APPLICATION OF DATA ANALYTICS — CASE STUDIES

By

Jiangmin Yu

Bachelor of Science in Material Science and Engineering
Wuhan University of Technology
Wuhan, China
2004

Master of Science in Computer Science
Wuhan University of Technology
Wuhan, China
2006

Submitted to the Faculty of the
Graduate College of
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTORATE OF PHILOSOPHY
May, 2017

APPLICATION OF DATA ANALYTICS — CASE STUDIES

Dissertation Approved:

Dr. Eric Chan-Tin

Dissertation Advisor

Dr. K. M. George

Committee Member

Dr. Nohpill Park

Committee Member

Dr. Weihua Sheng

Committee Member

Acknowledgments

I am grateful to many people for their support throughout the Ph.D. program. First, I would like to give a special acknowledgment to my chair and mentor, Dr. Eric Chan-Tin. His patience, expertise and availability have been invaluable for my study and research. He has provided me opportunities to work on projects, given me a wealth of knowledge regarding the research, and made my life at Oklahoma State University an enjoyable experience.

I would also like to thank my committee members, Dr. K. M. George, Dr. Nohpill Park, and Dr. Weihua Sheng, for their helpful insight and support throughout my study at Oklahoma State University.

Any level of success I have made is directly attributed to my friends at Oklahoma State University. Jiyoung Shin, Michael Farcasin and Weiqi Cui provided a great deal of guidance. I am really fortunate that I joined a team with an outstanding cohort of individuals. I look forward to continuing our friendships in the future.

I would like to thank my family for their always support. Mom and Dad, thank you for loving me so much. Finally, I would like to thank my wife, Feibo. Thank you for your support to my study and research. Thank you for all your time for taking great care of Beverley. Thank you for being my best friend and by my side every step of the way.

Name: JIANGMIN YU

Date of Degree: MAY, 2017

Title of Study: APPLICATION OF DATA ANALYTICS — CASE STUDIES

Major Field: COMPUTER SCIENCE

Abstract: Data analytics is the technique of finding knowledge by examining raw data. It is an important tool for researchers to verify existing knowledge or infer new knowledge. In this dissertation, we focus on anonymous traffic and privacy-aware systems. Our research is divided into three data analytics case studies. We use data analytics to learn from and improve existing systems. Tor, an anonymous network, is designed to protect Internet users from traffic analysis attacks. Researchers have shown that traffic analysis like timing attack and website fingerprinting attack are still realistic and can be used to deanonymize Tor users. We first analyze the anonymity of Tor itself; we show that a timing attack can be used to bypass the anonymity provided by Tor. We also propose a schema to identify this type of timing attack. Our second case study is about website fingerprinting. We propose a new realistic cover traffic algorithm to mitigate website fingerprinting attacks. Our algorithm reduces the accuracy of website fingerprinting attacks to 14% with zero latency overhead and 20% bandwidth overhead. Our third case study is about webbrowser fingerprinting in anonymous communications. We analyze the network traffic generated by web browsers and show that features of webbrowsers can be inferred with high probability.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

Data analytics is the technique of finding knowledge by examining raw data. It is the process of collecting data, analyzing data and drawing information. With data analytics, researchers can verify existing knowledge and infer new knowledge. Figure 1.1 shows the process of data analytics. Firstly, raw data is collected from various sources, such as existing databases, legacy file systems etc.. Raw data then is processed to remove redundant or unused information. Real raw data usually contains lots of missing, even erroneous information. We clean the raw data and load it in a nicely-formatted table such as a database. Finally, we start analyzing the data. There are two types of analysis: 1) exploratory analysis and model building. Exploratory analysis is to reveal features of data; usually graphs are drawn to help understand the data, and 2) knowledge is expressed in terms of models; statistical algorithms or data mining techniques are used to build these models.

In this dissertation, we use data analytics to learn from and improve existing systems. Three data analytics case studies are conducted in this dissertation. These case studies are about privacy-aware systems. The aim is to improve anonymity and preserve privacy. In the first case study, we analyze the anonymity of Tor itself. We show that a circuit clogging attack can be used to bypass the anonymity provided by Tor. We also propose a schema to identify circuit clogging attacks. Our second case study is about website fingerprinting. We propose a new realistic cover traffic

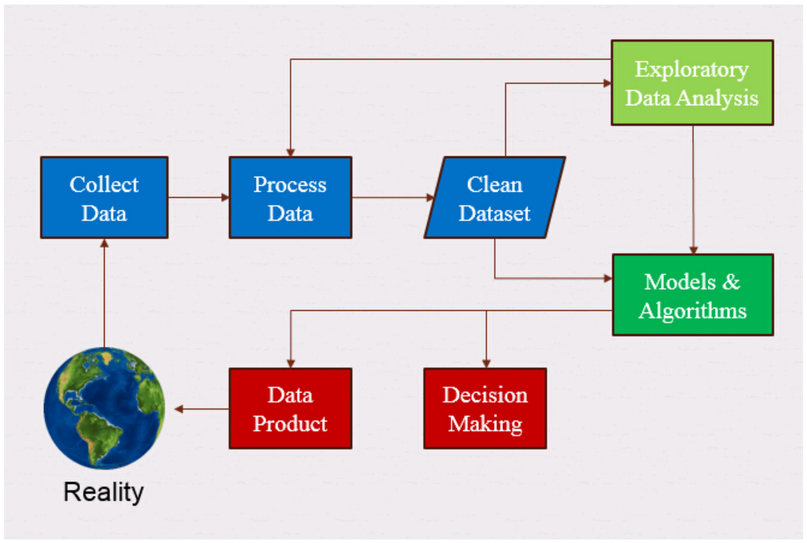


Figure 1.1: Process of data analytics

algorithm to mitigate website fingerprinting attacks. Our algorithm reduces the accuracy of website fingerprinting attacks from 92% to 14% with zero latency overhead and 20% bandwidth overhead. Our third case study is about webbrowser fingerprinting in anonymous communications. We analyze the network traffic generated by web browsers and show that features of webbrowsers can be inferred with high probability.

1.1 Case 1: Timing Attack on Tor

Tor is a popular anonymity-providing network used by over 500,000 users daily. The Tor network is made up of volunteer relays. To anonymously connect to a server, a user first creates a circuit, consisting of three relays, and routes traffic through these proxies before connecting to the server. The client is thus hidden from the server through three Tor proxies. If the three Tor proxies used by the client could be identified, the anonymity of the client would be reduced. One particular way of identifying the three Tor relays in a circuit is to perform a circuit clogging attack. This attack requires the client to connect to a malicious server (malicious content, such as an advertising frame, can be hosted on a popular server). The malicious server alternates between sending bursts of data and sending little traffic. During the burst period, the three relays used in the circuit will take longer to relay traffic due to the increase in processing time for the extra messages. If Tor relays are continuously monitored through network latency probes, an increase in network latency indicates that this Tor relay is likely being used in that circuit. We show, through experiments on the real Tor network, that the Tor relays in a circuit can be identified. A detection scheme is also proposed for clients to determine whether a circuit clogging attack is happening. The costs for both the attack and the detection mechanism are small and feasible in the current Tor network [3].

1.2 Case 2: Website Fingerprinting

Website fingerprinting attacks have been shown to be able to predict the website visited even if the network connection is encrypted and anonymized. These attacks have achieved accuracies as high as 92%. Mitigations to these attacks are using cover/decoy network traffic to add noise, padding to ensure all the network packets are the same size, and introducing network delays to confuse an adversary. Although these mitigations have been shown to be effective, reducing the accuracy to 10%, the overhead is very high. The latency overhead is above 100% and the bandwidth overhead is at least 40%. We introduce a new realistic cover traffic algorithm, based on a user's previous network traffic, to mitigate website fingerprinting attacks. Our algorithms reduce the accuracy of attacks to 14% with zero latency overhead and about 20% bandwidth overhead.

1.3 Case 3: Webbrowser Fingerprinting

Webbrowser fingerprinting is a powerful tool to identify an Internet end-user. Previous research has shown that the information extracted from webbrowsers can uniquely identify an end-user. To collect webbrowser specific information, intentional JavaScript codes are embedded in web pages. In this research, we show that fingerprinting characteristics of a webbrowser can also be collected by solely checking the network traffic data generated when browsing a website. We collect network traffic data generated by browsing the homepage of the most popular websites. Based on this data, we show that the browser fingerprinting characteristics can be inferred with high accuracy. Among these characteristics, type of webbrowser can be identified with over 70% accuracy rate, usage status of popular plug-ins like JavaScript and flash can also be accurately identified [4].

CHAPTER 2

Timing Attack on Tor

2.1 Introduction

Anonymity and censorship-resistant systems are becoming more prominent due to the recent unrest in Egypt and other countries [5–7]. To avoid prosecution and identification by oppressive authorities, citizens of these countries require a system that allow them to maintain their anonymity on the Internet. Various anonymization-providing services exist nowadays [8–14], with Tor [13, 14] being one of the most popular ones with over $500,000$ users [15]. Tor is known as a "low-latency" service as interactive applications such as web-browsing, chat, and remote connections (VPN and ssh for example) can be used on top of Tor.

A client, who wants to connect to a remote server anonymously, uses Tor as a proxy. All the connections and messages go through Tor first, then to the server. Thus, the server believes that the connection is coming from Tor, which hides who the real client is. The Tor system is made up of a network of relays. Each *relay* is a volunteer machine. The client picks three relays from the network to form a circuit: the *entry* node, *middle* node, and *exit* node. The client establishes a connection with the entry node, then using the entry node as a proxy, extends that connection to the middle node, and finally, extends the same connection to the exit node. Then, the client hops through each of the relays to connect to the server. In this case, the server believes the connection is coming from the exit relay. Only the entry relay knows who the client is, but believes the destination is the middle relay.

The Tor project started around 2004 and has been growing in popularity since. It is used by citizens of oppressive regimes, dissidents, whistle-blowers, journalists, and governments' military for anonymous communication. Currently, there are over $500,000$ users [15] in Tor and over $3,000$ relay nodes [15]. The premise of anonymity relies on the three relays used by the client to be non-colluding (especially the entry and exit nodes). Moreover, the identity of the three relays used by a client to connect to a server is hidden. If an adversary could somehow identify the three relays used by a client, this breaks some of the anonymity of the client as it reveals which three Tor relays the client chose. It has been shown in [16] that after the Tor relays in the circuit have been identified, the identity of the client is also leaked. Thus, deanonymizing the three relays used by a client is the first step towards identifying which client is communicating with which server. This has a huge impact on Tor as the anonymity of any Tor user can be compromised.

There exists many attacks [17–25] in the literature on deanonymizing the client and the three Tor relays, such as timing attacks, network flow attacks, and circuit clogging attacks. In this research, we revisit the *circuit clogging* attack described in [20] and show that it is still applicable in the current Tor network. The three relays used in Tor is called a circuit. In a circuit clogging attack, the premise is that a client creates a circuit and connects to a server using that circuit. The server or parts of the content of the server (for example an advertising frame) is malicious. The malicious content alternates between sending a lot of data and sending very little data. If there were a direct connection between the client and the server, there were no issue with the extra data. Since each Tor relay can be part of multiple circuits, when even one circuit is busy delivering a lot of data, that Tor relay will slow down. In this particular attack, the goal is to identify the three relays used by a client.

Every relay in the Tor network is continuously monitored, for example, by creating a 1-hop circuit or through system pings. When a relay slows down during the period of sending burst data (sending a lot of data), its monitoring will show a spike in network latency. The three Tor relays that show an increase in network latency in the monitoring are most likely the three relays used in the circuit by the client.

The first described circuit clogging attack [20] on Tor was performed a few years ago when Tor was still in its infancy. There were only 50 Tor relays at the time, of which 13 were used to demonstrate the attack. Now, there are over $3,000$ relays and Tor is heavily used for web traffic and bulk downloads, such as BitTorrent [26] or peer-to-peer file download traffic. We revisit this attack and show that this circuit clogging attack to identify the three nodes in a Tor circuit is still applicable today. Although the authors in [22] claimed that this attack is no longer possible, their experiments were limited. We performed a full-scale experiment with real Tor nodes. Our proposed circuit clogging attack and network experiments are slightly different from the experimental setup in [20] or [22], as shown in Section 2.3. Section 2.5 shows that a distinct pattern can be observed for the Tor relays in the circuit versus Tor relays not in the circuit. The pattern shows that during the burst period, an increase in network latency can be observed, while during the "sleep" period, the network latency decreases to normal. Through experiments performed on the real Tor network, all the Tor relays used in a circuit under a circuit clogging attack can be identified over 78% of the time. The bandwidth required to perform the circuit clogging attack is less than 30KB/second, and the bandwidth needed to monitor all the $3,000$ Tor relays is 391KB/second.

A detection scheme for clients is also proposed. The detection mechanism probes all the user's created circuits for timing information. If it detects a high and unexpected increase in network latency, the user can disconnect from the server and destroy the affected circuit. Experiments indicate that the proposed scheme can detect over 85% of attacks. The scheme also incurs a very low overhead, requiring less than 3KB/second extra bandwidth to operate. The scheme proposed detects when a possible circuit-clogging attack is occurring; it does not prevent circuit-clogging attacks.

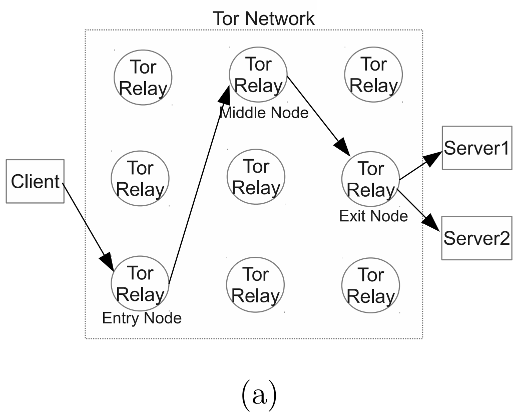
The rest of the research is organized as follows. Section 2.2 gives a more detailed description of how Tor works. An outline of related work is also provided in Section 2.2. In Section 2.3, we describe the specific circuit clogging attack from [20] and how our experiment is set up. A possible solution to identify when a clogging attack is taking place, is given in Section 2.4. The results of our experiment are shown in Section 2.5. Finally, in Section 2.6, we discuss future work and conclude.

2.2 Background

We provide a more detailed description of how Tor works, as well as, a threat model for the attack. The different attacks on Tor to identify the client or the circuit are then outlined.

2.2.1 Tor

Tor [13, 14], released around 2004, is the second generation onion router [27]. The Tor network consists of three main entities: directory servers, relays, and clients. The directory servers are trusted and they keep track of all the relays in the network. Every relay contacts the directory servers to register itself as a relay and upload its key and configuration, such as open ports, and advertised bandwidth. The directory servers regularly form a consensus of all the relays, and sign the consensus document. The servers also monitor the relays' advertised bandwidths. Each client contacts the directory servers to download the consensus document to obtain a list of all the relays



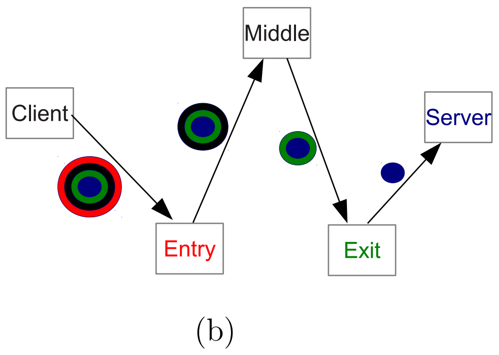


Figure 2.1: (a) Circuit creation in Tor: the client randomly selects three Tor relays in the Tor network to be its entry, middle, and exit nodes. The same circuit can be used for different connections (called streams) to different servers. (b) Onion routing and encryption: A client connects to the server through Tor. The client wants to send a message (blue) to the server. The client first adds a layer to the message, encrypted with the exit node's key (green). Then the client adds a second layer, encrypted with the middle node's key (black), and finally, that whole message is encrypted with the entry node's key (red). At each step, each relay peels off its layer and forwards the message to the next relay.

and their status. A client can also serve as a relay. The directory servers are currently hard-coded in Tor, whereas the relays are volunteer machines.

Tor works as follows. A client contacts the directory servers and downloads the consensus document of all the active relays. The client then needs to construct a circuit to connect to servers on the Internet. To contact a server, the client proxies the connection through a circuit. To build a circuit, the client randomly selects three Tor relays: an *entry* node, a *middle* node, and an *exit* node. The client first establishes an encrypted connection with the entry node. The client then extends that circuit, by going through the entry node, to establish an encrypted connection with the middle node. The client further extends the circuit by establishing another encrypted connection with the exit relay. Encryption and authentication of each relay are possible since every relay's key is part of the consensus document. Once the circuit is built, the client uses it to proxy connections over, to contact Internet servers. Circuits are built using onion routing, such that the connection from the client to each of the three relays is onion-encrypted. None of the relays see what the client is sending to the other relays. For example, the entry node cannot tell that the client is extending the circuit to a particular exit node, as the message to the middle node is encrypted with the middle node only. Figure 2.1 shows graphically how circuits and onion encryption work.

Anonymity is achieved since no entity in the network knows who the client and server are. The entry node only knows that the client is communicating with the middle node. The middle node knows that a machine (entry node) is communicating with another machine (exit node). The middle node cannot tell that it is the middle node of a circuit. Similarly, the exit relay knows that a machine (middle node) is communicating with a server. Finally, the server believes that the connection is coming from the exit relay. To prevent an adversary from potentially controlling a large fraction of entry nodes in all circuits, a client randomly selects three Tor relays as its entry guard nodes. This means that the client will only pick one of these three entry guards as its first entry node in any circuit the client creates. The middle and exit nodes in the circuit are still randomly selected from the consensus document. It is noted that there are over $3,000$ relays in the Tor network, with about 800 of these relays marked as exit relays.

Due to the heavy cost, in terms of encryption and processing, in creating circuits, a client can use one circuit for multiple connections (or streams) to different servers. By default, each circuit is used for at least 10 minutes before it is recycled and a new one created. If a connection is still active in a circuit, that circuit is not destroyed but no new connections are created through that circuit. Circuits can be created in parallel to increase efficiency. Rate limiting is applied end-to-end using TCP. However, Tor also implements its own rate limiting. Every connection is associated with a token bucket. Once a connection runs out of tokens, no new packets can be accepted or delivered until previous packets have been received.

In Tor, every packet is a cell, and each cell's size is 512 bytes. By default, for each relay, the size of a circuit is $1,000$ cells while the size of a stream is 500 cells. Within each circuit, cells are scheduled in a First-In-First-Out (FIFO) fashion. There are currently two scheduling algorithms that can be used within each relay to decide which circuit's cells get processed next. The original algorithm uses round-robin to select the next cell to process. The round-robin is performed among all the circuits. The newer algorithm uses an exponential weighted moving average (EWMA) to determine which circuit to process the next cell from. Each cell is weighted exponentially based on the number of cells in each circuit. Circuits with fewer cells thus have higher priority. This scheduling algorithm is the recommended one from the consensus document.

2.2.2 Threat Model

The threat model in this research is the same as that used in the original Tor paper [14] and in the current literature on Tor. The adversary is only local to each entity or ISP in Tor, and is assumed to not have global capabilities. The adversary can be either passive or active. Attackers can drop packets, relay packets, modify packets (note that cells in Tor are encrypted), delay packets, and passively eavesdrop on all packets. An adversary can listen on the communications at multiple relays, clients, and/or servers, but not all of them. Moreover, the Tor relays, clients, and servers can be malicious but the directory servers are honest and trusted.

2.2.3 Related Work

The current literature contains many publications describing attacks on Tor and other anonymity-providing services [16, 18–23, 28]. Most of these attacks use some sort of timing analysis to determine which relays, clients, servers, or paths are used. The simplest attack in Tor, is for the adversary to be in control of both the entry and exit nodes in a circuit. Using timing information, the adversary can correlate that the two nodes are actually part of the same circuit, and from there, can deduce that a particular client is communicating with a particular server. If the adversary controls f relays out of n relays, then the probability that the entry and exit nodes selected belong to the adversary is $(\frac{f}{n})^2$. This probability is even lower with the introduction of entry guards.

In 2005, Murdoch and Danezis [20] showed a low-cost traffic analysis attack on Tor, which allowed them to identify the three relays used in a circuit. The attack is a variant of the circuit clogging attack, where the adversary attempts to overload the three relays used, which in turn, increases the network latency timing of these three relays. Since all other relays are unaffected, an increase in timing of network latency of three nodes, leads the attacker to conclude that these three nodes are the three relays in the circuit. This attack assumes that the server or part of the content being served is controlled by the adversary. From contacting the directory servers, the adversary also knows all the Tor relays in the network. It can then monitor all the Tor relays for timing information. The adversary runs a probe client and server. It then constructs a one-hop (one-node) circuit to each Tor relay. The probe client uses that circuit to connect to the probe server. The probe keeps each circuit alive and periodically performs a measurement of the network latency of each circuit. Once the malicious server receives a connection from Tor, it starts the circuit clogging attack. The attack consists of a period of low traffic, followed by a period of high traffic, followed by a period of low traffic, and so on. During the period of high traffic, the three relays in the circuit become overloaded as they have to process more cells. This, in turn, leads to an increase in network latency measured by the probe. As all other relays are unaffected, the three relays, experiencing an increase in network latency, are likely the three nodes in the circuit. Once the circuit is identified, the attack can be further extended [16] to narrow down the list of possible clients which created that circuit. The authors of [16] measured the network latency of the whole circuit to calculate the network latency between the entry node and the server. From there, it was trivial to calculate the network latency between the victim and the entry node. The list of possible victims can then be narrowed down using that extra information (network latency) obtained.

However, the investigation of this low-cost circuit clogging attack was originally performed in 2005, when Tor consisted of only 50 relays and few users. The primary use of Tor then, was for remote connections, web traffic, and anonymous chats. Experiments were performed using 13 out of the 50 relays. Nowadays, Tor consists of thousands of relays, with hundreds of thousands of users. Usage of Tor is also more diverse, consisting of short and small web traffic and longer and more bandwidth-intensive file downloads, such as BitTorrent [26]. It is widely believed that this attack is no longer possible due to the changes in the Tor network mentioned above, such as increased traffic and more diverse traffic. As far as we know, there was no specific countermeasure implemented, other than a better scheduling algorithm for circuits. Evans *et al.* [22] presented a variant of the clogging attack using *long paths*. In that paper, they replicated the Murdoch and Danezis attack and showed that it was no longer applicable in the Tor network of 2008, due to the noise in the Tor network and the loss of the effects of the attack in the noise.

We show that a variant of the circuit clogging attack from the 2005 paper [20] is still possible today. This allows an adversary to identify the three relays used in a circuit, and can be a stepping stone to actually identify clients of the Tor network. We note that in our proposed attack, only two out of the three relays are public Tor relays; the third one is under the adversary's control. Our attack is more comprehensive than previous attacks [20,22] as the experiments include all the Tor relays, and the periods of on/off attack are longer. The attack is also performed on the real Tor network. The experiment setup is also slightly different, as shown in the next section.

2.3 Attack Design

We now describe our circuit clogging attack. It is very similar to the clogging attack from Murdoch and Danezis [20]. The four entities are the *client* (victim), *burst server*, *probe*, and *Tor*. The client refers to any user connecting to servers anonymously, using the Tor network. *Tor* represents the three relays used in the circuit created by the client. The three Tor relays are randomly selected from the real Tor network. The burst server is a malicious server the client connects to, either directly or indirectly. For example, the burst server can serve advertisement content when the client visits a major popular website. The goal of the burst server is to introduce enough traffic in the connection with the client to identify the three Tor relays used in the circuit. The probe entity is controlled by the same adversary that controls the burst server; the objective of the probe is to perform network measurements of all the Tor relays. The probe measures the time to route a message through each Tor relay in the network. For consistency, the measurement is performed by creating a one-hop circuit through each Tor relay and measuring the network latency. The probe entity consists of three components, all hosted on the same physical machine: a probe client, probe server, and a public Tor exit node. Although the Tor exit node is a public exit relay, its exit policies are restrictive to only allow exit traffic to the probe server. Moreover, this exit node advertises low bandwidth and is regularly turned off to prevent it from gaining the “fast” and “stable” flags in the consensus document, which in turn, decreases the probability that it will be used by other circuits. Since actual one-hop circuits are not allowed in Tor, this exit node is needed to simulate a one-hop circuit. Since the exit node is hosted on the same physical machine as the probe client and server, and is not used by other circuits, the overhead introduced is small and consistent among all the network probe measurements. The burst server machine and the probe machine are time synchronized so that the increase in network latency measurements can be

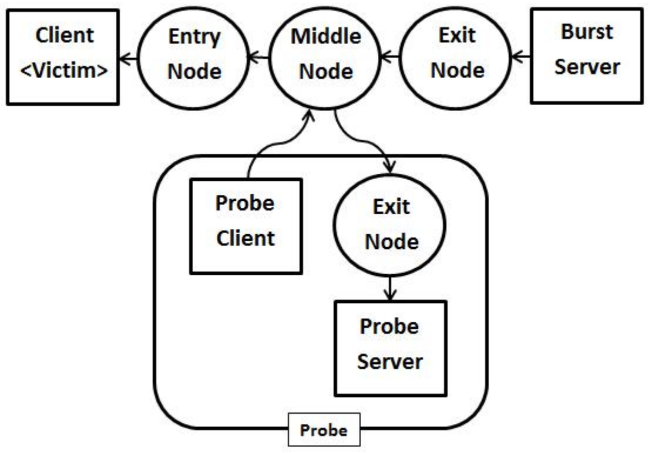


Figure 2.2: The layout of our proposed attack experiment. The burst server could be a physical server or part of the content served by another server, for example an advertisement server. The circuit created by the client consists of real Tor relays. The probe's three components are hosted on the same physical machine. The probe exit node is set up such that it only accepts exit connections to the probe server.

correlated with the time of the burst traffic period. All the entities of our experiments are shown in Figure 2.2.

The burst server introduces burst of traffic to the circuit, in an attempt to disrupt the timing information measured by the probe. The burst server alternates between a period of burst traffic and a period of low traffic. Ideally, the Tor network traffic is constant enough that introducing noise or extra messages temporarily will increase the network latency of the three Tor relays measured by the probe. If this experiment is performed multiple times, the identity of the three Tor relays used in the circuit can be identified, and the identity of the client can be leaked [16]. The burst server starts by sending very few packets for a period T_I, to obtain probe measurements during the initial period. The burst server then sends a burst of packets for a period of T_A, then sends very few packets for a period of time T_S, and so on. During the burst period, it is expected that the probe will measure an increase in network latency for the three Tor relays used in the circuit, but no increase in network latency in all other relays. We next describe in more detail how the probe works.

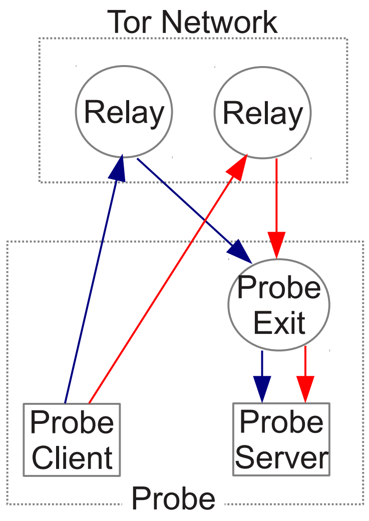


Figure 2.3: The probe works by creating "one-hop" circuit to each Tor relay, with the exit node being the probe exit node. In this example, the Tor network is shown to have only two relays.

The probe client creates a circuit with each of the relays of the Tor network as the entry node, and chooses the probe exit relay as the exit node. There is no middle node. Since the exit node is controlled by the probe, essentially, a "one-hop" circuit is created. Figure 2.3 illustrates how the probe works. Once a circuit is created, at regular intervals, the probe client sends a timestamp t_1 to the probe server through the circuit. The server records the time t_2 that it received the timestamp. Since the probe client and probe server are located on the same physical machine, no time synchronization is required. Also, since the probe exit node is not used for other circuits, the latency through the exit node and the extra time for processing is minimal. That small extra time is also consistent for all the probe measurements and does not affect any of the timing information. The network latency of each circuit (each relay node p) is $t_p = t_2 - t_1$.

The adversary needs to control the probe's Tor exit node. It is possible for the probe's Tor exit node not to appear in the public consensus document; an adversary could be more stealthy that way. This requires setting up private directory servers, which mirror the real Tor directory servers. This does not affect the results of the experiments, just whether the probe's exit relay is public or not. Our modification to the original circuit clogging attack [20] also requires that the Tor exit node used in the circuit, shown in Figure 2.2, be controlled by the adversary. We plan to relax that requirement in future work. Although the Tor exit node advertises a low bandwidth to the directory servers (10 KB/s), it is allowed to relay as much traffic as the network bandwidth allows; this is to prevent our two Tor exit nodes (probe and client circuit) to be used in other circuits.

2.4 Proposed Detection Scheme

We propose a scheme that can be used by all clients using Tor to detect when a circuit clogging attack is happening. Recall that a circuit clogging attack, like the one described in the previous section, can reduce the anonymity of all Tor users. When a possible circuit clogging attack is detected, the client can disconnect from the server and destroy the affected circuit. This detection scheme can also be used by the Tor operators, such as the directory servers, to monitor the Tor network for circuit clogging attacks.

The idea behind the proposed scheme is to use a probe to measure the network latency of each circuit. This is similar to the adversary using a probe to build one-hop circuits to each Tor relay. When the client creates a circuit, it also starts a client probe through the same circuit. The client probe regularly sends a timestamp ts_1 through the circuit to the victim probe server. The probe server replies with the same message ts_1. Once the client probe receives the reply from the probe server, it calculates the current time ts_2, and the RTT or network latency for that circuit is $ts = ts_2 - ts_1$. Both the victim client and the victim probe are hosted on the same physical machine. Both the burst server and the victim probe server are also hosted

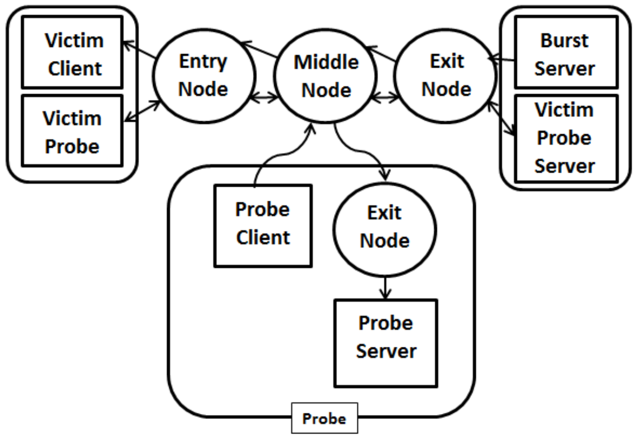


Figure 2.4: The layout of the proposed detection scheme for circuit clogging attacks. The two new entities are the victim probe and the victim probe server. The probe works similar to the attack probe, except the whole circuit is monitored instead of individual relays.

on the same server. Figure 2.4 depicts the setup for the proposed detection scheme.

In our experiments, we set up both a victim probe and a victim probe server. In a real attack, the victim probe server is not needed. The network latency of the whole circuit can be obtained through the lower layers. For example, the TCP sequence numbers and corresponding timestamps can be examined to determine the network latency for each circuit. Also, a public trusted server, similar to the Tor directory servers, can be hosted to reply to victim probes. As the next section shows, the costs required to run such a server are not very big. The challenges in setting up a public probe server is outside the scope of this research. All the clients need to run is an extra process or thread for the victim probe, which sends a probe at regular intervals through all the created circuits.

It can be argued that clients can also set-up one-hop circuit for each of the relays used. However, since Tor, by default, blocks one-hop exits, the client will have to also host a Tor exit node, which might not be feasible nor practical.

2.5 Evaluation

2.5.1 Experimental Setup

All our experiments were performed during February 2013 using Tor version 0.2.3.25. The client, burst server, and probe were hosted on different machines, and were time synchronized. In a real attack, only the burst server and the probe are controlled by an adversary. Before each experiment, the latest consensus document was downloaded from the Tor directory servers. Only the Tor relays with the “fast” and “stable” flags were chosen. The Tor node selection uses these criteria as well, so our experiment is close to what would happen in a real attack. Two Tor relays were randomly chosen to be the entry and middle relay in the circuit used by the client to connect to the burst server. The exit relay is hosted on our server; we plan to relax this constraint later. Two other Tor relays were also chosen, as a control case. In our experiments, we are only probing four Tor relays, not the whole Tor network – an actual attack would have to probe the whole Tor network. The entry and middle relays are referred as Tor_C, while the two other Tor relays are referred as Tor_R. The probe creates a “one”-hop circuit to each of the four relays and sends a network measurement every 5 seconds. The initial time period T_I was set to 15 minutes, while the burst attack period T_A was set to 5 minutes, and the sleep period T_S was set to 5 minutes. Each experiment was run for 60 minutes. Instead of creating one single connection, the client is multi-threaded and sets up 5 threads to connect to the server. All the 5 threads use the same circuit. In a real setting, a server can redirect a client to 5 different content servers; many web browsers download multiple parts of a server in parallel. For each experiment, the four Tor relays are randomly selected from the list of Tor nodes.

Figure 2.5 shows a probe’s network measurement of a Tor relay used in the circuit. Each probe was sent at intervals of 5 seconds. It can be seen in the figure that during

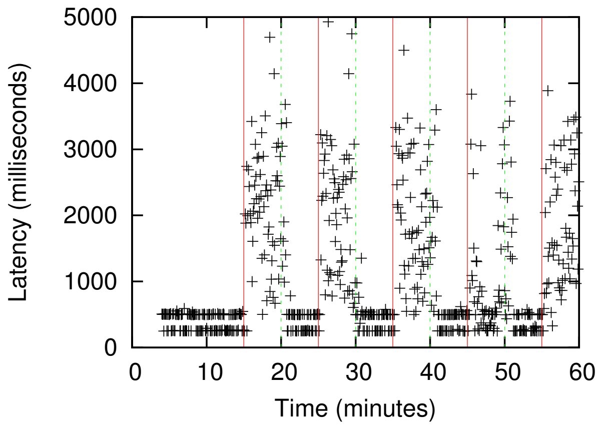


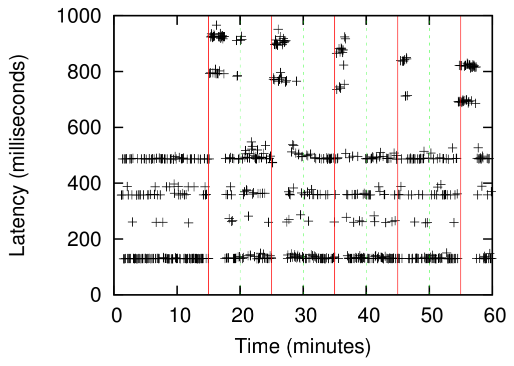
Figure 2.5: Probe network measurement for a Tor relay used in the client circuit. The burst attack period starts at the red (solid) vertical line and stops at the green (dashed) vertical line. The pattern of burst/sleep periods can be clearly seen.

the burst period (the beginning of which is indicated by the red darker solid vertical line), the latency measured increases, and during the sleep period (the beginning of which is indicated by the green lighter dashed vertical line), the latency decreases to that of the initial period. The initial period is used to set the baseline for the average network latency for each relay.

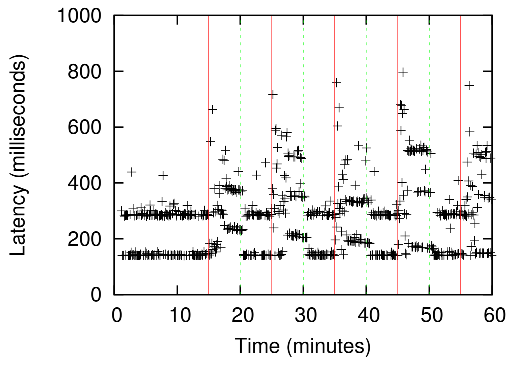
The victim probe also is set to measure the latency of the circuit every 5 seconds.

2.5.2 Results of Attack

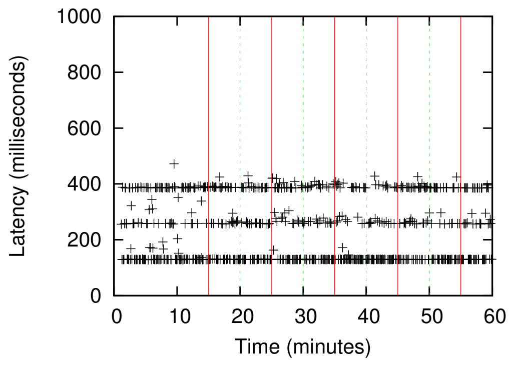
Figure 2.6 shows an example of an experiment run. The figure shows the probe's network latency measurement for the four Tor relays used in that experiment: two of them were part of the circuit, and the other two were not part of the circuit. Figure 2.6(a) and (b) refer to the entry and middle node in the circuit, respectively, while Figure 2.6(c) and (d) refer to the other two Tor relays. These two Tor relays ((c) and (d)) act as a control case, representing all the relays in the Tor network. The probings of these two relays should show no difference during the burst attack period and during the sleep period. However, the entry and middle node should show



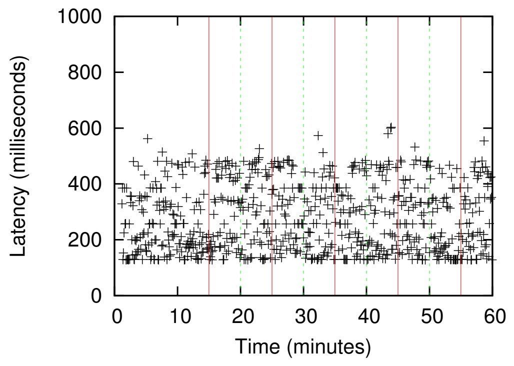
(a) Entry relay



(b) Middle relay



(c) Other relay



(d) Other relay

Figure 2.6: The probe's network latency measurement over time for (a) the entry relay, (b) the middle relay, (c) and (d) the other two Tor relays not part of the circuit.

a distinct pattern, which is seen in Figure 2.6.

The probe machine is time-synchronized with the burst server and knows when the different periods (initial/burst/sleep) happen. The initial time period T_I is used as a baseline. The baseline time t_b is calculated as the average of all the network latency times during the initial period T_I. The average time for each period is also calculated. For the entry and middle relays' measurements, it is expected that the average time during the burst periods will be higher than t_b, and the average time during the sleep periods will be about the same as t_b. For the two other Tor relays Tor_R, it is expected that the average time for all the periods will be similar to the average initial time t_b. Due to noise and variations in probe measurements, such as a Tor relay being used in a bandwidth-intensive circuit, the average time for the sleep periods for Tor_C and for all periods for Tor_R might be higher than the average initial time t_b. If this happens, this is called a *false positive*. These regular variations in the network latencies measured lead to using a threshold value α to determine whether the average time t_a for a period indicates a burst period or a sleep period. If $t_a \geq \alpha \times t_b$, then this indicates that the network latency measured is high enough that it indicates a burst period. Relays experiencing such high network latencies during actual burst periods are marked as possible entry or middle relays used in the circuit. The value of the threshold α is varied from $1.0 - 5.0$.

The varying threshold produces different numbers of false positives (a random Tor relay accidentally marked as being part of the circuit used to connect to the burst server) and different numbers of correct predictions or numbers of true positives (correct Tor relay identified as being either entry or middle relay in the circuit used to connect to the burst server). The Receiver Operator Characteristic (ROC) [29] curve shows the trade-off between the false positive rate and the true positive rate. Figure 2.7 illustrates the ROC curve for our experiments, when varying the threshold α. The line $y = x$ (green line in the figure) shows the true positive rate and the false positive rate when randomly determining whether a Tor relay is part of the circuit or not (50% probability of being correct). The vertical line from $(0, 0)$ to $(1, 1)$ shows a perfect classifier where all the Tor relays part of the circuit are correctly identified and there are no false positives. Figure 2.7 shows that our experiments fall between the perfect classifier and random classifier. This means that our attack is better than random but not perfect. The area under the curve (AUC) shows the trade-off between the true and false positive rate. The perfect classifier has an area under the curve of 1.0 and the completely random classifier has an AUC of 0.5. The area under the curve for our attack is 0.78. The area under the curve when using one thread for the attack, instead of five threads, is 0.68 (ROC curve not shown). This means that

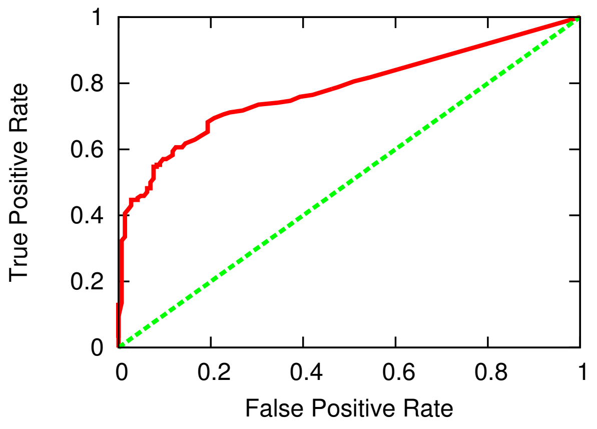


Figure 2.7: The Receiver Operator Characteristic (ROC) curve for the probe measurements. The area under the curve (AUC) is 0.78 and the equal error rate is 28%. The $y = x$ line indicates the random classifier.

using more threads is more effective at performing the circuit clogging attacks than using just one thread. The equal error rate indicates when the false positive rate is equal to the true positive rate. The lower the equal error rate, the more accurate is the system. Our attack achieved an equal error rate of 28%, which means that our attack is accurate.

The results indicate that the circuit clogging attack is still possible in the current Tor network. The probings of Tor relays for measuring the network latency is effective at determining when a burst attack period occurs and at identifying whether a relay is used in the circuit by the client to connect to the malicious burst server. We note that our attack is conservative: more server threads, a higher burst of messages, and a longer running time with more burst/sleep periods will make the attack more accurate. However, since our experiments were performed on the real Tor network, we did not want to affect the load on the Tor relays unnecessarily.

2.5.3 Detection Scheme

The proposed detection scheme was run at the same time as the attack. Each probe measurement is per circuit created by the client. In a real setting, the client does

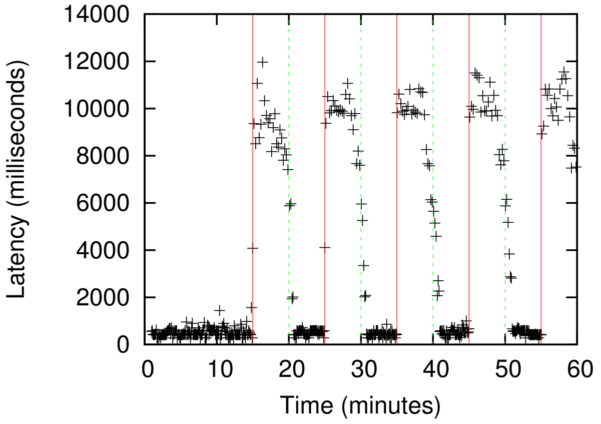


Figure 2.8: One experiment showing the client's probe of a circuit used for connecting to the burst server. The periods of burst/sleep can be clearly seen in the figure.

not know when the burst periods are, which is different for the attacker's probe. For experimental simplicity, the victim machine and the burst server are also time-synchronized. This allows us to determine the baseline and calculate the number of correct predictions of circuit clogging attack happening. Figure 2.8 shows a time versus network latency measured graph for the victim probe in a circuit that is used to communicate with the busrt server. It can be clearly seen in the figure that during the burst period, the network latency measured increases and gradually goes back to normal when the burst period ends. The increase in network latency is also more pronounced than for the attack probes, as seen in Figure 2.6; the y-axis scales are different. For the one-hop attacker probe circuits, only one relay is affected, whereas for the client's three-hop circuits, three relays are affected.

Similar to the attack probe measurement, an initial time period T_I is needed to set the baseline average time t_c. The baseline time can be obtained by the client probe connecting to the client probe server some time before connecting to the burst server, or through the average of all other circuit times, or through a network-wide latency average in the consensus document. When the measured circuit network latency exceeds the baseline initial time t_c, this circuit is marked as suspicious. If the burst period was active at that time, then this counts as a correct prediction. Our proposed detection scheme is able to accurately detect 87.5% of all burst periods; that is, it can predict when a circuit clogging attack is happening 87.5% of the time.

The proposed detection scheme is thus accurate. Although the scheme is not perfect, once a possible attack is detected, all the connections streamed over that circuit can be reported to the user, with a list of all the servers. After the detection scheme is run for some time, the repeated servers on the reported list are suspicious and can be further investigated. This whole detection and reporting mechanism can be performed transparently to the user. The challenges in authenticating this list and preventing "bad-mouthing" of honest servers are outside the scope of this research. The proposed detection scheme allows the client to destroy a suspicious circuit, to preserve the user's anonymity. The proposed detection mechanism has some limitations, such as false positives and the overhead/cost in creating a new circuit. The user has control when to destroy a suspicious circuit. The user can disconnect as soon as the circuit is flagged as suspicious, or it can wait until k "bursts" are detected.

2.5.4 Costs

The total costs and overhead for performing the burst attack (hosting the malicious content) and the probe machine are small, as will be shown in this section.

The bandwidth used in our burst server during the burst attack period is less than 30 KB/second, on average. If the malicious content is hosted on Amazon EC2 [30], the cost to run one server for one circuit is \$11.38 per month. The storage and processing needed to maintain the burst server are minimal, and the costs for storage and processing are assumed to be minimal.

The bandwidth required to probe one Tor relay is 130.4 bytes/second. Probings are performed every 5 seconds. If a more fine-grained measurement is required, the interval between measurement can be decreased, which will increase the costs to run the probe machine. With 3,000 relays, this is $3000 * 130.40 = 391,200$ bytes/second, which is a bandwidth of about 391 KB/second. The cost to run the probe on Amazon EC2 is \$121.91 per month. The processing needed for each network latency measurement is minimal. The storage required is to record all the network latency measurements, which consist of timestamps. Each timestamp is only 13 bytes. For a 60 minute experiment, this is 9.36KB for probing one relay, or 28MB storage for all the relays.

The bandwidth required for a client to run the proposed detection mechanism is 130.4 bytes/second, for each circuit the client creates. With possibly 20 circuits created, this is an extra bandwidth of 2,608 bytes/second or 2.6 KB/second which is minimal. The interval for each probe is 5 seconds; this interval can be decreased for an increase in extra bandwidth needed but faster detection. If the victim probe server is a public server, the bandwidth required to reply to 5-minutes probe from 500,000 clients using 20 circuits is 21.7MB/second or 1.9TB/day, which is \$5,870 on Amazon EC2. In general, the overall costs of running a victim probe, including processing, storage, and network, are minimal. The benefit of running a probe, however, is real, and allows a client to detect the occurrence of a circuit clogging attack. This helps to preserve the client's anonymity.

2.6 Discussion and Conclusion

We showed that a circuit clogging attack is still possible in the current Tor network, contrary to previous claims. The goal of a circuit clogging attack is to identify all the Tor relays used in a circuit. A client uses the circuit to connect to a malicious server (or malicious content hosted on a honest server). The malicious server periodically switches between burst mode and sleep mode. During burst modes, the server sends a burst of messages, while during the sleep modes, the server sends a few messages only. Since all the relays in the Tor network are monitored through probe network latency measurements, a spike in network latency is observed for the three Tor relays used in the circuit during the burst periods. The costs to perform a circuit clogging attack are also very low, making it a practical attack. We showed that the Tor relays used in a circuit can be accurately identified. Moreover, the false positive rate is low as only some other Tor relays not used in the circuit are accidentally identified as being part of the circuit.

A circuit clogging attack detection mechanism is also proposed. The scheme uses a probe to monitor all the circuits created by the client, instead of each Tor relay. Once an increase in network latency from a previously recorded baseline time is measured, the server is flagged as suspicious. The client can then disconnect from the server and destroy the affected circuit. Through experiments on the real Tor network, the proposed detection scheme has an accuracy of over 85%.

This research showed that the anonymity of a person using Tor is reduced since the Tor relays used can be identified. This can be a stepping stone towards narrowing down the possible users behind these relays. The detection scheme proposed allows a user to detect possible occurrences of circuit clogging attacks. With over $500,000$ users daily, the attack has huge potential consequences. The proposed detection scheme can help hundreds of thousands of people stay anonymous on the Internet.

Although a detection mechanism is better than nothing, a prevention algorithm would be best. We leave as future work designing a scheduling algorithm that can prevent circuit clogging attacks, such as [28]. The current experimental set-up requires that the Tor exit node in the circuit be under the control of the attacker. We performed 15 experiments where all the nodes in the circuit are public Tor relays and with one server thread. Figure 2.9 shows the ROC curve. The area under the curve

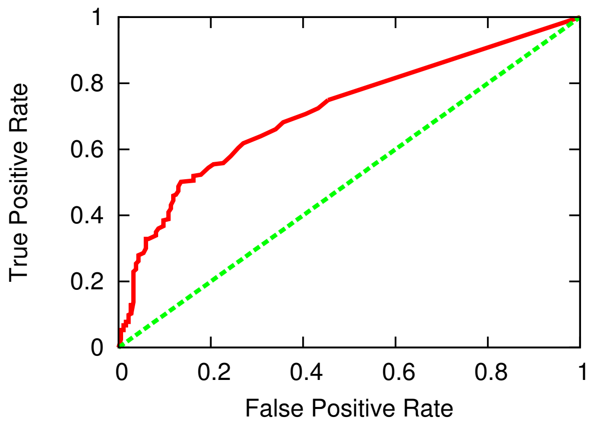


Figure 2.9: The Receiver Operator Characteristic (ROC) curve for the probe measurements, where all the relays in the client circuit are public Tor relays. The area under the curve (AUC) is 0.72. The $y = x$ line indicates the random classifier.

(AUC) was 0.72, which is close to an AUC of 0.78 when the exit relay belonged to the adversary. Based on this promising preliminary result, we plan on extending our attack such that the exit node used in the circuit is a regular Tor exit node. We will explore ways to improve the accuracy of the attack, such as using multiple server threads, varying the time of the burst/sleep periods, and modifying the amount of data sent during the burst periods.

The current attack identifies the Tor relays in the circuit; future work will identify which of the relays are the entry, middle, and exit relays. This would require more fine-grained probe measurements. One of the anonymity improvements in Tor is to use entry guards, a fixed set of three relays used as entry relay in any circuit. We will also analyze whether using entry guards leaks any information, as the user could be more easily identified. If the same entry relay is found in circuits, this can leak information about the user. The possible impact of the attack on bridges and hidden servers is left as future work.

CHAPTER 3

Website Fingerprinting

3.1 Introduction

Website fingerprinting violates the privacy expected from a user when she is using an anonymizing service such as a proxy or Tor [13]. The goal of website fingerprinting attacks [31] is to determine the website visited by a victim. The adversary, in this case, is usually local, for example on the same network or the Internet Service Provider, and can observe all the network traffic sent by the victim. These attacks are effective and are very accurate in successfully identifying the websites. The accuracy is over 90% even when the network traffic is encrypted or anonymized through a proxy. Since the adversary knows who the user is and can accurately guess what websites she is visiting, the user has no privacy.

Various defenses against website fingerprinting attacks [32–36] have been proposed. The defenses include padding so that every packet has the same size, cover traffic to generate enough noise to fool the adversary, or introducing network delays between network packets. Although they have been shown to be effective, the overhead introduced by these defenses is very high. The latency overhead is above 100% and the bandwidth overhead is from 50% to over 100%.

Our **contribution** is a new cover traffic algorithm that generates just enough noise to mitigate website fingerprinting attacks. Our algorithm also has zero latency

Mitigation	Accuracy (%)	Latency Overhead (%)	Bandwidth Overhead(%)
No Defense	91%	0%	0%
CS-BuFLO [36]	22%	173%	130%
Tamaraw [33]	10%	200%	38%
WTF-PAD [1]	15%	0%	54%
Our Algorithm	14%	0%	20%

Table 3.1: Comparison of our algorithm's accuracy and overhead with previous mitigation schemes. We showed the lowest accuracy numbers for the other schemes, regardless of algorithms used. The table is based from [1].

overhead and lower bandwidth overhead than current schemes. Our algorithm generates "realistic" cover traffic; it collects the network traffic from a user, then uses that historical network traffic data as training set to feed the cover traffic generation algorithm. The generated noise thus will look exactly like a website that a user has previously visited. This prevents website fingerprinting attacks and introduces little bandwidth overhead.

Table 3.1 shows a comparison of our proposed algorithm with existing mitigation techniques. Our algorithm has comparable accuracy with the other schemes, zero latency overhead, and lower bandwidth overhead. The table shows the lowest accuracy (best-case for the mitigation) regardless of the classification algorithm used. Our algorithm has zero latency overhead since we are only introducing cover traffic. No padding or delays are introduced.

Section 3.2 describes website fingerprinting attacks and our threat model. The design of our proposed cover traffic algorithm is provided in Section 3.3. The setup for our experiments is outlined in Section 3.4 and Section 3.5 shows the effectiveness of our proposed algorithm. A survey of related work is given in Section 3.6 while future work is discussed in Section 3.7.

3.2 Background

3.2.1 Website Fingerprinting

Website fingerprinting aims to determine the website visited by examining the network trace sent by a victim's webbrowser. That trace is usually encrypted and sent over a proxy or an anonymous network like Tor [13, 14] so that the network contents cannot be analyzed. The only information that can be observed are the packet sizes, the direction of the packets, the time interval between packets, and the number of packets sent and received.

A packet trace PT consists of n network packets. PT also consists of the tuple $< \Sigma_{i=0}^{n} N_i, N_s, N_c >$, where N_i is each individual network packet, N_s is the total number of packets from server to client, N_c is the total number of packets sent from the client to the server. Each network packet N_i forms the tuple $< S_i, T_i, D, M_s, M_c >$, where S_i is the size of the packet, T_i is the time interval until the next packet, D is the direction of the network packet (from client to server or from server to client), M_s is the number of packets from the server to the client, and M_c is the number of packets from the client to the server. M_s and M_c denotes each "train" of packets, that is, there are usually a few packets from client to the server, followed by several packets from server to client. On the other hand, N_c and N_s denotes the total number of packets for the whole packet trace.

3.2.2 Classification

Previous work [2, 31–33, 37–47] have achieved a classification accuracy of around 90% in both the open and closed world settings. A closed world is where the set of training packet traces are the same as the testing set. An open world setting is where there is a small set of monitored/sensitive packet traces among a larger set; the goal is to detect if a packet trace belongs to one of these monitored websites.

To perform classification, various features have been used such as number of outgoing and incoming packets, total size of incoming and outgoing packets, and cumulative size of packets. If the Tor network is used, some features that have also been considered include the Tor cells before and after. Various algorithms have also been used such as k-nearest neighbors (K-NN), support vector machine (SVM), random decision forests, edit distances, Jacard index, and Naive Bayes.

In this work, we used the same features as those mentioned in [2]: cumulative size of packets sampled at regular intervals over the whole packet trace, number of incoming packets, total size of incoming packets, number of outgoing packets, and total size of outgoing packets. We also used the random decision forests as this was used by the most recent paper [47].

3.2.3 Mitigations

Various defenses against website fingerprinting attacks have been proposed such as padding of packets to a fixed size and cover traffic (noise) to mask the real packet trace. They have all been shown to be somewhat effective reducing the accuracy of website fingerprinting to about 10% to 30%. However, all of these defenses incur high latency overhead or high bandwidth overhead or both.

Our proposed algorithm achieves a similar reduction in accuracy while keeping the overhead manageable. Our cover traffic generated is realistic instead of random as it depends on what the user has done previously. A threshold value for the amount of cover traffic generated can also be chosen by the user. Our algorithm experiences zero latency overhead.

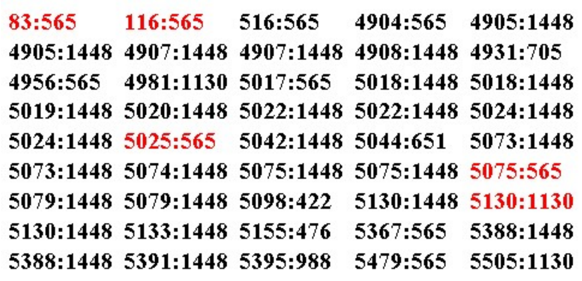
3.2.4 Threat Model

The threat model is a local adversary that can see all the network traffic from a user. The adversary cannot decrypt the contents of the network packets but can observe the metadata such as packet sizes, direction of the packets, and the timings of packets. The adversary can also look at the IP headers to determine the source and destination IP addresses and port numbers. The goal for the adversary is to guess the website or webpage from only the encrypted network packet trace.

3.3 Proposed Noise Algorithm

Our proposed algorithm to generate cover traffic is novel since it generates realistic noise rather than random noise or random padding. The noise generated is learned from the network traffic generated by the user's webbrowser. The information recorded is the network traffic trace without the payload contents: each incoming and outgoing packet's size, and the time interval between packets and "train" of packets. A train is a set of incoming packets with size of MTU (Maximum Transmission Unit) with the last packet size less than MTU. Usually an outgoing web request is following by one or more trains of incoming packets. Replaying this recorded network traffic will simulate that user's browsing habit. Our hypothesis is that if the cover traffic generated is similar to what the user usually does, this will provide a better noise in preventing website fingerprinting and also reduce the bandwidth overhead since this would be traffic that the user usually generates anyway. It has already been shown that if a client visits several webpages at the same time [45], then it is hard for an adversary to identify the webpage visited.

Instead of replaying the web requests to the actual servers which would use up resources on these servers, we set up our own simple webserver. Our algorithm can



83:565	116:565	516:565	4904:565	4905:1448
4905:1448	4907:1448	4907:1448	4908:1448	4931:705
4956:565	4981:1130	5017:565	5018:1448	5018:1448
5019:1448	5020:1448	5022:1448	5022:1448	5024:1448
5024:1448	5025:565	5042:1448	5044:651	5073:1448
5073:1448	5074:1448	5075:1448	5075:1448	5075:565
5079:1448	5079:1448	5098:422	5130:1448	5130:1130
5130:1448	5133:1448	5155:476	5367:565	5388:1448
5388:1448	5391:1448	5395:988	5479:565	5505:1130

Figure 3.1: Sample of recorded network traffic. The format is <timestamp>:<packet size>. Red indicates outgoing packets.

be implemented as a plugin for Firefox (Tor Browser Bundle). It will send a web request padded to a certain packet size to our webserver through the Tor network. The request will contain the total size of data that the server has to send back and the time that the data should be sent. Both the client plugin and webserver do not have to send any content; only pad the packets to the specified packet size. The generated network traffic will be transferred over the Tor network; a local adversary will not be able to determine which packet is noise.

The algorithm will first record traffic of a web page, then parses the recorded traffic trace. Packets are put into two sets based on whether they are incoming packets or outgoing packets. For each set, packets are organized by trains of packets. For each train, the size and timestamp of each packet is recorded. Trains of packets are listed in order by the timestamp of the first packet in the train.

Figure 3.1 shows an example sample of a recorded network traffic. The only information recorded is the relative time between packets and the packet sizes. Both incoming and outgoing packets are recorded. The format of the sample shown in the figure is as follows: <timestamp>:<packet size>. The actual time is not relevant; only the time difference between two packets is used. This is the time difference between the current packet's timestamp and the next packet's timestamp. The packet size is the TCP-level packet size. A red packet size indicates an outgoing packet.

Taking the packet traces from Figure 3.1 as the example, the following two tables are built. Table 3.2 shows the parsed outgoing packets set, denoted as TS_out. Table 3.3 shows the parsed incoming packets set, denoted as TS_in. Most webbrowsing network traces have a higher number of incoming packets than outgoing packets. Moreover, the size of the incoming packets is higher than outgoing packets, which are usually web requests for a URL resource (such as jpg, html, etc...). This is typical of web traffic and is reflected in the tables. Generating noisy cover traffic works as follows.

I. Randomly select one traffic train from TS_{out} and TS_{in} each, denoted as T_{out} and T_{in} respectively. The total size of T_{out} is S_{out} and the total size of T_{in} is S_{in}.

II. Construct a cover traffic request with size S_{out}.

III. Send this request to the noise server.

IV. The server will reply back with data of size S_{in} in WT_{in} milliseconds. WT_{in} is the time difference between the first packet of T_{in} and the first packet of the next traffic train after T_{in} in the incoming traffic train set.

V. Wait for some time WT_{out}, where WT_{out} is the time difference between the first packet of the chosen outgoing traffic train T_{out} and the first packet of the next outgoing traffic train.

VI. Repeat steps 1 to 5 until the total incoming traffic from the noise server is equal to the size of all the incoming packets of the recorded traffic trace.

As an example, let's suppose outgoing traffic train 3 is selected from TS_{out} and incoming traffic train 8 is selected from TS_{in}. Our algorithm will create a new cover traffic request to send to the noise server. The request will ask the server to send back

Traffic Train ID	Time and Packets
1	83:565
2	116:565
3	5025:565
4	5075:565
5	5130:1130
6	5560:565

Table 3.2: The parsed outgoing traffic train set.

Traffic Train ID	Time and Packets
1	516:565
2	4904:565 4905:1448 4905:1448 4907:1448 4907:1448 4908:1448 4931:705
3	4956:565
4	4981:1130
5	5017:565 5018:1448 5018:1448 5019:1448 5020:1448 5022:1448 5022:1448 5024:1448 5024:1448
6	5042:1448 5044:651
7	5073:1448 5073:1448 5074:1448 5075:1448 5075:1448 5079:1448 5079:1448 5098:422
8	5130:1448 5130:1448 5133:1448 5155:476
9	5367:565
10	5388:1448 5388:1448 5391:1448 5395:988
11	5479:565
12	5505:1130

Table 3.3: The parsed incoming traffic train set.

data of size $1448 + 1448 + 1448 + 476 = 4820$ bytes with a time of $5367 - 5130 = 237$ milliseconds. The request contains only total size of data to be sent and the time. For example, the server only needs to send padded data with size 4820. The lower level network interface will determine how to send each packet – if the MTU is 1448, packet size will be $1448 + 1448 + 1448 + 476$. The outgoing packet will be of size 565 bytes. Since the actual contents of the packet is small, the rest of the packet is padded. To simplify the example, we ignore packet headers. When the noise server receive this cover traffic request, it will send back data of size 4820 bytes and sleep for 237 milliseconds before responding to next request. At the same time, the client side waits for 55 milliseconds, which is the time difference between the first packets of the outgoing traffic train 4 and outgoing traffic train 5 from Table 3.2. This process is repeated until the sum of all the incoming packet sizes from the server is equal to the recorded traffic trace. The reason for waiting on both client and server sides is to ensure that the generated noise traffic is well distributed to look more realistic. This generated cover traffic can achieve better performance in terms of obfuscating the overall traffic collected by a website fingerprinting attacker.

The user can choose as a parameter, the size of the cover traffic s. Since the cover traffic mimicks the websites that the user has previously visited, the bandwidth used will be doubled. To minimize the bandwidth overhead, each train size could be reduced by a factor of s. The incoming train will thus have a total packet size of $0.5 \times S_{in}$ in time WT_{in}. This reduces the bandwidth overhead and generates fewer packets.

We emphasize that the cover traffic is only between the client's webbrowser and the cover traffic webserver through Tor. The only data sent are padded data so that the packets are of a certain pre-determined size. The cover traffic generated will look realistic as it is traffic that was generated by the user. This recorded network traffic is only stored locally on the browser.

We expect our algorithm to effectively mitigate website fingerprinting attack since it has already been shown that cover traffic is effective. We expect that our algorithm will have lower bandwidth overhead since the amount of noise generated can be modified. Moreover, there is no extra latency added as no padding or network delay is introduced. Our algorithm only generates cover traffic to another website.

3.4 Experimental Setup

We utilized the dataset provided by [2], which consists of $1,125$ webpages and 40 instances of each webpage. Each instance contains the timestamp of each packet along with the packet sizes (negative packet sizes indicate outgoing packet). We implemented the noise generation algorithm described in Section 3.3.

Due to the new data generated by our noise generation algorithm, we could not re-use the authors [2] SVM algorithm. Instead we use the standard Weka [48] tool and experimented with different classification algorithms: Support Vector Machine (SVM), decision tree (REPTree in Weka), neural network (Multi-layer perceptron), linear regression, and random forest.

The six classification features used in our experiments are similar to thoseused in [2]. The first four features are: total size of outgoing packets, total size of incoming packets, total number of outgoing packets, and total number of incoming packets. The remaining two features are the sampled cumulative representation of packet size. There are two ways to calculate the cumulative packet size: c is the cumulative size of packets size where an outgoing packet has a negative packet size and a is the cumulative size of packets size where both outgoing and incoming packet sizes are denoted as positive numbers. The number of samples used n can be varied and will be taken at equidistant points in the packet trace. For example, if there are 75 packets and $n = 100$, a sample is taken every 0.75 packet. To determine the packet size of the 0.75th packet, the linear interpolation is calculated. If the 0th packet size was 10 and the 1st packet size was 20, the 0.75th packet size is $(0.75 * (20 - 10)) + 10 = 17.5$.

We compared our proposed cover traffic algorithm with the basic cover traffic scheme. The latter works as follows. When a user visits a website, the basic scheme will randomly pick another website to also visit. As shown by [45], having two simultaneous website visits significantly lowers website fingerprinting accuracy.

The original dataset contained $1,125$ webpages, many from the same website. We filtered out webpages of the same website and used 91 websites as our base training dataset [1]. The dataset [2] contained timestamps and packet sizes. Merging the original website packet trace with the noise packet trace is relative straightforward. Since there are 40 instances of each website, we randomly picked one instance as the noise data to merge with the original packet trace.

We considered two different basic cover traffic algorithms. The first one always picks the same webpage (but possibly different instances). The second one randomly picks from a set of 10 webpages different from the 91 previously selected. The second case provides a more diverse set of webpages and noise to be added.

Our noise generation algorithm "learning" dataset consists of a further 10 webpages where the packet traces are recorded. For each of the original 91 webpages, we ran our algorithm to generate one packet trace of noise and merge that trace with the original webpage packet trace.

3.5 Evaluation

Table 3.4 shows the accuracy of different classification algorithms including either only feature c or both features a and c. The first four features are always included. As

[1]Note that since each webpage is a unique website, we used webpage and website interchangeably from now on.

Algorithm/Attributes considered	c	a and c
DecisionTree	57.34%	58.86%
SVM	29.25%	28.70%
NeuralNetwork	11%	25%
Regression	71.13%	70.30%
RandomForest	81.99%	81.77%

Table 3.4: Accuracy of five statistical and machine learning algorithms considering the two sets of attributes a and c mentioned in [2]. The sample size n used is 100.

shown in the table, adding feature a, the cumulative total of the absolute value of all packet sizes, does not improve the accuracy of website fingerprinting attacks by much. Unlike previous work, the SVM algorithm had a very low accuracy. The reason for this is due to the parameters used for the machine learning algorith. When changing some parameters, SVM achieved an accuracy of 68%. We plan on exploring SVM further. The table shows that Random Forest, which is a collection of decision trees, performs the best at 82% accuracy. The sample size n was set to the recommended 100 from [2]. We considered only the Random Forest classification algorithm in the rest of this research.

We varied the sample size n from 10 to 200. Figure 3.2 shows the accuracy when considering including either only feature c or both features (a and c). As observed, the accuracy is stable at 82% when the sample size is greater than 50. Thus we pick sample size $n = 100$ and without feature a in future experiments.

Figure 3.3 shows the classification accuracy for varying amount of noise added to original traces. Figure 3.4 shows the bandwidth overhead in % of the extra netwok traffic generated. The two basic cover traffic algorithms are indicated by $k = 1$ for adding the same one website as noise each time and by $k = 10$ for randomly adding one of ten websites as noise. The x-axis indicates the amount of noise s added. When

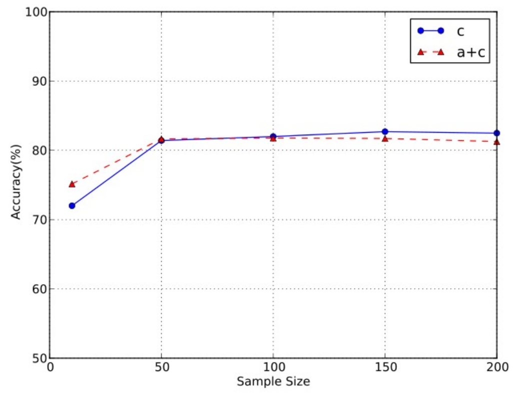


Figure 3.2: The accuracy using the Random Forest algorithm when varying the sample size. Note the y-axis does not start at 0.

$s = 1.0$, this means the whole packet trace is added as noise. When $s = 0.5$, only half of the packet trace is added as noise, that is, every other packet is added as noise to preserve the time intervals. For the basic cover traffic cases ($k = 1$ and $k = 10$), we are "simulating" the noise generated; in a real-world setting, this would be hard to achieve without controlling the server – in this case, the browser could send random packets. We show different values of s to compare with our algorithm. As more noise is added (s increases), the accuracy decreases, as expected. Similarly, the bandwidth overhead also increases as more noise is added. Our proposed noise generation algorithm achieves the same accuracy regardless of the amount of noise; this is because we are generating realistic noise that can more effectively hide a user's real traffic rather than generating random noise. Our algorithm's bandwidth overhead is the same as the basic cases. However, even with $s = 0.25$, the overhead is 20% and the accuracy is 14%. Since our proposed algorithm generates random packet traces based on real recorded network traffic, we ran our experiments five times; the graphs show the average of the five experiments. For these experiments, the training dataset used in the Random Forest classification algorithm is the original 91 webpages and the testing dataset is the new webpages with noise added.

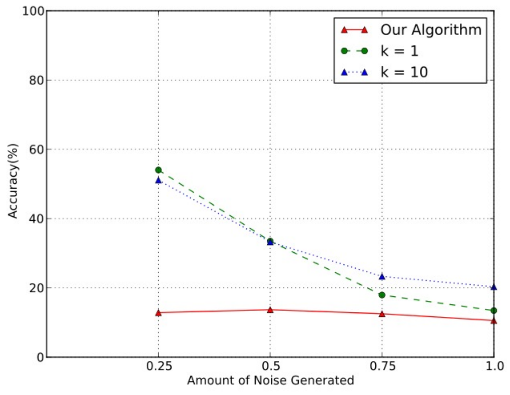


Figure 3.3: The accuracy using the Random Forest algorithm when introducing different kinds of cover traffic.

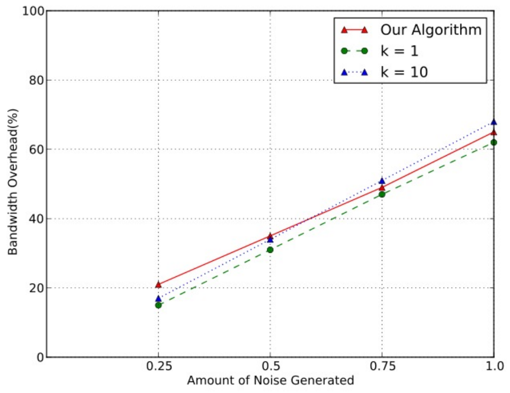


Figure 3.4: The bandwidth overhead when introducing different kinds of cover traffic.

Intuitively, accuracy should decrease as more noise is added. However, in Figure 3.3, we found that in our algorithm, $s = 0.25$ has a lower accuracy than $s = 0.5$. We hypothesized that this could be due to the features being considered. Recall that two of the five features are the total number of incoming packets and the total size of incoming packets. When s is lower, the number of incoming packets is lower. To verify our hypothesis, we re-ran our algorithms without considering these two features of incoming packets. Figure 3.5 shows the result. It can be seen that without these two features, accuracy decreases as noise generated increases, which is expected. This shows that the attributes for total size of incoming packets and total number of incoming packets help the website fingerprinting adversary in successfully identifying the correct website (increase in accuracy). Without these two features, our proposed algorithm performs even better as the accuracy is reduced to under 10% when $s = 1.0$. Previous work [45] has shown that the number of incoming packets is one of the most useful attributes in classification for website fingerprinting. We also considered not including the incoming packet features; the results are shown in Figure 3.6. The results are expected as well but the change in accuracy is not as obvious as Figure 3.5 since the number of outgoing packets is about the same regardless of the value of s.

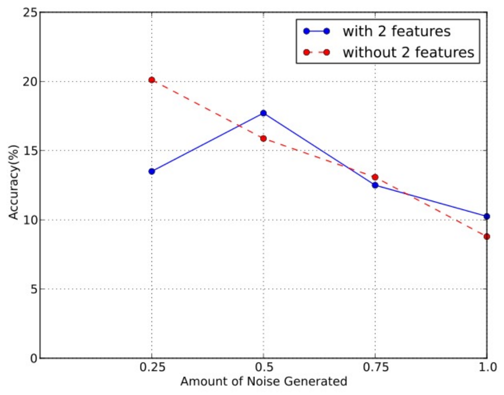


Figure 3.5: The accuracy using the Random Forest algorithm when considering the incoming packet features or not.

Table comparing our results vs others (CS-BUFLO, Tamaraw, WTF-PAD, etc...)

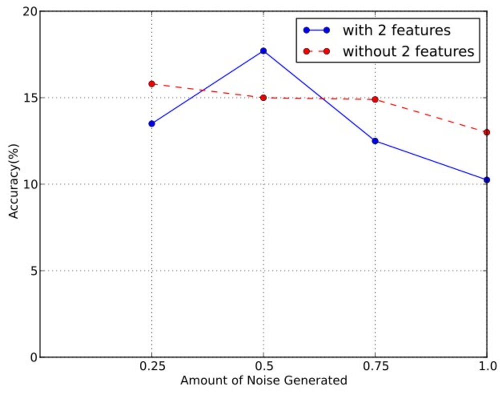


Figure 3.6: The accuracy using the Random Forest algorithm when considering the outgoing packet features or not.

3.6 Related Work

It has been shown that analyzing encrypted network traffic can reveal the websites and webpages visited [2, 31–35, 37–47, 49]. Since the payload is encrypted, only the metadata is available such as packet sizes, number of packets, direction of packets, and time interval between packets. A training dataset is built. Then, given a network traffic trace, machine learning techniques are used to predict the website visited. Previous results have shown that websites can be recognized with a high accuracy. More recent research results have looked at anonymized network traces such as using Tor instead of a simple HTTPS proxy. Although initial results showed that Tor provided adequate protection against website fingerprinting, more advanced data parsing techniques show that websites can be recognized with a fairly high accuracy even when the website trace is over Tor. The consequences of website fingerprinting is censorship or prosecution by the government if the user visits a forbidden website. It has been argued [50] that website fingerprinting is not a practical attack due to the large number of webpages and the false positive would be high. Website fingerprinting attacks have also been extended to identify the webbrowser used [4], which could lead to user identification and linking as most users utilize a unique webbrowser (based on fonts installed, languages, plugins, etc...) [51, 52].

Website fingerprinting is one type of network traffic analysis. There has been other work on network traffic analysis [53] and traffic analysis resistant protocols [24, 54, 55]. Network traffic analysis is usually performed for censorship [56]. Various techniques to avoid censorship have been proposed, using traffic morphing [57] to disguise the network traffic as VoIP [58, 59] or using other covert channels [60–62]. It has, however, been shown that it is still possible to see through this obfuscation [63–65].

Using cover/dummy/fake traffic to mask a user's activities has been proposed before [66]. It has been shown that this mechanism can be countered or the cover traffic removed [67, 68] to reveal the user's activities. Cover traffic is useful to mask real web search queries by performing many other unrelated and random search queries. Cover traffic can also be used to make network traffic analysis harder by adding unrelated network-level packets. Our algorithm generates realistic cover traffic making it harder for website fingerprinting attacks to accurately guess the website from the observed packet trace. Another scheme, Track Me Not [69], focused on web search queries and generating fake web searches, but [70] has shown that web search queries obfuscation can still be analyzed.

Various website fingerprinting defenses have been proposed [1, 32–36, 42]. They all make use of some sort of padding, delaying sending of packets, or adding cover traffic. Many of these defenses have high latency and/or bandwidth overhead and have been shown to be somewhat effective in mitigating website fingerprinting attacks. Our proposed cover traffic defense has zero latency overhead and lower bandwidth overhead while maintaining a high level of effectiveness.

Traffic morphing [57] is another possible defense against website fingerprinting attacks. It attempts to modify the shape and patterns of network traffic such that it looks different. For example, Stegotorus [71] attemps to make Tor network traffic look like HTTPS. Similarly, [58, 59] attempt to morph Tor traffic to look like VoIP traffic so that network traffic analysis or deep packet inspection will not allow Tor traffic to be blocked or identified; VoIP traffic is usually allowed. However, [64, 65] have shown that these traffic morphing schemes can be circumvented. We are not proposing to modify network traffic patterns. Our algorithm generates realistic cover traffic to mask the original packet trace.

3.7 Conclusion

We showed that our proposed cover traffic (noise generation) algorithm mitigates website fingerprinting attacks as effectively as current existing schemes. However, the bandwidth overhead is only 20%, much lower than existing schemes. The latency overhead is also 0%.

We plan to expand this work in considering more webpages for both the training dataset and our learning algorithm. A more detailed study on the different classification algorithms and parameters used will also be performed. For this research, we used an existing dataset; we plan on implementing the webbrowser plug-in and deploy on the Tor browser bundle. Since our algorithm records the packet traces of the user, even though the storage is local only for packet size and timestamp, with no identifiable information such as IP address, it has to be determined if the algorithm could leak any information by generating the cover traffic. Further improvements to our algorithm can be made, such as, if a user has multiple tabs open at the same time, no noise is needed. This would reduce the bandwidth overhead.

CHAPTER 4

Webbrowser Fingerprinting

4.1 Introduction

Webbrowser fingerprinting is widely used in nowadays Internet environment. Many Web services employ webbrowser fingerprinting techniques to track end users. The identification of an end user can import great benefits to web services providers. For example, an advertisement service provider can export new car ad to a user if the user once access auto-vender websites and his/her browser's fingerprinting is recorded by the website. Among these webbrowser fingerprinting techniques, browser plugins, cookies and Javascripts codes are used to collect fingerprinting charactics. These tools can collect webbrowser specific information such as webbrowser' producer, minor version, plug-in usages and font usages etc. Combining these information, an end user can be accurately identified.

The goal of this research is to determine whether the webbrowser fingerprinting characteristics can be obtained based solely on network traffic. The adversary is not the server, but any entity between the client and the server. The adversary thus can only eavesdrop on the network traffic. Relying only on the network traffic, the goal is then to identify the webbrowser being used. The effects of identifying the webbrowser are tracking users across different sessions and multiple connections to different servers, or injecting malware specific to a webbrowser. As shown in [51, 52], users can be almost uniquely identified from their webbrowser information.

All the work on webbrowser identification such as [51, 72] have assumed that the identifier entity is located at the server's side. That entity has the ability to set cookies, look at user agent strings, and so on. Using this method, a lot of information can be collected about the webbrowser such as the type of browser (Firefox, Internet Explorer), the major and minor version of the software, the fonts installed, plugins installed/enabled, and the web history of that user. All this information can be used to uniquely identify and track users across different sessions over time and across different website visits.

For network traffic identification, the closest work is website fingerprinting [43,44]. Looking at anonymized and encrypted network traffic, an adversary can identify the website being visited. This is useful for censoring particular websites. Our work focuses on network traffic analysis and uses similar techniques to identify webbrowsers.

Threat Model: The threat model considered in this research is that the client machine, the webbrowser used, and the server are all honest and free of any malware or bugs. The adversary can see the traffic between the client and the server. The adversary could be the client's ISP or any router on the path from the client to the server. The network traffic could be encrypted or in plain-text or through an anonymizing network such as Tor [13]. The adversary can only eavesdrop on the network traffic; it cannot modify, drop, or inject any network packet.

Previous works [32, 41] have proved that websites can be accurately identified by checking traffic data transfered between webbrowser and web server. In their paper, one important assumption is that the attacker knows victim's browser and its configuration. In this research, our research shows that when accessing a specific website, the traffic data between webbrowser and web server is seriously affected by the webbrowser specific characteristics. These webbrowser specific characteristics, like browser type and plug-in usages, can be accurately identified based on traffic data. For the websites used in our experiments, the identification accuracy rate ranges from 72% to 85%. Popular webbrowser plug-ins like JavaScript and flash usage status can be also be accurately identified.

The research is organized as follows. Section 4.2 provides an overview of previous work on webbrowser and website fingerprinting and identification. Our experimental setup is described in more details in Section 4.3 and the results shown in Section 4.4. Finally, a discussion of what the results imply for the future of webbrowsers is given in Section 4.5.

4.2 Related Work

It has been shown [31,32,41–44] that analyzing network traffic can reveal the websites and webpages being visited. The authors looked mostly at packet sizes and packet direction. They collected a training dataset from specific websites and clustered the dataset. Given a network traffic trace, the authors used machine learning techniques to predict the website visited. Website fingerprinting can be used when users want to hide who they are communicating with, for example, by using Tor [13]. These users might want to bypass any censorship mechanism blocking certain websites. Using website fingerprinting, the censor can identify users accessing censored web content. The latest paper [44] showed that specific websites can be accurately identified, only by examining the network trace. Our work differs in that although we are analyzing network traffic, the traffic analysis is used to identify and track webbrowsers, not to fingerprint websites visited.

Panopticlick [51, 52] showed how browser information, such as user agent string, fonts installed, languages supported, and plugins enabled, can be used to uniquely identify users. Every user has an almost unique fingerprint and this can be used to track users visiting different websites or across different visits to the same website. The main goal of identifying users is to track them. This is especially useful for advertising. The EverCookie [73] is a persistent storage that is not cleared when the browsers cleans its cache and cookies. This permanent cookie can be used to track users. [72] showed how users can still be tracked without setting any cookies. FPDetective [74] did a wide survey of popular Internet websites and determined which website was performining browser fingerprinting. Our work is different because the fingerprinting is not at the server side; we assume the server is honest. Instead, the adversary can only eavesdrop on network traffic between the client and the server.

In an attempt to hide their Internet activities, remain anonymous, or bypass censors, anonymity-providing schemes such as Tor [13, 14] and Anonymizer [75] have been deployed. These schemes only hide who is communicating with whom. Network traffic analysis can still be performed to identify any website visited. Server-side fingerprinting can still be performed to identify users and webbrowsers. There are tools [76, 77] that attempt to remove identifying or sensitive information from web headers. However, they don't provide complete protection as shown in [44, 72].

Network traffic analysis have been around for some time [20, 55]. They can be used to detect the application being used (such as Skype), or more commonly, to deanonymize users. Various countermeasures such as traffic morphing [57] have been proposed. A survey of network traffic analysis attacks and countermeasures is given in [54]. In our work, we leverage network traffic analysis to identify and track users from their webbrowser information.

Software	Version
Client Operating System	Windows 7
Web Browser 1	Google Chrome 29.0
Web Browser 2	Mozilla Firefox 24.0
Web Browser 3	Internet Explorer 10.0
Web Browser 4	Tor Browser Bundle 3.6.1
Network Protocol Analyzer	Wireshark 1.10.1
Data Mining Tool	Weka 3.6.10

Table 4.1: Software used during experiments and classification.

4.3 Experimental Setup

4.3.1 Collecting Data

Our assumption is that we know which web site the user is visiting, and we need to identify who is visiting the web site, that is, to get the fingerprinting information of the webbrowser which the user is using. The information we can use to reach this goal is only the network traffic generated when browsing a web page. In this research, network traffic data are collected for four types of webbrowsers and homepages of 9 top Alexa [78] websites. Our experiments are done on Windows 7 platform. The browsers we run experiments upon are Google Chrome, Internet Explorer, Firefox and Tor Browser Bundle. The first three are widely used web browsers. Tor Browser Bundle [76] is a pre-configured Firefox browser, it integrates Tor software to the webbrowser and takes advantage of Tor to protect users' anonymity. Wireshark is used to record TCP/IP traffic data. Table 4.1 shows the detail of softwares we used.

We use top 9 most popular web sites (downloaded from Alexa web site [78]),

Number	Website
1	www.google.com
2	www.yahoo.com
3	www.facebook.com
4	www.amazon.com
5	www.qq.com
6	www.taobao.com
7	www.live.com
8	www.ebay.com
9	www.youtube.com

Table 4.2: Websites used for experiments.

open them in web browsers and record TCP/IP traffic data. Some web sites support localization, they will automatically switch to different versions of web pages according to the region where we launch the browser. When using Tor Browser bundle, the region of our browser is determined by the exit node of the Tor circuit. In our experiments, we specify the region for google.com and yahoo.com as `https://www.google.com/ncr` and `https://us.yahoo.com/?p=us` respectively. In this way, we can ensure that each time we are visiting the same website. Table 4.2 lists the detail of web sites we used.

To run experiments automatically, we wrote a Java program. This program will iterate all these 9 websites and 4 browsers. For each iteration, the following 5 steps are taken:

I. Launch Wireshark to record traffic data.

II. Wait 30 seconds then launch one web browser and connect to a website.

III. Wait 300 seconds then kill web browser.

Browser plugin status				
flash	enable	enable	disable	disable
JavaScript	disable	enable	enable	disable

Table 4.3: plugin usage status of browser.

IV. Save traffic data to file and wait 30 seconds then Quit Wireshark.

V. Wait 300 seconds then go to step 1 and start another experiment.

For each experiment, we open a website with a specific web browser and then record all the TCP/IP packets between the web browser and the web server. In step 1, we launch Wireshark first. In step 2, we wait 30 seconds. The 30 seconds delay is to ensure we dont miss any packets. in step 3, we keep the browser running for 600 seconds so that all the contents of the web page have been downloaded. In step 4, the traffic data is recorded to a file like windows_chrome_exp1_amazon_20130915. We use file name to record information like OS platform, browser type, web site name and experiment number. To avoid experiments influencing each other, for example, web clients receive packets comes from web servers we visited in the previous experiment, in step 5, we wait another 300 seconds.

Javascript and flash are two popular plug-ins. With JavaScript, web pages are more user friendly. With flash, we can watch videos on web pages. However, JavaScript and flash can also put users at risk. To avoid this, some users tend to disable them. Taking the plug-ins usage into consideration, we define a browser to be in four different status, Table 4.3 shows the detail of these four status.

Except for Tor Browser Bundle [76], we perform 10 experiments for each browser, web page and plugin usages status combination. In total we have 1440 experiments ($4 \times 9 \times 40$). Our experiments were performed from February 2, 2014 to April 18, 2014.

4.3.2 Processing Data

To analysis these traffic data, for each experiment, we transfer the packets information to a data series based on the direction of packets (outgoing and incoming) and time series they are recorded. We sum up the packet size for a specific time frame interval based on the direction of packets. For outgoing packets, we use negative numbers to represent them by multiplying the sum of packets size with -1. We put the browser type at the end of the data series as class attribute. For each web site, we have 160 ($10 \times 4 \times 4$) experiments. Correspondingly we have 160 data series.

Weka [48] is a popular open source data mining tool, it provides lots of powerful classification algorithms. In this research, Weka [48] is used as the classification tool. To use Weka [48], we put all the 160 data series to a file with .arff file format (the standard Weka dataset file format). To get the best classification model, we tried ZeroR, OneR, C4.5, NaiveBayes and SVM (support vector machine) algorithms. These algorithms are all implemented in Weka [48]. ZeroR is the base line of classification, basically, it is randomly guessing based on probability. oneR is trying to find one attribute to classify instances. The test model we used is 10-fold cross-validation. In this test model, the dataset is randomly reordered and then split into 10 folds of equal size. In each iteration, one fold is used for testing and the other 9 folds are used for training the classifier. The test results are collected and averaged over all folds. Cross validation is quite useful in dealing with small datasets since it utilizes the greatest amount of training data from the dataset [79].

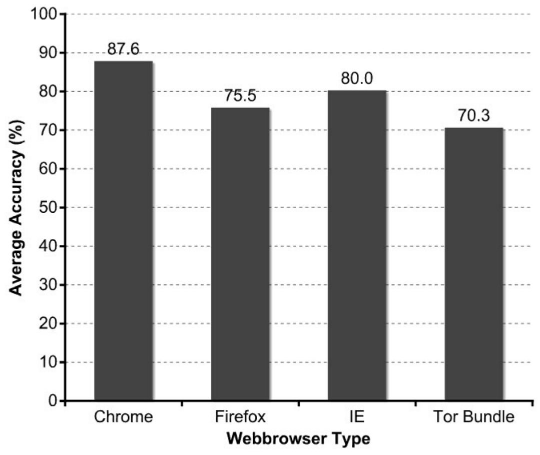


Figure 4.1: Average identification accuracy for different type of browsers when analysis time frame is 2 seconds.

4.4 Results

4.4.1 Webbrowser type identification

The type of the webbrowser is one of the most common webbrowser fingerprinting characteristics. Webbrowser type and version information usually can be obtained by checking the http user-agent header. Our assumption is that we can not see the plaintext of the packets content. All we have is the time series of packet size. Classification models are built upon these time series and the type of classification algorithm.

Figure 4.1 shows the average accuracy rate of identifying browser type for all nine web sites with C4.5 classification algorithm. In this research, we use accuracy rate to represent the performance of a classification model. Suppose 40 experiments with 10 experiments for each one of the four type of browsers, 8 experiments are correctly classified as using Chrome, 7 experiments are correctly classified as using Firefox, then the accuracy rate of identifying Chrome and Firefox is 80% and 70% correspondingly.

The accuracy rate of identifying browser type is slightly different for specific browsers. We can see that the best one is for Google Chrome which is identified with 87.27% average accuracy rate. That means that for all 360 experiments which use Chrome, 317 (360×0.88) of them are correctly identified, and 43 of them are identified as others. Tor browser bundle has lowest identification average accuracy which is around 69.99%. In our dataset, we have 4 different types of browsers, so the base line probability of identifying the correct browser type is 25%. Our worst accuracy rate, which is 70% for Tor bundler, is still 2.8 times of the base line accuracy.

For different websites, the accuracy rate of identifying browser type also changes. Table 4.4 shows the exact accuracy rate for nine websites. Though the accuracy rate changes for different websites, we can see that for each specific website, when browsing with different browsers, their traffic data series have significant difference. Among these 9 websites, www.facebook.com has the best average accuracy which is about 85% and www.qq.com has the worst average accuracy which is about 73%.

Though 70% accuracy rate is pretty good comparing to the 25% base line accuracy, there are still some improvements we can make to enhance it. Like Cai's work [32], we can round packets size to a multiple of 600 and remove noises of traffic data. In future works, we will try these kinds of data refine to get higher accuracy.

4.4.2 Webbrowser plug-ins identification

Besides the browser type, plug-in usage status is also important characteristic to fingerprint a browser. In this research, our assumption is that we have 13 combinations of browser type and plug-in usages. For browser Chrome, IE and Firefox, each of them can have 4 possible plug-in usage combination regarding the enabling or disabling of flash and JavaScript plug-in. For Tor bundle, we use the default plug-in setting.

We run C4.5 classification algorithm against the packet size series, the test model

	Identification accuracy (%)				
Websites	Chrome	Firefox	IE	Tor bundle	Average Accuracy for 4 browsers
www.amazon.com	89.74	82.05	64.10	75.68	77.89
www.ebay.com	84.62	87.18	82.05	70.15	81.00
www.facebook.com	92.50	80.00	87.50	81.08	85.27
www.google.com	95.00	70.00	77.50	64.86	76.84
www.live.com	84.62	82.05	89.74	63.64	80.01
www.qq.com	89.74	61.54	74.36	64.86	72.63
www.taobao.com	76.92	76.92	84.62	69.35	76.95
www.yahoo.com	85.00	72.50	75.00	70.27	75.69
www.youtube.com	90.00	67.50	85.00	73.30	78.95
Average	87.27	76.53	79.36	69.99	78.29

Table 4.4: Idenfification accuracy for specific website.

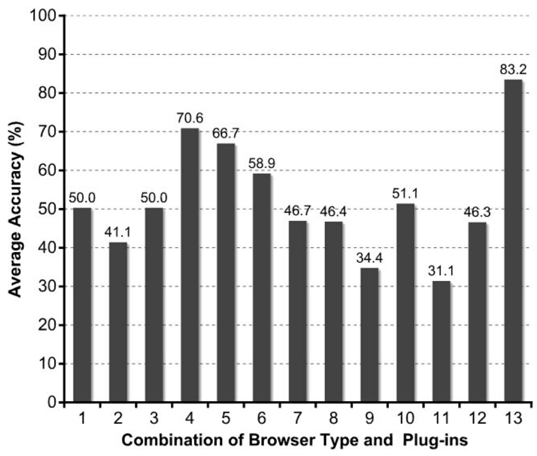


Figure 4.2: Average identification accuracy for different combination of browser and plug-ins when analysis time frame is 2 seconds. See table 4.5 for the detail of the combination.

we used is 10 fold cross verification, similar for identifying browser type. Figure 4.2 shows that for all 9 websites, the average identification accuracy of combinations of browser type and plug-ins. Besides Tor bundle, which we use default plug-in setting, the highest identification accuracy is for combination number 4 which is about 70.62% and the lowest one is for combination number 11, which is about 31.11%. Since there are 13 classes of browser type and plug-ins usage combination, the baseline probability of identifying one combination is less then 7.7% (1/13 = 0.0769) . Our worst case of accuracy, which is 31.11% is more then 4 times of the baseline probability. When the time frame of the packets size is 2 seconds, the average accuracy of all 13 combination is about 52.0%, which is 6.7 times higher than the baseline probability.

Table 4.5 shows the detail of combination of browser type and plug-in usage.

For a specific website, the average identification accuracy of all 13 combination changes slightly. For websites like YouTube, eBay and Amazon, they have relatively higher average identification accuracy. The reason is that their web pages contains more videos/images, and they use JavaScript to control how these videos/images are displayed on web pages. Enabling or disabling JavaScript and flash have much more influence to the network traffic than other websites.

No.	Browser type	Flash plug-in	JavaScript plug-in
1	Chrome	disable	enable
2	Chrome	disable	disable
3	Chrome	enable	enable
4	Chrome	enable	disable
5	Firefox	disable	enable
6	Firefox	disable	disable
7	Firefox	enable	enable
8	Firefox	enable	disable
9	IE	disable	enable
10	IE	disable	disable
11	IE	enable	enable
12	IE	enable	disable
13	Tor bundle	N.A.	N.A.

Table 4.5: Different combinations of browser type and plug-ins usage status.

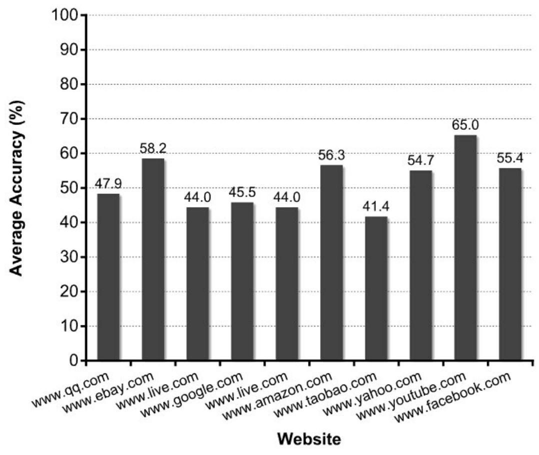


Figure 4.3: Average identification accuracy of each website for different combination of browser and plug-ins when analysis time frame is 2 seconds.

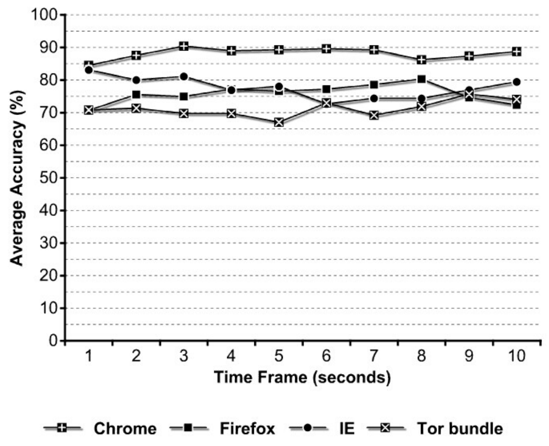


Figure 4.4: Average identification accuracy of C4.5 algorithm for different type of browsers and different time frames.

4.4.3 Varying time frames and classification algorithms

To get the best result, we tried time frames from 1 second to 10 seconds to generate the different data series. The basic idea is that when time frame is smaller, the data series we get contains finer information about the traffic. On the other hand, when time frame is smaller, it will take more time to run the classifier and model is also easier to suffer over-fitting. Besides the better accuracy rate of classification, we also need some kind of trade-off between the accuracy rate and model complexity. Figure 4.4 shows that for algorithm C4.5, the time frames we take has slight influence to the accuracy rate of identifying browser type.

When classifying a dataset, what kind of classification algorithm we choose usually have great influence to the performance of the final classification model. In this research, to find the best classification model, we tried zeroR, oneR, C4.5, Naive-Bayes and SVM algorithms. Table 4.6 shows the average browser type identification accuracy rate of all four type of browsers and nine websites when using different classification algorithms.

Among all these 5 algorithms, the C4.5 algorithm has best performance. It has

seconds	zeroR	oneR	C4.5	NaiveBayes	SVM
1	24.29	67.51	80.81	72.47	24.50
2	24.29	68.52	80.56	65.74	24.78
3	24.29	67.53	81.89	63.25	24.29
4	24.29	66.34	81.09	62.92	24.29
5	24.29	65.97	79.94	61.15	24.29
6	24.29	66.59	80.65	62.97	24.29
7	24.29	66.57	79.29	62.70	24.29
8	24.29	66.61	80.90	62.39	24.29
9	24.29	66.79	80.24	63.32	24.29
10	24.29	66.79	80.41	60.28	24.29

Table 4.6: Identification accuracy for different time frame and classification algorithm.

around 80% accuracy and the accuracy is relatively stable for different time frames. Accuracy of NaiveBayes algorithm seems to increase as time frames become smaller. But as we take smaller time frames, say 0.5, its accuracy did not increase.

SVM works badly in our case, there are two reasons: 1, data series used in this research are all based on raw data, we have no data rounding or normalization. 2, to ensure all traffic data are recorded, for each experiment, we collect all traffic data in 5 minutes starting from the launch of accessing a web page. This results that we have lots of zeros and small random numbers in the data series because a web page usually can be loaded in less then 1 minute. These zeros and small random numbers in data series decrease the classification accuracy of SVM greatly.

4.4.4 Summary

Our research has shown that we can identify the webbrowser type with a high accuracy rate by solely checking the time series of network packets size. Popular plug-ins usage status can also be identified with a good accuracy rate as well. The methodology used in this research is straightforward and can be easily reused for different webbrowser fingerprinting purpose.

4.5 Discussion and Conclusion

We have shown that it is possible to identify webbrowsers characteristics through network traffic analysis only, even if the network traffic is encrypted and through an anonymized service such as Tor. The consequences of this research are numerous. The webbrowsers and plugins used can be identified through network traffic analysis only. Users can potentially be identified and tracked across sessions and visits to different websites. The identification entity does not have to be part of the server. It can be any router along the path from the user to the server, for example, the ISP of the server or the user.

Our research also shows that the traffic data is seriously affected by webbrowser specific characteristics. This suggests that when we perform website fingerprinting based on traffic data, the webbrowser plays an important role and need to be considered in future work.

There remains more work to be done in this research area. More experiments need to be performed. The number of experiments per website need to be increased. The number of websites tested need to be increased. The reason why the difference of traffic data exists is also an interesting topic.

CHAPTER 5

Conclusion and Future Work

In this dissertation, we use data analytics to learn from and improve existing systems. We focus on privacy-aware systems and three data analytics case studies are conducted.

In the first case study, we showed that a circuit clogging attack is still possible in the current Tor network. The costs to perform a circuit clogging attack are also very low, making it a practical attack. We showed that the Tor relays used in a circuit can be accurately identified. Moreover, the false positive rate is low as only some other Tor relays not used in the circuit are accidentally identified as being part of the circuit. A circuit clogging attack detection mechanism is also proposed. The scheme uses a probe to monitor all the circuits created by the client. Once an increase in network latency from a previously recorded baseline time is measured, the server is flagged as suspicious. The client can then disconnect from the server and destroy the affected circuit. Through experiments on the real Tor network, the proposed detection scheme has an accuracy of over 85%. This research showed that the anonymity of a person using Tor is reduced since the Tor relays used can be identified. This can be a stepping stone towards narrowing down the possible users behind these relays. The detection scheme proposed allows a user to detect possible occurrences of circuit clogging attacks. With over $500,000$ users daily, the attack has huge potential consequences. The proposed detection scheme can help hundreds of thousands of people stay anonymous on the Internet. The current attack identifies the Tor relays in the circuit; future work will identify which of the relays are the entry, middle, and exit relays. This would require more fine-grained probe measurements. One of the anonymity improvements in Tor is to use entry guards, a fixed set of three relays used as entry relay in any circuit. We will also analyze whether using entry guards leaks any information, as the user could be more easily identified. If the same entry relay is found in circuits, this can leak information about the user. The possible impact of the attack on bridges and hidden servers is left as future work.

In the second case study, we showed that our proposed cover traffic (noise generation) algorithm mitigates website fingerprinting attacks as effectively as current existing schemes. However, the bandwidth overhead is only 20%, much lower than existing schemes. The latency overhead is also 0%. We plan to expand this work in considering more webpages for both the training dataset and our learning algorithm. A more detailed study on the different classification algorithms and parameters used will also be performed. For this research, we used an existing dataset; we plan on implementing the webbrowser plug-in and deploy on the Tor browser bundle. Since our algorithm records the packet traces of the user, even though the storage is local only for packet size and timestamp, with no identifiable information such as IP address, it has to be determined if the algorithm could leak any information by generating the cover traffic. Further improvements to our algorithm can be made, such as, if a user has multiple tabs open at the same time, no noise is needed. This would reduce the bandwidth overhead.

In the third case study, we have shown that it is possible to identify webbrowsers characteristics through network traffic analysis only, even if the network traffic is encrypted and through an anonymized service such as Tor. The consequences of this research are numerous. The webbrowsers and plugins used can be identified through network traffic analysis only. Users can potentially be identified and tracked across sessions and visits to different websites. The identification entity does not have to be part of the server. It can be any router along the path from the user to the server, for example, the ISP of the server or the user. Our research also shows that the traffic data is seriously affected by webbrowser specific characteristics. This suggests that when we perform website fingerprinting based on traffic data, the webbrowser plays an important role and need to be considered in future work. There remains more work to be done in this research area. More experiments need to be performed. The number of experiments per website need to be increased. The number of websites tested need to be increased. The reason why the difference of traffic data exists is also an interesting topic.

Bibliography

[1] M. Juarez, M. Imani, M. Perry, C. Diaz, and M. Wright, "Toward an efficient website fingerprinting defense," in *ESORICS*, 2016.

[2] A. Panchenko, F. Lanze, A. Zinnen, M. Henze, J. Pennekamp, K. Wehrle, and T. Engel, "Website fingerprinting at internet scale," in *Proceedings of the 23rd Internet Society (ISOC) Network and Distributed System Security Symposium (NDSS 2016)*, 2016.

[3] E. Chan-Tin, J. Shin, and J. Yu, "Revisiting circuit clogging attacks on tor," in *Availability, Reliability and Security (ARES), 2013 Eighth International Conference on*, pp. 131–140, IEEE, 2013.

[4] J. Yu and E. Chan-Tin, "Identifying webbrowsers in encrypted communications," in *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, pp. 135–138, ACM, 2014.

[5] J. Cowie, "Egypt leaves the internet." http://www.renesys.com/blog/2011/01/egypt-leaves-the-internet.shtml.

[6] J. Vargas, "How an egyptian revolution began on facebook." http://www.nytimes.com/2012/02/19/books/review/how-an-egyptian-revolution-began-on-facebook.html.

[7] J. Brodkin, "Iran reportedly blocking encrypted internet traffic." http://arstechnica.com/tech-policy/2012/02/iran-reportedly-blocking-encrypted-internet-traffic/.

[8] "I2P anonymous network." http://www.i2p2.de/.

[9] G. Danezis, R. Dingledine, and N. Mathewson, "Mixminion: Design of a Type III Anonymous Remailer Protocol," in *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, pp. 2–15, May 2003.

[10] U. Möller, L. Cottrell, P. Palfrader, and L. Sassaman, "Mixmaster Protocol — Version 2." IETF Internet Draft, July 2003.

[11] M. Rennhard and B. Plattner, "Introducing MorphMix: Peer-to-Peer based Anonymous Internet Usage with Collusion Detection," in *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2002)*, (Washington, DC, USA), November 2002.

[12] M. Rennhard and B. Plattner, "Practical anonymity for the masses with morphmix," in *Proceedings of Financial Cryptography (FC '04)* (A. Juels, ed.), pp. 233–250, Springer-Verlag, LNCS 3110, February 2004.

[13] "Tor." https://www.torproject.org/.

[14] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[15] "Tor metrics portal." https://metrics.torproject.org/.

[16] N. Hopper, E. Y. Vasserman, and E. Chan-Tin, "How much anonymity does network latency leak?," in *Proceedings of the 14th ACM conference on Computer and communications security (CCS)*, pp. 82–91, 2007.

[17] A. Back, U. Möller, and A. Stiglic, "Traffic analysis attacks and trade-offs in anonymity providing systems," in *Proceedings of Information Hiding Workshop*

(IH 2001) (I. S. Moskowitz, ed.), pp. 245–257, Springer-Verlag, LNCS 2137, April 2001.

[18] G. Danezis and A. Serjantov, "Statistical disclosure or intersection attacks on anonymity systems," in *Proceedings of 6th Information Hiding Workshop (IH 2004)*, LNCS, (Toronto), May 2004.

[19] V. Shmatikov and M.-H. Wang, "Timing analysis in low-latency mix networks: Attacks and defenses," in *Proceedings of ESORICS 2006*, September 2006.

[20] S. J. Murdoch and G. Danezis, "Low-cost traffic analysis of Tor," in *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, IEEE CS, May 2005.

[21] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker, "Low-resource routing attacks against Tor," in *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2007)*, (Washington, DC, USA), October 2007.

[22] N. Evans, R. Dingledine, and C. Grothoff, "A practical congestion attack on tor using long paths," in *Proceedings of the 18th USENIX Security Symposium*, August 2009.

[23] X. Wang, S. Chen, and S. Jajodia, "Network flow watermarking attack on low-latency anonymous communication systems," in *IEEE Symposium on Security and Privacy*, pp. 116 –130, may 2007.

[24] P. Mittal, A. Khurshid, J. Juen, M. Caesar, and N. Borisov, "Stealthy traffic analysis of low-latency anonymous communication using throughput fingerprinting," in *Proceedings of the 18th ACM conference on Computer and Communications Security (CCS 2011)*, October 2011.

[25] Y. Gilad and A. Herzberg, “Spying in the Dark: TCP and Tor Traffic Analysis,” in *Proceedings of the 12th Privacy Enhancing Technologies Symposium (PETS 2012)*, Springer, July 2012.

[26] “BitTorrent.” http://www.bittorrent.com/.

[27] D. M. Goldschlag, M. G. Reed, and P. F. Syverson, “Hiding Routing Information,” in *Proceedings of Information Hiding: First International Workshop* (R. Anderson, ed.), pp. 137–150, Springer-Verlag, LNCS 1174, May 1996.

[28] J. McLachlan and N. Hopper, “Don’t clog the queue: Circuit clogging and mitigation in P2P anonymity schemes,” in *Proceedings of Financial Cryptography (FC ’08)*, January 2008.

[29] T. Fawcett, “An introduction to roc analysis,” *Pattern Recogn. Lett.*, vol. 27, pp. 861–874, June 2006.

[30] “Amazon EC2.” http://aws.amazon.com/ec2/.

[31] A. Hintz, “Fingerprinting websites using traffic analysis,” in *Proceedings of Privacy Enhancing Technologies workshop (PET 2002)* (R. Dingledine and P. Syverson, eds.), Springer-Verlag, LNCS 2482, April 2002.

[32] X. Cai, X. Zhang, B. Joshi, and R. Johnson, “Touching from a distance: Website fingerprinting attacks and defenses,” in *Proceedings of the 19th ACM conference on Computer and Communications Security (CCS 2012)*, October 2012.

[33] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, “Effective attacks and provable defenses for website fingerprinting,” in *Proceedings of the 23rd USENIX Conference on Security Symposium*, SEC’14, (Berkeley, CA, USA), pp. 143–157, USENIX Association, 2014.

[34] R. Nithyanand, X. Cai, and R. Johnson, "Glove: A bespoke website fingerprinting defense," in *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, WPES '14, (New York, NY, USA), pp. 131–134, ACM, 2014.

[35] X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg, "A systematic approach to developing and evaluating website fingerprinting defenses," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, (New York, NY, USA), pp. 227–238, ACM, 2014.

[36] X. Cai, R. Nithyanand, and R. Johnson, "Cs-buflo: A congestion sensitive website fingerprinting defense," in *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, WPES '14, (New York, NY, USA), pp. 121–130, ACM, 2014.

[37] Q. Sun, D. R. Simon, Y.-M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu, "Statistical identification of encrypted web browsing traffic," in *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, (Berkeley, California), May 2002.

[38] G. D. Bissias, M. Liberatore, D. Jensen, and B. N. Levine, "Privacy vulnerabilities in encrypted http streams," in *Proceedings of the 5th International Conference on Privacy Enhancing Technologies*, PET'05, (Berlin, Heidelberg), pp. 1–11, Springer-Verlag, 2006.

[39] M. Liberatore and B. N. Levine, "Inferring the Source of Encrypted HTTP Connections," in *Proceedings of the 13th ACM conference on Computer and Communications Security (CCS 2006)*, pp. 255–263, October 2006.

[40] L. Lu, E.-C. Chang, and M. C. Chan, *Website Fingerprinting and Identification Using Ordered Feature Sequences*, pp. 199–214. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[41] R. W. Dominik Herrmann and H. Federrath, "Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial nave-bayes classifier," in *Proceedings of the 2009 ACM workshop on Cloud computing security*, CCSW '09, (New York, NY, USA), pp. 31–42, ACM, 2009.

[42] A. Z. Andriy Panchenko, Lukas Niessen and T. Engel, "Website fingerprinting in onion routing based anonymization networks," in *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*, WPES '11, (New York, NY, USA), p. 103114, ACM, 2011.

[43] X. Gong, N. Borisov, N. Kiyavash, and N. Schear, "Website detection using remote traffic analysis," in *Proceedings of the 12th Privacy Enhancing Technologies Symposium (PETS 2012)*, Springer, July 2012.

[44] T. Wang and I. Goldberg, "Improved website fingerprinting on tor," in *Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society*, WPES '13, (New York, NY, USA), pp. 201–212, ACM, 2013.

[45] T. Wang and I. Goldberg, "On realistically attacking tor with website fingerprinting," in *Privacy Enhancing Technologies Symposium (PETS)*, 2016.

[46] R. Spreitzer, S. Griesmayr, T. Korak, and S. Mangard, "Exploiting data-usage statistics for website fingerprinting attacks on android," in *Proceedings of the 9th ACM Conference on Security &#38; Privacy in Wireless and Mobile Networks*, WiSec '16, (New York, NY, USA), pp. 49–60, ACM, 2016.

[47] J. Hayes and G. Danezis, "k-fingerprinting: A robust scalable website fingerprinting technique," in *25th USENIX Security Symposium (USENIX Security 16)*, (Austin, TX), pp. 1187–1203, USENIX Association, Aug. 2016.

[48] "Weka." `http://www.cs.waikato.ac.nz/ml/weka/`.

[49] M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt, "A critical evaluation of website fingerprinting attacks," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, (New York, NY, USA), pp. 263–274, ACM, 2014.

[50] "Experimental defense for website traffic fingerprinting." https://blog.torproject.org/blog/experimental-defense-website-traffic-fingerprinting.

[51] P. Eckersley, "How unique is your web browser?," in *Proceedings of the 10th International Conference on Privacy Enhancing Technologies*, PETS'10, (Berlin, Heidelberg), pp. 1–18, Springer-Verlag, 2010.

[52] "Panopticlick." https://panopticlick.eff.org/.

[53] B. Miller, L. Huang, A. D. Joseph, and J. D. Tygar, *I Know Why You Went to the Clinic: Risks and Realization of HTTPS Traffic Analysis*, pp. 143–163. Cham: Springer International Publishing, 2014.

[54] S. Le Blond, D. Choffnes, W. Zhou, P. Druschel, H. Ballani, and P. Francis, "Towards efficient traffic-analysis resistant anonymity networks," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, (New York, NY, USA), pp. 303–314, ACM, 2013.

[55] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Peek-a-boo, I still see you: Why efficient traffic analysis countermeasures fail," in *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, May 2012.

[56] M. C. Tschantz, S. Afroz, Anonymous, and V. Paxson, "SoK: Towards Grounding Censorship Circumvention in Empiricism," *IEEE Symposium on Security and Privacy*, 2016.

[57] C. Wright, S. Coull, and F. Monrose, “Traffic morphing: An efficient defense against statistical traffic analysis,” in *Proceedings of the Network and Distributed Security Symposium - NDSS ’09*, IEEE, February 2009.

[58] A. Houmansadr, T. J. Riedl, N. Borisov, and A. C. Singer, “I want my voice to be heard: Ip over voice-over-ip for unobservable censorship circumvention.,” in *NDSS*, 2013.

[59] H. M. Moghaddam, B. Li, M. Derakhshani, and I. Goldberg, “Skypemorph: Protocol obfuscation for tor bridges,” in *Proceedings of the 19th ACM conference on Computer and Communications Security (CCS 2012)*, October 2012.

[60] D. Fifield, N. Hardison, J. Ellithorpe, E. Stark, D. Boneh, R. Dingledine, and P. Porras, “Evading censorship with browser-based proxies,” in *Proceedings of the 12th International Conference on Privacy Enhancing Technologies*, PETS’12, (Berlin, Heidelberg), pp. 239–258, Springer-Verlag, 2012.

[61] Q. Wang, X. Gong, G. T. Nguyen, A. Houmansadr, and N. Borisov, “Censorspoofer: Asymmetric communication using ip spoofing for censorship-resistant web browsing,” in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS ’12, (New York, NY, USA), pp. 121–132, ACM, 2012.

[62] J. Holowczak and A. Houmansadr, “Cachebrowser: Bypassing chinese censorship without proxies using cached content,” in *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS ’15, (New York, NY, USA), pp. 70–83, ACM, 2015.

[63] L. Wang, K. P. Dyer, A. Akella, T. Ristenpart, and T. Shrimpton, “Seeing through network-protocol obfuscation,” in *Proceedings of the 22Nd ACM*

SIGSAC Conference on Computer and Communications Security, CCS '15, (New York, NY, USA), pp. 57–69, ACM, 2015.

[64] A. Houmansadr, C. Brubaker, and V. Shmatikov, "The parrot is dead: Observing unobservable network communications," in *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, (Washington, DC, USA), pp. 65–79, IEEE Computer Society, 2013.

[65] J. Geddes, M. Schuchard, and N. Hopper, "Cover your acks: Pitfalls of covert channel censorship circumvention," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer &#38; Communications Security*, CCS '13, (New York, NY, USA), pp. 361–372, ACM, 2013.

[66] C. Diaz and B. Preneel, "Taxonomy of mixes and dummy traffic," in *Proceedings of I-NetSec04: 3rd Working Conference on Privacy and Anonymity in Networked and Distributed Systems*, August 2004.

[67] N. Mallesh and M. Wright, "Countering statistical disclosure with receiver-bound cover traffic," in *Proceedings of 12th European Symposium On Research In Computer Security (ESORICS 2007)* (J. Biskup and J. Lopez, eds.), vol. 4734 of *Lecture Notes in Computer Science*, pp. 547–562, Springer, September 2007.

[68] C. T. Simon Oya and F. Pérez-González, "Do dummies pay off? limits of dummy traffic protection in anonymous communications," in *Proceedings of the 14th Privacy Enhancing Technologies Symposium (PETS 2014)*, July 2014.

[69] D. Howe and H. Nissenbaum, "Trackmenot: Resisting surveillance in web search," *On the Identity Trail: Privacy, Anonymity and Identify in a Networked Society*, 2008.

[70] S. Peddinti and N. Saxena, "On the privacy of web search based on query obfuscation: A case study of trackmenot," in *Privacy Enhancing Technologies* (M. Atal-

lah and N. Hopper, eds.), vol. 6205 of *Lecture Notes in Computer Science*, pp. 19–37, Springer Berlin Heidelberg, 2010.

[71] Z. Weinberg, J. Wang, V. Yegneswaran, L. Briesemeister, S. Cheung, F. Wang, and D. Boneh, "StegoTorus: A camouflage proxy for the Tor anonymity system," in *Proceedings of the 19th ACM conference on Computer and Communications Security (CCS 2012)*, October 2012.

[72] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna, "Cookieless monster: Exploring the ecosystem of web-based device fingerprinting," in *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, (Washington, DC, USA), pp. 541–555, IEEE Computer Society, 2013.

[73] "Evercookie." http://samy.pl/evercookie/.

[74] G. Acar, M. Juarez, N. Nikiforakis, C. Diaz, S. Gürses, F. Piessens, and B. Preneel, "Fpdetective: Dusting the web for fingerprinters," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer &#38; Communications Security*, CCS '13, (New York, NY, USA), pp. 1129–1140, ACM, 2013.

[75] "Anonymizer." https://www.anonymizer.com/.

[76] "Tor browser bundle." `https://www.torproject.org/projects/torbrowser.html.en`.

[77] "Privoxy." http://www.privoxy.org/.

[78] "Alexa." `http://www.alexa.com/topsites`.

[79] "Weka manual." `http://www.cs.waikato.ac.nz/ml/weka/documentation.html`.

VITA

Jiangmin Yu

Candidate for the Degree of

Doctor of Philosophy

Thesis: APPLICATION OF DATA ANALYTICS — CASE STUDIES

Major Field: Computer Science

Biographical:

Education:

Completed the requirements for the Doctor of Philosophy in Computer Science at Oklahoma State University, Stillwater, Oklahoma in May, 2017.

Completed the requirements for the Master of Science in Computer Science at Wuhan University of Technology, Wuhan, China in 2006.

Completed the requirements for the Bachelor of Science in Material Science and Engineering at Wuhan University of Technology, Wuhan, China in 2004.

Experience: I graduated from Wuhan University of Technology in 2006. After that, I worked as a software engineer in Shanghai China for about 6 years. In 2012, I joined the Ph.D. program at Oklahoma State University.

Professional Memberships: