User Identification in Dynamic Web Traffic via Deep Temporal Features

Jihye Kim*
Korea Army Academy
Yeongcheon, Republic of Korea

John V. Monaco
Naval Postgraduate School
Monterey, CA

***Abstract*—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. These requests are often made on behalf of user actions, such as typing on the keyboard or pointing at an HTML element. As a result, the timings of the user's actions are strongly correlated with the timings of packets that carry these events. In this work, we examine several dynamic web applications and the ability to measure human behavior in encrypted network traffic by using deep temporal features. Our approach relies on the ability to accurately detect a subset of packets that correspond to user actions. Leveraging recent work in keystroke dynamics, we show that user identification can be performed with modest accuracy utilizing the packet timings induced by a user typing into a search engine. While this tool could be used by forensic investigators to perform target identification among encrypted network traffic, it also raises a privacy concern in which an on-path remote adversary able to detect these packets may infer user behaviors.**

I. Introduction

Web applications that process sensitive information have become prevalent. Many of these applications operate in real time, with components split between the client and the server. Consequently, side-channel attacks across the web are getting more attention [1]. Encrypted web traffic can reveal surprisingly substantial information that threatens user privacy. User inputs are exposed primarily through packet timing and size side-channels. This leads to serious security challenges such as device/website fingerprinting [2], [3], [4], [5], user identification [6], content extraction [7], and session hijacking [8]. Some of these issues arise from the use of Asynchronous JavaScript and XML (Ajax) [9]. Ajax enables browsers to send and receive data from the server after the initial web page has been loaded. Often times, this traffic takes on the form of a series of much smaller requests in response to user input.

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. These requests are often made on behalf of user actions, such as typing on the keyboard or clicking on HTML element. It is common practice for websites that accept text input within a form to provide an "autocomplete" feature that suggests a pre-populated list of values based on previous aggregate user inputs and the content of web pages [10]. For example, many search engines contain an autocomplete graphical user interface (GUI) widget that generates traffic in response to a single keystroke or mouse click. This traffic contains features that may reveal each state transition within the application. As a result, a remote observer can infer sensitive user inputs or server responses based on time or size properties of the Ajax requests, for example by associating the size of an autocomplete server response with individual characters to recover a user's search query [1].

In this work, we examine the extent to which remote user identification of dynamic web traffic can be performed via deep temporal features. This is possible because the user's keystroke dynamics are revealed by the temporal patterns of Ajax request packets. We demonstrate that these packets, triggered by keyboard input, can be accurately detected within a TCP connection maintained by the browser, and as more Ajax packets are gathered the ability to identify individual users increases substantially. We propose a system that can automatically detect these packets and either identify or verify the user based on packet timings. Ajax packet detection is performed by taking the longest increasing subsequence (LIS) of packet sizes within a TCP connection, as packet size gradually increases with each subsequent request. Recent advances in keystroke dynamics are leveraged to perform user identification [11]. We utilize a recurrent neural network (RNN) trained with triplet loss to extract deep temporal features from the sequence of detected packet timings. Considering the encrypted traffic generated by a user typing into a search engine, we show that user identification can be performed with accuracy that rivals previous generation keystroke dynamics systems and verification accuracy nears the performance attained with timestamps measured on the host. To summarize, our contributions are:

- We discuss the characteristics of modern dynamic web traffic and how this is often correlated with user behavior.
- We develop a method to detect Ajax packets in search engines and describe a general technique that can be used to match Ajax packets to user input events.
- Using the detected packets, we perform user identification and verification with up to 500 users.
- Through analysis of several websites and an ablation study, we propose several mitigations that do not severely impact the performance of the website.

*Work done while a student at the Naval Postgraduate School.

The rest of the paper is organized as follows: Section II briefly reviews background and related works to our approach, including dynamic web traffic, remote side-channel attacks, and keystroke biometrics. Section III describes our methodology in detail, including the threat model, packet detection, and user identification. Section IV summarizes the results achieved using our approach. Section V discusses the results in the context of both forensic applications and privacy implications. Finally, Section VI concludes the paper by proposing several mitigations and identifying areas of future work.

II. Background and related work

A. Dynamic web traffic

Web traffic can be analyzed through either deep packet inspection [12] or header-based features [13], [14]. Header-based traffic analysis is increasingly used and has become more relevant to user privacy with the growing usage of strong encryption that makes content-based traffic analysis difficult [15]. Features derived from packet headers, such as inter-arrival time and payload size, may reveal specific individual application functionality (e.g., website fingerprinting [16]) as well as user and device behavior. Thus, it is worthwhile to investigate header-based features in classifying *dynamic* web traffic.

Web browsers are the primary tool through which most Internet users access web servers. Modern web browsers (i.e., Google Chrome, Mozilla Firefox, Microsoft Edge, Apple Safari) provide a JavaScript application programming interfaces (API) that enables the browser to make 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 autocomplete, chat applications, and mapping services.

As an example, 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 [9]. Ajax provides a means of implementing asynchronous client-server communication to send the partially completed query to the server and retrieve the list of search suggestions before the form is submitted. This web traffic carries the user input to the server and updates to the document object model (DOM) on the client.

Ajax broadly refers to the ability of a web page to make asynchronous updates. The API utilized and 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 by using three interfaces: headers, request, and response, which correspond to the core components of HTTP. However, Fetch implements only a subset of XHR capabilities. Most recent browsers are compatible with both XHR and Fetch APIs.

Ajax requests that are invoked by user input (mouse click, keypress, etc.) may reveal the timing of the user's action. The extent to which packet times are correlated with user actions depends on how much variability there is in the latency between the action and the packet being sent over the wire. There may be many factors involved, such as the client hardware, system load, and network congestion. Most importantly, however, is the way in which the web page processes the input event and generates an Ajax request. The event processing model describes the way in which the browser detects input events before making any request [19].

There are generally two kinds of processing models: 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 keydown or keyup 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 keydown and keyup events. We found that *Google Search* uses a callback registered to keydown events, consistent with prior work and that *DuckDuckGo Search* also uses a callback registered to keydown events, which differs from prior work in which a callback was registered to keyup events [19].

B. Remote side-channel attacks

Side-channel attacks have been widely studied using a variety of information sources, such as timing, power analysis, electromagnetic, and acoustic [20]. Remote side-channel attacks, or those that can be carried out over a network, have become increasingly important [1]. This kind of attack generally utilizes header-based traffic analysis to extract information such as packet inter-arrival timings or packet sizes.

Chen et al. [1] reviewed the status and direction of side-channel leaks in web applications considering both upstream and downstream traffic. The root causes of many side-channel vulnerabilities examined are based on fundamental design features of web applications: frequent small interactions, stateful communication, and diversity in the contents exchanged during state transitions. Chen et al. examined actual information leaks by constructing an attack on several high-profile web applications, including health, tax, investment, and search engines. Unique "web flow vectors" were obtained by clicking through different page elements. The user's actions are reconstructed by an attacker who identifies the packet sizes that match server responses, which may carry suggestion lists and other page updates. While this approach detects Ajax events from traffic emitted by the server, our approach considers only traffic emitted by the client.

Meng et al. [17] investigated the feasibility of recovering personalized keystroke timing information by using

Google Suggestions (GS), one example of an interactive rich JavaScript application. They analyzed the timing side-channel of GS by reverse-engineering the communication model from obfuscated JavaScript code. In their experiment, 11 participants were asked to install a plugin on their browser to capture the keystroke timings of GS queries and their keystrokes were collected. For each key pair with at least 20 samples, the mean inter-keystroke timing was determined with an error of less than 20% from their experiments. This result suggests that the timing of the queries over the network can reconstruct a user's typing pattern. In this 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.

The automated detection of side-channel leaks expands to side-channel attacks that threaten user privacy. Song et al. [21] studied timing attacks on Secure Shell (SSH) which is designed to provide a secure channel between two hosts. Taking advantage of the fact that encrypted SSH packets leak the size of payload and inter-keystroke timing information, they 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.

Recent work has been conducted on web search engines to examine the feasibility of keystroke timing attacks on HTTP traffic [19] later extending this to a complete attack [7]. For the experiment, 1,000 queries were collected on each of five different search engines (Google, Bing, DuckDuckGo, Baidu, and Yandex) that implement autocomplete. The behavior of each website was characterized in the perspective of packet size, event processing model, event censoring, and information gain. For some of the search engines examined, keystrokes can be detected based on the increasing pattern of packet sizes, and the key-press time intervals are faithfully preserved in the packet inter-arrival times. The vulnerability of websites that generate dynamic traffic varies significantly based on the accuracy of Ajax packet detection.

C. Keystroke biometrics

As a form of behavioral biometrics, keystroke dynamics has been regarded as one of the most efficient and economical means of user identification [22]. 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 accuracy needed for wide usage deployment [11]. In this work, our method of user identification is based on the typing pattern of users as seen through Ajax request packets. Idiosyncratic typing behaviors, including the temporal dynamics observed through the keyboard, are observed among Ajax packtes and enable user identification.

Keystroke biometric systems are largely categorized as either fixed-text and free-text. Fixed-text systems are those in which a pre-defined string is typed, such as a username, password, or PIN. In comparison, free-text systems are designed to identify or verify users typing any text. Prior works on free-text systems utilize timing features based on the interval between key-press and release events (i.e., the duration a key is held down for), as well as the latency between consecutive key-presses (i.e., the flight time between successive keystrokes). In many works, these features are conditioned either on particular keys or key groups [22], [23], [11].

Acien et al. [11] developed a user authentication system, TypeNet, for free-text keystrokes based on a Siamese RNN architecture. This approach was effective when scaling up to 100k users, achieving close to a 5% error rate on average. It outperforms previous state-of-the-art algorithms and approaches the performance of fixed-text systems. We leverage a variant of the TypeNet model developed in [11] 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 [11] by discretizing key-press time intervals through rounding to reduce noise introduced by packet jitter, introduce an embedding layer in the model, and train the model using triplet loss with online triplet mining.

Recently, Whiskerd et al. [24] examined the ability to utilize keystroke biometrics on web search engine traffic from desktop computers and mobile devices. They conducted an experiment in two scenarios: on encrypted network traffic by using packet metadata, and with decrypted traffic through an HTTPS proxy (mitmproxy). They 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 using conventional machine learning classifiers (naive Bayes and k-nearest neighbor). The identification performance resulted in error rates of 5-24% depending on the scenario. 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 these works in three important ways. 1) We utilize only the timing of Ajax packets, which correspond to key-press timings; therefore, duration features cannot be taken because key-release 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 similar to [25], which makes use of a sequence of key-press timings to identify users. 3) Our sequence-based feature extraction model can be leveraged for both user identification and verification scenarios.

III. Methodology

A. Threat model

We assume a remote passive observer 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 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 observer can detect the Ajax packets by inferring the patterns of packet size among TLS traffic.

Detecting Ajax packets over a TCP connection rather than individual page loads enables an observer to acquire key-press timings that potentially span multiple search queries. This obviates the need to detect individual page loads, which is difficult in practice [26], [5]. 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. This concept is based on the HTTP persistent connection, which allows multiple requests/responses to be sent over a single TCP connection rather than creating a new connection for each request. The HTTP/2 protocol allows multiple concurrent requests/responses to be multiplexed within a single TCP connection [27]. HTTP/2 multiplexing is supported by most modern browsers including Google Chrome that we used for the experiment. The observer 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.

B. 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 [28]. 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 totalling 810 characters on average. The dataset contains a wide variety of typing speeds, ranging from 1.5 to 22 keystrokes per second. 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, since these do not result in Ajax requests and can unintentionally invoke keyboard shortcuts.

The process for data collection is summarized in Figure 1. Data collection was performed over approximately five days on an Intel NUC10FNK running Ubuntu 20.04 LTS. We replayed keystrokes from the 500 user dataset by emitting key-press and key-release events in real time through the uinput module [29]. The uinput module is a kernel module that enables the creation of virtual devices and emulating device events from user space

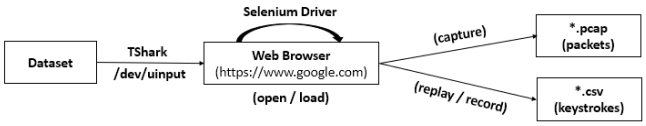

Fig. 1: Process to replay keystrokes and capture network traffic from dynamic websites.

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 Google Chrome (v.87). 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 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 filter 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 record traffic from 15,000 search queries (500 users $\times$ 15 sessions $\times$ 2 browsers).

C. 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 longest increasing subsequence (LIS) of packet sizes within the TCP connection since background traffic generally does not exhibit this behavior. This approach has previously achieved 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.

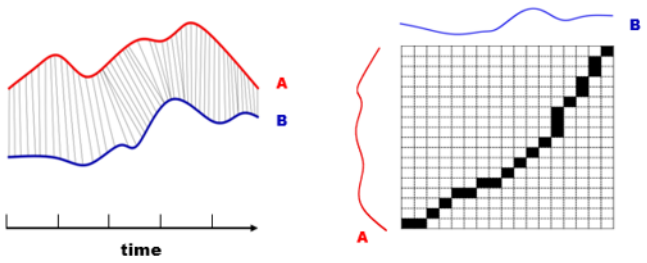

Fig. 2: A graphical description of Dynamic Time Warping

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 arranging the timestamp sequences (host vs packet) to identify regions of similarity. We adapted dynamic time warping (DTW) that is used for an optimal alignment between two temporal sequences which may vary in length. 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.

Figure 2 shows an example of DTW alignment: (left) is the general depiction of aligning two sequences, and (right) shows the optimal warping path between two time-series with the minimum cost. Through DTW alignment between key-press timings and packets on *Google Search*, we determine that the average delay between matched events in each sequence is from 10ms to 15ms, i.e., packets are emitted roughly 10ms after key press events. We use this approach as another source of ground truth to evaluate the LIS approach and find that LIS compared to DTW has over 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.

D. User identification

1) Feature extraction: We build and train a recurrent neural network (RNN) that extracts a vector of deep temporal features from a sequence of timestamps. An RNN is suitable for temporal data, such as a time series, because the inputs can vary in length. This makes it suitable for training the dataset we used that consists of free-text keystrokes.

The intervals between packet timings are first computed. 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

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

where the sequence τ has length N. This requires $N+1$ timestamps. Note that if the Ajax packets corresponded exactly to key-press times, τ would represent the sequence of key-press latencies, i.e., time between successive keydown events.

TABLE I: LSTM network architecture.

Layer	Output Shape	Num. 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

Because of packet jitter, which may result from variations in event processing on the host, the intervals τ do not match exactly the key-press intervals that would be obtained on the host. To mitigate this, we quantize τ by rounding the intervals to the nearest increment of b ms, forming the sequence of discrete tokens:

$$s_i = \left\lfloor \frac{\tau_i}{b} + 0.5 \right\rfloor \tag{2}$$

where $b = 5$ms. We choose b that 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. We additionally cap τ_i at a maximum of 2 seconds. There are two reasons why we set the maximum of 2 seconds for the tokens. First, most keystroke time 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 function f that outputs a fixed-length vector

$$f(\mathbf{s}) = \mathbf{x} \tag{3}$$

where $\mathbf{x}$ is an L2 normalized feature vector of length 128. Distances between the embedded vectors form the basis for user identification and verification.

2) LSTM architecture and triplet training: The function f is a RNN based on the TypeNet model developed in [11] which contains two stacked long short-term memory (LSTM) layers with batch normalization before each layer and dropout between the LSTM layers. Because we tokenize the intervals, we introduce an embedding layer as the first layer of the network. 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. The structure of our model is shown in Table I.

The model is trained using triplet loss, a method that learns to rank distances between samples belonging to the same or different classes [30]. Triplet loss is effective when the number of samples per class is small [31]. In our case, we assume

there are only two samples per class, which represent two separate TCP connections. During each batch of training, the model is presented with triplets, each of which include an anchor sample, a positive sample belonging to the same class as the anchor, and a negative sample belonging to a different class than the anchor. The triplet loss function forces the anchor-positive distance to be smaller than the anchor-negative distance.

Model training is performed with approximately 117k 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 key-press timings are utilized for training. This differs from previous work on keystroke biometric systems in which both key-press and key-release timings are utilized. We train the model for 150 epochs with adaptive moment estimation (Adam) optimization, 256 batch size, and online semi-hard triplet mining [30].

3) 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 metrics for evaluation: user identification and user verification. Identification accuracy is measured by rank-1, rank-5, and rank-50 classification accuracy. In user verification, the goal is to verify that a given sequence of Ajax packets belongs to a particular user identity. Verification performance is measured by balanced accuracy, i.e., 1 - the equal error rate (EER), the point at which the rates of false positive and false negative are equal on the receiver operating characteristic (ROC) curve obtained by varying a distance threshold d.

IV. Results

A. Event packet detection

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, we observed that LIS detection can fail under several conditions: background packets with similar size (false positive), Ajax packets with a different size (false negative), and multiple key-press events buffered into a single Ajax request (a kind of false negative). After typing about 15 characters on *Google Search*, the Ajax packets increase by about 20 to 25 bytes. This is due to an additional parameter "gs_mss" added to the request URL [19]. Next, we found on several websites that implement dynamic content that multiple keys are merged into a single packet when the typing speed exceeds a threshold. The speed at which this occurs differs per website: on *DuckDuckGo Search* (implements an autocomplete feature similar to *Google Search*) and *Google Docs* (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. *Google Search* has a comparatively lower threshold resulting in fewer false negatives.

Finally, because we did not exclude Backspace and Delete keys from our data capture, Ajax packet size could potentially decrease when edits are made to a partially completed query. Figure 3 compares typical packet size growth (user1) with fast typing and error correction (user2), showing that Ajax packet sizes do not always follow a steady increase in size. 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, and a general and robust method of Ajax packet detection remains an item for future work.

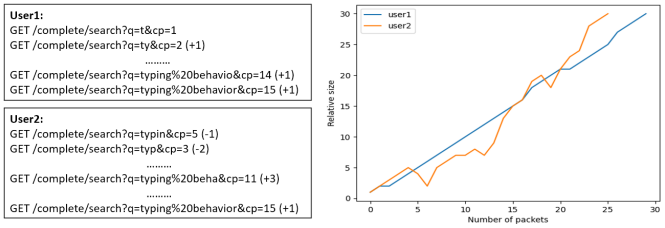

Fig. 3: Query scenarios in diverse user behavior: slow typing (user1) vs fast typing with error correction (user2)

TABLE II: 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

We compare the relative performance of each detection method: LIS, DTW alignment, and payload inspection. The accuracy, false positive rate (FPR) and false negative rate (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 II summarizes these results.

The LIS method achieves 96.0% accuracy with 6.5% FNR on average, which is high enough to support user identification by using the temporal features of detected packets. However, the result did not show perfect 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.

B. User identification and verification

User identification and verification are performed by taking the Euclidean distance between the query sample feature vector 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% (5 profiles) and 10% (50 profiles), respectively, when comparing to the

TABLE III: Summary of user identification (rank-N classification accuracy) and verification (balanced accuracy) performances. Packet timings=the model developed in this work using only Ajax packet time intervals; Press timings=key-press times obtained on the host, which would match packet timings with perfect Ajax packet detection and no timing jitter; All timings=key-press and key-release timings on the host using a TypeNet model.

Features	Identification Accuracy			Verification Accuracy
	Rank-1	Rank-5	Rank-50	
All timings (on host)	87.4	98.2	99.4	98.2
Press timings (on host)	59.0	88.6	99.8	97.8
Packet timings	**49.0**	**77.8**	**98.6**	**96.0**

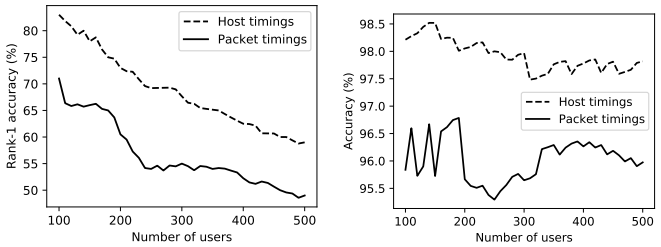

Fig. 4: Identification (left) and verification (right) accuracy as the number of users is scaled up.

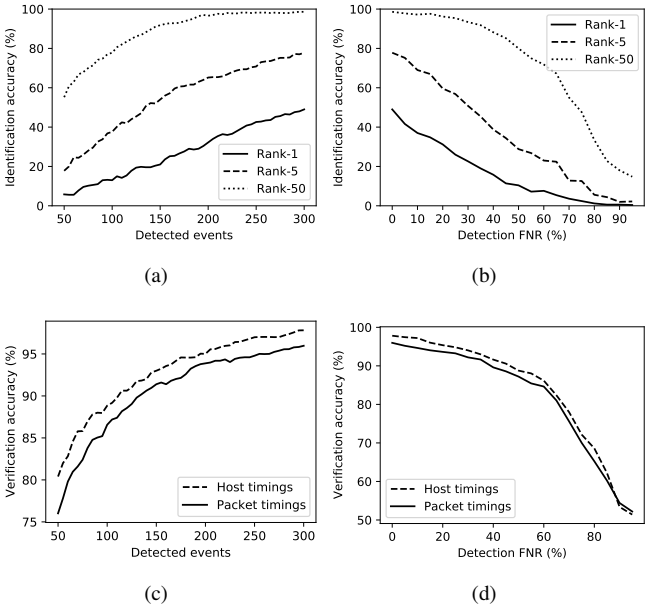

Fig. 5: Identification accuracy vs sample length (a) and detection rate (b), and Verification accuracy vs sample length (c) and detection rate (d).

query to the 500 user profiles. User verification performance is evaluated by balanced accuracy.

We compare user identification and verification performance with packet timings obtained from detected Ajax packets to the key-press timings as measured on the host. This scenario reflects the performance that could be achieved with perfect Ajax packet detection and no additional noise introduced by packet timings. We additionally evaluate the performance obtained by the TypeNet model developed in [11] which utilizes both key-press and key-release timings as well as key codes. Note that all timings scenario leverages keystroke temporal features on the host, not any features of Ajax packets.

Table III summarizes user identification and verification performance for three different scenarios. In all scenarios there are 500 users with 1 training sample (one-shot learning) and 1 testing sample, each comprised of 300 keystrokes. Rank-1 identification accuracy with packet timings is 49%, while using timings 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, i.e., the set of user profiles can be accurately reduced by up to 90%.

Figure 4 summarizes identification and verification accuracy 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 which is consistent with the results in [11].

Aggregating several hundred Ajax packet timings is key to accurate user identification/verification. We scaled the size of each sample (both training and testing) from 50 packets to 300 packets, shown in Figure 5. 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 (b), where we evaluate accuracy as the FNR of Ajax packet detection increases. Rank-1 identification accuracy is halved at approximately 25% Ajax packet FNR.

Verification accuracy decreases with the shorter sample lengths by about 75%, which is a similar result of the higher identification accuracy with the longer sample lengths, as described in Figure 5 (c). When varying Ajax packet detection FNR, verification accuracy declines to about 50% (chance accuracy for binary classification).

V. Discussion

For Ajax packet detection, a more general and robust method 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 to account on the website, 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. Thus the accuracy of Ajax packet detection depends on both temporal noise and differences in packet size. Extending Ajax packet detection to a real-world environment remains an item for future work.

Although identification and verification accuracy using timings only on the host is higher than packet timings, several

results stand out and warrant further examination. In user identification, Rank-50 accuracy of all timings is lower than that of key-press timings on the host. This is not a significant difference and may be attributed to model variance, i.e., differences in model performance based on different initial parameters, dataset shuffling, etc. In user verification, the accuracy is not consistently proportional to the number of users. We can observe that accuracy fluctuates, also likely attributed to differences among users as a single new user is added to the population. However, verification accuracy tends to stabilize with a larger number of users, consistent with [11].

A. Forensic applications

Identifying and locating cyber adversaries has become increasingly difficult due to anonymizing technologies. A forensic investigator aiming to locate a particular target or verify the presence of a specific adversary faces numerous challenges when that individual takes measures to hide their identity or obfuscate geolocation (e.g., by using a VPN or other form of proxy). Relying on user behavioral metrics as observed in network traffic could form the basis for locating a target or attributing two different TCP sessions to the same user, although it is not clear how and if this technique would support litigation.

The general technique we described may also be applicable to scenarios beyond the on-path observer described. We assumed that an observer is able to capture HTTPS traffic, i.e., HTTP over TLS, which encrypts at the transport layer and leaves TCP/IP headers exposed. However, there may be scenarios where an observer has access to traffic encrypted at the link layer, e.g., using the WPA2 (Wi-Fi Protected Access II, part of the 802.11i standard). In this case, TCP/IP headers are encrypted, but packet sizes are still exposed. Therefore, a size-based detection could still be performed, albeit on all the traffic generated by a particular device (based on MAC address) rather than a particular TCP connection. The LIS method described could then still be utilized as we have found the presence of background traffic to have little effect on Ajax packet detection accuracy.

B. Privacy implications

Like other biometric modalities, e.g., face recognition and gait analysis, the ability to identify users among encrypted web traffic can reduce user privacy on the Internet. User identification can be leveraged as an intermediate goal to achieve better personalization or target identification [32]. Behavior information can be combined with other tools to enhance performance, especially to identify/verify users on a large scale. There are several privacy implications from an adversarial point of view.

First, we consider casual users who access the Internet through a public or untrusted network. An adversary who observes the dynamic network traffic can analyze user behaviors by detecting Ajax packets, and subsequently track users. This form of tracking users through behavioral patterns is more more difficult to evade since it does not need explicit tracking techniques, such as HTTP cookies, and has come under increased scrutiny in recent years [33], [34].

In addition to the causal user, there are risks to privacy-conscious users who take extra measures to conceal their identity. Users who connect to the Internet through a VPN are at risk of leaking their identity so long as the VPN does not actively modify the packet sizes, which would make detection more difficult, or timings, which would make identification more difficult if detection were still possible. This is concerning especially for users who seek to circumvent censorship, such as in regimes where Internet access is strictly regulated.

Lastly, users on the Tor network, which is an overlay network designed to obfuscate location [35], [36], may be deprived of anonymity in the presence of dynamic web traffic since Tor exposes the packet 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.

VI. Conclusion

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 isolate Ajax packets with high accuracy, presumably through a distinct pattern of packet sizes, and the preservation of key-press latencies in the Ajax packet time intervals. When surveying potential candidates that exhibit both characteristics, we examined several websites in which this kind of attack becomes much more difficult primarily due to the lack of the second. These sites suggest that a simple mitigation is to implement a timeout mechanism for dynamic content triggered by user input, merging multiple input events into a single Ajax request when the rate exceeds some threshold. In this way, original key-press latencies are not recoverable from the Ajax packets so long as typing speed exceeds the timeout. An alternative mitigation would be to add a random amount of padding or to normalize to the packet size (e.g., fixed Tor cell size), making Ajax packet detection more difficult. Packet size-based detection would largely fail as this is based on increasing packet size. This mitigation will cause decreased user identification capabilities since identification relies on Ajax detection accuracy.

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 web traffic over a larger and more diverse set of websites and web applications also remains an ongoing area of research. Extensions may be possible to web applications that provide diverse functionalities such as emailing services, gaming services, and chat applications. Among applications that have real-time communication between the client and server, there is potential for information leakage leaked about user identities and actions through encrypted network traffic.

REFERENCES

[1] 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.

[2] 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, 2005.

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] J. Baek, J. Kim, and W. Susilo, "Inspecting tls anytime anywhere: A new approach to tls interception," in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*. Association for Computing Machinery, 2020, p. 116–126.

[13] K. Borders and A. Prakash, "Quantifying information leaks in outbound web traffic," in *2009 30th IEEE Symposium on Security and Privacy*, 2009, pp. 129–140.

[14] J. Zimmermann, A. Clark, G. Mohay, F. Pouget, and M. Dacier, "The use of packet inter-arrival times for investigating unsolicited internet traffic," in *First International Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE'05)*, 2005, pp. 89–104.

[15] M. Nasr, A. Houmansadr, and A. Mazumdar, "Compressive traffic analysis: A new paradigm for scalable traffic analysis," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery, 2017, p. 2053–2069.

[16] X. Gong, N. Kiyavash, and N. Borisov, "Fingerprinting websites using remote traffic analysis," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, 2010, p. 684–686.

[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)*. IEEE, 2019, pp. 212–217.

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

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

[22] 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)*. IEEE, 2016, pp. 1–6.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

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

[28] 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.

[29] 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

[30] 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.

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

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

[33] 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.

[34] 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.

[35] 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.

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