NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

USER IDENTIFICATION IN DYNAMIC WEB TRAFFIC VIA DEEP TEMPORAL FEATURES

by

Jihye Kim

March 2021

Thesis Advisor: Vinnie Monaco
Second Reader: John D. Fulp

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** March 2021	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE USER IDENTIFICATION IN DYNAMIC WEB TRAFFIC VIA DEEP TEMPORAL FEATURES	**5. FUNDING NUMBERS**
6. AUTHOR(S) Jihye Kim	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

Web applications that process sensitive information have become prevalent. Modern web applications rely heavily on dynamic content (i.e., page updates made by the browser using an XMLHttpRequest, and more recently the JavaScript Fetch API). Ajax technology provides fast client-server communication, which generates web traffic that updates the document object model (DOM) object in the browser interface often induced by user input. Therefore, the user's actions are strongly correlated with timing and size of packets that carry Ajax requests. This research aims to characterize the relationship between keystroke dynamics and Ajax packets in dynamic web traffic. We investigate several dynamic web applications and the ability to measure human behavior in encrypted network traffic. Two approaches to Ajax packet detection are proposed and evaluated: longest increasing subsequence (LIS), which uses packet sizes, and dynamic time warping (DTW), which uses keystroke and packet timings. From the detected packets of recognized patterns, we examine the extent to which remote user identification in dynamic web traffic can be performed. We use a recurrent neural network (RNN) trained with triplet loss to extract deep temporal features from the detected packet timings. Leveraging recent work in keystroke dynamics, we show that user identification can be performed with modest accuracy utilizing the packet timings invoked by a user typing in a web search engine.

14. SUBJECT TERMS network traffic analysis, keystroke biometrics, dynamic web traffic, side channel attack, recurrent neural network, triplet loss, user identification	**15. NUMBER OF PAGES** 87
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

USER IDENTIFICATION IN DYNAMIC WEB TRAFFIC VIA DEEP TEMPORAL FEATURES

Jihye Kim
Captain, Republic of Korea Army
BAS, Korea Military Academy, 2013

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2021

Approved by: Vinnie Monaco
Advisor

John D. Fulp
Second Reader

Gurminder Singh
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Web applications that process sensitive information have become prevalent. Modern web applications rely heavily on dynamic content (i.e., page updates made by the browser using an XMLHttpRequest, and more recently the JavaScript Fetch API). Ajax technology provides fast client-server communication, which generates web traffic that updates the document object model (DOM) object in the browser interface often induced by user input. Therefore, the user's actions are strongly correlated with timing and size of packets that carry Ajax requests. This research aims to characterize the relationship between keystroke dynamics and Ajax packets in dynamic web traffic. We investigate several dynamic web applications and the ability to measure human behavior in encrypted network traffic. Two approaches to Ajax packet detection are proposed and evaluated: longest increasing subsequence (LIS), which uses packet sizes, and dynamic time warping (DTW), which uses keystroke and packet timings. From the detected packets of recognized patterns, we examine the extent to which remote user identification in dynamic web traffic can be performed. We use a recurrent neural network (RNN) trained with triplet loss to extract deep temporal features from the detected packet timings. Leveraging recent work in keystroke dynamics, we show that user identification can be performed with modest accuracy utilizing the packet timings invoked by a user typing in a web search engine.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

1 Introduction **1**
1.1 Research Questions 3
1.2 Contributions 3
1.3 Thesis Organization 4

2 Background and Related Work **5**
2.1 Dynamic Web Traffic 5
2.2 Remote Side-Channel Attacks 9
2.3 Keystroke Dynamics 15

3 Design and Methodology **19**
3.1 Overview 19
3.2 Threat Model 20
3.3 Data Collection 21
3.4 Ajax Packet Detection 23

4 User Identification **31**
4.1 Feature Extraction 31
4.2 Recurrent Architectures 33
4.3 Triplet Training 36
4.4 Classification 37
4.5 Summary 39

5 Experimental Results **41**
5.1 Event Packet Detection 41
5.2 User Identification and Verification 48

6 Conclusion **55**
6.1 Summary 55

6.2 Key Takeaways 55
6.3 Discussion 57
6.4 Conclusion and Future Work 58

List of References **61**

Initial Distribution List **69**

List of Figures

Figure 2.1 Architecture of dynamic web application and threat model 6

Figure 2.2 Flow of callback and polling model 8

Figure 2.3 Classification of side-channel attacks 10

Figure 3.1 Overview of methodology 19

Figure 3.2 HTTP/2 multiplexing traffic 21

Figure 3.3 The capture process 22

Figure 3.4 Head of CSV with 2-Tuple 23

Figure 3.5 Definition of false positive rate (FPR) and false negative rate (FNR) for a detection system 25

Figure 3.6 An example of Ajax packet detection by using LIS algorithm . . 26

Figure 3.7 A procedure to detect Ajax packets using dynamic time warping (DTW) alignment 27

Figure 3.8 An example of Ajax packet detection by the DTW approach from one of the evaluation datasets: (a) an optimal warping path between packets and keystrokes, (b) a linear graph with (packet timing-latency) and keypress timing, (c) a distribution of difference between packet and keypress timing, (d) a distribution of difference between packet and keypress timing (histogram) 29

Figure 4.1 Histogram of time interval distribution: original values of τ (up) and the result after tokenization at b = 5ms (down) 32

Figure 4.2 Different recurrent architectures: (a) recurrent neural network (RNN), (b) long short term memory (LSTM), (c) Bidirectional LSTM, (d) gated recurrent unit (GRU). Adapted from [51], [52]. . 34

Figure 4.3 The triplet loss function: to minimize the distance of anchor-positive and to maximize the distance of anchor-negative. Adapted from [14]. 37

Figure 4.4 (a) equal error rate (EER), (b) area under the ROC curve (AUC)-receiver operating characteristic (ROC) curve (area under the receiver operating characteristics (AUROC)) 38

Figure 5.1 Arduino key generation code 42

Figure 5.2 Website behavior: packet time interval distribution at DuckDuckGo (left) and packet size distribution at GoogleDocs (right) 43

Figure 5.3 Analysis of web traffic: a packet rate per a second (up) and the result of Ajax packet detection (down) 45

Figure 5.4 An example of Ajax packet pattern in Google Search 46

Figure 5.5 Query scenarios in diverse user behavior: general behavior(user1) vs fast typing with error correction(user2) 46

Figure 5.6 Identification accuracy vs sample length (left) and detection rate (right) 50

Figure 5.7 Identification accuracy (Rank-1, 5, 50) as the number of users (test) is scaled up (100-500) 51

Figure 5.8 Comparison of verification error rate: (a) EER, (b) EER of all timings, (c) EER of packet timings on the host, (d) EER of packet timings detected by our method 52

Figure 5.9 Verification accuracy (left) and EER (right), as the number of users is scaled up (100-500) 53

Figure 5.10 Verification accuracy vs sample length (left) and detection rate (right) 53

Figure 5.11 AUC-ROC Curves 54

Figure 6.1 An example of keystroke and packet time intervals of subgraph matching problem 59

List of Tables

Table 2.1 Comparison of callback and polling model 9

Table 3.1 Data collection overview 23

Table 4.1 LSTM network architecture 35

Table 5.1 Summary of website behavior (1) 43

Table 5.2 Summary of website behavior (2) 44

Table 5.3 Statistics of Ajax packet detection 47

Table 5.4 Summary of Ajax packet detection performance 48

Table 5.5 Factors used to evaluate metrics 49

Table 5.6 Identification accuracy (Rank-1, 5, 50) and Loss as the number of users (train) is scaled up (10k-170k) 49

Table 5.7 Summary of user identification (rank-N classification accuracy) and verification (accuracy) performances. 54

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

Ajax	asynchronous JavaScript and XML
API	application programming interface
AUC	area under the ROC curve
AUROC	area under the receiver operating characteristics
CSV	comma separated values
DOM	document object model
DTW	dynamic time warping
ECDSA	elliptic curve digital signature algorithm
EER	equal error rate
FNR	false negative rate
FPR	false positive rate
GPU	graphics processing unit
GRU	gated recurrent unit
GUI	graphical user interface
GS	Google Suggestions
HMM	hidden Markov model
HTML	hypertext markup language
HTTP	hypertext transfer protocol
HTTPS	HTTP over TLS

ICMP internet control message protocol

KNN K nearest neighbors

LIS longest increasing subsequence

LSTM long short term memory

LTS long term support

PIN personal identification number

RNN recurrent neural network

ROC receiver operating characteristic

RSA Rivest Shamir Adleman

SaaS software-as-a-service

SSH secure shell

TCP transmission control protocol

TLS transport layer security

TNR true negative rate

TPR true positive rate

UTC coordinated universal time

VoIP voice over internet protocol

XHR XMLHttpRequest

Acknowledgments

First, I would like to express the deepest gratitude to my thesis advisor, Dr. Vinnie Monaco. It has been my greatest luck to meet such a superb professor here, starting with his AI class. I sincerely thank him for his insightful advice, continuous enduring encouragement, and for sharing his great works and precious materials with me. He actually has shown himself as the real expert that I have vaguely wanted to be.

I am also very grateful to the second reader, John D. Fulp, for his invaluable teaching at his classes, and helpful comments for this thesis. His lectures were very clear and perfectly presented, enabling me to build a much stronger background in areas related to this thesis, as well as general network security.

I want to thank both the Republic of Korea Ministry of National Defense and the Republic of Korea Army for giving me this precious chance to study at the Naval Postgraduate School. In particular, there have been a lot of obstacles due to the COVID pandemic, but I'd like to thank both organizations for helping me to finish my education as planned. Also, I am thankful to the professors in the Department of Computer Engineering at Korea National Defense University for their continuous encouragement.

Lastly, I sincerely appreciate my family, friends, and colleagues in Korea, who are always giving me their full trust and support. Going forward, I will always do my best on my duty and try to be a better person.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

Privacy is a major concern of users while using the internet. User tracking and disclosure of sensitive information are some of the threats that internet users face. Internet privacy is the individual's right to selectively control one's own information on the internet, which is based on the definition of privacy as "a right to be let alone" [1]. To protect users' privacy on the internet, many measures have been taken, such as using encrypted networks, anonymizing proxies, and multi-factor authentication. Nevertheless, internet privacy issues are increasing due to the nature of the web, which is inherently open, dynamic, and prone to information leakage [2].

Web applications that process sensitive information have become prevalent. Modern web applications rely heavily on dynamic content (i.e., page updates made by the browser using an XMLHttpRequest, and more recently the JavaScript Fetch application programming interface (API)). To enhance user experience, most dynamic web services use the asynchronous JavaScript and XML (Ajax) technology, which provides fast client-server communication that generates web traffic even with only a small input from users. Ajax is one of the prominent technologies being used to develop rich and more interactive web applications. However, Ajax also amplifies the possible attacks in a web environment [3] since more frequent interaction can allow unintended information leakage, for example by revealing search engine queries. A key feature of Ajax is that it enables browsers to retrieve data from a server more intelligently, not reloading the entire page but downloading only the updated parts of a page, without disturbing the user's activity. Therefore, the user's actions are strongly correlated with the timing and size of packets that carry even small events.

Moreover, encrypted web traffic does not eliminate the risk of this information leakage; a surprisingly substantial amount of information may be revealed since user inputs are exposed primarily through packet timing and size side-channels. A remote adversary can take advantage of this kind of unintended input-based side-channel leakage in diverse ways. This leads to serious security challenges such as device/website fingerprinting [4]–[7], user identification [8], content extraction [9], and session hijacking [10]. Some of these issues arise from the use of Ajax technology [3].

Autocomplete is one of the widely used features that relies on Ajax technology. Many websites that accept text input within a form provide an "autocomplete" feature that suggests a pre-populated list of values based on previous aggregate user inputs and the content of web pages [11]. Most web search engines carry autocomplete graphical user interface (GUI) widgets that produce traffic replying to a single mouse click or a keyboard input. This traffic contains features that may reveal each state transition within the application [12]. As a result, an attacker can infer sensitive user inputs or server responses based on the time or size properties of the Ajax requests on web search engines.

In this thesis, we study how the web page processes user input and invokes the requests. Each web application has various unique web-flow vectors depending on its web API and event processing model. The potential for information leakage through web traffic depends on each website. While surveying potential candidates for this study, we chose the most widely used search engine, Google Search, which provides the autocomplete feature. Based on our analysis of website behavior, Google leaks relatively more information through Ajax web traffic compared to other search engines.

Next, we consider what information can be found through the patterns of Ajax triggered by user inputs. Among the user inputs, we use keystroke dynamics as a form of behavior biometrics since it is regarded as one of the most efficient and economical means of user identification [13]. Previous works on keystroke dynamics have mostly been conducted by temporal features obtained on the host. We expand it to a remote scenario in which a passive adversary is observing encrypted web traffic. An on-path remote passive adversary can extract Ajax packets that are induced by a user typing in a search query box in a web search engine. We examine the extent to which remote user identification can be achieved from observing dynamic web traffic.

Diverse machine learning models have been utilized for user identification. In our study, recurrent neural network (RNN) is suitable because it fits well with temporal data (Ajax packet timings) and can accommodate inputs of various lengths. By leveraging recent work in keystroke dynamics, we use an RNN trained with triplet loss, a method that learns to rank distances between samples belonging to the same or different classes [14].

1.1 Research Questions

Considering the characteristics of Ajax-based web applications, the research questions that this thesis aims to address are as follows:

- How do the dynamic web services process user input and invoke the requests?
- Is there any way to detect user input events over encrypted web traffic? How accurate is the detection?
- How can a remote adversary take advantage of unintended input-based side-channel leakages? How does this infringe on user privacy?

1.2 Contributions

This research aims to figure out the patterns between keystroke dynamics and Ajax packets generated from dynamic web traffic on the most widely used web search engine, Google, which provides the autocomplete feature. From the detected packets of recognized patterns, we examine the extent to which remote user identification can be achieved by observing dynamic web traffic. To summarize, our contributions are:

- We discuss the characteristics of modern dynamic web traffic and how these are often correlated with user behavior. Ajax-based web applications produce more responsive and faster web traffic even with smaller user inputs.
- We develop a method that can automatically detect Ajax packets in search engines and describe a general technique that can be used to match Ajax packets to user input events. We introduce two approaches that can detect packets without checking the payload: longest increasing subsequence (LIS) for packet sizes, and dynamic time warping (DTW), which is a new approach achieved by leveraging keystroke and packet timings.
- Using the detected packets, we perform both user identification and verification with up to 500 users. We utilize a RNN trained with triplet loss to extract deep temporal features from the detected packet timings.
- We survey several websites that are potentially vulnerable to this kind of attack. Through the analysis of several websites and an ablation study, we propose several mitigation measures that do not severely impact the performance of the website.

1.3 Thesis Organization

The rest of the thesis is organized as follows:

Chapter 2 reviews background and works related to our approach, including research on dynamic web traffic, remote side-channel attacks, and keystroke dynamics.

Chapter 3 describes our methodology for threat model, data collection and Ajax packet detection.

Chapter 4 discusses the user identification and verification part of our study in detail, including feature extraction, specification of a recurrent neural network, triplet training, and lastly, classification.

Chapter 5 provides the detailed experimental results achieved by using our approach.

Chapter 6 summarizes this work, and discusses experimental results, mitigation, and broad implications. Lastly, it concludes the thesis and identifies areas for future work.

CHAPTER 2:
Background and Related Work

Various kinds of side-channel attacks inevitably lead to privacy issues using the internet. Due to the recent increase in web use and corresponding increase in processing sensitive user data, side-channel attacks in the web industry have been getting more attention. User input events are captured almost in real-time and transmitted to enable the detection of user actions, which poses a particular threat to privacy in the web environment. Even event timestamps alone on encrypted traffic can leak a user's behavior patterns that may reveal the user's identity and application contents. This means that anonymity cannot be guaranteed because time information is leaked equally even through Tor, an overlay network designed to obfuscate user location [15], [16].

While interacting with web applications, users generate characteristic patterns of web traffic by their input events, which can be detected [17]. This interaction causes unintended information leakage, for example by revealing search engine queries, that can threaten privacy [9]. Threats have increased with the use of Ajax technology, which supports fast and responsive interactions. Sensitive information that can be exposed through web applications includes the identity of user, the device that the user used, and the contents that the user entered. This leads to serious security challenges that violate user privacy. We examine the feasibility of whether remote user identification from dynamic web traffic can be achieved by using such information leakage. This chapter reviews background information and works related to our approach in the areas of dynamic web traffic, remote side-channel attacks, and keystroke dynamics.

2.1 Dynamic Web Traffic

A web application is software that provides a service for users through a web browser. As the use of software-as-a-service (SaaS) has increased, "thin clients" have replaced many desktop applications used to process sensitive information in a wide range of areas. Web applications are divided into browser-side and server-side components. Modern web applications provide dynamic web services that can display different contents in accordance with the user's inputs and need, both client-side and server-side scripting, with frequent

communication. This interactive feature inevitably exposes web traffic flows, which makes it more vulnerable to side-channel attacks even when communicating over an encrypted network. The architecture of the dynamic web application and threat model considered in this work are shown in Figure 2.1. We assume that a remote passive adversary basically just sees the encrypted traffic but does not interact with it.

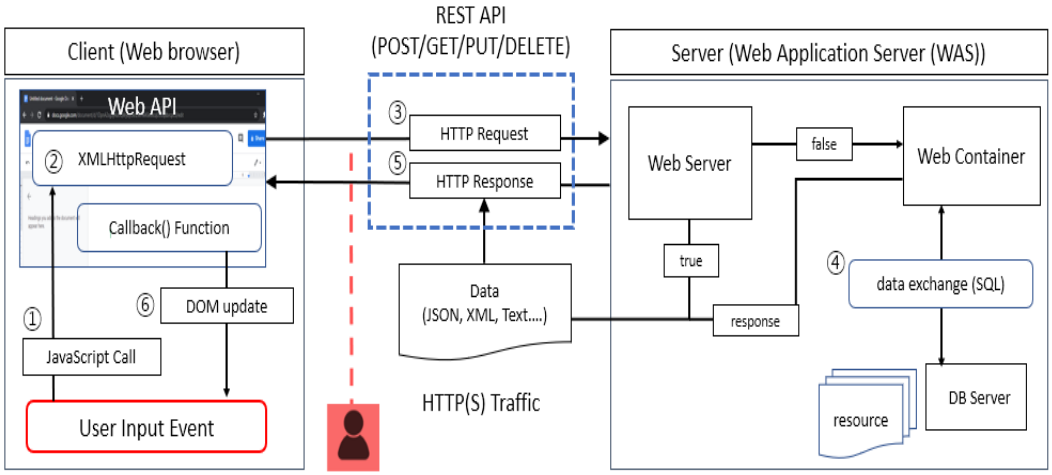

Figure 2.1. Architecture of dynamic web application and threat model

Web browsers are the primary tool through which users interact with web servers. Most modern web browsers (e.g., Google Chrome, Microsoft Edge, and Mozilla Firefox) provide JavaScript API that allow the browser to make hypertext transfer protocol (HTTP) requests to get or update page resources after a page has loaded. These web APIs are essential for websites that require asynchronous processing of dynamic content, such as mapping services, document editors, and search engines.

Web search engines provide an autocomplete feature that allows users to view recommended suggestions while typing search queries [17]. Partially completed search queries typed by the user on the client browser are used to predict what the user might search for, as suggestions are updated for each printable character entered [3]. Ajax provides a means of implementing asynchronous client-server communication to send the partially completed queries and retrieve the list of search suggestions. This web traffic carries the user input to the server and updates the document object model (DOM) on the client.

As depicted in Figure 2.1, the front-end JavaScript communicates with the back-end server by using HTTP or HTTP over TLS (HTTPS) with regards to user input events such as a keypress [17]. Even a user input event as small as one keypress generates web traffic and updates DOM objects in the browser interface through the API [12], which makes web applications more interactive. When the HTTP request is transmitted to the web application server, the server gets or updates data from the DB server and passes the result back to the user in response to the HTTP request.

Ajax broadly refers to the ability of a web page to make asynchronous updates. The API utilized and the way in which Ajax is implemented varies across websites. The most widely used Ajax APIs are XMLHttpRequest (XHR) objects and the more recent Fetch API. XHR is an event-based model that manages inputs, outputs, and states within an object. Events are generated separately upon each state change of a request. This approach differs from Promise-based asynchronous programming implemented by the Fetch API [18]. A Promise is an object embedded in JavaScript intended to simplify asynchronous communication compared to the callback functions in XHR. The Fetch API is optimized for HTTP through the use of three interfaces: headers, requests, and responses, which correspond to the core components of HTTP. Fetch, however, implements only a subset of XHR capabilities. Most recent browsers are compatible with both XHR and Fetch APIs.

Considering how Ajax requests are made in reply to user input events, and ultimately comparing keyboard event timings to packet timings, we should understand how the web page processes user input and invokes the requests. The event processing model represents the procedure by which the browser becomes aware of input events before making any request [19]. Event processing models are mainly divided into callback and polling. In a callback model, Ajax requests are made within some function registered as a callback to an input event, such as a keypress or keyrelease DOM event. In comparison, a polling model checks for new input at regular intervals and invokes an Ajax request when new input is detected. We investigated which event processing model is currently used in several search engines by setting breakpoints within the browser and tracing the path of keypress and keyrelease events. We found that Google uses a callback registered to keypress events, consistent with prior work and that DuckDuckGo also uses a callback registered to keypress events, which differs from the prior work in which a callback was registered to keyrelease events [19].

A callback model is event-driven, which means handling events immediately will induce the request without waiting for other events to finish. The callback process does not need to check any changes periodically and react to the events correspondingly. For example, a hypertext markup language (HTML) DOM keypress or keyrelease input event trigger HTTP requests to notify the changes. The delay between input event time and request time mainly relies on the running time of the callback function. If this running time is stable between consecutive events, the time intervals between events are reliably conserved with packet inter-arrival times [19]. The main benefit of using the callback model is that the application does not have to spend any process resources to monitor the events periodically, which means that it can work more efficiently.

In a polling model, the process monitors input events regularly at fixed time intervals, as depicted in Figure 2.2. It saves the contents periodically using HTTP request packets regardless of any detection of user inputs. Because the requests depend on each time cycle, the time delay from input events to the request packet is related to the timing of the event itself. Some multiple of the timer period closely aligns to the inter-arrival of packet timing [19]. The choice of polling interval is important in that it can buffer multiple events together, which would reduce network load if the time interval is optimal, whereas short intervals will cause inefficiency and put a heavy burden on network due to unnecessary traffic, resulting in delayed updates in the case of long intervals.

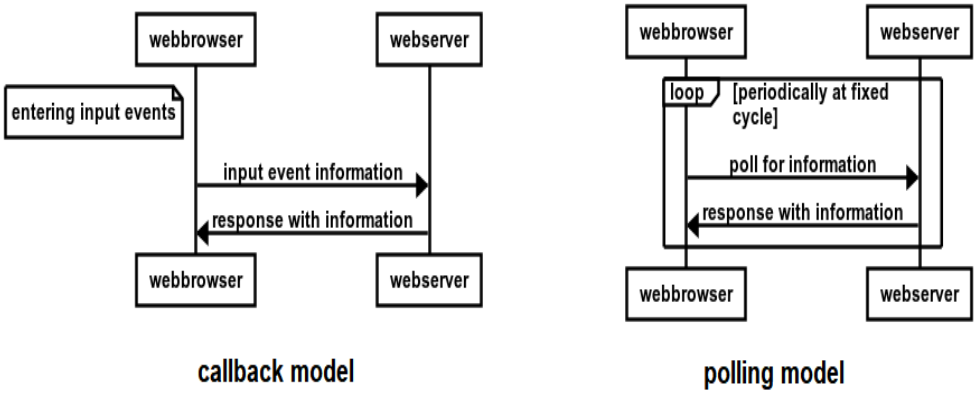

Figure 2.2. Flow of callback and polling model

Generally, if the application is in an environment that needs fast processing or performs just a single task, for example in gaming where real-time inputs are needed, the callback model is the better choice. In contrast, the polling model works well in the case of lower frequency actions such as email or message sending, file downloading, and controller input. Table 2.1 summarizes and compares the characteristics of the callback and polling models.

Table 2.1. Comparison of callback and polling model

Feature	Callback	Polling
number of requests	same as input events	less than/same
delay	execution time of callback function	polling rate
loop cycles	not required	required
environment	needs faster processing	comparatively slower
side-channel attacks	more vulnerable	less vulnerable

2.2 Remote Side-Channel Attacks

Over the last two decades, side-channel attacks have been widely studied in diverse contexts using different forms of information that accompany the processing (i.e., timing, power analysis, electromagnetic, and acoustic) to identify the vulnerabilities of the targets. It is difficult to determine how a system will behave at the time of design [20]. According to [21], attackers typically use patterns that arise from the design of programs used, user actions, and the interaction between the client and the server to carry out side-channel attacks. This means that certain patterns can characterize the side-channel information.

Figure 2.3 illustrates a taxonomy of side-channel attacks. We categorize remote side-channel attacks in terms of three classes: kinds of side-channel information leaks, attack targets, and threat models. In this subsection, we discuss remote side-channel attacks in web traffic in particular. The attacks in the other fields are discussed in the next subsection. In a network, side-channel information leaks are related to packet metadata, such as timing or sizes, and these can be exploited as attacks that threaten user privacy.

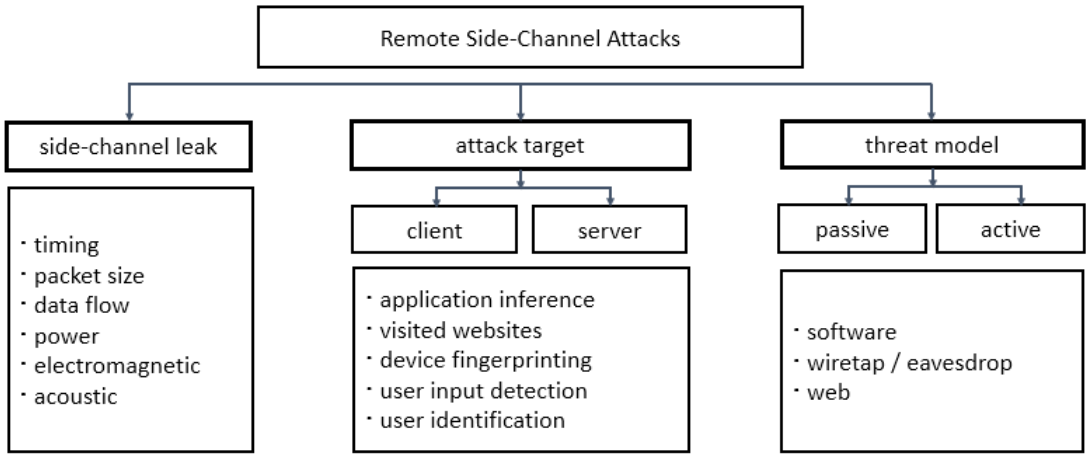

Figure 2.3. Classification of side-channel attacks

2.2.1 Side-Channel Attacks in Dynamic Web Traffic

In [19], Monaco examined the feasibility of keystroke timing attacks on several popular search engines. For the experiment, 1000 queries were collected on each of five different search engines (Google, DuckDuckGo, Bing, Yandex and Baidu) that implement autocomplete, and used two web browsers (Chrome, Firefox). Based on the collected dataset, the behavior of each website was characterized from the perspective of packet size, event processing model, censoring and information gain. Keystrokes were recognized by the pattern of increasing packet sizes, and the time intervals between keystrokes were well conserved in the packet inter-arrival times on some of the search engines examined [19]. Hence, the vulnerability of websites that generate dynamic traffic varies significantly based on the accuracy of Ajax packet detection.

Based on the results of prior work, Monaco [9] developed an attack model called KREEP (Keystroke Recognition and Entropy Elimination Program) to demonstrate a client-side attack on search engines and discuss countermeasures. KREEP consists of five stages: keystroke detection, tokenization, dictionary pruning, word identification, and beam search. This attack only focuses on HTTP request autocomplete traffic, which means it is client independent, and uses the differences of packet size and packet inter-arrival times. The datasets for the experiment were collected in both Chrome and Firefox, and consisted of a total of 16,000 queries: 4,000 queries x 2 search engines (Google, Baidu) x 2 web

browsers. As a result, keystrokes detection was performed with nearly perfect accuracy in both websites and browsers. Lastly, network padding, generating dummy traffic, and merging requests were proposed for the countermeasures against the attack.

Chen et al. [12] reviewed the status and direction of side-channel information leaks in web applications both server-side and client-side. The root causes of many side-channel vulnerabilities are based on the fundamental design features of web applications: frequent small interactions, stateful communication, and diversity in the contents exchanged during state transitions. The researchers examined actual information leaks by constructing an attack on several high-profile web applications, including health, tax, investment, and search engines. As an example, when they used the functionality "add health records" in OnlineHealth, the tab design revealed the record type and each tab click created a web flow. Moreover, because of the auto-suggestion widget, the user's actual inputs could be distinguishable by attackers who identify the packet sizes that match the response of the suggestion lists. Unique "web flow vectors" were obtained by clicking through different page elements. Thus, the user's actions can be reconstructed by attackers who identify the packet sizes that match server responses, which may carry suggestion lists and other page updates [12]. While this approach detects Ajax events from traffic emitted by the server, our approach considers only traffic emitted by the client.

After this research, in response to the need for countermeasures, Chen et al. [22] proposed the automatic tool called "Sidebuster" for automated detection and quantitation of side-channel leaks in web applications. This technique is mainly conducted on the server-side, and needs two steps: First, analyze the information flows of web applications to detect the interactions induced by sensitive user data between client and server; second, conduct the dynamic examination on the network to determine the degree of information leaks being revealed. They surveyed six applications' interactions between client and server to detect user information. As a result, they found that each application had idiosyncratic side-channels. The health profile program included multiple widgets that leak information by interactions, and the tax program had side-channels in its control flows. Some applications exposed user actions and sensitive information through the traffic generated by interactions (e.g., a link selection or a button click) [22]. This research is meaningful in that it proposed the first automatic tool that can detect and quantify side-channel leaks in web application development.

As such, it is certain that server-side attacks are being actively conducted. Meng et al. [17] inspected the feasibility of recovering personalized keystroke timing information by using Google Suggestions (GS), which is one example of an interactive rich JavaScript application. They analyzed the timing side-channel of GS by reverse-engineering the communication model, which is made by the obfuscated JavaScript code. In their experiment, 11 participants installed a plugin on their browser to capture keystroke timings of queries and their keystrokes were collected. As a result, the average of the inter-keystroke timing was decided with a less than 20% error rate for each key pair with at least 20 samples. This result suggests that typing patterns can be reconstructed by the timing of the queries over the network [17]. In our work, we demonstrate that typing speed can be recovered with much higher accuracy on modern search engines, and that modest user identification accuracy is achieved with up to 500 users.

In another study, Chapman et al. [23] developed a black-box crawling system that can log web traffic on the web application, which mimics the interactions of actual users using standard web browsers. This system produces both web states and user behaviors with generated web traffic. They tested on several existing web applications, including search engines, the United Kingdom's National Health Service Symptom Checker, and Google Health to evaluate their approach. The researchers created crawl specification files and executed the crawlers on each application in a variety of hardware. When the crawlers finished the examination of the web application, the researchers quantified the leaks as a multi-class classification problem. As a result, the best accuracy rate was 96.3% on the Bing Suggestions data set, which means they successfully identified the queries [23]. Thus, the researchers demonstrated the automated detection of side-channel leaks by developing their own web application crawling system.

Recently, Goetherm et al. [24] introduced a new kind of timing attack that infers the multiplexing of network protocols and the concurrent handling of requests by applications. Instead of depending on sequential timing, the concurrency-based timing attacks exploit the relative difference of timing between two requests that are currently carried out. The attack was applied in diverse areas such as web applications over HTTP/2 or Wi-Fi authentication, and on a Tor network. For web-based timing attacks, they used the nghttp2 C library and the TCP_CORK option to ensure that a single TCP segment contained two TLS records. After that, they created sets of request pairs to evaluate the attack performance. In conclusion,

they proved that the concurrency-based timing attacks not only outperform the original timing attacks but they are being more resilient to diverse network conditions [24]. By using concurrency, this research proposed a more powerful remote timing attack that is not subjected to network jitter.

2.2.2 Other Remote Side-Channel Attacks

Aside from the ones already mentioned, a number of remote side-channel attacks have been actively studied, in the context of cryptosystem or protocol implementation [25], [26], secure shell (SSH) [27], voice over internet protocol (VoIP) [28], Skype [29], cloud domain [30] and others. The research has demonstrated the practicality of side-channel attacks in a variety of areas.

Brumley et al. [25] showed that the client is able to extract a Rivest Shamir Adleman (RSA) private key stored on the server not only in a local network but also on virtual machines, by measuring the time needed for an OpenSSL server to respond to the queries. The researchers identified a vulnerability of timing attack in the implementation of OpenSSL and expanded it to a key recovery attack of a transport layer security (TLS) server that authenticates with ECDSA signatures [26]. In the acoustic emanations area, Asonov et al. [31] demonstrated attacks based on emanations produced by the PC keyboard and the clicks. By leveraging the difference between sounds, the researchers used a neural network to distinguish the keys.

Song et al. [27] studied timing attacks on SSH, which is designed to provide a secure channel. Taking advantage of the fact that encrypted SSH packets leak the payload size and inter-keystroke timing information, the researchers recovered users' typing patterns from the network traffic and developed a hidden Markov model (HMM) to predict key sequences from the inter-packet timings. The tool developed as part of this attack, Herbivore, first demonstrated the possibility of remote keystroke timing attacks with network traffic.

White et al. [28] unveiled privacy threats against VoIP communications by segmenting the sequence of observed packet sizes and analyzed the extent of the adversary's ability to reconstruct parts of VoIP conversations. First, the study found phoneme boundaries from the sequence of packet sizes and classified them by using their profile HMM model. After identifying word boundaries from the flow of categorized phonemes, the researchers translated the subsequences of phonemes into English words. The result generally represented

adequately understandable translations with respect to machine translation.

Expanded from VoIP research, Benoit Dupasqueir et al. [29] investigated information leakage in detail from Skype, a widely used VoIP application. They compared time series by adopting DTW and Kalman filter algorithms to encrypted Skype traces in an ideally noise-free environment. The accuracy was able to reach from 60% to 83%, which demonstrates that the privacy provided by Skype is not guaranteed.

Fingerprinting is a technique to identify a device, operating system, or web browser by using a certain attribute of the device. By calculating clock skew in device hardware, Kohno et al. [4] studied the remote physical device fingerprinting area without the fingerprinted device's known cooperation. They mostly used a transmission control protocol (TCP) timestamp-based approach because most systems use TSopt clocks that operate at lower frequencies than the internet control message protocol (ICMP) replay messages, which include 1,000 Hz clocks. For fingerprinting to work, two fundamental properties are needed: each different device has a measurably different clock skew and clock skews are relatively constant over time [4]. Overall, the research of Kohno et al. [4] proved that remote physical device fingerprinting through timestamps is feasible with various methods.

Other work examined a remote website fingerprinting attack to prove the feasibility of using remote traffic analysis to learn sensitive information. By using the ordered packet size sequences and DTW distance metric, Gong et al. [32] developed a fingerprint detection technique. To evaluate the detection technique, they made a training dataset of fingerprint samples from the top 1000 US sites by Alexa while monitoring the victim's traffic. Most of the results showed high accuracy with low false-positive rate (maximum:0.5%), but several sites were invulnerable to attack. The result also revealed sufficient detection ability in the deanonymization attack. There is a limit, however, in that the study implemented a stable fingerprint only when using the front page of the website, leaving room for future work for dynamic web application traffic.

2.2.3 Countermeasures

Meanwhile, many studies have been conducted to develop countermeasures to mitigate these unintended leakages and enhance privacy. Countermeasures can be largely divided into two types: obfuscation of traffic (e.g., padding, mimicking, and morphing) and obfuscation of

user inputs against identification exposure over encrypted traffic. HTTPOS, which stands for HTTP/HTTPS with obfuscation, is a browser-side method proposed in [33] to prevent the sensitive information leaks from web applications that use TCP and HTTP. This method obfuscates encrypted traffic by modifying packet size, timing, flow size, and web object size, which is effective for mitigating the existing attacks.

Similarly, a technique for obfuscating keystroke time intervals at the keystroke event level was proposed in [34]. The researchers used two different methods of time interval mixing methods to add noise to the time intervals while arriving: a delay mix and an interval mix. A delay mix adds a random delay to the event time while an interval mix adds a random delay to the time interval from the prior event. Both ways were tested by using publicly available keystroke datasets. They could get the results that led to about a 20 to 50% reduction of identification accuracy, but a much longer time lag was needed for the lower ability to identify users. In conclusion, the result in this work suggests that it is feasible to obfuscate keystroke behavior with a delay without the user noticing, which provides a direction for a more robust defense mechanism. As recent related work implies, each vulnerability and mitigation measure related to side-channel attacks is generally specific to individual application functionality. Thus, it is worthwhile to investigate side-channel leaks in dynamic web applications that provide diverse functions such as an autocomplete, remote document editing, chat applications, and mapping services.

2.3 Keystroke Dynamics

As a form of behavioral biometrics, keystroke dynamics has been regarded as one of the most efficient and economical means of user identification [13]. Keystroke dynamics capture typing behaviors believed to be distinctive to an individual and hard to duplicate [35]. However, the accuracy of keystroke biometric systems struggles to compete with traditional biometrics, such as face and fingerprint. Only recently have keystroke biometric systems been able to scale up to thousands of users and approach the accuracy needed for wide usage deployment [36].

We can obtain mainly two kinds of information from keystroke dynamics: temporal and spatial. Temporal information leverages the sequence of keystroke timings such as keypress (t_p) and keyrelease (t_r) [37]. Such information uses either temporal features of individual

keys or time intervals between key pairs [37]. On the other hand, the spatial information exposes exactly which key was entered. This study aims to identify rather than merely detect keys used. In this work, our method of user identification is based on the temporal features of users' typing patterns as seen through Ajax request packets. Idiosyncratic typing behaviors, including the temporal dynamics observed through the keyboard, are used to distinguish between users.

Keystroke biometric systems are also largely categorized as either fixed-text or free-text. Fixed-text systems are those in which a pre-defined string is typed, such as a username, password, or personal identification number (PIN). In comparison, free-text systems are designed to identify or verify users typing any text. Prior work on free-text systems utilize timing features based on the interval between keypress and release events (i.e., the duration a key is held down), as well as the latency between consecutive keypresses (i.e., the flight time between successive keystrokes). In many works, these features are conditioned either on particular keys or key groups [13], [36], [38].

Tappert et al. [35] examined the long-text input for keystroke biometric system as a way of identification and authentication. A K nearest neighbors (KNN) classifier with Euclidean distance was employed for both identification and authentication and achieved sufficient accuracy. The researchers utilized 239 feature measurements, which characterized a user's keypress duration times, transition times and typing speed, among other factors. They could extract diverse features from the keystroke dynamics. They also figured out that reasonably accurate user identification can be obtained from the input of 300 keystrokes [35]. In our work, we scaled the size of each sample (both training and testing) to 300 Ajax packets, which corresponds to 300 keystrokes.

Acien et al. [36] developed a user authentication system, TypeNet, for free-text keystrokes based on a Siamese RNN architecture. This approach was effective when scaling up to 100,000 users, achieving close to a 5% error rate on average. It outperformed previous state-of-the-art algorithms and approaches the performance of fixed-text systems. We leverage a variant of the TypeNet model developed in [36] and show that user identification can be performed by utilizing the packet timings induced by a user typing into a search engine. We extend the work of [36] by discretizing keypress time intervals through rounding to reduce noise introduced by packet jitter, introducing an embedding layer in the model, and training

the model using a triplet loss.

Recently, Whiskerd et al. [39] examined the ability to utilize keystroke biometrics on web search engine traffic from desktop computers and mobile devices. They conducted the experiment in two scenarios: one on encrypted network traffic by using packet metadata, and the other with decrypted traffic through an HTTP proxy (mitmproxy). The researchers used a short fixed-text dataset consisting of 7-30 keystrokes per sample and leveraged only packet timing for user identification. The test cases showed that it is feasible to distinguish users in a small group by using conventional machine learning classifiers (naive Bayes, KNN). The identification performance resulted in error rates of 5-24% depending on the scenario [39]. Compared to this, we leverage a much larger group of 500 users and up to 300 keystrokes of free-text aggregated over several search queries that occur within the same TCP connection.

Our approach differs from the previous works in three important ways. 1) We utilize only the timing of Ajax packets, which correspond to keypress timings; therefore, duration features cannot be taken because keyrelease timings are not available. 2) The key name is not known, and the only information available is the sequence of Ajax packet timings; therefore, timings cannot be conditioned on different keys. This scenario is most similar to [40], which uses only a sequence of keypress time intervals to identify users. 3) Our sequence-based feature extraction model can be leveraged for both user identification and verification scenarios.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:
Design and Methodology

3.1 Overview

This chapter introduces our methodology and the measurement setup for extracting user behavior over the encrypted network traffic. This research has two goals: 1) to figure out the patterns between keystroke dynamics and Ajax packets from dynamic web traffic; 2) to examine the extent to which remote user identification in dynamic web traffic can be performed from the detected packets of recognized patterns. We consider Google Search, the most widely used web search engine that provides the autocomplete feature, for the experiment.

We first build a system that can replay pre-recorded keystrokes with precise timing and capture the corresponding keystrokes and packets. We show that the Ajax request packets can be distinguished from other background traffic. The adversary can monitor and analyze this flow remotely from the victim accessing a website with dynamic content while the system we built replays and records keystrokes. Then, we develop and train an RNN that extracts a vector of deep temporal features from a sequence of timestamps.

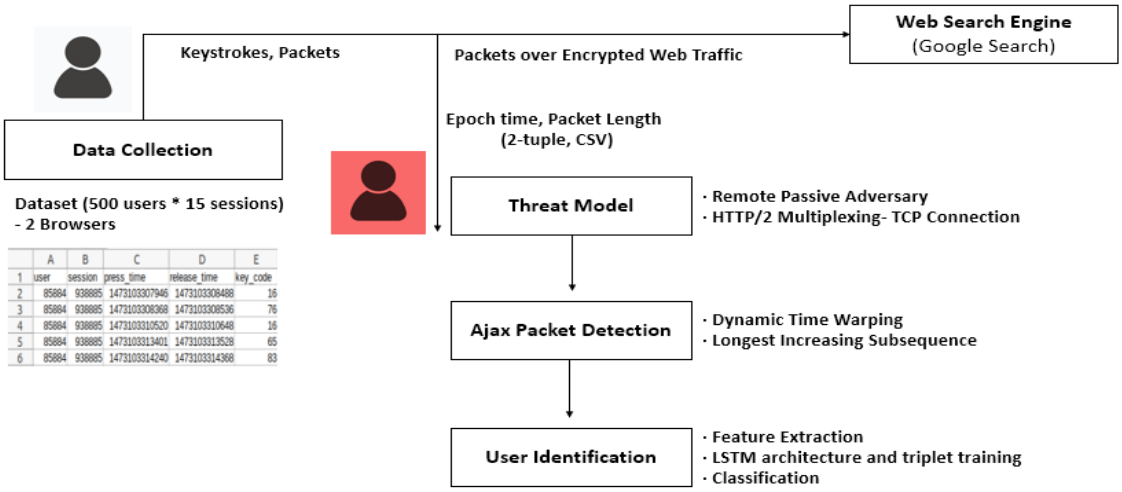

Figure 3.1. Overview of methodology

The overview of methodology is illustrated in Figure 3.1., which is divided into four parts: threat model, data collection, Ajax packet detection, and user identification. We explain each part in more detail in this chapter. User identification is discussed in detail in the next chapter.

3.2 Threat Model

We assume a remote passive adversary that can eavesdrop on encrypted network traffic from the victim accessing a website with dynamic content. The traffic includes packets emitted by the victim typing search queries into GS as well as other background traffic. We assume that the TCP connection over which Ajax requests are sent is maintained over the course of several search queries, which we verified to be the case for all major web browsers. The adversary can detect the Ajax packets by inferring the patterns of the packet size through the observed TLS traffic.

Detecting Ajax packets over a TCP connection rather than individual page loads enables an attacker to acquire keypress timings that potentially span multiple search queries. This obviates the need to detect individual page loads, which is difficult in practice [7], [41]. Therefore, we assume that the victim does not close and reopen the browser while entering several consecutive search queries. In this case, the browser will utilize the same TCP connection based on the HTTP persistent connection, which permits multiple requests and responses to be sent to over a single TCP connection instead of creating a new connection for each request.

The concept of HTTP/2 multiplexing is shown in Figure 3.2. The HTTP/2 protocol allows multiple concurrent requests and responses to be multiplexed within a single TCP connection [42]. HTTP/2 multiplexing is supported by most modern browsers, including Google Chrome, which we used for the experiment. The adversary attempts to identify users by first detecting Ajax packets within a TCP connection and then extracting a vector of temporal features from the Ajax packet timings. These features are relatively unique to individual users, enabling the observer to perform user identification (i.e., link the identities of two different TCP connections).

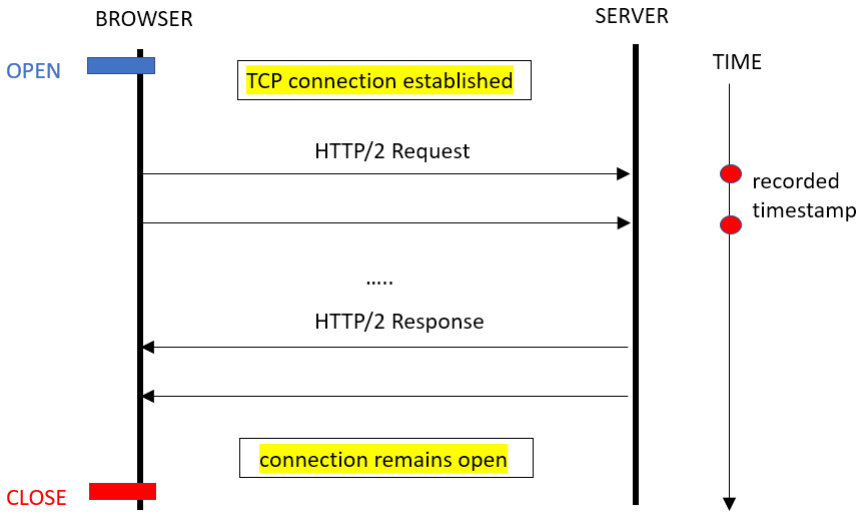

Figure 3.2. HTTP/2 multiplexing traffic

3.3 Data Collection

We used a subset of a large-scale dataset of typing behavior containing over 136 million keystrokes that were collected from 168,000 users observed by Aalto University during three months [43]. The size of the Aalto dataset is approximately 5GB, and the sentences typed by users ranged from 15 to about 150 keystrokes, including Space and Backspace. The sentences were randomly chosen from the Enron mobile email corpus and English Gigaword Newswire corpus. Each user typed 15 sentences totaling 810 characters on average. The dataset contains a wide variety of typing speeds, ranging from 1.5 to 22 keystrokes per second. The dataset was saved in comma separated values (CSV) file format with five columns: user, session, press time, release time, and keycode.

From this dataset, we randomly chose 500 users with 15 sessions and at least 600 keystrokes total. This subset of keystrokes is replayed in real time using a setup that mimics a user typing a search query in a browser. Data from the remaining users with at least 600 keystrokes each are used to train a model that extracts deep temporal features from a sequence of timestamps. This model is then applied to the 500 independent users. Note that we excluded function keys, such as Ctrl and Alt, from the dataset since these typically do not result in Ajax requests and can unintentionally invoke keyboard shortcuts.

The process for data collection is summarized in Figure 3.3. Data collection was performed over approximately five days on an Intel NUC10FNK running Ubuntu 20.04 long term support (LTS). We replayed keystrokes from the 500 users dataset by emitting keypress and keyrelease events in real time through the uinput module [44]. The uinput module is a kernel module that enables the creation of virtual devices and emulation of device inputs from the user space by writing to the `/dev/uinput` device. These events trigger interrupts as if they were generated by an actual keyboard and propagate to the application with the input focus, which is a web browser in our case. We used Selenium Driver for the automatic control of both Chrome(v.87) and Firefox(v.84). For each session, the web browser was opened and `https://www.google.com` was loaded by the browser through Selenium Driver. After loading the web page, the network capture was started by running TShark in the background. Tshark is a command-line interface capable of the same functions achievable with Wireshark and used for capturing packets while replaying keystrokes. Both packets and keystrokes used the epoch time with millisecond resolution, which is the standardized time that reflects the number of seconds that have passed since 00:00:00 coordinated universal time (UTC) on 1 January 1970.

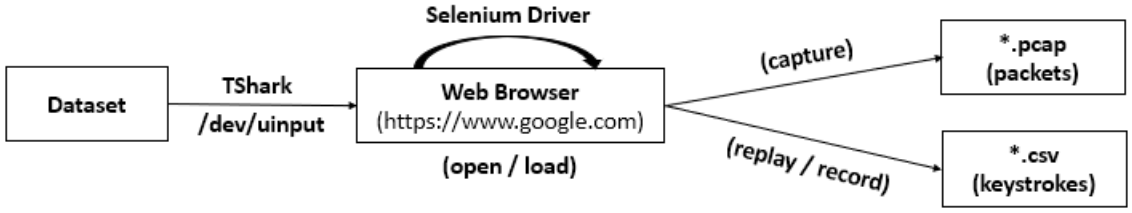

Figure 3.3. The capture process

TShark created *.pcap files which included background traffic in addition to the Ajax packets generated during the capture. A two-second delay occurred before replaying keystrokes, and each session was saved as a separate pcap file after finishing the replay. Packet timestamps in epoch format and TCP segment lengths were obtained from each pcap file. The TCP segment lengths were used for Ajax packet detection and the timestamps for user identification. Because we are interested only in upstream traffic, we filtered based on the source client IP address. Without loss of generality, additional background traffic is omitted, including TCP ACKs and HTTP/2 PING packets, both of which have a distinct segment length. In total,

we recorded traffic from 15,000 search queries (500 users x 15 sessions x 2 browsers). An overview of data collection is described in Table 3.1.

Table 3.1. Data collection overview

Subjects	Contents
Data collection duration	5 days
Number of users	500
Number of sessions	15
Number of browsers	2
Average number of keystrokes per user	710
Average number of keystrokes per session	48

The captured pcap was converted to CSV format to compare with the keystrokes. We used a Python script with the TShark command for conversion. We extracted only epoch time and packet size that were needed for Ajax packet detection and saved it as 2-tuple. Figure 3.4 is an example of a CSV file of extracted features. The features were chosen because user inputs are exposed primarily through packet timing and size side-channels.

	time	length
0	1.609800e+12	206
1	1.609800e+12	213
2	1.609800e+12	215
3	1.609800e+12	216
4	1.609800e+12	216

Figure 3.4. Head of CSV with 2-Tuple

3.4 Ajax Packet Detection

Ajax packet detection is performed over an entire TCP session, which enables aggregating keystrokes from multiple search queries. We use the approach of detecting the Ajax packets through the patterns of packet sizes: if the packet size increases cumulatively according to the user's typing on the website, the Ajax packets can be detected by finding the LIS of

packet sizes within the TCP connection since background traffic generally does not exhibit this behavior. This approach achieves near-perfect Ajax packet detection accuracy (F-score > 0.99) without the use of Delete or Backspace keys [19].

To measure the detection accuracy of the LIS approach and check the ground truth, we decrypt the TLS connection (all traffic collected is HTTPS) and inspect the payloads. TLS packets can be decrypted by using a pre-master secret key / RSA key or through a transparent TLS proxy. On the client, we set the SSLKEYLOGFILE environment variable before each capture which logs each TLS session key to a file. Note that this method does not work with all cipher suites and in some cases we had to disable Diffie Hellman-based cipher suites to force an RSA-based connection.

In addition to payload inspection, we also develop our own method to evaluate ground truth without inspecting payloads. The way to compare the timings between keystrokes and packets entails utilizing the concept of sequence alignment, which is a way of laying out the timestamp sequences (host vs. packet) to identify the similarity between them. We adapt DTW, which is used for an optimal alignment between two temporal sequences that may vary in speed. DTW provides a non-linear alignment as well as the similarity between two time series. The DTW approach can be useful to assess whether the website leaks the timings of input events without performing payload inspection and can be used to measure the accuracy of a particular detector.

The detection method determines whether the packet is an Ajax packet or not, so we categorize it as a binary classification problem. The performance of each detection method is measured by classification accuracy for which binary classification is a special case. The confusion matrix of a binary classifier is shown in Figure 3.5, where columns refer to ground truth and rows refer to predictions made by the model.

We compare the predicted values with actual values and compute four different metrics: true positive rate (TPR), true negative rate (TNR), FPR, and FNR.

- **True Positive Rate**: True positive / Actual positive
- **False Positive Rate**: False positive / Actual negative
- **False Negative Rate**: False negative / Actual positive (= 1 - TPR)
- **True Negative Rate**: True negative / Actual negative (= 1 - FPR)

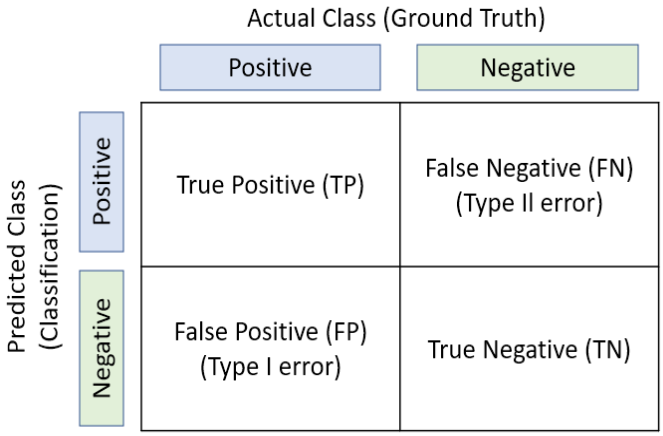

Figure 3.5. Definition of false positive rate (FPR) and false negative rate (FNR) for a detection system

- **Accuracy**: (TP+TN) / (TP+TN+FP+FN)

The TPR is the rate at which packets are correctly detected and the TNR is the rate at which the background packets are successfully ignored. Since there is always a trade-off between TPR and TNR, which part to put the weight on should be decided depending on the usage of the classifier. Note that the perfect detection has TPR = TNR = 1 [37]. Generally, TPR, TNR should be higher and FNR and FPR should be lower for the better performance. We compare the performance of each method with Accuracy, FPR, and FNR.

3.4.1 Longest Increasing Subsequence

Given an input sequence $X = \langle x_1, x_2, ..., x_n \rangle$ of numbers, a longest increasing subsequence (LIS) of X is the longest subsequence where the values are increasing in value. In other words, we look for the longest sequence of indices $i_1, ...i_k$ such that

$$i_1 < i_2 < ... < i_k, \quad x[i_1] < x[i_2] < ... < x[i_k]. \tag{3.1}$$

The LIS can be determined efficiently with a dynamic programming algorithm.

We have been able to confirm that the packet size pattern follows this rule on most websites, which means that the packet size tends to increase as the characters typed into a text input field increase. When surveying potential candidates for the experiment, we examined

several websites: Google Search, DuckDuckGo, GoogleDocs, and Amazon. Each website has a unique model to process input events (i.e., how it handles events, executes functions, and whether it sets any timeout threshold). Thus, the absolute packet size and the amount of change is different for each website and this characteristic leads to several issues that make perfect detection difficult. Figure 3.6 is one example of Ajax packet detection by using the LIS algorithm at Google. As we can see from Figure 3.6, the packets sizes are not strictly following LIS, and hence, the parameter setting can be different for higher accuracy (e.g., contains the equal or decrease case). This especially true since our dataset includes the Delete and Backspace keys, which can make packet sizes decrease.

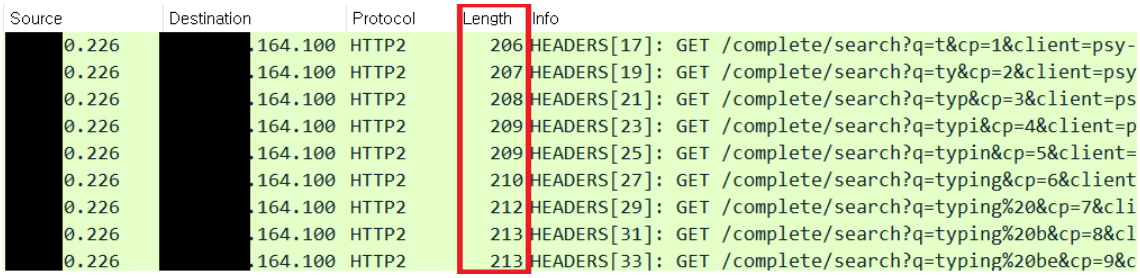

Figure 3.6. An example of Ajax packet detection by using LIS algorithm

3.4.2 Dynamic Time Warping

The way to compare the temporal features between keystrokes and packets requires leveraging the concept of sequence alignment. Among them, we adapted DTW, which is especially effective for calculating a similarity between two temporal sequences that may have various speeds. The DTW algorithm has gained popularity due to its efficiency measuring time series similarity with different phases by allowing the elastic conversion and minimizing the effects of shifting and distortion in time [45].

The objective of DTW is to compare two given time sequences $X = x_1, x_2, ..x_i.., x_x$ of length |X| and $Y = y_1, y_2, ..y_j.., y_y$ of length |Y|. A warping path $W = w_1, w_2, ..., w_k$, $max(|X|, |Y|) \leq K < |X| + |Y|$, where K is the length of the warp path and k^{th} element of the warping path is $w_k = (i, j)$, where i is an index from X, and j is an index from Y [46]. The optimal warping path is the minimum-distance between two time series. By leveraging DTW alignment combined with cross-correlation, we develop a new approach to detect Ajax packets. Figure 3.7 describes the procedure we used. It is mainly divided into

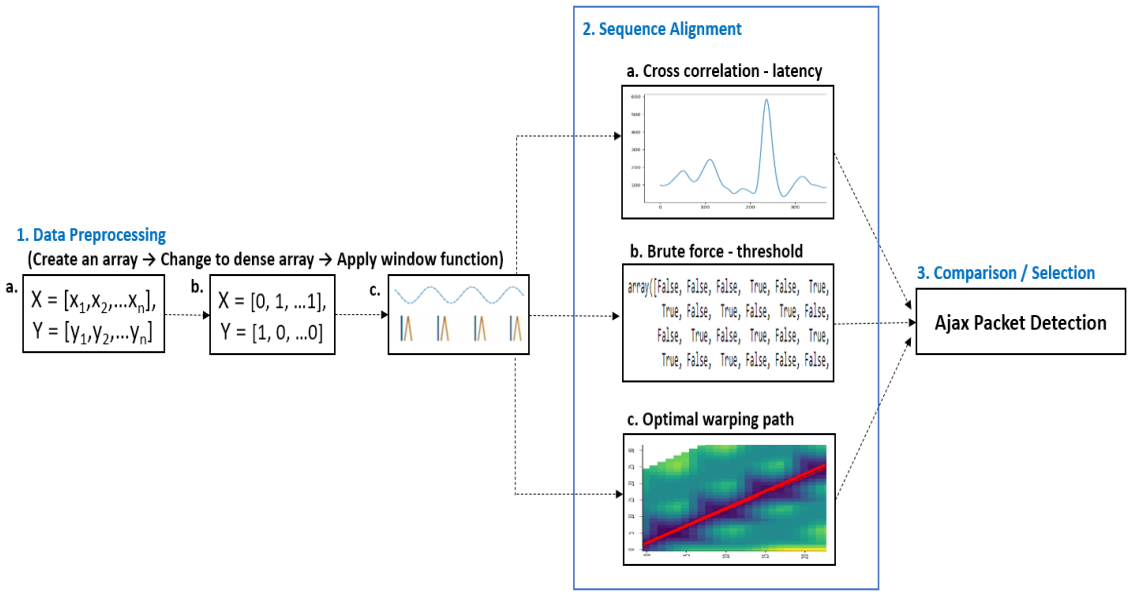

Figure 3.7. A procedure to detect Ajax packets using DTW alignment

three parts: data processing, sequence alignment, and comparison/selection. The detailed explanation is as follows:

1. Data Processing
   (a) Create an array of each temporal sequence:
   $X = [x_1, x_2, ..., x_n]$ (an array of packet timing),
   $Y = [y_1, y_2, ..., y_n]$ (an array of keystroke press timing)
   (b) Change X, Y to dense arrays that consist of 0 and 1 for comparison
   (e.g., $X = [0, 1, ..., 0]$, $Y = [1, 0, ..., 1]$ (X[x_n] = 1, Y[y_n] = 1))
   (c) Apply a window function to each array for continuity
2. Sequence Alignment
   (a) Cross-correlation: calculate the "latency" between two arrays
   (b) Brute force: calculate the distance between each pair of two arrays and set the "threshold" (referring to latency at 2-(a)) to extract packets
   (c) DTW: calculate the optimal warping path using the latency at 2-(a).
3. Comparison / Selection
   (a) Extract the pairs for which the distances are smaller than the threshold at 2-(b).
   (b) Compare 3-(a) and 2-(c) and make a final choice for Ajax packets

Cross-correlation is another method that is used to compare two different time series and determine how well they match each other. Note that the size of the window function and threshold should be different depending on each input. As a result of our experiment, we were able to detect Ajax packets accurately by leveraging the DTW approach on most websites. However, this approach did not work well with fast typing speed because of the threshold of auto-saving / autocomplete (about 300ms) in the case of GoogleDocs and DuckDuckGo.

Figure 3.8 is an example of Ajax packet detection achieved by the DTW approach from one of the evaluation datasets: (a) an optimal warping path between packets and keystrokes, (b) a linear graph with (packet timing - latency) and keypress timing, (c) a distribution of difference between packet and keypress timing, and (d) a distribution of difference between packet and keypress timing (histogram). This result shows the relation between packet and keypress timing for Google Search, which means they are well preserved under a constant delay time, though there is some noise.

Through the procedure to detect Ajax packets by using DTW alignment on Google Search, we determined that the average delay between matched events in each sequence is from 10ms to 15ms (i.e., packets are emitted roughly 10ms after keypress events). This calculated delay was almost consistent with the delay from the actual ground truth. We can observe that packet timing is almost consistent with keypress timing after subtracting a delay, which is described in Figure 3.8 (b). Figure 3.8 (c), (d) describes the offset between packet and keypress timings that are matched with the optimal warping path, including background packets.

Therefore, we use this approach as another source of ground truth to evaluate the LIS approach and find that LIS compared to DTW has a more than 90% true positive rate in detection. Inspecting failure cases reveals that errors arise when typing speed exceeds a threshold and multiple characters are merged into a single Ajax packet.

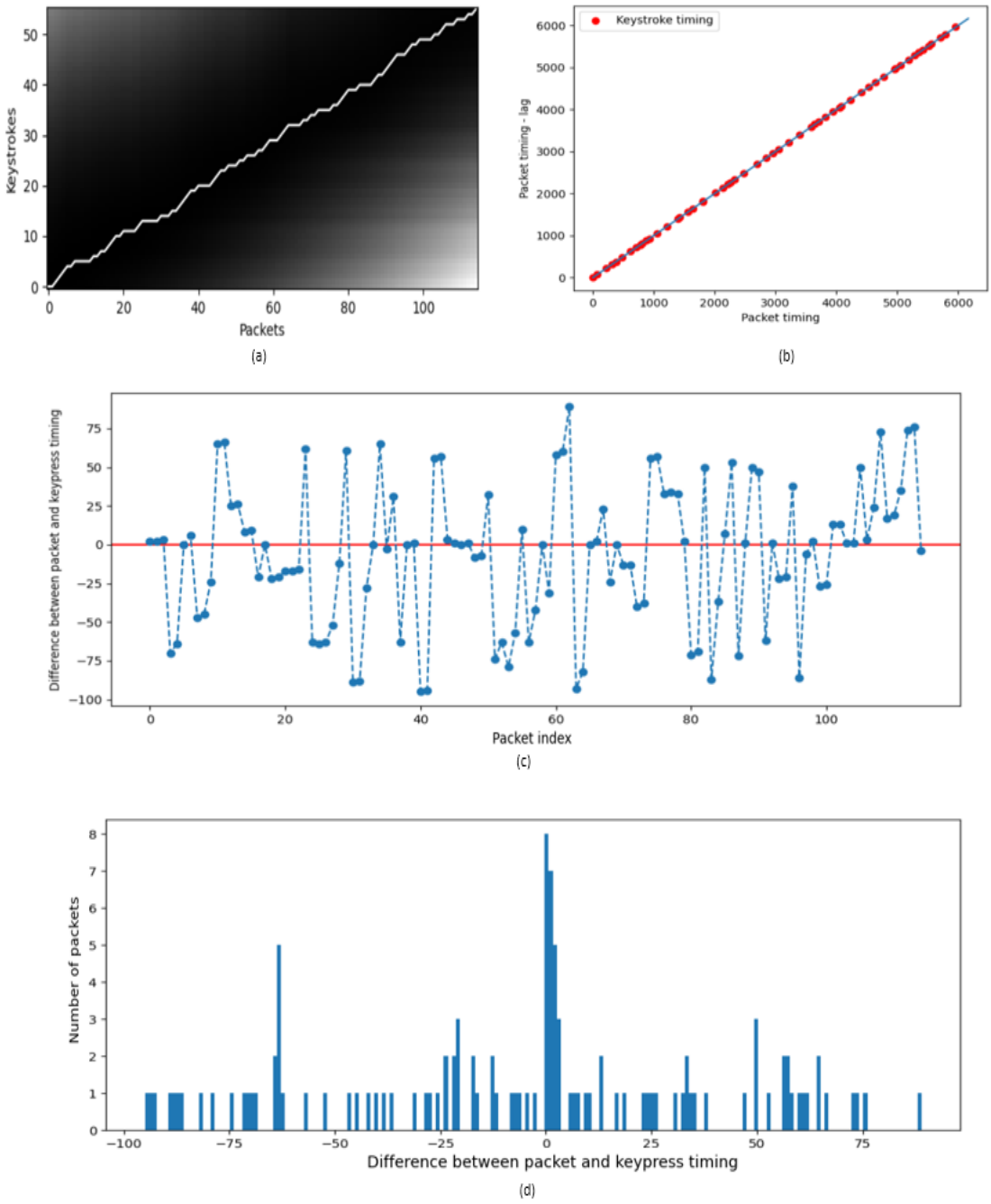

Figure 3.8. An example of Ajax packet detection by the DTW approach from one of the evaluation datasets: (a) an optimal warping path between packets and keystrokes, (b) a linear graph with (packet timing-latency) and keypress timing, (c) a distribution of difference between packet and keypress timing, (d) a distribution of difference between packet and keypress timing (histogram)

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
User Identification

In the previous chapter, we described Ajax packet detection through the patterns of packet size side-channels identified by the LIS algorithm. In this chapter, we perform remote user identification in dynamic web traffic from the detected packets. We discuss four steps for user identification: feature extraction, specification of a recurrent neural network, triplet training, and classification. We leverage the extracted packet timestamp information from the detection to perform user identification. Feature extraction is described as the way in which the feature vector is extracted from the raw packet timestamp information. Then, we elaborate on our training methods in regard to RNN and triplet loss functions. Finally, we describe how to perform user identification and verification by comparing the distance between the embedded vectors of train/test dataset.

4.1 Feature Extraction

Feature extraction is a method of performing transformations of original features to produce more informative and non-redundant features with the goal of representing the data in a way that is more amenable to machine learning [47]. We build and train an RNN that extracts a vector of deep temporal features from a sequence of timestamps. We first compute the intervals between packet timings. Given a sequence of Ajax packet timestamps in millisecond resolution t_i for $0 \leq i \leq N + 1$, the sequence of time intervals is taken as

$$\tau_i = t_{i+1} - t_i \tag{4.1}$$

where the sequence τ has length N. This requires N+1 timestamps. Note that if the Ajax packets corresponded exactly to keypress times, τ would represent the sequence of keypress latencies (i.e., time between successive keydown events). Because of variations in event processing on the host and packet jitter, however, the intervals τ do not exactly match the keypress intervals that would be obtained on the host. Packet delay jitter may be caused by various factors, such as: poor hardware performance, network congestion, and taking different paths to the destination [48]. We consider an ideal scenario in which jitter in

minimized by collecting traffic at the network interface of the victim. The motivation for this is to isolate packet timing jitter to the web page only, eliminating network effects. After that, we quantize τ by rounding the intervals to the nearest increment of b ms, forming the sequence of discrete tokens:

$$s_i = \left\lfloor \frac{\tau_i}{b} + 0.5 \right\rfloor \tag{4.2}$$

We determine the optimal b value as 5ms, which is large enough to eliminate most of the jitter introduced by the web page. This allows for some variation in the Ajax packet timings such that packet time intervals will be mapped to the same interval on the host with minor noise introduced.

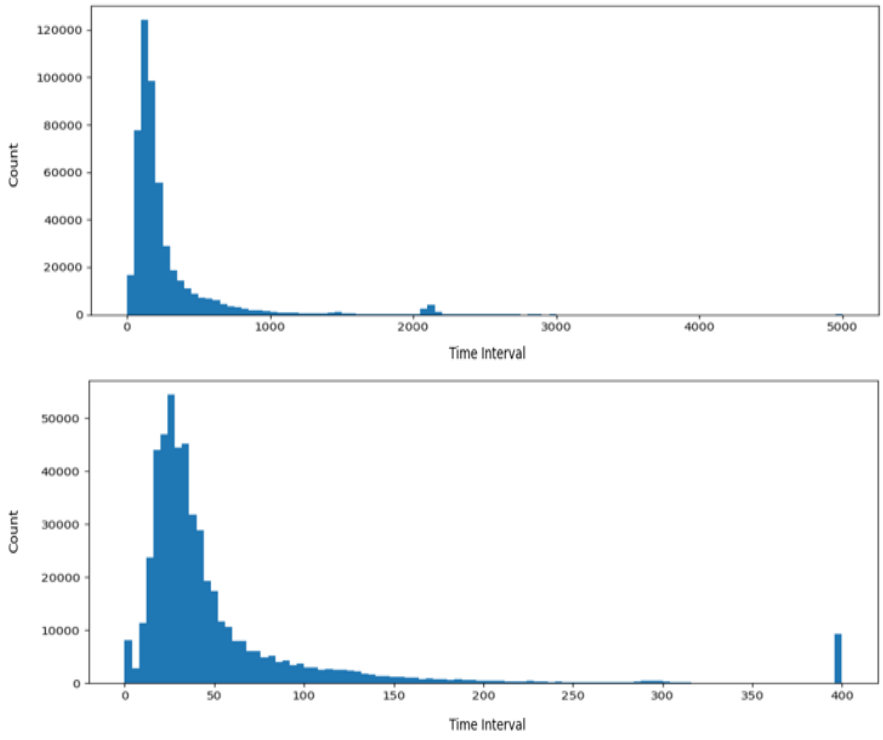

Figure 4.1. Histogram of time interval distribution: original values of τ (up) and the result after tokenization at $b = 5$ms (down)

Additionally, we analyze the time interval (τ) distribution, which is depicted in Figure 4.1: the original values of τ (up) and the result after tokenization at $b = 5$ms (down). We cap τ_i at

a maximum of 2 seconds. There are two reasons why we set the maximum of 2 seconds for the tokens, which can be observed in Figure 4.1 (a). First, most keystroke timing intervals are below 2 seconds. Faster typists typically also demonstrate more consistent behavior than slower typists; thus larger intervals are less indicative of user identity. Second, the collected data has approximately 2 seconds between each search query since we combined Ajax packets from several consecutive queries. As a result, the tokens s_i are bounded, $0 \leq s_i \leq 401$ with $b = 5$ms. The sequence $\mathbf{s}$ is provided as input to a recurrent model f that outputs a fixed-length vector

$$f(\mathbf{s}) = \mathbf{x} \tag{4.3}$$

where $\mathbf{x}$ is an L2 normalized feature vector of length 128. L2 norm ("least squares" norm) is defined as the square root of the sum of the squares of the values in each dimension [49]. Euclidean distances between these embedded vectors form the basis for user identification and verification.

4.2 Recurrent Architectures

An RNN is a neural network for sequential data, which maintains a hidden state as input is fed to the network [50]. It fits well for temporal data such as time series since the inputs can vary in length. This makes it suitable for training the dataset we used that consists of free-text keystrokes. Figure 4.2 depicts the different recurrent architecture by the order: from vanilla RNN, long short term memory (LSTM), bidirectional LSTM, to gated recurrent unit (GRU).

A vanilla RNN is the most basic form of the RNN, which processes the sequence $\mathbf{x}$ as an input vector by applying a recurrence formula. It consists of a single tanh activation function and shares the parameters between units. However, an RNN is unable to handle long-term dependencies, which means it is difficult to process by using the distant state from the current node.

An LSTM is a modified version of an RNN that aims to solve the vanishing gradient problem (i.e., the difficulty of learning long-term dependencies), which consists of three gates: input gate, forget gate, and output gate. An LSTM utilizes these gates to remove or add information

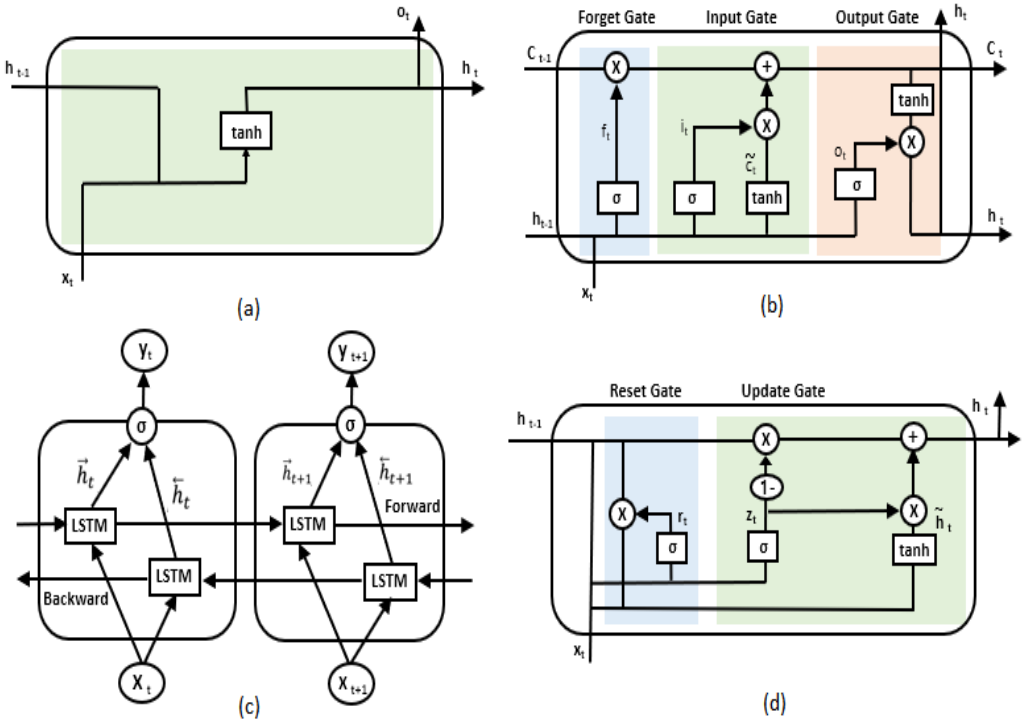

Figure 4.2. Different recurrent architectures: (a) RNN, (b) LSTM, (c) Bidirectional LSTM, (d) GRU. Adapted from [51], [52].

to cell state [51] through the operation of sigmoid and tanh activation function of each gate. A "forget gate" determines whether the past hidden output will be forgotten or not, and an "input gate" decides which values update. Leveraging these gates, an LSTM generally performs better than an RNN.

Unfortunately, an LSTM can only predict using past sequences, not future information. As depicted in Figure 4.2 (c), a bidirectional LSTM solves this problem by considering both forward and backward operations, to conduct a parallel operation. However, a bidirectional LSTM does not always bring better performance and has the disadvantage of higher computational cost. Finally, a GRU is similar to an LSTM, but simpler in that it has fewer parameters and no output gate. A GRU utilizes two gates: a reset gate and an update gate. It provides a reduced training time with comparable performance but normally fits better with smaller datasets. The performance was worse than that of the LSTM in the case of our experiment since our datasets may be comparatively larger for the GRU.

Among the RNN architectures we discussed, the performances of the LSTM and the bidirectional LSTM were similarly high; the LSTM was slightly better. This means our datasets do not seem to be affected by future information as much as information about the past. Also, considering the graphics processing unit (GPU) memory and the time cost, we chose the LSTM architecture.

The structure of our model is shown in Table 4.1. The model f is an RNN based on the TypeNet model developed in [36] which contains two stacked LSTM layers of 128 units. Note that the number of units is the dimension of the hidden state or size of the output. We introduce an embedding layer as the first layer of the network since we tokenize the intervals. An embedding layer uses the integers in $\mathbf{s}$ to index a dense matrix, a technique commonly used in natural language processing where dictionary size can grow to hundreds or thousands of tokens. We found the embedding layer to be essential to obtaining identification accuracy with the packet timings that approaches that of using the timings measured on the host.

Table 4.1. LSTM network architecture

Layer	Output Shape	No. Params
Input	$N \times 1$	0
Embedding	$N \times 16$	6416
Batch Norm	$N \times 16$	64
LSTM	$N \times 128$	74240
Dropout (0.5)	$N \times 128$	0
Batch Norm	$N \times 128$	512
LSTM	128	131584
L2 Norm	128	0

To prevent the overfitting, we use batch normalization before each layer and dropout between the LSTM layers. Batch normalization is used to normalize the activations of input volume before going to the next layer and can be used for any set of activations in the network [53]. It has been shown to have advantages in stabilizing the training process by reducing the number of epochs and providing various learning rates. Dropout prevents the overfitting by changing the network architecture through randomly dropping the connections as probability p at each training session. We set p as 0.5, which seems to be the optimal rate.

The model is trained with approximately 117,000 users in the Aalto dataset that contain

at least 600 keystrokes. We evaluate the model with data from the 500 users held out for traffic capture. Because this model takes as input a single time series, only keypress timings are utilized for training. This differs from previous work on keystroke biometric systems in which both keypress and keyrelease timings are utilized. We train the model for 150 epochs with adaptive moment estimation (Adam) optimizer (ϵ=10^{-3}, β=0.9, β_2=0.999), 256 batch size, and online semi-hard triplet mining with margin (α=1.0) [14].

4.3 Triplet Training

Triplet loss is considered one of the best-performing loss functions on embedding tasks [54] (i.e., projecting high-dimensional data to a lower dimension). Our model is trained using triplet loss, a method that learns to rank distances between samples belonging to the same or different classes [14]. Triplet loss is especially effective when the number of samples per class is small [55], and it generally performs better than training with other loss function such as softmax, according to [56]. We choose triplet loss because we assume there are only two samples per class in our dataset, which represent two separate TCP connections.

During each batch of training, the model is presented with triplets, each of which includes an anchor sample(a), a positive sample(p) in the same class as the anchor, and a negative sample(n) in a different class from the anchor. The formula of triplet loss is

$$Loss(a, p, n) = \max(\underbrace{\| f(a) - f(p) \|^2}_{d(a,p)} - \underbrace{\| f(a) - f(n) \|^2}_{d(a,n)} + \alpha, 0) \tag{4.4}$$

where a, p, n are the anchor, positive, and negative samples, respectively, and α is a margin that is imposed between positive and negative pairs [57].

The number of triplets to choose from grows rapidly with the size of the dataset. Triplet mining is the process of finding triplets that lead to better model optimization, which are generally categorized into online and offline methods. The main difference between them is the time to generate triplets. In online triplet mining, triplets are created within the mini-batches of data during the training phase. Conversely, the data processing is conducted before the training in offline generation [58]. We choose the online triplet method since it is more efficient and is able to produce more triplets for a single batch of inputs [57]. Triplets

are also mainly divided into three categories based on the definition of the loss: easy triplets, hard triplets, and semi-hard triplets. Each definition depends on the relative position of the negative. Selection of the proper triplet greatly influences the performance of the training model [14]. We compute the triplet loss with semi-hard negative mining, which means the negative instance is farther than the positive to the anchor, but still has a positive loss due to the margin we set. We use the TripletSemiHardLoss function in the `Tensorflow Addons` library [59].

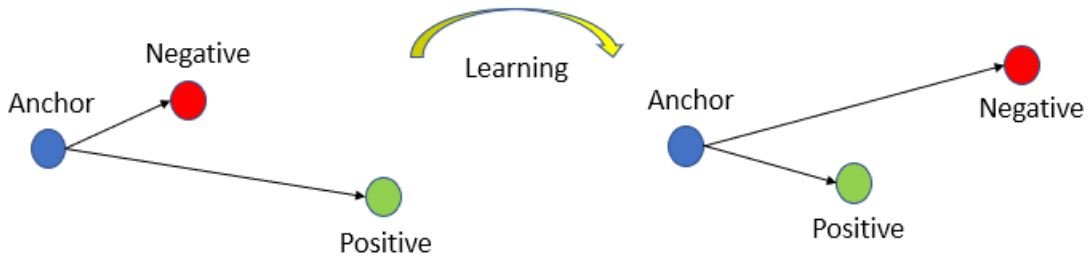

Figure 4.3. The triplet loss function: to minimize the distance of anchor-positive and to maximize the distance of anchor-negative. Adapted from [14].

Figure 4.3 describes the training process of triplet loss function: the triplet loss forces the anchor-positive distance to be smaller than the anchor-negative distance. Therefore, this training method is determined by triplets that carry hard positive cases (close anchor-positive) and hard negative cases (close anchor-negative) [60]. These hard samples generate triples that create gradients with a large magnitude (i.e., the performance of triplets is related to the training sampling method). We shuffle the training dataset before the batches are created at each epoch to regularize the model. To help avoid overfitting, it is important to shuffle the randomly dataset during the training procedure.

4.4 Classification

The embedded vectors produced by the model are compared using Euclidean distance, and both user identification and verification are performed with only a single template sample (i.e., one-shot learning scenario). We consider two different adversaries: an adversary who attempts to identify a user by linking the keystrokes observed in two different TCP connections; and an adversary who attempts to verify the identity of a user.

In user identification, the goal is to correctly match a given connection to one of the templates. After the training procedure, we compare the feature vector of the test sample against samples in the training set by using the triplet network. Identification accuracy is measured by Rank-1, Rank-5, and Rank-50 classification accuracy. Ranked accuracy is mostly used to measure the accuracy of the neural network model. Rank-1 accuracy is the percentage of top prediction, which has the same meaning as the standard accuracy. Rank-5 accuracy selects the five highest probabilities. It is common to consider rank-n accuracy with n>1 when the number of labels is high.

In user verification, the goal is to verify that a given connection belongs to the same user as a reference connection. It is a binary classification problem to determine whether the user is genuine or an impostor. We measure verification performance by calculating the area under the ROC curve (AUC), equal error rate (EER), and accuracy. Figure 4.4 describes the concepts of EER in (a) and AUC in (b). The EER is the point at which the rates of false positive and false negative are equal on a distance threshold d. The lower the EER value, the higher the accuracy of the model.

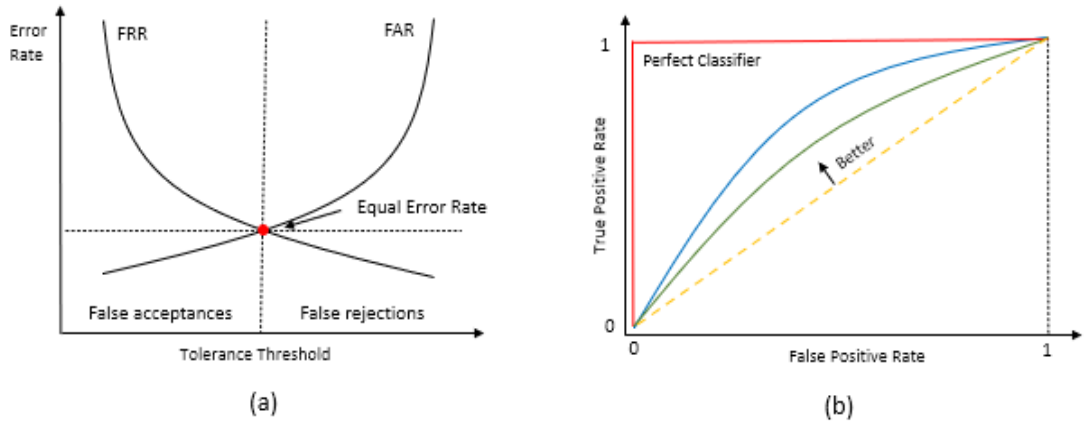

Figure 4.4. (a) EER, (b) AUC-receiver operating characteristic (ROC) curve (area under the receiver operating characteristics (AUROC))

The ROC curve is a probability curve that consists of the x-axis with TPR and y-axis with FPR. The AUC is a performance metric obtained by integrating the ROC curve to obtain the total area under the curve [61]. Whereas EER measures only a single point on the ROC curve, the AUC captures the complete system performance by considering all distance

thresholds. The red line in Figure 4.4 represents the perfect classifier where the AUC is 1. The closer the graph gets to the value of 1, the better the performance we get to distinguish between classes. We leverage this concept to compare the verification performance with diverse scenarios.

4.5 Summary

We have described the whole parts of our methodology over two chapters so far. In summary, we set two important assumptions for the experiment: 1) a remote adversary can eavesdrop on traffic from the victim, and 2) the victim utilizes the same TCP connection. From the data collected by the system we built, we performed Ajax packet detection by leveraging the patterns of packet timing and size side-channels. Then, we extracted features from the detection and utilized them as inputs for our model. The model we used is based on RNN, which is suitable for temporal data. We elaborated on our training methods in regard to LSTM with diverse techniques and triplet loss functions. After that, we evaluated the performance of user identification and verification with diverse scenarios. The next chapter introduces the experimental results for each method we performed.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5:
Experimental Results

This chapter discusses our experimental results and is organized in two parts: Ajax packet detection, and user identification/verification. In Ajax packet detection, we analyze the website behavior and compare the relative performance of each detection method. After that, user identification and verification accuracies are evaluated for several different scenarios.

5.1 Event Packet Detection

There are mainly three approaches for Ajax packet detection that we described in Chapter 3: longest increasing subsequences, dynamic time warping, and inspecting payload. The user's typing a search query results in Ajax requests containing a URL parameter that gradually increases in length, which enables LIS detection to achieve a high accuracy. However, there are also several exceptions in which LIS detection can fail, including when: background packets have a similar size (false positive), Ajax packets have a different size (false negative), and multiple keypress events are buffered into a single Ajax request (a kind of false negative). Additionally, Ajax packet size could potentially decrease when edits are made to a partially completed query since we did not exclude Backspace and Delete keys from our data capture.

The dynamic time warping approach leverages packet and keystroke timings to minimize the latency between them. This latency depends on the way Ajax requests are made in response to user input events, so each website has a different latency. We use DTW as another source ground truth to evaluate the LIS method since DTW is a verified way to measure time series similarity by utilizing the concept of sequence alignment. DTW, however, has to deal with packet jitter since it uses temporal features only. Also, DTW tends to be affected by generated background traffic. Future work should consider a better way to extract timing for a more robust alignment, such as comparing the sequence of time intervals.

There are three ways to decrypt packets: using a pre-master secret key, RSA key, or transparent TLS proxy. We use a pre-master secret key, which was obtained by setting the SSLKEYLOGFILE environment variable before starting to capture the traffic. The auto-

complete feature makes payload detection possible since the client requests an HTTP/2 GET that contains the partially completed query in the URL after each keypress. Therefore, we can evaluate the exact performance of each method by inspecting the payload.

5.1.1 Analysis of Website Behavior

We investigate the susceptibility of each website to leak information over encrypted traffic using Ajax packet timings. We survey and analyze several popular websites that implement dynamic contents as potential candidates for the experiment: Google Search, DuckDuckGo, GoogleDocs, and Amazon. We use Arduino, which is an open-source electronics platform that can read inputs and turn them into outputs [62], to maintain the same condition for user input (e.g., the keystroke time interval is set between 100ms to 300ms). Figure 5.1 is a part of the keystroke generation code for Arduino to examine the website behavior.

```
char* alphabet[] = {'t', 'y', 'p', 'i', 'n', 'g', ' ', 'b', 'e', 'h', 'a', 'v', 'i', 'o', 'r'};
int numKeystrokes = 15;

if (startKeyboard) {
  for (int i = 0; i < numKeystrokes; i++) {
    randnum = random(100, 300); // mindelay = 100, maxdelay = 300
    Keyboard.press(alphabet[i]);
    delay(randnum); // time interval between keypress and keyrelease
    Keyboard.release(alphabet[i]);
    delay(randnum); // time interval between keystroke events
  }
```

Figure 5.1. Arduino key generation code

Table 5.1 describes each website's behavior and detection performance. The TPR of each detection method is calculated in the same manner: typing the short sentence "typing behavior," which consists of short fixed-text input (15 characters) in the Chrome browser. We use Wireshark and the keylogging software Biologger [63] to compare the timing between keystrokes and packets. Note that the performance can vary according to several factors such as the length of data entered and the typing speed.

The websites that we investigated use a callback event model with keypress and XHR rather than Fetch API. Also, we can observe that multiple keys are merged into a single packet when the typing speed exceeds a threshold on several websites. The speed at which this occurs differs per website: on DuckDuckGo Search (which implements an autocomplete feature

Table 5.1. Summary of website behavior (1)

Website	Event Model	Latency (ms)	TPR (%)		Threshold (ms)
			LIS	DTW	
Google	keypress (XHR)	10–15	100	86.7	100
DDG	keypress (XHR)	10–20	92.3	76.9	300
GoogleDocs	keypress (XHR)	200–250	42.9	85.7	2–300
Amazon	keypress (XHR)	300–400	93.3	100	100

similar to Google Search) and GoogleDocs (which implements an autosave feature triggered by keydown events), multiple keys are merged into a single request when keystrokes occur within less than 300 ms of each other. On the other hand, Google and Amazon merged multiple keys within a comparatively lower threshold (100ms) into a single packet.

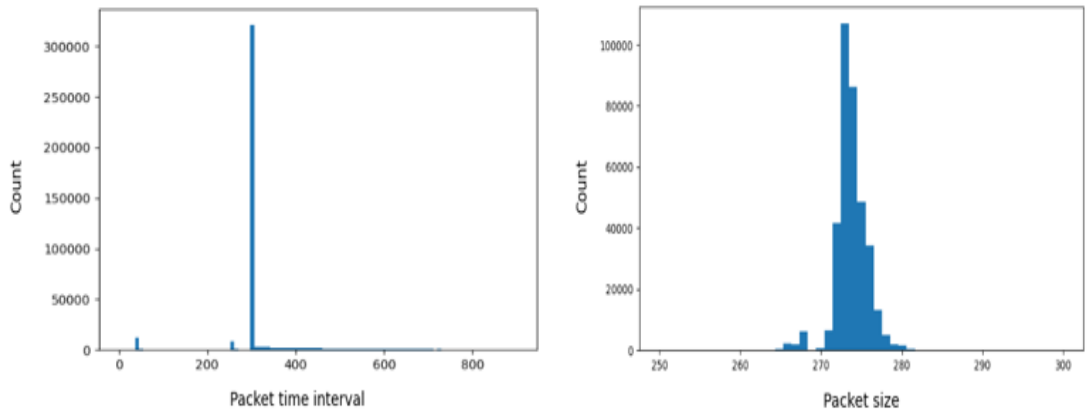

Figure 5.2. Website behavior: packet time interval distribution at DuckDuckGo (left) and packet size distribution at GoogleDocs (right)

Figure 5.2 is the histogram of packet time interval distribution at DuckDuckGo (left) and packet size distribution at GoogleDocs (right). We can observe that the time intervals are converged at its threshold (300ms), which means it is difficult to distinguish users through temporal features at DuckDuckGo. This result differs from that of prior work in that a callback registered to keyup events and did not have such a threshold [19]. DuckDuckGo seems to have changed its autocomplete function compared to the previous result investigated. In the case of GoogleDocs, the packet sizes do not follow the LIS pattern. The packet

size is determined not by the query length but by the number of inputs that are contained in a single packet. Most packet sizes are between 270 and 280 at GoogleDocs. Thus, the detection accuracy of the LIS approach for GoogleDocs is 42.9%, which is much lower than other websites.

Furthermore, GoogleDocs and Amazon have much longer latencies compared to other websites. GoogleDocs is a document collaboration application, which may produce a longer delay than the autocomplete feature since it does not have to save documents in such a short time or more time is required to save documents. We also found that the pattern of packets is different on GoogleDocs depending on its log-in state. When an anonymous user who is not the owner of the document modifies the shared document, an additional POST packet is generated at each Ajax packet. The POST packet size is similar to Ajax packets, and this makes it more difficult to distinguish the Ajax packets based on the packet size. Amazon would be a good candidate since Ajax packets are almost perfectly detected. On the other hand, there are also disadvantages in that its latency is unstable as well as the user's tendency to search queries on e-commerce sites by using just simple words. Table 5.2 summarizes the protocols that the website used to generate Ajax packets and the decrypted payload of Ajax packets. Most websites use the HTTP/2 protocol but some of them still use HTTP like the Amazon case.

Table 5.2. Summary of website behavior (2)

Website	Protocol	Payload
Google	HTTP2	HEADERS[N]: GET /complete/search?q=...
DDG	HTTP2	HEADERS[N]: GET /ac/?q=...
GoogleDocs	HTTP2	DATA[N] (application/x-www-form-urlencoded)
Amazon	HTTP	GET /api/2017/suggestions?session-id=...

As we describe in tables 5.1 and 5.2, each website has its own behavior in response to user input events.

Figure 5.3 is an analysis of web traffic at Google Search: a packet rate per a second (up) and the result of Ajax packet detection (down). At first, many packets occur due to TLS handshake (Client/Server Hello - Change Cipher Spec) to establish a connection between a client and a server and to load a web page at Google. This includes HTTP/2 SETTINGS

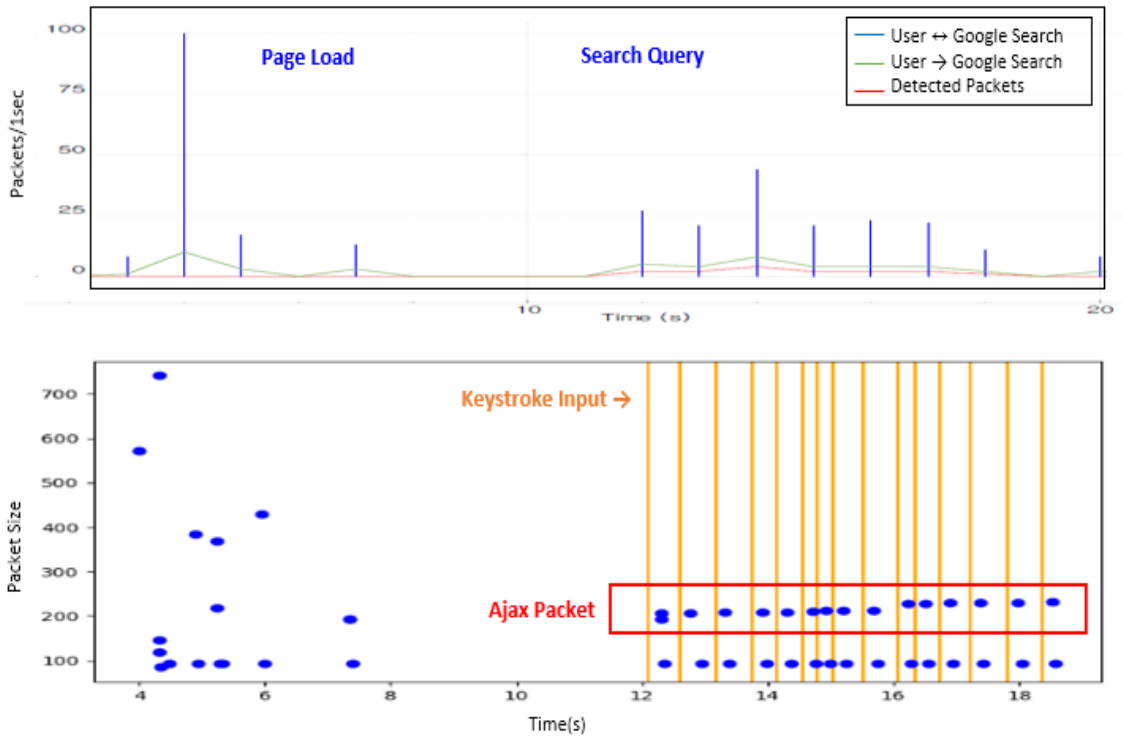

Figure 5.3. Analysis of web traffic: a packet rate per a second (up) and the result of Ajax packet detection (down)

for initial configuration parameters that affect how endpoints communicate, and HTTP/2 WINDOW_UPDATE to implement flow control [64]. After the page loading, packets tend to occur the same rate as typing speeds. Not only Ajax packets but also other background packets are generated by the keystroke inputs. We can also observe the consistent latency between keypress and packet time, and Ajax packets that have the LIS pattern in Figure 5.3 (down). Note that smaller size packets that occur at the same rate as Ajax packets are HTTP/2 PING packets.

Each website has its own behavior in response to user input events. Additionally, we found that the Ajax packets increase by about 20 bytes after typing about 15 characters in Google Search. This is due to an additional parameter "gs_mss" added to the request URL [19], as described in Figure 5.4. Despite this exception, Google Search has a comparatively lower threshold and delay resulting in fewer false negatives. For this reason, we choose Google Search for our experiment to perform user identification.

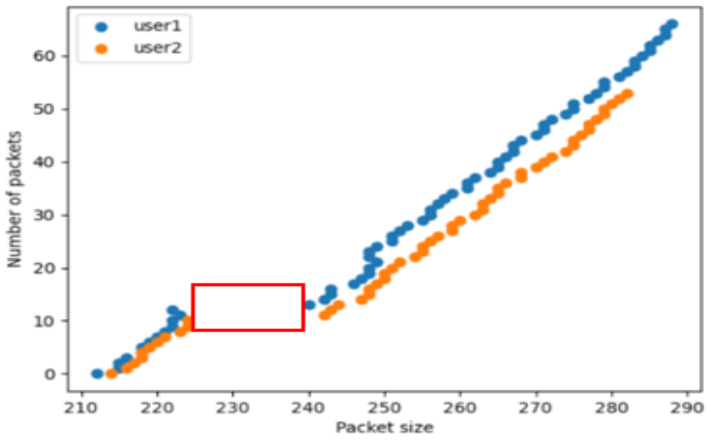

Figure 5.4. An example of Ajax packet pattern in Google Search

User1:
GET /complete/search?q=t&cp=1
GET /complete/search?q=ty&cp=2 (+1)
.....
GET /complete/search?q=typing%20behavio&cp=14 (+1)
GET /complete/search?q=typing%20behavior&cp=15 (+1)

User2:
GET /complete/search?q=typin&cp=5 (-1)
GET /complete/search?q=typ&cp=3 (-2)
.....
GET /complete/search?q=typing%20beha&cp=11 (+3)
.....
GET /complete/search?q=typing%20behavior&cp=15 (+1)

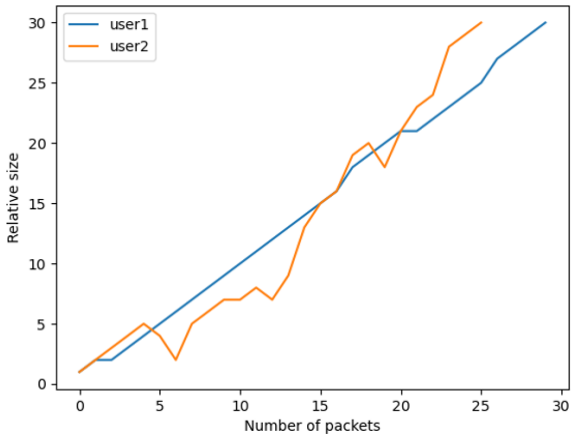

Figure 5.5. Query scenarios in diverse user behavior: general behavior(user1) vs fast typing with error correction(user2)

Figure 5.5 compares general behavior (user1) with fast typing with error correction (user2) in Google Search, showing that Ajax packet sizes do not always follow the LIS pattern with a constant rate. Note that we exclude the variance of Ajax packet sizes due to the "gs_mss" parameter. Ajax packet size could potentially decrease when edits are made to a partially completed query since we did not exclude Backspace and Delete keys from our data capture. Therefore, we set the increasing range of LIS for detection as -5 to 20 since we should consider the additional parameter as well as error correction. These parameters represent the smallest and largest change, respectively, that subsequences can undergo in

Ajax packet sizes. This makes the detection algorithm a bit more constrained than LIS.

5.1.2 Ajax Packet Detection Performance

The performance is measured for each detection method. Ground truth is extracted based on the fact that Google Search Ajax packets contain a "complete" string in the URL of the HTTP2 GET request. Note that this requires payload inspection and is website dependent. After that, the accuracy is computed by averaging TPR and TNR. Table 5.3 describes the statistics of Ajax packet detection for all packets, LIS packets, and DTW packets by the protocol and header.

Table 5.3. Statistics of Ajax packet detection

Method	HTTP/2 (GET / POST / PING / Others) (%)	TLSv1.2(1.3) (%)
All	93.4 (41.4 / 2.6 / 34.6 / 14.8)	6.6
LIS	99.9 (97.3 / 2.6 / 0 / 0)	0.0001
DTW	99.9 (84.9 / 0.01 / 14.9 / 0.001)	0.0002

Note that we only focus on TLS packets from source IP (user) to destination IP (Google Search) while opening the web browser and entering queries. Except for HTTP/2 over TLS, other TLS packets are mostly used for TLS handshake, which is a negotiation between browser and server to establish the connection. LIS and DTW did not detect these packets since the sizes are different from Ajax packets and they are made before any user input events. HTTP/2 packets mostly consist of GET headers that indicates requests for specified resources that include the autocomplete feature.

We compare the relative performance of each detection method: LIS, DTW alignment, and payload inspection. The accuracy, FPR and FNR are reported for LIS using both payload inspection and DTW alignment as a reference (ground truth), as well as DTW alignment using payload inspection as a reference. Table 5.4 summarizes these results based on the dataset we used.

As a result, we can observe that the LIS method records 96.0% accuracy with 6.5% FNR on average, which is high enough to examine further for user identification by using the temporal features of detected packets. On the other hand, the result did not show perfect

Table 5.4. Summary of Ajax packet detection performance

Method	Truth	Accuracy	FPR	FNR
LIS	Payload	96.0	1.6	6.5
LIS	DTW	96.9	0.38	5.9
DTW	Payload	93.0	4.0	10.1

detection due to the various issues mentioned earlier, because of background packets with similar sizes and some exceptions of Ajax packets with a different size. DTW as another source of ground truth suffers from different issues, such as packet jitter making detection more difficult, since it is based only on timing features. A general and more robust method of Ajax packet detection remains an area for future work.

5.2 User Identification and Verification

After training the model with triplet network, we evaluate the model with data from 500 users. We use a one-shot learning scenario, which means that predictions are made using only one training sample per user. We compare user identification and verification performance with three different scenarios: all keystroke timings, packet timings on the host, and packet timings detected by our method. The scenario for packet timings on the host assumes perfect Ajax packet detection and no additional noise introduced by packet timings. For the all keystroke timings scenario, we evaluate performance obtained by the TypeNet model developed in [36], which utilizes more diverse temporal features extracted by two consecutive keys as well as an additional features for the keycodes. Note that there are 500 users with one training sample and one testing sample, each comprised of 300 keystrokes in all scenarios.

We consider identification and verification accuracy dependent on three factors: the number of users, number of detected events, and detection FNR we used. We analyze and compare the performance by changing these variables. Table 5.5 summarizes the factors that we used for evaluation.

Table 5.5. Factors used to evaluate metrics

Subjects	Ranges	Metrics
Number of users (train)	10k–170k	Ranked Accuracy (1,5,50)
Number of users (test)	100–500	Ranked Accuracy (1,5,50)
		EER
Number of Ajax packets	50–300	Accuracy
Detection FNR	0–100	Accuracy

5.2.1 User Identification

We perform user identification by taking the Euclidean distance between the query sample and each of the 500 user profiles. We evaluate user identification performance based on the rank-N classification accuracy, for $N \in \{1, 5, 50\}$. Rank-1 accuracy represents the rate of unequivocally matching a query to that of the correct user. Rank-5 and Rank-50 accuracies represent the rate of placing the correct user's profile among the top 1% (five profiles) and 10% (50 profiles), respectively, when comparing the query to the 500 user profiles.

Table 5.6. Identification accuracy (Rank-1, 5, 50) and Loss as the number of users (train) is scaled up (10k-170k)

Number of users (train)	Rank-1	Rank-5	Rank-50	Loss
10k	33.2	69.6	98.0	0.63
50k	51.8	82.8	98.8	0.57
100k	50.0	81.2	99.0	0.61
170k	49.0	77.8	98.6	0.66

Table 5.6 describes the Rank-N accuracy with Loss according to the number of users in the model training set (not the evaluation training set). From 10,000 to 170,000 users we tested, the accuracy did not differ much, but 50,000 users of the training set recorded the highest Rank-1 accuracy (51.8%). Moreover, Rank-50 accuracy is not proportional to Loss or Rank-1 accuracy. In future work, it will be necessary to study the optimal training sampling method to get higher accuracy in the triplet network.

Figure 5.6 represents the relation between identification accuracy and sample length (left) and detection rate (right). Aggregating several hundred Ajax packet timings is a key for

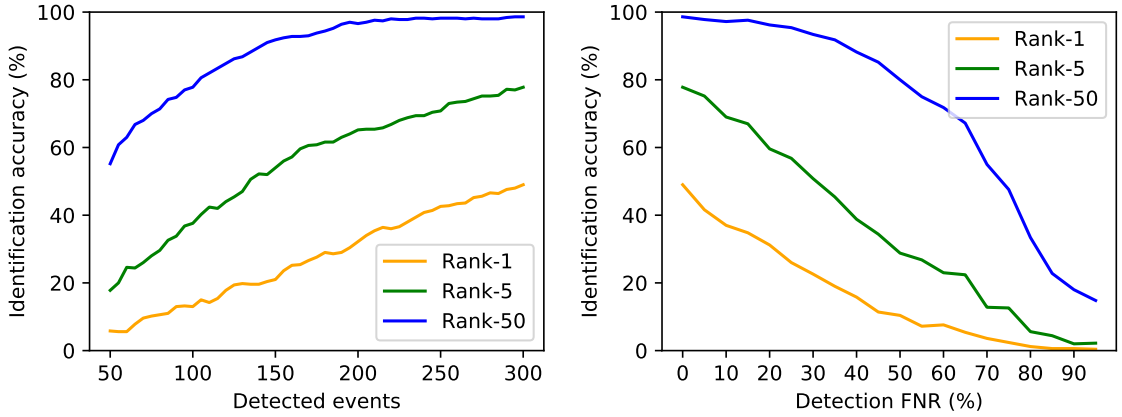

Figure 5.6. Identification accuracy vs sample length (left) and detection rate (right)

accurate user identification. We scaled the size of each sample (both training and testing) from 50 packets to 300 packets. Rank-1 identification accuracy steadily increases with the longer sample lengths, suggesting that higher accuracy could be obtained if more data were available. Likewise, user identification largely depends on a low FNR in detecting Ajax packets. This is shown in Figure 5.6 (right), where we evaluate accuracy as the FNR of detection increases. Rank-1 identification accuracy is halved at approximately 25% FNR.

Figure 5.7 describes each rank-*N* identification accuracy as the number of users is scaled up from 100 to 500 for both the timings on host and packet timings scenarios. We can observe that accuracy for Rank-1 and Rank-5 identification steadily decreases as the number of users increases. In the case of Rank-50 accuracy, however, it is not proportional to the number of users, but rather increases about 0.4% after 200 users in packet timings on the host. Since Rank-50 accuracy matches the answer with any of the 50 highest probabilities in the model, it is not as sensitive to population size with only a couple hundred users, but could potentially decrease as the population size increases.

5.2.2 User Verification

User verification is a binary classification problem about whether the user is genuine or an impostor. We measure verification performance by calculating AUC, EER, and accuracy. The EER is the point at which the rates of false positive and false negative are equal on

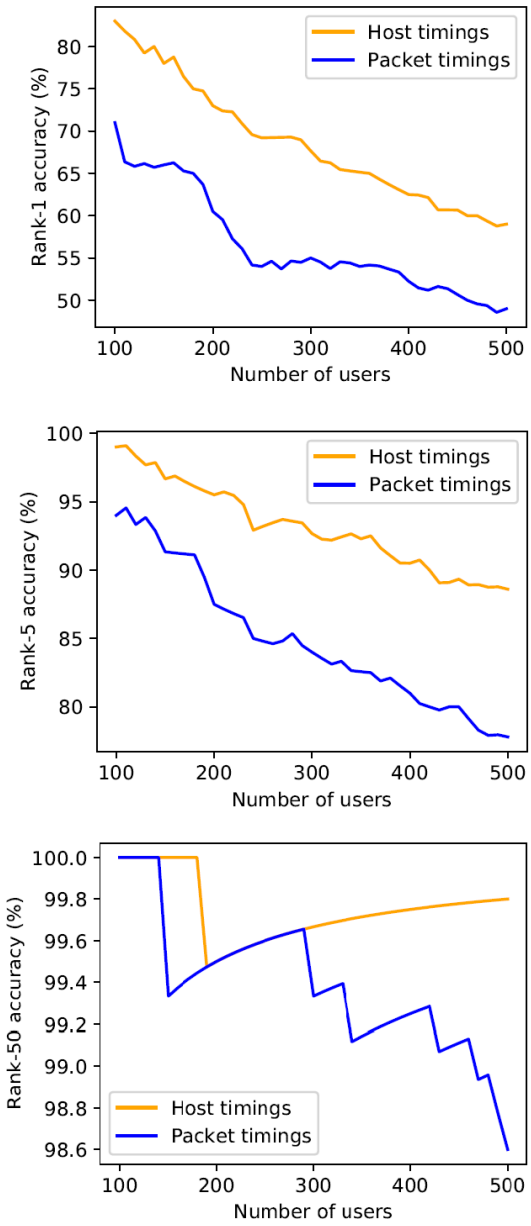

Figure 5.7. Identification accuracy (Rank-1, 5, 50) as the number of users (test) is scaled up (100-500)

the ROC obtained by varying a distance threshold d, which is described in Figure 5.8 (a). Figure 5.8 (b), (c), (d) describes the EER of each scenario: all timings, packet timings on the host, and packet timings detected by our method.

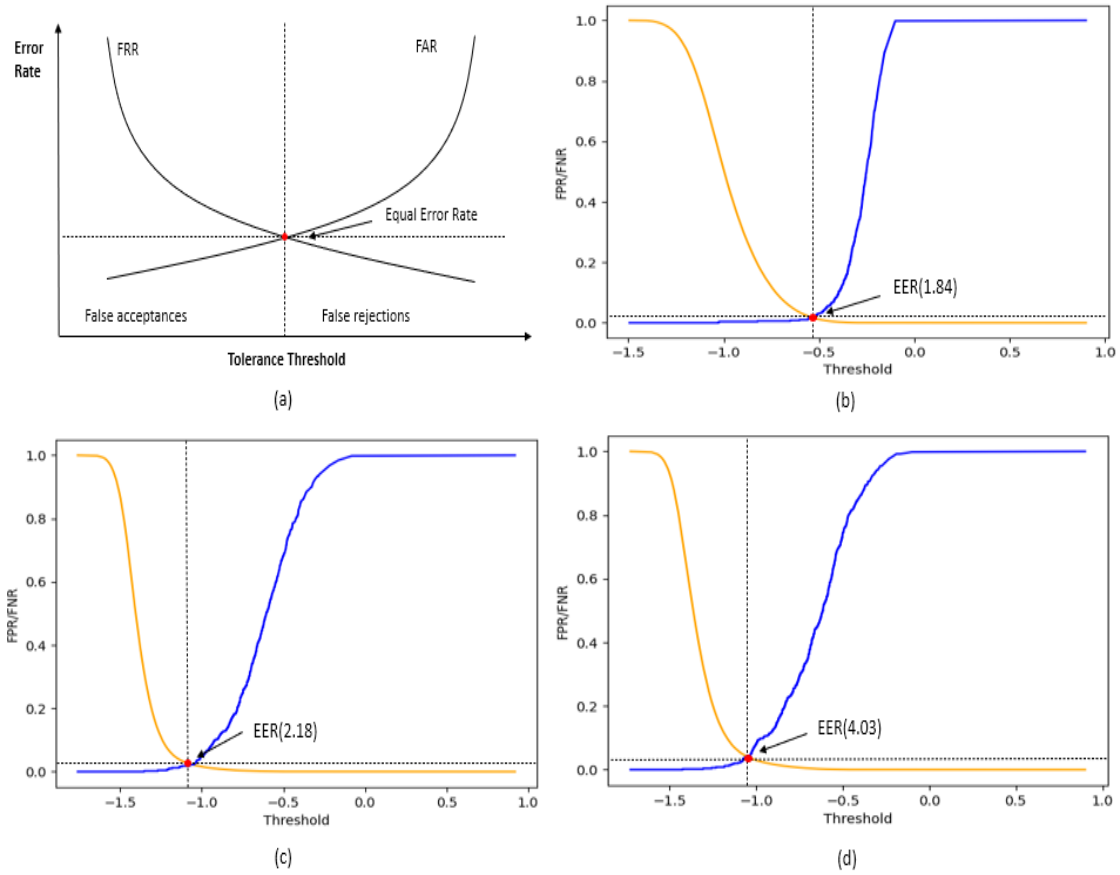

Figure 5.8. Comparison of verification error rate: (a) EER, (b) EER of all timings, (c) EER of packet timings on the host, (d) EER of packet timings detected by our method

Figure 5.9 also illustrates each verification accuracy rate and EER as the number of users is scaled up from 100 to 500 for both the timings on host and packet timings scenarios. While identification accuracy steadily declines with the additional users, verification performance does not significantly increase.

Figure 5.10 describes the relation between verification accuracy and sample length (left) and detection rate (right). Verification accuracy decreases with the shorter sample lengths by about 75%, which is a similar result of the identification that longer sample lengths guarantee the higher accuracy. For the detection FNR, verification accuracy declines by 50% as the detection rate decreases, since it is a binary classification problem.

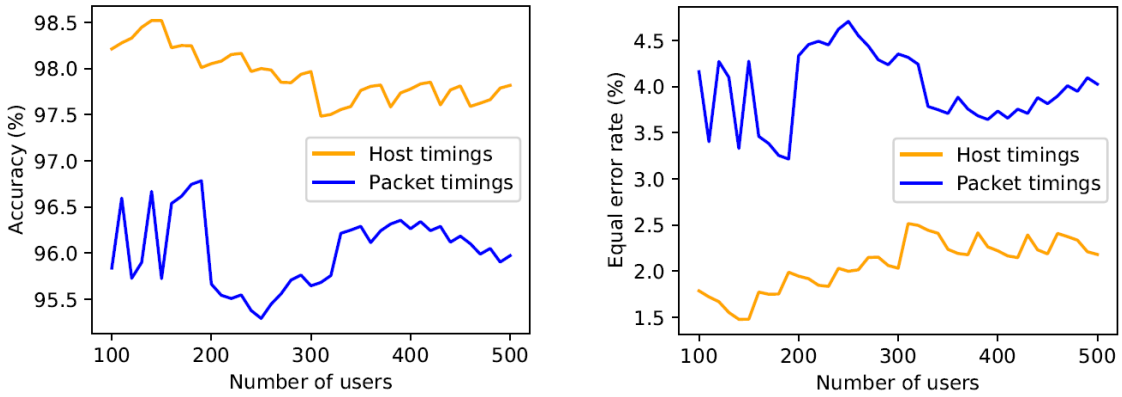

Figure 5.9. Verification accuracy (left) and EER (right), as the number of users is scaled up (100-500)

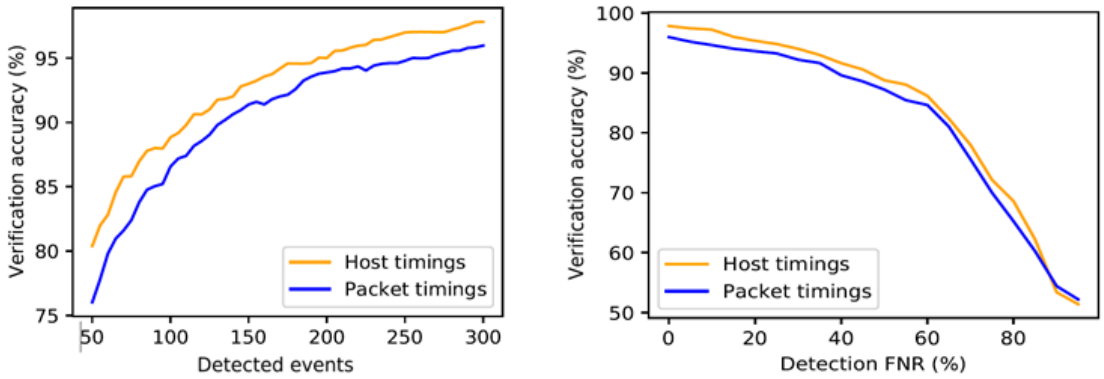

Figure 5.10. Verification accuracy vs sample length (left) and detection rate (right)

Figure 5.11 depicts the AUC-ROC curve for each scenario. The random classifier would have an AUC of 0.5, like the blue dotted line in Figure 5.11. We can observe that all AUCs we measure are near 1, which means having nearly perfect validation performance. Similar to previous results, the order of performance achieved in each scenario is all timings (0.996), host timings (0.995), and packet timings (0.989).

5.2.3 Summary

Table 5.7 summarizes user identification and verification performance for three different scenarios. Rank-1 identification accuracy with packet timings is 49%, while using timings

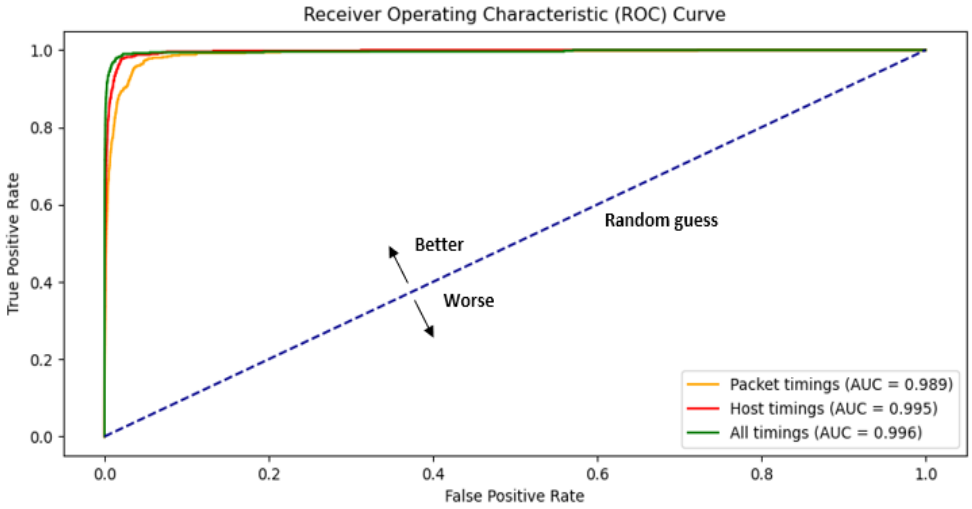

Figure 5.11. AUC-ROC Curves

Table 5.7. Summary of user identification (rank-N classification accuracy) and verification (accuracy) performances.

Features	Identification			Verification
	Rank-1	Rank-5	Rank-50	Accuracy
All timings	87.4	98.2	99.4	98.2
Host timings	59.0	88.6	99.8	97.8
Packet timings	49.0	77.8	98.6	96.0

on the host is 59%. The drop in performance can be attributed to the noise introduced by packet timings in addition to imperfect Ajax packet detection. However, nearly perfect rank-50 identification accuracy is achieved, suggesting that the set of user profiles can be accurately reduced by up to 90%. We also examined the degree to which performances is affected by three types of factors: the number of users, the number of Ajax packets, and detection FNR. Although accuracy using timings only on the host is higher than packet timings, several results stand out and warrant further examination such as the comparison of Rank-50 accuracy.

CHAPTER 6:
Conclusion

In this chapter, we summarize the research performed and discuss experimental results, mitigation, and broad implications. Lastly, we draw conclusions and identify areas for future work.

6.1 Summary

We have demonstrated that patterns of user behavior and identity can be exposed through side-channel information such as timing and packet size over encrypted dynamic web traffic. In our experiment, we built a system that can replay keyboard events and capture the corresponding keystrokes and network traffic. We detected Ajax packets from captured files based on the identified patterns by using two approaches (i.e., LIS for packet sizes and DTW for packet timings). Both approaches obtained over a 90% true positive rate on average in detection, which was high enough to support user identification. We used an RNN with triplet loss to extract deep temporal features from the detected packet timings and evaluated the performance. The results of user identification yielded an accuracy of 49.0% for Rank-1, 77.8% for Rank-5, and 98.6% for Rank-50. Verification performance was measured by EER and accuracy, which was recorded as 4.0% and 96.0%, respectively. In conclusion, we demonstrated that user identification and verification can be performed with modest accuracy by using the packet timings.

6.2 Key Takeaways

1. How do the dynamic web services process user input and invoke the requests? The event processing model represents a way for the browser to recognize input events before making a request. Google Search, which we used for the experiment, implements a callback model that invokes requests in accordance with the keypress event occurrence. Also, we focused on the use of Ajax technology, which supports fast and responsive interactions on dynamic web use. Ajax provides a better user experience, but it may also increase the threats related to privacy. When the user input event occurs, an Ajax API such as XHR or Fetch initiates requests in response to the event and communicates with the server by

using HTTP. Thus, even a small user input such as one keystroke can trigger web traffic through the API, which makes dynamic web services more interactive as well as vulnerable to diverse attacks.

2. Is there any way to detect user input events over encrypted web traffic? How accurate is the detection? The patterns of user input events can be exposed through side-channel information: timing and packet size. Two primary approaches were discussed for Ajax packet detection without inspecting the payload: LIS for packet sizes, and DTW, which leverages keystroke and packet timings. LIS is an algorithm that finding the longest subsequence where the values are increasing. The absolute packet size and the amount of change are different at each website. Dynamic time warping can measure the similarity between two time series. We developed this method as a new way for Ajax packet detection, and it is divided into three main parts: data preprocessing, sequence alignment, and comparison/selection. First of all, by leveraging keystroke press timing and packet timing, we create two time series into arrays, convert them to dense arrays, and apply the window function for continuity. After that, we calculate the latency respectively: using the concept of cross-correlation, setting the threshold, and using the optimal warping path. Lastly, a final choice for Ajax packets is made by comparing the result of each calculation. Both LIS and DTW methods could detect over 93% of Ajax packets at Google, which is high enough to examine further for user identification.

3. How can a remote adversary take advantage of unintended input-based side-channel leakages? How does this infringe on user privacy? A remote passive attacker can infer sensitive information from user inputs or server responses based on the timing and size side-channels. The sensitive information may include the identity of the user, the device that the user used, and the contents that the user entered. In this thesis, we focused on user identification by referring to the temporal features of users' typing patterns through Ajax request packets. By leveraging the sequence-based feature extraction model, both user identification and verification were performed. We compared the result with different scenarios: using all keystroke timings and packet timings on the host. The performance of our method was lower than the others, but it was still feasible to identify users, and high enough to verify users. This user's identity can be exposed without any agreement or notification, which can bring more subsequent security challenges such as tracking user activity or enabling identity fraud. This can lead to serious problems that violate user

privacy.

6.3 Discussion

6.3.1 Experimental Results

There are two main points that need further examination in our experimental results. First, a more general and robust method for Ajax packet detection is needed. This requires control of several different variables when analyzing the network traffic. Depending on the network access technology that users connect to, the type of devices and browsers they use, and whether the user logged in or not, the size of packets or the generated background packets can vary. Other variables, such as packet jitter and noise related to temporal features, can also affect Ajax packet detection. The accuracy of detection is a combination of all these factors. The verification of more diverse situations is necessary to perform user identification in a real world environment.

Next, several results from the user identification and verification processes need additional analysis. First, user identification and verification performance is not proportional to the number of users in the training dataset for the feature extraction model. The highest accuracy was obtained when training with 50k users, although this variance may be due to the selection of users in the training sets. Further examination is needed to determine the optimal number of training sets in a triplet network. In user identification, we can observe that Rank-50 accuracy slightly increases as additional test users are added. Also, Rank-50 accuracy of all timings is lower than that of host packet timings. This is not a significant difference and may be attributed to model variance (e.g., differences in model performance based on different initial parameters, data shuffling), but needs further examination. In user verification, accuracy is not consistently proportional to the number of users. The accuracy fluctuates, and it may likely be attributed to differences among users as a single new user is added to the population, since it is a binary classification problem. It is also necessary to consider alternative sampling methods to extract test users from the entire dataset.

6.3.2 Mitigations

Dynamic web content triggered by user input events can leak user identity in encrypted network traffic. Two key characteristics make this possible: the ability to distinguish Ajax

packets with high accuracy, presumably through a distinct pattern of packet sizes, and the preservation of keypress latencies in the Ajax packet time intervals. When we surveyed potential candidates that exhibit both characteristics, we found that this kind of attack becomes much more difficult, primarily due to the timing threshold for generating events on several websites (e.g., DuckDuckGo, GoogleDocs). These sites suggest that a simple mitigation measure is to implement a timeout mechanism for dynamic content triggered by user input. In other words, multiple input events are merged into a single Ajax request when the rate exceeds some threshold. In this way, original keypress latencies are not recoverable from the Ajax packets so long as typing speed exceeds the timeout. An alternative mitigation method would be to normalize packet size, making detection more difficult.

6.3.3 Privacy Implications

There are several privacy implications related to this research from the adversary's point of view. We can think of the risk to users who access the internet through a public/untrusted network or the government/business network. Both scenarios still expose the timing or size side-channel. A remote passive adversary who can observe the generated network traffic on the same network can analyze user behaviors by detecting Ajax packets, and subsequently track user actions or devices. This form of tracking users through behavioral patterns is more dangerous since it does not need explicit tracking techniques, such as leveraging HTTP cookies [65]. This field has come under increased scrutiny in recent years [65], [66]. Also, user identification can be leveraged as an intermediate goal to achieve better personalization or target identification [67]. Behavior information can be combined with other tools to enhance performance, especially to identify/verify users on a large scale.

6.4 Conclusion and Future Work

Similar to other biometric modalities (e.g., face recognition and gait analysis), the ability to identify users among encrypted web traffic is a double-edged sword. All users on the internet may still be exposed to risks and their privacy violated by being tracked. Not all privacy issues can disappear, but the applications of this research should be a step in the right direction for enhancing user privacy and designing a more effective defense mechanism.

There are several promising directions for future work. Detecting user behavior in dynamic

web traffic depends on accurately detecting Ajax packets, which remains a difficult problem to solve generally. A characterization of dynamic web traffic behavior over a more diverse set of websites also remains an ongoing area of research. By analyzing the behavior of more diverse websites and their interactions with browsers, we can measure the degree of information leakage and figure out how to generate the traffic at each website.

Developing a more robust Ajax packet detection method is also a promising direction for future work. By leveraging the time intervals, instead of raw timestamps, we may detect Ajax packets more accurately. This utilizes the concept of a subgraph pattern matching problem, which is to find all distinct embeddings of a graph pattern from a given large target graph and a query graph [68]. We can make two sets (N x N), (M x M) by using a brute force way to calculate each distance between raw timestamps for both keystrokes and packets, and may find the Ajax packets comparing the values based on the subgraph matching algorithm, which is described in Figure 6.1.

0	517	1071	1665	2053	2461	2680	2942
517	0	554	1148	1536	1944	2163	2425
1071	554	0	594	982	1390	1609	1871
1665	1148	594	0	388	796	1015	1277
2053	1536	982	388	0	408	627	889
2461	1944	1390	796	408	0	219	481
2680	2163	1609	1015	627	219	0	262
2942	2425	1871	1277	889	481	262	0

980	1040	1576	1630	1965	2027	2376	2426	2590
551	611	1147	1201	1536	1598	1947	1997	2161
381	441	977	1031	1366	1428	1777	1827	1991
0	60	596	650	985	1047	1396	1446	1610
60	0	536	590	925	987	1336	1386	1550
596	536	0	54	389	451	800	850	1014
650	590	54	0	335	397	746	796	960
985	925	389	335	0	62	411	461	625
1047	987	451	397	62	0	349	399	563
1396	1336	800	746	411	349	0	50	214
1446	1386	850	796	461	399	50	0	164
1610	1550	1014	960	625	563	214	164	0

Figure 6.1. An example of keystroke and packet time intervals of subgraph matching problem

In addition, user identification and verification performance can likely be improved, for instance, by optimizing the rounding base for inputs, the model hyperparameters, and the training method (e.g., the number of training users, triplet mining). Especially, we found that we can achieve better results from the optimal rounding base, which can be gained by analysis of distribution of timing jitter.

Lastly, users on the Tor network, which is an overlay network designed to obfuscate location [15], [16], may not guarantee anonymity in the presence of dynamic web traffic, since

Tor still exposes the timing information. Ajax packet detection would be more difficult due to the normalized packet sizes on Tor, but could potentially be performed through temporal features rather than size features (e.g., by detecting a page load and selecting packets that occur at about the same rate as typing speed after that). This form of remote user identification remains an item for future work.

List of References

[1] J. Rubenfeld, "The right of privacy," *Harvard Law Review*, vol. 102, no. 4, pp. 737–807, 1989.

[2] A. Bouguettaya, A. Rezgui, and M. Y. Eltoweissy, "Privacy on the web: Facts, challenges, and solutions," *IEEE Security Privacy*, vol. 1, no. 6, pp. 40–49, Nov 2003.

[3] S. A. Sharma and B. L. Menezes, "Implementing side-channel attacks on suggest boxes in web applications," in *Proceedings of the First International Conference on Security of Internet of Things*, 2012, pp. 57–62.

[4] T. Kohno, A. Broido, and K. C. Claffy, "Remote physical device fingerprinting," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 2, pp. 93–108, May 2005.

[5] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna, "Cookieless monster: Exploring the ecosystem of web-based device fingerprinting," in *2013 IEEE Symposium on Security and Privacy*, 2013, pp. 541–555.

[6] G. Fowler, "Think you're anonymous online? A third of popular websites are 'fingerprinting' you," *The Washington Post*, vol. 31, Nov 2019.

[7] 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*, 2014, pp. 263–274.

[8] T. Watanabe, E. Shioji, M. Akiyama, K. Sasaoka, T. Yagi, and T. Mori, "User blocking considered harmful? An attacker-controllable side channel to identify social accounts," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2018, pp. 323–337.

[9] J. V. Monaco, "What are you searching for? A remote keylogging attack on search engine autocomplete," in *28th Security Symposium USENIX Security 19*, 2019, pp. 959–976.

[10] Z. Qian and Z. M. Mao, "Off-path TCP sequence number inference attack: How firewall middleboxes reduce security," in *2012 IEEE Symposium on Security and Privacy*, 2012, pp. 347–361.

[11] L. C. Loh, "Autocomplete: Dr Google's "helpful" assistant?" *Canadian Family Physician*, vol. 62, no. 8, pp. 622–623, 2016.

[12] S. Chen, R. Wang, X. Wang, and K. Zhang, "Side-channel leaks in web applications: A reality today, a challenge tomorrow," in *2010 IEEE Symposium on Security and Privacy*, 2010, pp. 191–206.

[13] H. Çeker and S. Upadhyaya, "User authentication with keystroke dynamics in long-text data," in *2016 IEEE 8th International Conference on Biometrics Theory, Applications and Systems (BTAS)*, 2016, pp. 1–6.

[14] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 815–823.

[15] S. E. Oh, S. Li, and N. Hopper, "Fingerprinting keywords in search queries over tor," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 4, pp. 251–270, 2017.

[16] Tor Project, "Anonymity online," Accessed Mar. 27, 2021. [Online]. Available: torproject.org/

[17] C. M. Tey, P. Gupta, D. Gao, and Y. Zhang, "Keystroke timing analysis of on-the-fly web apps," in *International Conference on Applied Cryptography and Network Security*. Springer, 2013, pp. 405–413.

[18] M. Gaunt, "Introduction to fetch()," Accessed Mar. 20, 2019. [Online]. Available: https://developers.google.com/web/updates/2015/03/introduction-to-fetch

[19] J. V. Monaco, "Feasibility of a keystroke timing attack on search engines with autocomplete," in *2019 IEEE Security and Privacy Workshops (SPW)*, 2019, pp. 212–217.

[20] C. Herley and P. C. Van Oorschot, "SoK: Science, security and the elusive goal of security as a scientific pursuit," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 99–120.

[21] J. Demme, R. Martin, A. Waksman, and S. Sethumadhavan, "Side-channel vulnerability factor: A metric for measuring information leakage," in *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, 2012, pp. 106–117.

[22] K. Zhang, Z. Li, R. Wang, X. Wang, and S. Chen, "Sidebuster: Automated detection and quantification of side-channel leaks in web application development," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, 2010, pp. 595–606.

[23] P. Chapman and D. Evans, "Automated black-box detection of side-channel vulnerabilities in web applications," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, 2011, pp. 263–274.

[24] T. V. Goethem, C. Pöpper, W. Joosen, and M. Vanhoef, "Timeless timing attacks: Exploiting concurrency to leak secrets over remote connections," in *29th USENIX Security Symposium*, Aug. 2020, pp. 1985–2002. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/van-goethem

[25] D. Brumley and D. Boneh, "Remote timing attacks are practical," *Computer Networks*, vol. 48, no. 5, pp. 701–716, 2005.

[26] B. B. Brumley and N. Tuveri, "Remote timing attacks are still practical," in *European Symposium on Research in Computer Security*. Springer, 2011, pp. 355–371.

[27] D. X. Song, D. A. Wagner, and X. Tian, "Timing analysis of keystrokes and timing attacks on SSH," in *USENIX Security Symposium*, 2001, vol. 2001.

[28] A. M. White, A. R. Matthews, K. Z. Snow, and F. Monrose, "Phonotactic reconstruction of encrypted voip conversations: Hookt on fon-iks," in *2011 IEEE Symposium on Security and Privacy*, 2011, pp. 3–18.

[29] B. Dupasquier, S. Burschka, K. McLaughlin, and S. Sezer, "Analysis of information leakage from encrypted Skype conversations," *International Journal of Information Security*, vol. 9, no. 5, pp. 313–325, 2010.

[30] A. Aviram, S. Hu, B. Ford, and R. Gummadi, "Determinating timing channels in compute clouds," in *Proceedings of the 2010 ACM Workshop on Cloud Computing Security*, 2010, pp. 103–108.

[31] D. Asonov and R. Agrawal, "Keyboard acoustic emanations," in *2004 IEEE Symposium on Security and Privacy*, 2004, pp. 3–11.

[32] X. Gong, N. Borisov, N. Kiyavash, and N. Schear, "Website detection using remote traffic analysis," in *International Symposium on Privacy Enhancing Technologies*. Springer, 2012, pp. 58–78.

[33] X. Luo, P. Zhou, E. W. W. Chan, W. Lee, R. K. C. Chang, and R. Perdisci, "HTTPOS: Sealing information leaks with browser-side obfuscation of encrypted flows," in *Proceedings of Network and Distributed Systems Symposium (NDSS). The Internet Society*, 2011.

[34] J. V. Monaco and C. C. Tappert, "Obfuscating keystroke time intervals to avoid identification and impersonation," *arXiv preprint arXiv:1609.07612*, 2016.

[35] C. C. Tappert, M. Villani, and S.-H. Cha, "Keystroke biometric identification and authentication on long-text input," in *Behavioral Biometrics for Human Identification: Intelligent Applications*. IGI Global, 2010, pp. 342–367.

[36] A. Acien, A. Morales, R. Vera-Rodriguez, J. Fierrez, and J. V. Monaco, "Typenet: Scaling up keystroke biometrics," in *2020 IEEE International Joint Conference on Biometrics (IJCB)*, 2020, pp. 1–7.

[37] J. V. Monaco, "SoK: Keylogging side channels," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 211–228.

[38] J. V. Monaco and C. C. Tappert, "The partially observable hidden Markov model and its application to keystroke dynamics," *Pattern Recognition*, vol. 76, pp. 449–462, 2018.

[39] N. Whiskerd, N. Körtge, K. Jürgens, K. Lamshöft, S. Ezennaya-Gomez, C. Vielhauer, J. Dittmann, and M. Hildebrandt, "Keystroke biometrics in the encrypted domain: A first study on search suggestion functions of web search engines," *EURASIP Journal on Information Security*, vol. 2020, no. 1, pp. 1–16, 2020.

[40] N. A. Laskaris, S. P. Zafeiriou, and L. Garefa, "Use of random time-intervals (RTIs) generation for biometric verification," *Pattern Recognition*, vol. 42, no. 11, pp. 2787–2796, 2009.

[41] S. E. Coull, M. P. Collins, C. V. Wright, F. Monrose, M. K. Reiter *et al.*, "On web browsing privacy in anonymized netflows," in *USENIX Security Symposium*, 2007, pp. 339–352.

[42] I. Grigorik, "Introduction to HTTP/2," Accessed Sep. 3, 2019. [Online]. Available: https://developers.google.com/web/fundamentals/performance/http2

[43] V. Dhakal, A. M. Feit, P. O. Kristensson, and A. Oulasvirta, "Observations on typing from 136 million keystrokes," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 2018, pp. 1–12.

[44] The Linux Kernel Development Company. The Linux Input Documentation, "Uinput module," Accessed Jan. 31, 2021. [Online]. Available: https://www.kernel.org/doc/html/latest/input/uinput.html

[45] P. Senin, "Dynamic time warping algorithm review," *Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA*, vol. 855, no. 1-23, p. 40, 2008.

[46] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intelligent Data Analysis*, vol. 11, no. 5, pp. 561–580, 2007.

[47] S. Khalid, T. Khalil, and S. Nasreen, "A survey of feature selection and feature extraction techniques in machine learning," in *2014 Science and Information Conference*, 2014, pp. 372–378.

[48] E. J. Daniel, C. M. White, and K. A. Teague, "An interarrival delay jitter model using multistructure network delay characteristics for packet networks," in *the 37th Asilomar Conference on Signals, Systems Computers*, 2003, vol. 2, pp. 1738–1742 Vol.2.

[49] S. Z. Li and A. Jain, Eds., *L2 Norm*. Boston, MA: Springer US, 2009, pp. 883–883.

[50] G. Petneházi, "Recurrent neural networks for time series forecasting," *arXiv preprint arXiv:1901.00069*, 2019.

[51] C. Olah, "Understanding LSTM networks," Accessed Aug. 27, 2015. [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[52] H. Apaydin, H. Feizi, M. T. Sattari, M. S. Colak, S. Shamshirband, and K.-W. Chau, "Comparative analysis of recurrent neural network architectures for reservoir inflow forecasting," *Water*, vol. 12, no. 5, p. 1500, 2020.

[53] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*. PMLR, 2015, pp. 448–456.

[54] C.-Y. Wu, R. Manmatha, A. J. Smola, and P. Krahenbuhl, "Sampling matters in deep embedding learning," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2840–2848.

[55] A. Hermans, L. Beyer, and B. Leibe, "In defense of the triplet loss for person re-identification," *arXiv preprint arXiv:1703.07737*, 2017.

[56] C. Wang, X. Zhang, and X. Lan, "How to train triplet networks with 100k identities?" in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2017, pp. 1907–1915.

[57] O. Moindrot, "Triplet loss and online triplet mining in tensorflow," Accessed Mar. 19, 2018. [Online]. Available: https://omoindrot.github.io/triplet-loss

[58] M. Sikaroudi, B. Ghojogh, A. Safarpoor, F. Karray, M. Crowley, and H. R. Tizhoosh, "Offline versus online triplet mining based on extreme distances of histopathology patches," in *International Symposium on Visual Computing*. Springer, 2020, pp. 333–345.

[59] TensorFlow, "tfa.losses.TripletSemiHardLoss," Accessed Jan. 15, 2021. Available: https://www.tensorflow.org/addons/api_docs/python/tfa/losses/TripletSemiHardLoss

[60] B. Harwood, V. Kumar BG, G. Carneiro, I. Reid, and T. Drummond, "Smart mining for deep metric learning," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2821–2829.

[61] S. Narkhede, "Understanding AUC-ROC curve," *Towards Data Science*, vol. 26, pp. 220–227, 2018.

[62] Arduino, "Home page," Accessed Feb. 3, 2021. [Online]. Available: https://www.arduino.cc/

[63] J. V. Monaco, "Biologger," GitHub, Accessed Aug. 27, 2020. [Online]. Available: https://github.com/vmonaco/biologger

[64] Internet Engineering Task Force(IETF), "Hyper transfer protocol version2 (http/2)," Accessed May. 2015. [Online]. Available: https://tools.ietf.org/html/rfc7540

[65] C. Banse, D. Herrmann, and H. Federrath, "Tracking users on the internet with behavioral patterns: Evaluation of its practical feasibility," in *IFIP International Information Security Conference*. Springer, 2012, pp. 235–248.

[66] B. Krishnamurthy and C. E. Wills, "Generating a privacy footprint on the internet," in *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, 2006, pp. 65–70.

[67] Y. C. Yang, "Web user behavioral profiling for user identification," *Decision Support Systems*, vol. 49, no. 3, pp. 261–271, 2010.

[68] Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li, "Efficient subgraph matching on billion node graphs," *arXiv preprint arXiv:1205.6691*, 2012.

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California