer and 0.1 for OnlineWF, a difference of 0.24. We plot the difference between the Retracer and OnlineWF scores on a per-website basis in the right subplot. We observe that the Retracer method outperforms the OnlineWF method for over 85% of the websites, with an increase in the F_1 score of 0.91 in the best case.

We remark that Cherubin et al. did not have access to labeled entry traces and thus could only *estimate* OnlineWF performance. In a simplified evaluation using synthetic traces they found that incorrectly testing on exit rather than entry traces resulted in an overestimation of classifier accuracy of 5–18% [8, § 6.4]. Using GTT23, we reevaluated the accuracy of their estimation method and found that performance is overestimated for 90% of the websites we tested: 0.17 in the median and ranging from near zero to a worst case of 0.93. This range of overestimation is much greater than the previously estimated 5–18%. See Appendix F for more details.

Key Takeaway: Classifiers trained on Retracer entry traces outperform those trained on GTT23 exit traces when considering that they will be tested on entry traces in a real-world WF attack.

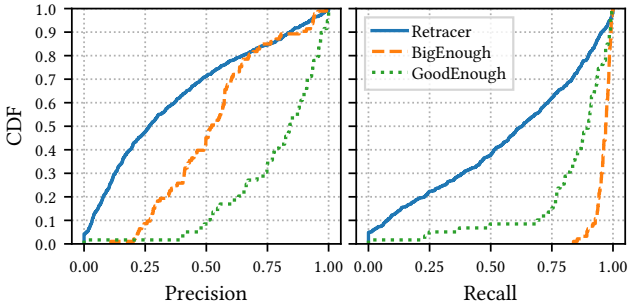

Figure 9: Performance of the classifiers trained and tested with each dataset. "Synthetic" traces lead to better performance than Retracer traces (transduced from GTT23).

5.4 Comparison to WF with Synthetic Datasets

In the previous sections, we studied WF by (1) considering traces created by real Tor users and measured by exit relays, and (2) testing on entry traces to more accurately reflect the WF threat model. In applying Retracer to GTT23, we believe our evaluation yields the best available estimate of the performance an adversary can achieve when directing WF attacks at real Tor users. In this section, we compare the real-world WF performance estimated with Retracer and GTT23 to the WF performance estimated with BigEnough and GoodEnough, two recent "synthetic" datasets that were programmatically collected using automated browsers (see §5.1.1).

5.4.1 Isolated Datasets. We first consider the performance that is achieved by isolating the datasets and independently evaluating WF on each. To perform the evaluation, we define the experiment Synthetic($D_{\text{Train}}, D_{\text{Test}}$) as follows:

(1) $\Lambda \leftarrow$ labels from $Test(D_{\text{Train}})$ with ≥ 100 traces

(2) $\forall \omega \in \Lambda$: train a model on traces from $Train(D_{\text{Train}}, \omega)$

(3) Test each trained model on traces from $Test(D_{\text{Test}})$

We conduct a Synthetic(BigEnough(60), BigEnough(40)) experiment in which we use 72,092 unique traces to train and test 93 classifiers, and a Synthetic(GoodEnough(Jan), GoodEnough(Feb)) experiment in which we use 24,476 unique traces to train and test 59 classifiers. We compare the performance evaluated in these isolated experiments to the performance we independently evaluated with Retracer in §5.2 (the two Experiment($\cdot$) runs with $M = 4$). The results are shown in Fig. 9. Overall, we find that the DF classifiers perform *much* better when trained and tested against synthetic traces than when trained and tested against Retracer traces. For example, the worst case and median precision is respectively 0.11 and 0.52 when testing the BigEnough classifiers, compared to just 0.0 and 0.27 when testing the Retracer classifiers. Similarly, the worst case and median recall is respectively 0.84 and 0.97 when testing the BigEnough classifiers, compared to just 0.0 and 0.63 when testing the Retracer classifiers. The GoodEnough classifiers tend to achieve higher precision and lower recall than the BigEnough classifiers, but the results support a similar conclusion: DF is considerably less effective at learning how to differentiate GTT23 traces than traces from the synthetic datasets, which may lead researchers using synthetic data to overestimate real-world WF performance.

5.4.2 Synthesized Datasets. Having found that WF is more challenging when faced with real-world data than with synthetic data, we now evaluate a training strategy in which the adversary supplements real-world traces with additional synthetic traces. The real-world traces are useful in teaching the classifier how to distinguish both positive and negative classes that it will encounter at test time, while synthetic traces may reinforce the classifier's concept of the chosen monitored set. Due to space, we summarize our experiment and results here (see Appendix G for full details).

We evaluate the effects of synthesized data on WF by training on an isolated Retracer dataset and an isolated BigEnough dataset as described in the previous section, and a third Combined dataset that is a concatenation of the two. Consistent with prior work [8, Fig. 4], we find a comparable performance distribution for both the Retracer and Combined classifiers, while performance for classifiers trained on the synthetic BigEnough dataset is poor with a maximum F_1 score of just 0.10. We further extend the analysis to consider the absolute difference in performance on a *per-website* basis, a new perspective. We find that the classifiers trained on the Combined dataset performed just slightly better than the Retracer dataset alone for about half of the websites, where in the best case the F_1 score increases by 0.17 and in the worst case it decreases by 0.12. More work considering more recent synthetic datasets is needed to understand if this result is statistically significant.

Key Takeaway: WF classifiers trained on the BigEnough and GoodEnough synthetic datasets broadly overestimate the performance that can be achieved when training and testing on real-world traffic patterns from Retracer, while supplementing real-world training data with synthetic examples offers questionable benefit.

6 Conclusion

In this paper we present Retracer, a novel method for cell trace transduction that employs high-fidelity network simulation and trace replay to produce traces in a target network position given traces from other positions. Retracer is important in the study of real-world WF because it enables researchers to utilize real labeled exit-side traces containing the natural behavior of real Tor users (such as those in GTT23 [23]) while at the same time accurately representing the WF threat model in which testing occurs on entry-side traces. Through extensive evaluation, we show that a real-world adversary training on exit traces could incur significant performance penalties that could be almost completely mitigated by applying Retracer to those traces prior to training. Our study makes other novel contributions to the study of WF, including the first analysis of feature importance considering real-world traces which is made possible by our study of individual website fingerprintability across thousands of trained website classifiers. We also provide novel insights concerning previous WF methods by revisiting them within a more realistic threat model. Based on our results, we believe that training and testing directly on traces from GTT23 is a reasonable approximation of an adversary that applies Retracer to exit traces prior to entry deployment (see Appendix H for a comparison).

Acknowledgments

This work was supported by the Office of Naval Research (ONR).

References

[1] Gunes Acar, Marc Juarez, and individual contributors. 2023. Tor-browser-selenium - Tor browser automation with Selenium. https://github.com/webfp/tor-browser-selenium.

[2] Alireza Bahramali, Ardavan Bozorgi, and Amir Houmansadr. 2023. Realistic website fingerprinting by augmenting network traces. In *ACM CCS 2023*. DOI: 10.1145/3576915.3616639.

[3] Alireza Bahramali, Ardavan Bozorgi, and Amir Houmansadr. 2023. Realistic website fingerprinting by augmenting network traces. https://github.com/SPIN-UMass/Realistic-Website-Fingerprinting-By-Augmenting-Network-Traces.

[4] Sanjit Bhat, David Lu, Albert Kwon, and Srinivas Devadas. 2019. Var-CNN: A data-efficient website fingerprinting attack based on deep learning. *PoPETs*, 2019, 4. DOI: 10.2478/popets-2019-0070.

[5] George Dean Bissias, Marc Liberatore, David D. Jensen, and Brian Neil Levine. 2005. Privacy vulnerabilities in encrypted HTTP streams. In *PET 2005*. Vol. 3856. DOI: 10.1007/11767831_1.

[6] Leo Breiman, Jerome Friedman, R.A. Olshen, and Charles J. Stone. 1984. *Classification and Regression Trees*.

[7] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. 2012. Touching from a distance: website fingerprinting attacks and defenses. In *ACM CCS 2012*. DOI: 10.1145/2382196.2382260.

[8] Giovanni Cherubin, Rob Jansen, and Carmela Troncoso. 2022. Online website fingerprinting: evaluating website fingerprinting attacks on Tor in the real world. In *USENIX Security 2022*. EPRINT: https://www.usenix.org/conference/usenixsecurity22/presentation/cherubin.

[9] Xinhao Deng, Qilei Yin, Zhuotao Liu, Xiyuan Zhao, Qi Li, Mingwei Xu, Ke Xu, and Jianping Wu. 2023. Robust multi-tab website fingerprinting attacks in the wild. In *2023 IEEE Symposium on Security and Privacy*. DOI: 10.1109/SP46215.2023.10179464.

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

[11] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. 2012. Peek-a-boo, I still see you: why efficient traffic analysis countermeasures fail. In *2012 IEEE Symposium on Security and Privacy*. DOI: 10.1109/SP.2012.28.

[12] Jiajun Gong, Wuqi Zhang, Charles Zhang, and Tao Wang. 2022. Surakav: generating realistic traces for a strong website fingerprinting defense. In *2022 IEEE Symposium on Security and Privacy*. DOI: 10.1109/SP46214.2022.9833722.

[13] Jamie Hayes and George Danezis. 2016. k-fingerprinting: A robust scalable website fingerprinting technique. In *USENIX Security 2016*.

[14] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. 2009. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *The Workshop on Cloud Computing Security*.

[15] Andrew Hintz. 2002. Fingerprinting websites using traffic analysis. In *PET 2002*. Vol. 2482. DOI: 10.1007/3-540-36467-6_13.

[16] James K Holland, Jason Carpenter, Se Eun Oh, and Nick Hopper. 2024. DeTorrent: an adversarial padding-only traffic analysis defense. *PoPETs*, 2024, 1.

[17] Rob Jansen, Kevin Bauer, Nicholas Hopper, and Roger Dingledine. 2012. Methodically modeling the Tor network. In *Workshop on Cyber Security Experimentation and Test*.

[18] Rob Jansen and Nicholas Hopper. 2012. Shadow: running Tor in a box for accurate and efficient experimentation. In *NDSS 2012*.

[19] Rob Jansen, Jim Newsome, and Ryan Wails. 2022. Co-opting linux processes for High-Performance network simulation. In *USENIX ATC 2022*. EPRINT: https://www.usenix.org/conference/atc22/presentation/jansen.

[20] Rob Jansen, Justin Tracey, and Ian Goldberg. 2021. Once is never enough: foundations for sound statistical inference in tor network experimentation. In *USENIX Security 2021*. EPRINT: https://www.usenix.org/conference/usenixsecurity21/presentation/jansen.

[21] Rob Jansen, Matthew Traudt, and Nicholas Hopper. 2018. Privacy-preserving dynamic learning of tor network traffic. In *ACM CCS 2018*. DOI: 10.1145/3243734.3243815.

[22] Rob Jansen and Ryan Wails. 2023. Data-explainable website fingerprinting with network simulation. *PoPETs*, 2023, 4. DOI: 10.56553/popets-2023-0125.

[23] Rob Jansen, Ryan Wails, and Aaron Johnson. A measurement of genuine Tor traces for realistic website fingerprinting. (2024). arXiv: 2404.07892 [cs.CR].

[24] Rob Jansen, Ryan Wails, and Aaron Johnson. GTT23: A 2023 dataset of genuine Tor traces. DOI: 10.5281/zenodo.10620519.

[25] Marc Juárez, Sadia Afroz, Gunes Acar, Claudia Díaz, and Rachel Greenstadt. 2014. A critical evaluation of website fingerprinting attacks. In *ACM CCS 2014*. DOI: 10.1145/2660267.2660368.

[26] Kate Keahey, Jason Anderson, Zhuo Zhen, Pierre Riteau, Paul Ruth, Dan Stanzione, Mert Cevik, Jacob Colleran, Haryadi S. Gunawi, Cody Hammock, Joe Mambretti, Alexander Barnes, François Halbach, Alex Rocha, and Joe Stubbs. 2020. Lessons learned from the Chameleon testbed. In *The USENIX Annual Technical Conference*.

[27] Gregory M Kurtzer, Vanessa Sochat, and Michael W Bauer. 2017. Singularity: scientific containers for mobility of compute. *PloS one*, 12, 5.

[28] Wladimir De la Cadena, Asya Mitseva, Jens Hiller, Jan Pennekamp, Sebastian Reuter, Julian Filter, Thomas Engel, Klaus Wehrle, and Andriy Panchenko. 2020. TrafficSliver: fighting website fingerprinting attacks with traffic splitting. In *ACM CCS 2020*. DOI: 10.1145/3372297.3423351.

[29] Shuai Li, Huajun Guo, and Nicholas Hopper. 2018. Measuring information leakage in website fingerprinting attacks and defenses. In *ACM CCS 2018*. DOI: 10.1145/3243734.3243832.

[30] Marc Liberatore and Brian Neil Levine. 2006. Inferring the source of encrypted HTTP connections. In *ACM CCS 2006*. DOI: 10.1145/1180405.1180437.

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

[32] Nate Mathews, James K Holland, Se Eun Oh, Mohammad Saidur Rahman, Nicholas Hopper, and Matthew Wright. 2023. SoK: a critical evaluation of efficient website fingerprinting defenses. In *2023 IEEE Symposium on Security and Privacy*. DOI: 10.1109/SP46215.2023.10179289.

[33] Se Eun Oh, Nate Mathews, Mohammad Saidur Rahman, Matthew Wright, and Nicholas Hopper. 2021. GANDaLF: GAN for data-limited fingerprinting. *PoPETs*, 2021, 2. DOI: 10.2478/popets-2021-0029.

[34] Se Eun Oh, Saikrishna Sunkam, and Nicholas Hopper. 2019. p-FP: extraction, classification, and prediction of website fingerprints with deep learning. *PoPETs*, 2019, 3. DOI: 10.2478/popets-2019-0043.

[35] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. 2016. Website fingerprinting at internet scale. In *NDSS 2016*.

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

[37] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. 2019. TESSERACT: eliminating experimental bias in malware classification across space and time. In *USENIX Security 2019*.

[38] Tobias Pulls. 2020. Towards effective and efficient padding machines for Tor. *arXiv preprint arXiv:2011.13471*.

[39] Tobias Pulls and Rasmus Dahlberg. 2020. Website fingerprinting with website oracles. *PoPETs*, 2020, 1. DOI: 10.2478/popets-2020-0013.

[40] Mohammad Saidur Rahman, Payap Sirinam, Nate Mathews, Kantha Girish Gangadhara, and Matthew Wright. 2020. Tik-Tok: the utility of packet timing in website fingerprinting attacks. *PoPETs*, 2020, 3. DOI: 10.2478/popets-2020-0043.

[41] Vera Rimmer, Davy Preuveneers, Marc Juárez, Tom van Goethem, and Wouter Joosen. 2018. Automated website fingerprinting through deep learning. In *NDSS 2018*.

[42] Meng Shen, Kexin Ji, Zhenbo Gao, Qi Li, Liehuang Zhu, and Ke Xu. 2023. Subverting website fingerprinting defenses with robust traffic representation. In *USENIX Security 2023*. EPRINT: https://www.usenix.org/conference/usenixsecurity23/presentation/shen-meng.

[43] Payap Sirinam, Mohsen Imani, Marc Juárez, and Matthew Wright. 2018. Deep fingerprinting: undermining website fingerprinting defenses with deep learning. In *ACM CCS 2018*. DOI: 10.1145/3243734.3243768.

[44] Payap Sirinam, Nate Mathews, Mohammad Saidur Rahman, and Matthew Wright. 2019. Triplet fingerprinting: more practical and portable website fingerprinting with N-shot learning. In *ACM CCS 2019*. DOI: 10.1145/3319535.3354217.

[45] Qixiang Sun, Daniel R. Simon, Yi-Min Wang, Wilf Russell, Venkata N. Padmanabhan, and Lili Qiu. 2002. Statistical identification of encrypted web browsing traffic. In *2002 IEEE Symposium on Security and Privacy*. DOI: 10.1109/SECPRI.2002.1004359.

[46] Paul F. Syverson, David M. Goldschlag, and Michael G. Reed. 1997. Anonymous connections and onion routing. In *1997 IEEE Symposium on Security and Privacy*. DOI: 10.1109/SECPRI.1997.601314.

[47] 2023. The Tor Metrics Portal. https://metrics.torproject.org.

[48] Tao Wang. 2020. High precision open-world website fingerprinting. In *2020 IEEE Symposium on Security and Privacy*. DOI: 10.1109/SP40000.2020.00015.

[49] Tao Wang. 2021. The one-page setting: A higher standard for evaluating website fingerprinting defenses. In *ACM CCS 2021*. DOI: 10.1145/3460120.3484790.

[50] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. 2014. Effective attacks and provable defenses for website fingerprinting. In *USENIX Security 2014*.

[51] Tao Wang and Ian Goldberg. 2016. On realistically attacking tor with website fingerprinting. *PoPETs*, 2016, 4. DOI: 10.1515/popets-2016-0027.

[52] Renjie Xie, Yixiao Wang, Jiahao Cao, Enhuan Dong, Mingwei Xu, Kun Sun, Qi Li, Licheng Shen, and Menghao Zhang. 2023. Rosetta: enabling robust TLS encrypted traffic classification in diverse network environments with TCP-aware traffic augmentation. In *USENIX Security 2023*.

Appendix

A NetAugment Details

In this section we provide an extended description of NetAugment. Note that we provided an overview in §2.4 while Bahramali et al. describe these operations in more detail [2, Alg. 1–4].

NetAugment operates on cell *bursts*, which are subsequences of cell traces such that all cells in a subsequence have the same direction and the cell before and after the burst are opposite-direction cells. More formally, a subsequence $C[i{:}j]$ of cell trace C is considered a burst if $(d_i = d_k)_{k=i}^{j}$, $d_{i-1} = d_{j+1}$, and $d_{i-1} \neq d_i$. NetAugment randomly applies one of three burst manipulations to a trace: (1) *modify incoming bursts*, where burst sizes are randomly increased for short cell traces and randomly decreased for long cell traces; (2) *insert outgoing bursts*, where incoming bursts are randomly split and outgoing bursts are inserted; and (3) *merge incoming bursts*, where outgoing bursts are randomly dropped and incoming bursts are merged. Finally, a shift operation is applied by dropping the last n cells and prepending n 0-sized cells.

B Network Simulation Details

In §4 and §5, Retracer replays traces from GTT23 inside of Shadow simulations of the Tor network. We run our simulation of Tor in Shadow using a blade server cluster in which each blade contains identical hardware: 1 TiB of RAM and 2×18 core Intel Xeon Gold 6354 CPUs (36 total cores and 72 total hyperthreads). Each blade is configured to run a minimal version of Debian 11 with Linux kernel `v5.10.0`, and the simulations are run in containers using Singularity [27]. We use Tor v0.4.7.10, Shadow at commit `e502d20ed`, tgen at commit `15d1eab3c`, oniontrace at commit `3696db432`, and tornettools at commit `9716a8682`.

In our experiments, we model Tor networks that represent 15% of the size and scale of the live Tor network using Tor metrics modeling data from the period 2023-01-01 to 2023-03-31. Our baseline Tor network model uses 946 Tor nodes and 1,129 traffic generation processes to create 447,084 circuits every 10 minutes and emulate the simultaneous traffic load of 112,851 users.

B.1 Simulated Network Performance

An important value in setting up the Shadow models is the load_scale value ℓ, which dictates how much background traffic is added into the simulated Tor network by the tgen traffic generation tools. To produce background traffic, tgen currently uses Markov models with parameters that were last updated in 2018 [21]. At that time, ℓ=1.0 was the default setting that produced the load that most closely matched the live Tor network in 2018. Since then, recent work has discovered that ℓ=2.0 produced load that more closely represents more recent Tor networks [22]. Thus, we use ℓ=2.0 in our baseline Tor network model, and larger values to represent more highly loaded networks. Recall that from this baseline, the Retracer method adjusts ℓ to different settings in order to replay traces across a more diverse set of network conditions. Changing ℓ causes the modeling tools to adjust the Markov model parameters to increase or decrease the number of background traffic circuits that are created during the simulation, which affects the congestion levels of the Tor relays.

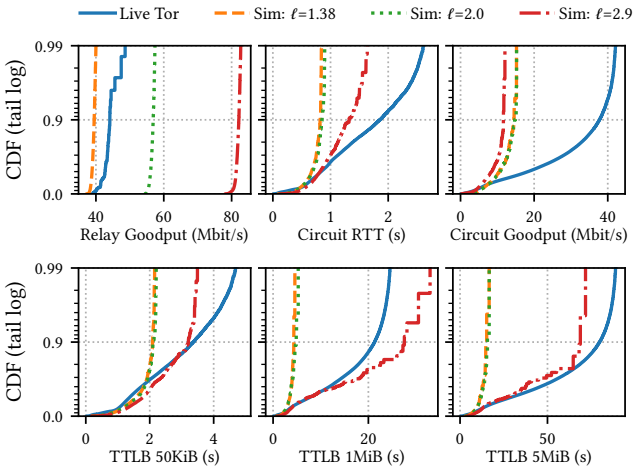

Figure 10: Cumulative distributions for six metrics commonly used to gauge Tor network performance. We compare metrics measured in the live Tor network during our modeling period with those measured in our Shadow simulations across the minimum, baseline, and maximum load values that we condsidered in our Retracer experiments (i.e., ℓ=1.38, ℓ=2.0, and ℓ=2.9, respectively).

We plot in Fig. 10 six metrics commonly used to gauge Tor network performance. We show the performance that was measured on the live Tor network during our modeling period [47], and compare it to the performance in the Shadow-simulated Tor networks for different values of ℓ. In particular, we show performance for the smallest, baseline, and greatest values of ℓ that we considered in §4 and §5, that is, respectively ℓ=1.38, ℓ=2.0, and ℓ=2.9. We observe that the performance in the Shadow simulations generally approximate the live Tor performance, and indeed the network becomes more congested for higher load values as indicated by higher round-trip and download times and lower circuit goodput. Thus, we confirm that our models produce a varied set of network conditions as intended, across which Retracer will replay traces.

B.2 Trace Replay Cost

Recall that the Retracer methodology replays GTT23 traces inside of Tor network simulations. The Retracer variant adds additional processes into the simulation: a replay server on each exit relay virtual host and 10 replay clients on new virtual hosts. In our Retracer experiments, we configure each replay client to replay 1,000 traces in parallel, and we replay a total of 115,000 traces across the 10 replay clients in each simulation.

We analyze the resource cost of the Retracer strategy by measuring the time and memory required to run the baseline simulation (with ℓ=2.0) with and without Retracer. The resources required to run the baseline simulations, and the added costs for replaying the traces with Retracer, are shown in Fig. 11. We observe an insignificant increase is memory (RAM) usage to run the replay processes (right subplot). However, we observe a larger increase in runtime (left subplot). Starting at 600 seconds into the simulation, the Retracer clients start running and replaying 1,000 traces in parallel,

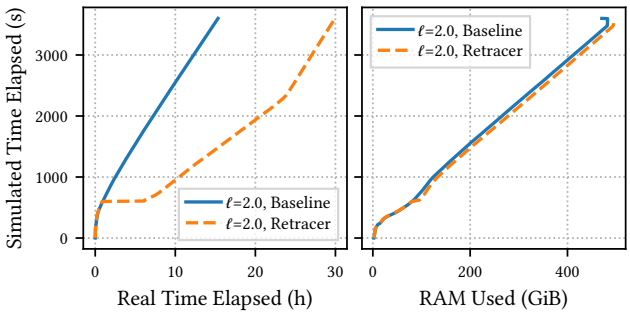

Figure 11: The resource costs required to run a single Shadow simulation of a 15% Tor network with and without Retracer replaying 115,000 traces in parallel during the simulation.

and each trace requires a new circuit to be built. Following the 600 second simulation time, ongoing circuit builds and data transfers for each trace further contributes to the increased runtime. In total, the baseline simulation without Retracer completes in 15.3 hours while the Retracer variant completes in 29.9 hours. Thus, we compute that on average each trace that is replayed adds about 2.2 seconds to the overall runtime of the simulation (7,871 traces can be replayed per added hour of runtime).

Note that we did not attempt to reduce the costs of replaying traces since optimizing the resource costs was not part of our research. However, there are some techniques that could limit the costs. For example, we run our experiments in simulated Tor networks that represent the live Tor network at a scale of 15%, but running them in smaller networks would significantly reduce the required resources. Additionally, the Shadow models include background traffic generators by default, but it is possible that those generators could be removed since the trace replay nodes may already produce adequate traffic. Future work could consider how these strategies impact the extracted entry traces and the performance of classifiers trained on them.

C Ethical Considerations

The measurements described in §4.2 were conducted using the live Tor network. We build approximately 80,000 circuits through Tor using a client under our control, and fetch a single webpage through each circuit. Although we only add a modest load to the network relative to the billions of circuits Tor is estimated to handle daily [31] and its total throughput of 300 Gbit/s [47], we spread our measurement over several days to limit our impact on the network and its users. Our own relay serves as one of three relays in every circuit we build; while measuring our circuits, our relay also contributes bandwidth to support general Tor usage. Finally, a goal of this work is to reduce the future necessity of such measurements.

D Description of Synthetic Datasets

Our evaluation in §5.4 considers two synthetic datasets: the GoodEnough dataset from Pulls [38], and the BigEnough dataset from Mathews et al. [32]. GoodEnough and BigEnough both include traces from different webpages per website to help improve dataset diversity. Additionally, both datasets include traces for *monitored* websites that the adversary considers sensitive, and traces for *unmonitored* background websites for open-world analysis. These features make them good candidates for comparison under a WF classification task to classifiers trained using Retracer traces whose patterns originate from GTT23.

D.1 GoodEnough

The GoodEnough dataset was collected from December 2019 to February 2020. During each of the three collection months, a dataset of 20,000 traces was collected. Half of those traces were collected from 50 websites selected from among the Alexa top-300. For each selected website, 10 of its webpages were selected and 20 traces collected per webpage, for a total of $50 \times 10 \times 20 = 10,000$ traces. The other half of the traces consists of a single trace from each of 10,000 webpages collected from the top URLs of reddit.com/r/all. These traces were collected to form the negative class during open-world analysis. All traces were collected using scripted Tor Browser instances configured not to use entry guards. Data was collected for each of the three different Tor Browser safety settings: Standard, Safer, and Safest. In our evaluation, we only use the Standard data, which is the default Tor Browser setting.

We removed 1,909 short traces from GoodEnough. Because GoodEnough contains traces collected in consecutive months, we partition the remaining traces into a training set of traces that temporally precedes a testing set of traces following best practices [37]. Consequently, we name the collection of traces from January 2020 as GoodEnough(Jan) and the collection of traces from February 2020 as GoodEnough(Feb). Note that GoodEnough label hashes were produced consistently with GTT23.

D.2 BigEnough

The BigEnough dataset was collected from November 2021 to January 2022. It is similar to GoodEnough and is collected using the same tools, but it includes more and different websites. Unlike GoodEnough, BigEnough provides a single crawl spanning its collection period and does not provide full crawls in different months. We obtained 161,254 total traces from the BigEnough authors. The monitored set was composed of 74,511 traces from the 121 most-popular websites as ranked by the Open PageRank Initiative. From each of those sites, 17 webpages were selected and visited 36 times on average. The unmonitored set was composed of the remaining 86,743 traces of front pages sampled from the remaining top websites in the ranking. We use the data from the default Standard Tor Browser safety level, but data from other levels was also collected.

We removed 56,686 short traces from BigEnough. Because BigEnough traces are not temporally separated, we partition the remaining traces into a training set BigEnough(60) and a testing set BigEnough(40) by page using random assignment such that 60% of the pages appear in BigEnough(60) and 40% appear in BigEnough(40). Thus, BigEnough(60) does not contain every page of every BigEnough website: a page might be a part of either BigEnough(60) or BigEnough(40) but not both, which ensures that unobserved pages will be encountered during testing as would occur in practice. Note that BigEnough label hashes were produced consistently with GTT23.

E Feature Importance

Table 5 shows an extended view of the results that we presented in Table 4 in §5.2. For the full meta-analysis, we used the following collection of trace-level features:

Trace Length The total number of cells in the trace.
Count Incoming (Interval) The total number of server-to-client cells in the specified cell interval. We used the intervals $[0, m)$ for $m \in \{10, 30, 50, 100\}$.
Count Incoming (All) The number of server-to-client cells.
Count Outgoing (All) The number of client-to-server cells.
Length of Longest Burst The number of cells in the longest burst.
Number of Bursts The total number of bursts in the trace.
CUMUL x The cumulative sum of cell direction values after the circuit is x% complete [35].

These trace features were aggregated on a domain-wise basis using 5 different summary statistics: (1) average, (2) minimum, (3) maximum, (4) standard deviation, and (5) median.

F Online WF Estimation Error

This section provides extended details and analysis from §5.3. In the OnlineWF method of Cherubin et al., the adversary continuously trains WF classifiers on real-world *exit* traces in real time [8], and in the WF threat model those classifiers should be tested on *entry* traces. However, the authors did not have access to real-world entry traces with which to test the classifiers they trained on exit traces. Thus, they were only able to *estimate* the performance of their own OnlineWF method by *both* training *and* testing on real-world exit traces, which is an inaccurate representation of the WF threat model. To understand the error in their estimation, they conducted a simplified evaluation using synthetic traces and found that incorrectly testing on exit rather than entry traces resulted in an overestimation of classifier accuracy of 5–18% [8, § 6.4].

Using GTT23, we reevaluate the accuracy of their method of testing on exit traces to estimate OnlineWF performance. We compare the performance of classifiers both trained and tested on exit traces to those trained on exit traces but tested on entry traces. We define the experiment Estimate($W_{\text{Train}}, W_{\text{Test}}$) as follows:
(1) $\Omega \leftarrow$ labels from *Test*(GTT23(W_{Train})) with ≥ 100 traces
(2) $\forall \omega \in \Omega$: train a model on *Train*(GTT23(W_{Train}), ω)
(3) Test each trained model on *Test*(GTT23(W_{Test}))
We define an experiment Actual($W_{\text{Train}}, W_{\text{Test}}$) as follows:
(1) Train as in Steps 1 and 2 of Estimate($W_{\text{Train}}, W_{\text{Test}}$)
(2) Test each trained model on *Test*(Retracer(W_{Test}, 1))
We use Estimate($\{1\}, \{2, 3, 4\}$) and Estimate($\{7\}, \{8, 9, 10\}$) to evaluate the performance estimated by Cherubin et al., and we use Actual($\{1\}, \{2, 3, 4\}$) and Actual($\{7\}, \{8, 9, 10\}$) to evaluate the OnlineWF performance we expect can be achieved in the real world. Across the 4 experiments, we trained and tested a total of 1,674 DF classifiers using a total of more than 1,033,020 unique traces.

Fig. 12 compares classifier performance across the 652 websites for which we have statistically rigorous estimates (i.e., at least 30 testing examples); larger values ($\rightarrow$ on the x-axis) indicate better performance. We plot the distribution over the absolute F_1 scores of the classifiers in the left subplot. We observe that the OnlineWF estimate in which the exit-trained classifiers are unrealistically

Table 5: Expanded set of important per-website trace features.

	Rank	Feature	Importance	ρ
Recall	1	Num traces	.26	.49
	2	Avg of Count Incoming $[0, 10)$	.21	.50
	3	Stddev of Count Incoming $[0, 100)$	.14	−.44
	4	Stddev of Count Incoming $[0, 10)$	.03	−.51
	5	Stddev of Count Incoming $[0, 30)$	.03	−.51
	6	Stddev of CUMUL 10	.02	−.28
	7	Median of Count Incoming $[0, 10)$	.02	.41
	8	Stddev of Count Incoming (All)	.02	−.39
	9	Stddev of Length of Longest Burst	.02	−.27
	10	Median of Length of Longest Burst	.02	.01
	11	Stddev of Count Incoming $[0, 50)$	.01	−.48
	12	Max of CUMUL 10	.01	−.07
	13	Stddev of Trace Length	.01	−.34
	14	Min of Count Outgoing (All)	.01	.03
	15	Avg of Length of Longest Burst	.01	.05
	16	Stddev of Number of Bursts	.01	−.20
	17	Stddev of CUMUL 20	.01	−.31
	18	Median of Number of Bursts	.01	−.03
	19	Avg of Number of Bursts	.01	−.07
	20	Avg of Count Incoming $[0, 30)$	.01	.33
Precision	1	Stddev of Count Incoming $[0, 10)$	.27	−.50
	2	Avg of Count Incoming $[0, 10)$	.10	.51
	3	Stddev of CUMUL 10	.04	−.22
	4	Max of Count Incoming $[0, 100)$	.04	−.18
	5	Median of Length of Longest Burst	.02	.06
	6	Num traces	.02	.12
	7	Stddev of Length of Longest Burst	.02	−.14
	8	Median of Count Incoming $[0, 10)$	.02	.47
	9	Max of CUMUL 10	.02	−.03
	10	Stddev of Count Incoming $[0, 30)$	.02	−.40
	11	Avg of Length of Longest Burst	.02	.05
	12	Min of Number of Bursts	.01	.08
	13	Stddev of CUMUL 20	.01	−.23
	14	Stddev of Trace Length	.01	−.25
	15	Stddev of Count Incoming $[0, 100)$	.01	−.30
	16	Min of Count Outgoing (All)	.01	.00
	17	Stddev of Count Incoming $[0, 50)$	.01	−.32
	18	Max of Count Outgoing (All)	.01	−.05
	19	Max of CUMUL 80	.01	.03
	20	Max of Length of Longest Burst	.01	−.08
F_1	1	Stddev of Count Incoming $[0, 10)$	.28	−.54
	2	Avg of Count Incoming $[0, 10)$	.11	.54
	3	Num traces	.05	.22
	4	Stddev of CUMUL 10	.05	−.25
	5	Median of Count Incoming $[0, 10)$	.04	.49
	6	Stddev of Count Incoming $[0, 50)$	.02	−.38
	7	Stddev of Count Incoming $[0, 30)$	.02	−.45
	8	Stddev of Length of Longest Burst	.02	−.18
	9	Stddev of Count Incoming $[0, 100)$	.02	−.35
	10	Min of Number of Bursts	.01	.06
	11	Stddev of Trace Length	.01	−.29
	12	Max of Count Incoming $[0, 100)$	.01	−.18
	13	Max of CUMUL 10	.01	−.02
	14	Median of Length of Longest Burst	.01	.06
	15	Avg of Length of Longest Burst	.01	.06
	16	Stddev of CUMUL 60	.01	−.28
	17	Median of Number of Bursts	.01	.04
	18	Min of Count Outgoing (All)	.01	.00
	19	Stddev of CUMUL 20	.01	−.26
	20	Stddev of CUMUL 50	.01	−.28

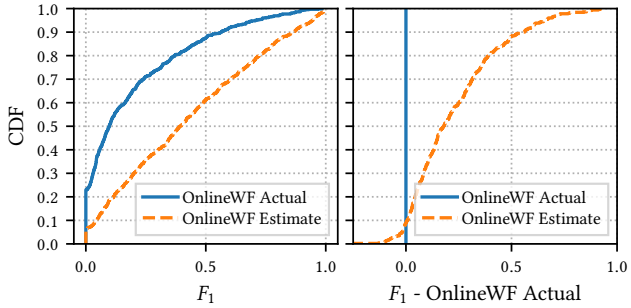

Figure 12: OnlineWF performance when testing on exit traces as in the estimation of Cherubin et al. [8], compared to actual performance when tested on Retracer entry traces. Using exit traces to estimate the test set leads to overestimation.

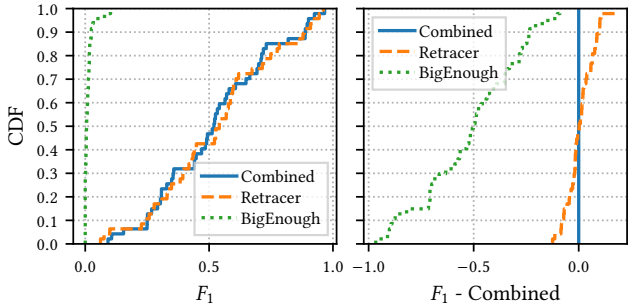

Figure 13: Distribution of the F_1 score over the classifiers that are trained on common labels with Retracer, BigEnough, and combined datasets and tested on Retracer entry traces.

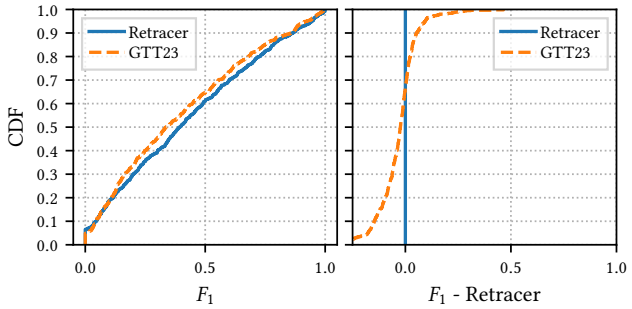

Figure 14: Training and testing directly on exit traces from GTT23 is a reasonable approximation of the more realistic Retracer method of training and testing on entry traces.

tested on exit traces generally tends to overestimate performance. The estimated F_1 scores are greater than the actual scores by 0.14 (0.02→0.16), 0.29 (0.1→0.39), and 0.34 (0.32→0.66) in the Q1, median, and Q3, respectively. The right subplot shows the extent of the overestimation relative to the actual scores on a per-website basis. We observe that performance is overestimated for 90% of the websites we tested: 0.17 in the median and ranging from near zero to a worst case of 0.93. This range of overestimation is much greater than the previously estimated 5–18% [8, § 6.4].

G WF Evaluation on Synthesized Data

This section provides extended details and analysis of the evaluation of the benefit of synthetic data that we summarized in §5.4.2.

We have found in §5.4 that WF is more challenging when faced with real-world data than with synthetic data. We now seek to understand the effectiveness of a training strategy in which the adversary supplements real-world traces with additional synthetic traces. The real-world traces are useful in teaching the classifier how to distinguish both positive and negative classes that it will encounter at test time, while synthetic traces may reinforce the classifier's concept of the monitored set. We define a Combine($D_{\text{Train}}, D_{\text{Test}}$) experiment as follows:

(1) $\alpha \leftarrow$ labels from $Test(D_{\text{Train}})$ with ≥ 100 traces
(2) $\beta \leftarrow$ labels from $Test(\text{BigEnough}(60))$ with ≥ 100 traces
(3) $\forall \omega \in \alpha \cap \beta$: train a model on traces from the concatenated $Train(D_{\text{Train}}, \omega) \parallel Train(\text{BigEnough}(60), \omega)$
(4) Test each trained model on traces from $Test(D_{\text{Test}})$

We evaluate the overall effect of synthesized datasets by running a Combine(Retracer($\{1\}, 4$), Retracer($\{2, 3, 4\}, 1$)) experiment and a Combine(Retracer($\{7\}, 4$), Retracer($\{8, 9, 10\}, 1$)) experiment, resulting in the training of 49 new DF classifiers. We compare the performance of the classifiers from the Combine($\cdot$) experiments with that of the BigEnough and Retracer classifiers with labels in $\alpha \cap \beta$ that were trained and tested as described in §5.

Fig. 13 shows the distribution of F_1 scores over the 47/49 common websites for which we observed at least 30 traces in the test set. In the left plot, we observe a comparable performance distribution for both the Retracer and Combined classifiers, while performance for classifiers trained on the synthetic BigEnough dataset is poor with a maximum F_1 score of just 0.10. These results are consistent with previous work [8, Fig. 4]. We further extend the analysis to consider the absolute difference in performance on a *per-website* basis, a new perspective. The right plot in Fig. 13 shows that the classifiers trained on the combined dataset performed just slightly better than the Retracer dataset alone for about half of the websites, where in the best case the F_1 score increases by 0.17 and in the worst case it decreases by 0.12. Future work should consider expanding this analysis in order to consider the effects of more recent synthetic datasets.

H WF with Retracer vs. GTT23

Here we seek to understand classier performance when both training and testing directly on exit traces from GTT23. If training and testing directly on exit traces from GTT23 yields similar performance to training and testing on entry traces from Retracer, then using GTT23 directly in lieu of Retracer may be a reasonable approximation that could prevent future researchers from incurring the added costs associated with Retracer. Thus, we simply compare the results from training and testing on exit traces from GTT23 (i.e., the two Estimate($\cdot$) experiments from Appendix G) with results from training and testing on Retracer entry traces (i.e., the two Experiment($\cdot$) experiments from § 5.2 with M=4). This comparison is shown in Fig. 14 and demonstrates that researchers who use GTT23 alone may already be able to closely approximate the Retracer method that a real-world adversary could use.