Browser Fingerprinting from Coarse Traffic Summaries: Techniques and Implications

Ting-Fang Yen[1], Xin Huang[2], Fabian Monrose[2], and Michael K. Reiter[2]

[1] Carnegie Mellon University, Pittsburgh, PA
[2] University of North Carolina, Chapel Hill, NC

Abstract. We demonstrate that the browser implementation used at a host can be passively identified with significant precision and recall, using only coarse summaries of web traffic to and from that host. Our techniques utilize connection records containing only the source and destination addresses and ports, packet and byte counts, and the start and end times of each connection. We additionally provide two applications of browser identification. First, we show how to extend a network intrusion detection system to detect a broader range of malware. Second, we demonstrate the consequences of web browser identification to the deanonymization of web *sites* in flow records that have been anonymized.

Keywords: Application fingerprinting, traffic deanonymization, malware detection, machine learning.

1 Introduction

On many large networks, the most fine-grained representation of network traffic that is feasible to collect is a coarse summary of each network flow or connection, e.g., flow formats as produced by CISCO NetFlow. Such formats typically include only source and destination addresses and ports, flow start and end times, and packet and byte counts. Limiting the data collected to this information can dramatically reduce the reporting bandwidth and storage requirements by orders of magnitude in comparison to full packet capture, and is widely supported today in commodity routers. Additionally, due to privacy concerns, network administrators are generally reluctant to share packet traces containing payload information, and so flow data presents a good compromise between privacy and utility. For these reasons, flow logging for traffic volume estimation is now common practice, and applications of flow logs for network intrusion detection are increasingly being studied[1].

In this paper we examine a novel use of flow logs, namely to infer the application software running on hosts whose traffic is represented in a flow log. We demonstrate this by focusing on a particular software application, namely web browsers, and show that the browser implementation on a host (e.g., Internet

[1] For example, the annual FloCon workshop is devoted to this topic (`http://www.cert.org/flocon/`).

U. Flegel and D. Bruschi (Eds.): DIMVA 2009, LNCS 5587, pp. 157–175, 2009.

Explorer (IE), Firefox, Opera, or Safari) can be determined given its web traffic in flow records, *without* access to payload information. More importantly, our techniques do not rely upon observing web retrievals that are unique to a single browser platform, e.g., Firefox checking for updates at the Firefox update server. We eschew such telltale events both because they tend to be relatively rare (e.g., Opera checks for updates only once per week) and so might not be represented in a flow log under consideration, but also because in the case of anonymized network data, such events may not be evident.

Rather, our techniques infer the browser implementation by applying machine-learning techniques to the behavioral features of the traffic in which it is involved when interacting with regular sites, as observed even in coarse flow records. It is arguably surprising that browser implementations could be discerned in this way, since a browser's network behavior is primarily determined by the content and structure of the pages it accesses. Moreover, classification could be complicated by various factors that are inherent in traffic, including variations in the users' browsing behavior or browser configuration, differences in the web page content being retrieved (both across different websites and in the same website over time), the client hardware configuration, and the different geographic locations from which the content is retrieved.

One of the contributions of this work is to evaluate the impact of the above factors on the classification accuracy of the browser type. We do so on the basis of web traffic induced by the four most popular browsers, as measured in retrievals of the main pages of the top 150 websites on the Internet[2] over the course of two months, and on the basis of web retrievals recorded at the border of the Carnegie Mellon University network. Our results show that even when the training and testing datasets are from different time frames, to different websites, and collected at different geographic locations, we were still able to achieve 75% classification precision and 60% recall (see Section 4).

Our focus of web browsers for this study is partly due to their relative importance among applications today, but is also due to the implications of their identification. A second contribution of our work is the demonstration of these implications, in two contexts. First, because of their widespread use, attacks that exploit vulnerabilities in specific browser implementations have also emerged. In this context, inferring the type of browser from traffic traces is beneficial for network intrusion detection systems that identify hosts infected by platform-dependent malware by observing suspicious traffic from hosts with similar software configurations. We describe an application of this in the TĀMD system [1], which previously classified hosts as similar on the basis of only their operating systems. Our techniques enable TĀMD to incorporate browser similarity into this evaluation.

Second, we demonstrate the consequences of web browser identification to the deanonymization of web *sites* in flow records that have been anonymized. We show that the identification of the web browser in flow records enables the application of per-browser website classifiers to yield a more precise deanonymization

[2] According to Alexa, `http://www.alexa.com/`

of the websites represented in the traffic than has previously been achievable from flow records. Specifically, we show that we can deanonymize websites visited by a host using a per-browser classifier with up to a 17% improvement in precision over the case in which we use a similarly trained generic classifier.

To summarize, our contributions include (i) techniques to identify the web browser represented in flow records of web page accesses; (ii) quantification of the impact of various factors on browser classification accuracy; (iii) the application of this technique to improve network intrusion detection systems; and (iv) the application of this technique to more accurately deanonymize the websites represented in anonymized network traffic.

2 Related Work

Many fingerprinting tools are *active* in nature, probing services with carefully crafted queries (e.g., those produced by `Nmap` and `Nessus`) to detect implementation-specific characteristics [2,3]. More relevant to our work are *passive* fingerprinting techniques that infer the implementations of network applications or operating systems based solely on observing the traffic they send. Passive fingerprinting tools and techniques are numerous, though most focus on identifying TCP/IP implementations and utilize specific information [4,5,6] that is unavailable in coarse flow records. While passive techniques have more recently been proposed to identify the *application* (e.g., peer-to-peer file transfers versus web retrievals) or the class of application (e.g., interactive sessions versus bulk-data transfers) reflected in packet traces [7,8,9,10,11], few proposals (e.g., [12,13,14,15]) have done so from coarse flow records. Moreover, to the best of our knowledge, none of these proposed techniques attempt to identify particular *implementations* of an application (e.g., the browser) from passive observations of flow records alone.

We explore in this paper the implications of browser identification for the problem of deanonymizing web *sites* in anonymized flow records. Several works have examined the susceptibility of anonymized traffic traces to deanonymization, e.g., [16,17,18]. Similar to our work, these approaches re-identify hosts or websites on the basis of their behaviors as exhibited in the anonymized traffic traces. However, none of these earlier works have taken into consideration the fact that on-the-wire behaviors are influenced by the particular implementation of their protocol peers. As we show later, first classifying the browser involved in a web retrieval can improve the fidelity with which one can deanonymize websites present in anonymized network flows.

We also demonstrate how reliable identification of the browser can be used to detect platform-dependent malware by identifying suspicious traffic coming from hosts with similar software configurations [1]. Like TĀMD, other network intrusion-detection systems employ fingerprints of host software platforms when detecting intrusions, though most generate these fingerprints actively (e.g., [19]). As far as we are aware, none do so passively on the basis of coarse flow information, however, and so our techniques might enhance a range of network intrusion-detection systems when flow information is all that is available.

3 Data Sets

The empirical analysis in this paper takes advantage of several sources of data recorded in the Argus (Audit Record Generation and Utilization System [20]) flow format. Argus is a real time flow monitor based on the RTFM flow model [21,22]. Argus inspects each packet and groups together those with the same attribute values into one bi-directional record. In particular, TCP flows are identified by the 5-tuple (source IP address, destination IP address, source port, destination port, protocol)[3], and packets in both directions are summarized into a single Argus flow record. The browser fingerprinting techniques we propose in this paper require only that each flow record include the source and destination IP addresses and ports, the protocol, and the total bytes and packets sent in each direction. In our data collection, however, we extend this basic flow record format with additional information — notably, the first 64 bytes of payload on the connection, and time-to-live (TTL) values in IP packet headers — for the sole purpose of determining ground truth of certain attributes to use in our evaluation. To be clear, this additional information is not used by our classifiers, and is only taken into consideration when determining the accuracy of our techniques and for extracting testing instances from live network data.

We use the following data sources in our evaluations:

The CMU dataset. This dataset consists of anonymized traffic from the edge routers of the wired CMU campus network, which includes one /16 subnet. We do not consider hosts (that is, IP addresses) from the wireless network, since those hosts typically have short-lived DHCP-assigned IP addresses, such that hosts using different browsers may be associated with the same address, leading to inconsistencies in the data. The rate of the traffic in the CMU dataset is about 5000 flow records per second, and was collected over six weeks from October to December 2007. We are interested in reducing this dataset only to web retrievals for the purposes of this paper, but one of the challenges in processing live network data is in accurately identifying the boundaries that separate website retrievals (c.f., [18,23]). In this work, we leverage the first 64 bytes of each flow to identify the start boundary of a website retrieval from a host internal to the CMU network. More specifically, we define a web retrieval to begin with a port-80 connection comprised of an HTTP request of the form "GET / ", as such a connection would be highly unlikely to be part of another retrieval. The web retrieval is then comprised of this flow and all subsequent flows originating from the same host in the next 10 seconds. Our choice of 10 seconds is based on empirical evaluations. The use of the flow payload for parsing web retrievals can be replaced, for example, by checking for a certain amount of idle time before a burst of web traffic [16], though we do not explore this alternative here. Incomplete retrievals, or those with less than three flows, do not carry enough information about the browser implementation in order for the

[3] Since Argus records are bi-directional, the source and destination IP addresses are swappable in the logic that matches packets to flows. However, the source IP address in the record is set to the IP address of the host that initiated the connection.

classifier to make a well-grounded decision, and so we only consider retrievals with more than three flows in our analysis.

As mentioned earlier, we examine the 64 bytes of available payload in each flow to infer the browser involved in the retrieval. Specifically, for the purposes of ground-truth, a host is identified to be using the Opera browser if the user-agent string in its HTTP request starts with the string "Opera". Firefox hosts are identified by the special "safe-browsing" requests issued by the browser to check the validity of the website being contacted (`https://wiki.mozilla.org/Phishing_Protection`). Due to the 64-byte restriction in the available payload length, we were not able to reliably identify hosts using IE and Safari in the CMU data set.

The PlanetLab-Native dataset. In order to perform our evaluations in which the CMU dataset serves as the testing data, we would like a training dataset from hosts that are diverse in terms of geography and hardware platform. PlanetLab [24] offers a platform that is generally available and that enables the retrieval of web pages from a wide range of hosts with different hardware configurations and geographic locations. To collect this dataset, we deployed a program to fourteen hosts across five PlanetLab networks; this program sequentially retrieved the front page (i.e., generating "GET / " HTTP requests) of the top 150 most popular websites in the U.S. (according to Alexa) repeatedly over the course of one month. Each web retrieval was comprised of the flows observed in the thirty seconds since the start of the retrieval. Machines on PlanetLab are required to run a Linux operating system, so we performed retrievals from Linux-compatible browsers, namely Firefox and Opera[4]. Recall that these two browsers are also the only ones reliably identifiable in the CMU dataset, and so the PlanetLab-Native dataset can serve well as training data for testing with the CMU dataset.

The PlanetLab-QEMU dataset. In an effort to develop a dataset that includes traffic for all of the major browsers (IE, Firefox, Opera and Safari), we utilized a processor emulator, QEMU [25], to run an emulated Windows operating system on PlanetLab hosts. As in the PlanetLab-Native dataset, we ran an automated program to sequentially retrieve the front page of the top 150 most popular websites repeatedly over the course of one month. Each web retrieval was comprised of the flows observed in the thirty seconds since the start of the retrieval. We deployed this emulated version of Windows on seven hosts across three PlanetLab networks[5].

Arguably, the PlanetLab datasets may not accurately represent website retrievals generated by actual user activities, where frequent visits to a particular website may result in much of the content being cached. To compensate for this effect, we set the browser cache sizes to be sufficiently large (400MB) so that objects would not be evicted from cache.

[4] To generate our PlanetLab-Native dataset, we used Firefox 2.0.0.16 and Opera 9.51.

[5] To generate our PlanetLab-QEMU dataset, we used IE 7.0, Firefox 2.0.0.13, Opera 9.51 and Safari 3.1.

Feature Selection

To capture browser-specific characteristics in network traffic, we extracted nine main features from each website retrieval, listed in Table 1. The mean, standard deviation, maximum, minimum, median, first and third quartile, inter-quartile range, and the cumulative sum, are also calculated for each flow statistic. Our feature selection strategy is based on examining the information gain associated with each of the statistics for the aforementioned nine main features. More specifically, using the PlanetLab-Native dataset, we select the top statistics whose cumulative information gain accounts for at least 90% of the overall information gain. These selected statistics are combined into a feature vector F_r for website retrieval r. Among the most important features are those associated with the byte and packet counts in each direction, the cumulative flow duration, and the retrieval duration. While we have not fully explored the root cause for all of these differences, they are related to the different orders in which the browsers retrieve objects on a given page, different numbers of objects retrieved in one connection, and the numbers of connections that can be active simultaneously. Of course, while these features play an important role in distinguishing different browser implementations in our tests, we acknowledge that they may not be optimal for distinguishing browsers not included in the training data, or future browser versions that behave fundamentally differently from the ones covered in this study. That said, the methodology outlined in this paper can be easily applied to incorporate new browser types into the classifier.

Table 1. Main features extracted for each retrieval

Flow Statistics	Byte count (in each direction)
	Packet count (in each direction)
	Flow duration
	Number of flows active simultaneously to this one
	Start time minus most closely preceding flow start time
Retrieval Statistics	Total number of flows
	Cumulative byte count from destination
	Cumulative flow duration
	Retrieval duration

4 Browser Identification from Flow Records

As discussed in Section 1, our first goal is to develop techniques for inferring the browser implementation that is participating in recorded flows that represent web retrievals from that browser. At first, it might seem that distinguishing the browser should be difficult, since a browser primarily serves to interpret and render the HTML and other types of content it receives. As such, its behavior should be primarily dictated by the content it is accessing.

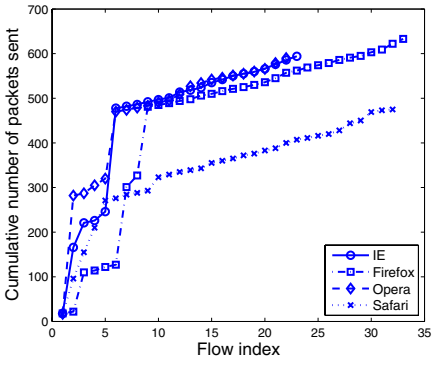

Fig. 1. Number of packets sent from the browser, accumulated over all flows that comprise the retrieval. Each retrieval is to `http://www.cnn.com/`.

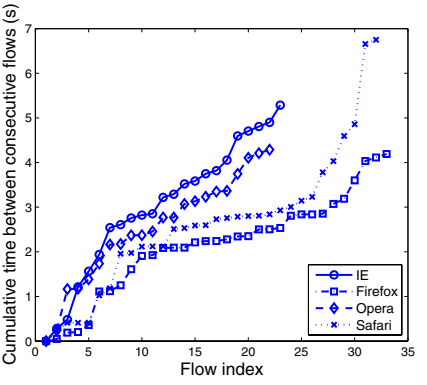

Fig. 2. Cumulative time between consecutive flows that comprise the retrieval. Each retrieval is to `http://www.cnn.com/`.

An example of why this intuition might not be true is shown in Figure 1, which shows just one feature (see Table 1) for the four most popular browsers (IE, Firefox, Opera, and Safari) when each retrieved `http://www.cnn.com/` at nearly the same time and from a host in the University of North Carolina campus network. The feature pictured is the number of packets sent from the browser, accumulated over all flows that comprise the retrieval. It is evident that in these retrievals, Firefox initiates more flows than the other browsers, Opera sends more packets in earlier flows, and Safari sends fewer packets overall. Figure 2 shows the start time of each flow minus the most closely preceding flow start time, accumulated over all flows in the retrieval. This feature clearly shows that certain browsers (e.g., Firefox) try to improve response time by multiplexing the retrieval of content across substantially more flows than other browsers.

However, using these differences to reliably determine the browser from flow records is not as straightforward as it may seem, and in particular is not as easy to automate as Figures 1–2 might suggest. Aside from the content and structure of the websites, users' browsing behavior, browser configuration, geographic location, and the client hardware configuration can also affect browser network behavior. As such, in the remainder of this section we test with what precision and recall an automatic classifier can distinguish among browsers in different scenarios.

More specifically, the classifier type that we utilize is Support Vector Machines (SVM)[6], which have been widely applied to many supervised learning problems [27,28]. Given two sets of labeled data, the SVM finds a hyperplane that separates the data and maximizes the distance to each data set. When multiple classes are involved, the SVM generates a group of pair-wise binary

[6] We utilize the SVM implementation included in the Weka machine learning package [26].

classifiers. Each binary classifier gives a vote to a class, and the final classification is the class with the highest vote. Loosely speaking, since an instance is classified depending on which side of the separating hyperplane it lies on, and not necessarily on how far from the hyperplane it is, there can be cases where an instance is misclassified if it is located "close" to the separating hyperplane.

To aid in our classification, we modify the aforementioned application of SVMs to incorporate a notion of "confidence". The confidence threshold is the minimum distance of the hyperplane from the testing instance, where only instances with distance to the hyperplane greater than the confidence threshold are classified. This allows the classifier to avoid making decisions in ambiguous situations that would likely result in incorrect classifications.

The general structure of each test described below is that we first train a browser classifier on one dataset and then classify each retrieval in another dataset to obtain a guess of the browser used in that retrieval. Each website retrieval is classified only if its distance to the separating hyperplanes is greater than the confidence threshold. The classifier then determines the type of browser used by host h to be the browser classified most often in h's retrievals. To avoid errors due to a host having a small number of retrievals, we only consider hosts with more than thirty classified retrievals in our analysis. Our choice of thirty retrievals was determined empirically, and provides a good balance between precision and the number of hosts classified from the dataset.

We denote the classification for host h to be $\mathsf{browserguess}(h)$, and the actual browser used by host h to be $\mathsf{browser}(h)$. Note that $\mathsf{browser}(h) = \perp$ if the actual browser for h could not be determined, which occurred in the CMU dataset in some cases; see Section 3. Also, $\mathsf{browserguess}(h) = \perp$ can result if the classifier makes no classification for h, since no overwhelming choice arises for h's retrievals. The precision and recall across all hosts in the test dataset is defined as follows:

$$\begin{aligned}
\mathsf{Precision} &= \Pr[\mathsf{browser}(h) = b \mid \mathsf{browserguess}(h) = b \neq \perp] \\
&= \frac{|\{h : \mathsf{browserguess}(h) = \mathsf{browser}(h)\}|}{|\{h : \mathsf{browserguess}(h) \neq \perp\}|} \\
\mathsf{Recall} &= \Pr[\mathsf{browserguess}(h) = b \mid \mathsf{browser}(h) = b \neq \perp] \\
&= \frac{|\{h : \mathsf{browserguess}(h) = \mathsf{browser}(h)\}|}{|\{h : \mathsf{browser}(h) \neq \perp\}|}
\end{aligned}$$

Keep in mind that a classifier that makes random guesses, i.e., classifying each host as a particular browser with $\frac{1}{n}$ probability, where n is the number of browsers, and a network where the browsers are distributed evenly among the hosts, the precision can only be expected to be $\frac{1}{n^2}$.

4.1 Tests on PlanetLab-QEMU Dataset

In an ideal web browsing scenario, only one website retrieval is taking place at any time, such that boundaries between consecutive retrievals are clearly delineated, and each webpage is allowed to fully download before the next one. While

this idealistic scenario will be compounded by many other issues in practice, we argue that tests in a controlled environment are valuable in that they enable us to better understand what factors influence classification the most.

We evaluate the results of browser identification under this setting using the PlanetLab-QEMU dataset. To simulate multiple hosts, each running a specific browser implementation, data from each host is separated by the browser that generated the traffic. This traffic pertaining to a specific browser from one host serves as testing data, while the classifier is trained on traffic from all other hosts, for each experiment. Since in some applications it will not be possible to obtain retrievals from every website that may be present in the testing data, we set the training data to be traffic from the top 100 websites, and use traffic from the remaining 50 websites (from top 100 to 150) for testing.

The precision and recall are shown in Figure 3, for confidence thresholds set to one of $\{0.35, 0.65, 0.95, 1.15, 1.30, 1.50\}$. The rise in precision is likely due to incorrect classifications being filtered out as a result of the increase in confidence threshold, to the point that most of a host's classified retrievals are then correct. On the other hand, recall decreases with the confidence since more hosts are unclassified (i.e., $\{h : \mathsf{browserguess}(h) = \perp\}$). In all cases the correct browser can be identified with at least 71% precision and recall, and the precision grows to 100% with recall at 43% as the confidence threshold is increased. These results show that browser implementations exhibit different traffic behaviors that can be accurately identified even in coarse flow records.

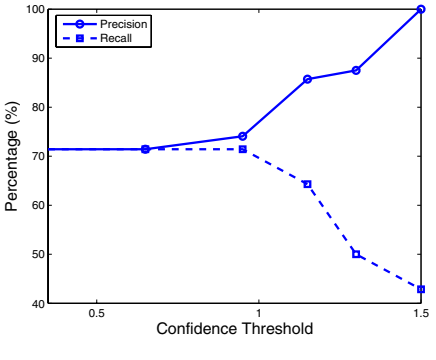

Fig. 3. Precision and recall for browser classification on the PlanetLab-QEMU dataset

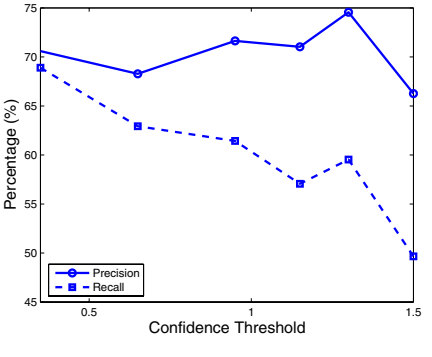

Fig. 4. Precision and recall for browser classification on the CMU dataset (Train: PlanetLab-Native, Test: CMU)

4.2 Tests on CMU Dataset

Unlike the controlled setting of the PlanetLab experiments, the CMU dataset provides a setting for evaluating our techniques on traffic recorded in the real world. That said, we remind the reader that for purposes of ground truth, we

could only reliably identify hosts using Firefox and Opera in the CMU dataset, and consequently, our analysis here is restricted to these cases. Out of those hosts, the vast majority of them used Firefox, and to not bias our results to that of a single-browser evaluation, we randomly select Firefox hosts in the CMU dataset but ensure that we have an equal number of Firefox and Opera hosts. The PlanetLab-Native dataset is used in this case for training a browser classifier.

Figure 4 shows the precision and recall for the CMU dataset, for confidence thresholds set to one of $\{0.35, 0.65, 0.95, 1.15, 1.30, 1.50\}$. The precision generally increases slightly with the confidence threshold, as instances that were incorrectly classified are now filtered out (because they were too close to the separating hyperplane), while recall decreases as a higher threshold leads to more unclassified instances (i.e., $\{h : \mathsf{browserguess}(h) = \perp\}$). As the confidence threshold increases, some hosts whose majority of retrievals were correctly classified now have those correct classifications filtered out, so that these hosts are left with more misclassified retrievals that cause the browser to be identified incorrectly; this results in a decrease in precision at the end of the curve. The peak in precision is 74.56%, when the confidence threshold is 1.30. We note that in this test (where the number of Firefox and Opera hosts are balanced) our precision is substantially greater than that of random guessing (i.e., 25%).

5 Applications to Network Intrusion Detection

TĀMD (Traffic Aggregation for Malware Detection) [1] is an intrusion detection system that passively observes traffic passing through an enterprise network border to identify internal hosts infected by stealthy malware, such as botnets and spyware. TĀMD exploits the observation that, however subtle, stealthy malware still needs to communicate to exfiltrate data to the attacker, to receive commands, or to carry out the commands. Moreover, since malware rarely infiltrates only a single host in a large enterprise, these communications should emerge from multiple hosts within coarse temporal proximity to one another (e.g., within an hour of one another). Based on these observations, TĀMD functions by finding new communication "aggregates" involving multiple internal hosts, i.e., communication flows that share common characteristics.

One of the characteristics on which TĀMD aggregates traffic is the *platform* of the internal hosts involved in sending or receiving that traffic, which is useful for identifying platform-dependent malware infections. That is, suspicious traffic common to a collection of hosts becomes even more suspicious if the hosts share a common software platform. Previously, forming platform aggregates in TĀMD was based solely on the hosts' operating systems. As such, malware that is application-dependent, such as malware that exploits Firefox only[7], might span multiple aggregates formed by O/S fingerprinting alone (if the exploit works on

[7] Examples of such application-dependent malware are the `Infostealer.Snifula` trojan that exploits Mozilla Firefox, the `MSIL.Yakizake` worm that exploits Mozilla Thunderbird, the `Imspam` trojan that sends spam through MSN or AOL Messenger, among others.

Malware traces [1]	Homogeneity threshold 70%	80%	90%
Bagle	0.25 (±4.95)	0.09 (±3.05)	0.09 (±3.05)
IRCbot	0.05 (±2.18)	0.01 (±0.99)	0.01 (±0.99)
Mybot	0.03 (±1.39)	0.00 (±0.00)	0.00 (±0.00)
SDbot	0.06 (±1.94)	0.00 (±0.00)	0.00 (±0.00)
Spybot	0.02 (±1.40)	0.00 (±0.00)	0.00 (±0.00)
HTTP bot	0.03 (±1.39)	0.00 (±0.00)	0.00 (±0.00)
Large IRC bot	0.19 (±3.05)	0.06 (±2.19)	0.06 (±2.19)

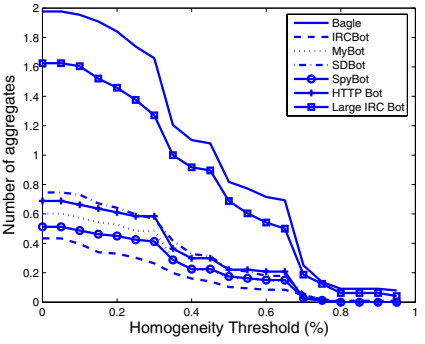

Fig. 5. Average number of aggregates per hour (± standard deviation) due to browser similarity, in addition to the identified malware cluster and to O/S aggregates. For descriptions of the malware, please refer to [1].

multiple operating systems) or might represent a small subset of an O/S aggregate (e.g., all Windows machines). In either case, the mismatch between the software fingerprinted (the O/S) and the software exploited (the browser) can cause platform aggregation to fail to detect an exploit.

Here we consider the impact of reliable *browser* fingerprinting on TĀMD. Specifically, we modified its platform aggregation function so that a platform aggregate is identified when the largest fraction of hosts sharing the same O/S *or* the same web browser is above a given threshold. In doing so, we are able to detect both platform-dependent and browser-dependent malware, while incurring only slight overhead.

To quantify this overhead, we followed the same experiments that were performed in that earlier work [1], which involved seven types of O/S-specific (but not browser-specific) malware. Briefly, the experiment consisted of overlaying recordings of malware traffic onto the CMU dataset, which was done by assigning malware traffic to originate from randomly selected internal hosts. More specifically, we assigned malware traffic to random internal hosts running the O/S that the malware exploits, as determined by the time-to-live (TTL) field in packets.

This combined data, consisting of the CMU dataset overlaid with malware traffic, is then given to TĀMD — configured to identify common host platforms based on their O/S or browsers, but otherwise configured identically as in [1] — in hourly batches, where the goal is to identify the single aggregate consisting of the malware traffic. The same experiment is repeated for each hour over three weeks in November and December 2007, for each of the seven different malware.

Figure 5 shows the number of browser aggregates, in addition to the malware aggregate and other O/S aggregates, that is identified by this new version of TĀMD that incorporates our browser classifier from Section 4, for different thresholds on the homogeneity of the platform aggregate, for each malware experiment. When the threshold is set to 90% (as it was for the original O/S-based platform aggregation in [1]), meaning that at least 90% of the hosts in the aggregate are required to share a common browser (which cannot be $\perp$), the number

of additional aggregates reported due to browser similarity on average per hour is 0.0229. This shows that incorporating browser fingerprinting into TĀMD induces a limited amount of additional cost, while giving TĀMD the ability to detect a wider range of malware, i.e., browser-dependent malware.

Other intrusion detection systems that operate on flow records (e.g., [29,30,31,32]) and approaches for profiling network traffic (e.g., [7,33,34]), can also potentially benefit from passive application fingerprinting. We plan to investigate this in future research.

6 Applications to Traffic Deanonymization

Website deanonymization techniques attempt to infer the actual web *sites* contacted in anonymized traffic traces, without examining the contents of the communication. In order to retain the utility of these datasets for networking research, IP addresses are typically anonymized in a consistent fashion, i.e., so that the same real IP address is mapped consistently to the same pseudonym in the anonymized dataset. This enables the behaviors of the anonymous web servers to be examined, however, which can sometimes lead to their deanonymization. As a trivial example, the larger number of bytes typically transmitted from the main page of `cnn.com` would enable it to be differentiated from `google.com`. Moreover, since a page retrieval can involve connections to multiple physical servers (e.g., image servers or content distribution networks), Coull et al. [18] also found that the sequential order of the servers contacted to retrieve objects on a webpage can enable websites to be differentiated. While previous works placed emphasis on observing traffic behaviors of the websites, to our knowledge, no study has accounted for this behavior as influenced by the particular implementation of their protocol peers, i.e., the browser. In what follows, we show that classifying the browser first can yield a more precise deanonymization of websites.

6.1 Feature Selection

As described in Section 3, we extract nine main flow features from each web page retrieval. While previously these features were calculated over all flows in a retrieval, in the case of website classification, we calculate these features for all flows *per physical server*, for each of the first five servers contacted. The features are then arranged according to the order that the server was contacted, i.e., for retrieval r, the feature vector is $\{F_{r1}, ..., F_{r5}\}$, where F_{rj} refers to the features derived from the flows to physical server j, for website retrieval r. Breaking down the retrieval features by physical server provides a finer-grained representation of the retrieval and an order to the physical servers, both of which have been utilized in previous website deanonymization efforts (e.g., [18]). Furthermore, to eliminate redundancies and reduce dimensionality, we selected a subset of those features that are most consistent across datasets, specifically by computing the correlation of each feature from one day of retrievals to `http://www.cnn.com/` in the PlanetLab-Native dataset to one day of such retrievals in the CMU dataset.

This yielded nine features: the byte and packet counts to/from the first server contacted, and the number of flows to each of the first five servers contacted.

We focus on deanonymizing those websites that are "stable", as judged by their standard deviation for the total number of flows, bytes, and packets, and also those websites that are complex enough, as judged by the total number of flows. It has been previously established [18] that websites with a high variability in their contents (e.g., `espn.com`) or those that are too simple (e.g., `google.com`, `orkut.com`) will typically not be identified accurately. Specifically, we determine a website as "stable" if the average number of flows from the first five servers contacted is greater than one, and the byte and packet counts to/from the first server has a small standard deviation, i.e., within twice the average value. In this way, we narrow down the list of websites that we will attempt to deanonymize in traffic traces to the front pages of 52 of the top 100 websites according to `alexa.com`.

6.2 Website Classifier

We build our website classifiers using Bayesian belief networks, which have been shown to yield good results [18]. Given a test instance, the classifier outputs a probability for each class, which is the likelihood of the instance belonging to that class, according to the model built from training data. The class with the highest probability is taken as the classification of the test instance. This may not always yield optimal classification, for example, in cases where the probabilities for several classes are close to each other, or when all of the probabilities are small.

To establish some notion of "confidence" on the classification, one way is to let the classifier make a decision only from classes with probabilities greater than a cutoff value, and only when there exist probabilities above the cutoff. Although this has limited impact when multiple classes have similar probabilities, it allows the classifier to provide answers based on more confident results, avoiding cases where uncertainty (small probabilities) are likely to cause incorrect classifications. The higher the cutoff parameter, the higher the probability of the test instance belonging to its class must be.

For the PlanetLab-QEMU dataset, we group the data by the browser that generated the traffic, as well as a combined group with traffic from all four browsers. This allows us to build four per-browser website classifiers (IE, Firefox, Opera, Safari), and one generic website classifier. The former are trained on traffic from a single browser type, while the latter is trained on combined browser traffic. In the following, we quantify the benefits of first classifying the browser in website deanonymization by applying these two types of classifier models separately and comparing their results. When testing with the CMU dataset, the browser type for each host is determined by our browser classifier developed in Section 4, using a confidence threshold set at 1.30. The per-browser website classifier is then applied to a website retrieval based on the browser determined for the host that performed the retrieval.

For each testing instance, i.e., each website retrieval, the classifier returns the class with the highest probability above the cutoff. If no probability larger than the cutoff exists, the instance is unclassified. Let the classification for retrieval r be $\mathsf{websiteguess}(r)$, and its actual website be $\mathsf{website}(r)$, where $\mathsf{website}(r) = \perp$ if the ground-truth website for retrieval r cannot be determined in the dataset (which only happens in the case of the CMU dataset). Then, the precision and recall are

$$\begin{aligned}
\mathsf{Precision} &= \Pr[\mathsf{website}(r) = s \mid \mathsf{websiteguess}(r) = s \neq \perp] \\
&= \frac{|\{r : \mathsf{websiteguess}(r) = \mathsf{website}(r)\}|}{|\{r : \mathsf{websiteguess}(r) \neq \perp\}|} \\
\mathsf{Recall} &= \Pr[\mathsf{websiteguess}(r) = s \mid \mathsf{website}(r) = s \neq \perp] \\
&= \frac{|\{r : \mathsf{websiteguess}(r) = \mathsf{website}(r)\}|}{|\{r : \mathsf{website}(r) \neq \perp\}|}
\end{aligned}$$

In the following tests, we only report results for cutoff values where the classifier is able to make at least thirty classifications. This is to avoid cases where not enough classifications can be made for the results to be representative.

6.3 Tests on PlanetLab-QEMU Dataset

Similar to the experiments described in Section 4.1, we first evaluate the results of website deanonymization under an ideal setting using the PlanetLab-QEMU dataset. In each experiment, the testing data consists of retrievals from one host, while the training data is from all other hosts. We apply each per-browser website classifier to retrievals determined to have been performed with that browser by our classifier in Section 4, to generate the per-browser results. We generate results for the generic website classifier by applying that classifier to all retrievals. Our tests are "closed-world", in the sense that only retrievals of the 52 selected websites (see Section 6.1) are tested.

Figure 6 and 7 show the precision and recall from the per-browser and generic website classifiers. Cutoff values range from 0.01 to 0.99, in steps of 0.01. The precision increases with the cutoff, but the recall decreases since some instances are not classified at higher cutoff values. The drops in precision are due to cases where correct classifications that do not have a high probability are filtered out by the cutoff value. The generic classifier was not able to classify more than thirty retrievals after the cutoff reaches 0.78, so we do not plot its results for cutoff values greater than 0.78. To present an alternate view depicting our overall accuracy, let $\mathsf{Precision}(c)$ and $\mathsf{Recall}(c)$ be the precision and recall, respectively, when the cutoff is set to be c. We then define the precision "integral", over the range $[c_{\min}, c_{\max}]$, to be

$$\sum_{c=c_{\min}}^{c_{\max}} \mathsf{Precision}(c)$$

and we define the recall "integral" similarly. $c_{\min}$ and $c_{\max}$ are defined as the endpoints of the range where both the per-browser and generic classifiers were

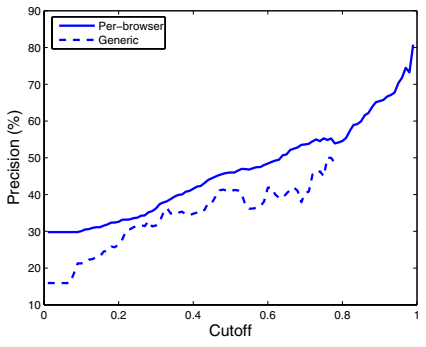

Fig. 6. Website classification precision on the PlanetLab-QEMU dataset

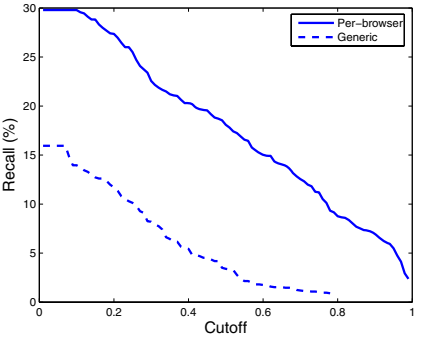

Fig. 7. Website classification recall on the PlanetLab-QEMU dataset

able to make enough classifications. The integral is a measure of how the classifier performs across different cutoff values, in that larger integrals show higher precision (or recall) overall. The integral of precision and recall over $[0.01, 0.78]$, in steps of 0.01, are shown in Table 2, with the generic case serving as baseline. The maximum difference in precision for per-browser and generic classifiers is 15.61%. While website deanonymization remains a challenging problem in practice, we note that the improvement in recall between per-browser and generic classifiers remains significant, across all cutoff values, where the average difference is 14.01% and the maximum difference is 16.11%.

Table 2. Comparing the precision and recall integrals on website classification on the PlanetLab-QEMU dataset

Classifier	Precision	Recall
Generic	26.16	5.34
Per-browser	+6.01	+10.93

6.4 Tests on CMU Dataset

To evaluate the impact of first classifying the browser on website deanonymization in a more realistic setting, we turn to the CMU dataset, with the PlanetLab-Native dataset serving as training data. Since the IP addresses are anonymized in the CMU data, we have no direct knowledge of the websites contacted. So, to build ground truth for the classification, we examined information available in the first 64 bytes of each flow payload. Specifically, the "Host" field in HTTP requests are extracted to identify the domain name of the websites. Of the 52 websites targeted for identification, we found only 23 in the CMU dataset in this way, and so used only these retrievals for testing (while the training data still

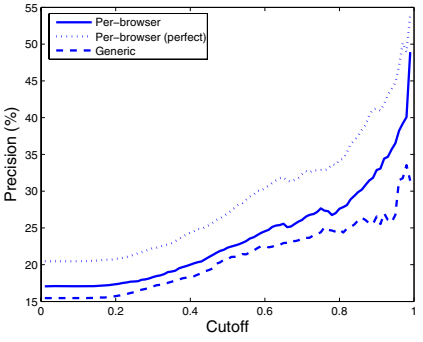

Fig. 8. Website classification precision on the CMU dataset (Train: PlanetLab-Native, Test: CMU)

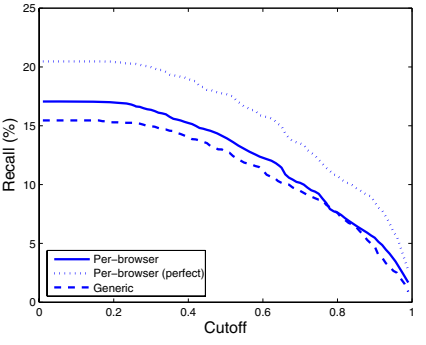

Fig. 9. Website classification recall on the CMU dataset (Train: PlanetLab-Native, Test: CMU)

Table 3. The integral of precision and recall on website classification in the CMU dataset (Train: PlanetLab-Native, Test: CMU)

Classifier	Precision	Recall
Generic	20.53	11.30
Per-browser	+2.73	+1.07
Per-browser (perfect)	+7.93	+4.38

consists of traffic to the 52 websites). Only retrievals from hosts whose ground-truth browser type could be determined were used (see Section 4).

For each retrieval to one of the chosen 52 websites (see Section 6.1) in the CMU dataset from a Firefox or Opera browser, we classify it using both the appropriate per-browser classifier (i.e., for the browser identified using the classifier of Section 4) and the generic website classifier, built using the PlanetLab-Native dataset. The results are shown in Figures 8 and 9, for the two cases when (i) our browser classifier from Section 4 is applied first, and (ii) when we assume perfect browser classification, i.e., the per-browser website classifier applied to a website retrieval is based on the actual browser that performed that retrieval, as opposed to the browser determined by our classifier. When our browser classifier is applied, the difference in precision between the per-browser and generic classifiers can reach close to 17% at high cutoff values. Table 3 shows the integral of precision and recall over cutoff values from 0.01 to 0.99, in steps of 0.01. The results in Figures 8 and 9 are calculated across all 52 websites.

However, for an attacker who is only interested in deanonymizing certain websites, such as those listed in Table 4, a classifier that is able to classify those websites well would be more useful than a general website classifier. For example, the per-browser classifier has a 84.62% precision for `dailymotion.com`, a 27.57% improvement to the generic classifier. These results point out that in live network

Table 4. The precision and recall for the per-browser classifier on some of the websites in the CMU dataset, when our browser classifier from Section 4 is applied first (Train: PlanetLab-Native, Test: CMU)

Website	Precision (%)		Recall (%)	
	Per-browser	Generic	Per-browser	Generic
adobe.com	17.59	0.00	9.55	0.00
aol.com	9.15	8.03	5.67	4.73
dailymotion.com	84.62	57.05	50.00	44.95
myspace.com	19.32	18.57	12.40	11.65
nytimes.com	21.15	16.26	12.26	9.13
wordpress.com	13.98	0.00	7.15	0.00
yahoo.com	45.52	29.60	29.81	19.78

traffic, classifying the browser first can bring a non-trivial advantage to website deanonymization.

7 Conclusion

In this paper we have explored the passive identification of browser implementations from coarse flow records. We have shown that browser implementations can be identified with substantial precision and recall, even using flow records from real traffic recorded at a different time and on a different network from the traffic used to train the classifier.

We have also demonstrated two applications of browser fingerprinting. In the first, we demonstrated how browser identification can be used to improve a network intrusion-detection system called TĀMD, by permitting the intrusion-detection system to identify aggregates of hosts on the network that share the same browser. Suspicious traffic is even more suspect when coming from such an aggregate, since this may indicate that these hosts have succumbed to a browser-specific exploit. Our browser fingerprinting techniques would enable TĀMD to detect more types of malware, i.e., those that are browser-dependent, while incurring slight overhead.

The second application of browser fingerprinting that we explored is deanonymization of network traffic. Our techniques assume that the traffic is anonymized using consistent pseudonyms, which is a common practice today; this enables traffic trace deanonymization by examining the trace for the retrieval characteristics of websites of interest. We demonstrated that improvements in deanonymizing retrieved websites can be achieved by first classifying the browser in use.

References

1. Yen, T.-F., Reiter, M.K.: Traffic aggregation for malware detection. In: Zamboni, D. (ed.) DIMVA 2008. LNCS, vol. 5137, pp. 207–227. Springer, Heidelberg (2008)
2. Comer, D.E., Lin, J.C.: Probing TCP implementations. In: Proceedings of the USENIX Summer 1994 Technical Conference (June 1994)

3. Padhye, J., Floyd, S.: On inferring TCP behavior. In: Proceedings of ACM SIGCOMM, August 2001, pp. 287–298 (2001)
4. Paxson, V.: Automated packet trace analysis of TCP implementations. In: Proceedings of ACM SIGCOMM, pp. 167–179 (1997)
5. Lippmann, R., Fried, D., Piwowarski, K., Streilein, W.: Passive operating system identification from TCP/IP packet headers. In: Proceedings of the ICDM Workshop on Data Mining for Computer Security (2003)
6. Beverly, R.: A robust classifier for passive TCP/IP fingerprinting. In: Barakat, C., Pratt, I. (eds.) PAM 2004. LNCS, vol. 3015, pp. 158–167. Springer, Heidelberg (2004)
7. Karagiannis, T., Papagiannaki, K., Faloutsos, M.: BLINC: multilevel traffic classification in the dark. In: Proceedings of ACM SIGCOMM, August 2005, pp. 229–240 (2005)
8. Bernaille, L., Teixeira, R., Akodkenou, I., Soule, A., Salamatian, K.: Traffic classification on the fly. ACM SIGCOMM Computer Communication Review 36(2), 23–26 (2006)
9. Hernandez-Campos, F., Nobel, A.B., Smith, F.D., Jeffay, K.: Understanding patterns of TCP connection usage with statistical clustering. In: Proceedings of 13th Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, September 2005, pp. 35–44 (2005)
10. Roughan, M., Sen, S., Spatscheck, O., Duffield, N.: Class-of-service mapping for QoS: A statistical signature-based approach to IP traffic classification. In: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement, October 2004, pp. 135–148 (2004)
11. Crotti, M., Dusi, M., Gringoli, F., Salgarelli, L.: Traffic classification through simple statistical fingerprinting. ACM SIGCOMM Computer Communication Review 37(1) (2007)
12. Collins, M.P., Reiter, M.K.: Finding peer-to-peer file-sharing using coarse network behaviors. In: Gollmann, D., Meier, J., Sabelfeld, A. (eds.) ESORICS 2006. LNCS, vol. 4189, pp. 1–17. Springer, Heidelberg (2006)
13. Zander, S., Nguyen, T., Armitage, G.: Automated traffic classification and application identification using machine learning. In: Proceedings of the 2005 IEEE Conference on Local Computer Networks (2005)
14. Moore, A.W., Papagiannaki, K.: Toward the accurate identification of network applications. In: Dovrolis, C. (ed.) PAM 2005. LNCS, vol. 3431, pp. 41–54. Springer, Heidelberg (2005)
15. Erman, J., Mahanti, A., Arlitt, M., Williamson, C.: Identifying and discriminating between web and peer-to-peer traffic in the network core. In: Proceedings of the 16th International World Wide Web Conference (May 2007)
16. Koukis, D., Antonatos, S., Anagnostakis, K.: On the privacy risks of publishing anonymized IP network traces. In: Proceedings of Communications and Multimedia Security, October 2006, pp. 22–32 (2006)
17. Coull, S.E., Wright, C.V., Monrose, F., Collins, M.P., Reiter, M.K.: Playing devil's advocate: Inferring sensitive information from anonymized network traces. In: Proceedings of the 2007 ISOC Network and Distributed System Security Symposium (February 2007)
18. Coull, S.E., Collins, M.P., Wright, C.V., Monrose, F., Reiter, M.K.: On web browsing privacy in anonymized NetFlows. In: Proceedings of the 16th USENIX Security Symposium, August 2007, pp. 339–352 (2007)

19. Shankar, U., Paxson, V.: Active mapping: Resisting NIDS evasion without altering traffic. In: Proceedings of the 2003 IEEE Symposium on Security and Privacy (May 2003)
20. QoSient LLC: Argus - auditing network activity, `http://qosient.com/argus/`
21. Brownlee, N., Mills, C., Ruth, G.: Traffic flow measurement: Architecture. RFC 2722 (1999)
22. Handelman, S., Stibler, S., Brownlee, N., Ruth, G.: New attributes for traffic flow measurement. RFC 2724 (1999)
23. Spiliopoulou, M., Mobasher, B., Berendt, B.: A framework for the evaluation of session reconstruction heuristics in web-usage analysis. INFORMS Journal on Computing 15(2) (2003)
24. Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., Bowman, M.: PlanetLab: an overlay testbed for broad-coverage services. ACM SIGCOMM Computer Communication Review 33(3), 3–12 (2003)
25. Bellard, F.: QEMU, a fast and portable dynamic translator. In: USENIX Annual Technical Conference, FREENIX Track (2005)
26. Witten, I., Frank, E.: Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann, San Francisco (2005)
27. Joachims, T.: Text categorization with support vector machines: Learning with many relevant features. In: Nédellec, C., Rouveirol, C. (eds.) ECML 1998. LNCS, vol. 1398. Springer, Heidelberg (1998)
28. Osuna, E., Freund, R., Girosit, F.: Training support vector machines: an application to face detection. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (June 1997)
29. Karasaridis, A., Rexroad, B., Hoeflin, D.: Wide-scale botnet detection and characterization. In: Proceedings of the 1st Workshop on Hot Topics in Understanding Botnets (April 2007)
30. Gates, C., Becknel, B.: Host anomalies from network data. In: Proceedings of the 6th IEEE Systems, Man and Cybernetics Information Assurance Workshop (June 2005)
31. Gu, G., Perdisci, R., Zhang, J., Lee, W.: Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In: Proceedings of the USENIX Security Symposium (August 2008)
32. Collins, M.P., Reiter, M.K.: Hit-list worm detection and bot identification in large networks using protocol graphs. In: Kruegel, C., Lippmann, R., Clark, A. (eds.) RAID 2007. LNCS, vol. 4637, pp. 276–295. Springer, Heidelberg (2007)
33. Xu, K., Zhang, Z., Bhattacharyya, S.: Profiling internet backbone traffic: Behavior models and applications. In: Proceedings of ACM SIGCOMM (August 2005)
34. Aiello, W., Kalmanek, C., McDaniel, P., Sen, S., Spatscheck, O., Van der Merwe, J.E.: Analysis of communities of interest in data networks. In: Dovrolis, C. (ed.) PAM 2005. LNCS, vol. 3431, pp. 83–96. Springer, Heidelberg (2005)