2019 IEEE Security and Privacy Workshops (SPW)

Feasibility of a Keystroke Timing Attack on Search Engines with Autocomplete

John V. Monaco
Naval Postgraduate School

***Abstract*—Many websites induce the browser to send network traffic in response to user input events. This includes websites with autocomplete, a popular feature on search engines that anticipates the user's query while they are typing. Websites with this functionality require HTTP requests to be made as the query input field changes, such as when the user presses a key. The browser responds to input events by generating network traffic to retrieve the search predictions. The traffic emitted by the client can expose the timings of keyboard input events which may lead to a keylogging side channel attack whereby the query is revealed through packet inter-arrival times. We investigate the feasibility of such an attack on several popular search engines by characterizing the behavior of each website and measuring information leakage at the network level. Three out of the five search engines we measure preserve the mutual information between keystrokes and timings to within 1% of what it is on the host. We describe the ways in which two search engines mitigate this vulnerability with minimal effects on usability.**

I. Introduction

It is becoming increasingly apparent that preventing side channel attacks is a major security challenge. The unintended leakage of information between processes can expose secrets, enable unauthorized access, and violate user privacy. Part of this difficulty stems from the difficulty in reasoning about the physical systems upon which software is executed [1].

Many side channel attacks exploit device behavior. This includes microarchitectural attacks, which expose hardware state leaked through the timing of software events on a device [2], and physical side channels, which expose hardware state through external measurements such as power consumption [3] and acoustics [4].

A class of side channel attacks that exploit human behavior has also emerged. Such attacks recognize a user's actions from their behavior sensed either externally or on the device. This includes attacks that recognize the keys a user types on a keyboard or mobile device based on the position of the hands sensed through channels such as WiFi signal distortion, motion of the device, and timings of keystrokes [5].

The manifestation of human behavior in network traffic can enable such attacks remotely, which have primarily leveraged the sizes of encrypted packets [6]. We explore how packet inter-arrival time might also leak information about user behavior. Many websites respond to user input events in near-real time, which requires low latency communication with the server. In this model, the timing of input events is exposed in the network traffic from the client to the server.

Autocomplete is a feature that has been incorporated into almost every major search engine. This service provides suggested queries to the user as they type based on the partially completed query, trending topics, and the user's search history [7]. Intended to enable the user to find information faster, autocomplete requires the user's client to communicate with the server when input events are detected. As a result, the user's keystroke timings can manifest in network traffic, potentially enabling a remote keylogging side channel attack [5]. We determine the feasibility of such an attack in this work.

Considering the autocomplete feature of several popular search engines, we aim to address the following questions:

1. *How much information about a search query is leaked in the network traffic generated by autocomplete?* Prior work has measured the mutual information, or information gain, between keyboard keys and keystroke timings observed on the host [8], [5]. This work has assumed that these timings can be detected and are faithfully preserved in packet inter-arrival times. We test this assumption using actual network traffic generated by several major search engines and measure the amount of information that is lost after the keyboard events pass through the web browser in transit to the server.

2. *What kind of event processing model is used by major search engines to implement autocomplete?* Search engines, and dynamic websites in general, differ in the way input events are processed and communicated to the server. In this regard, characterizing the input event processing model of each search engine will enable a better understanding of how web application design considerations can lead to or mitigate side channel attacks.

3. *What defenses that are currently implemented, if any, mitigate such an attack?* Towards an effective defense against keylogging side channels, we identify ways in which a search engine is less vulnerable to attack. Specifically, we are interested in defense mechanisms that can mitigate such an attack without greatly decreasing usability.

II. Background and related work

A. Keylogging side channels

A keylogging side channel attack aims to recover the keystrokes of a victim using a channel outside of the intended keyboard event processing pipeline. Such attacks date back 75 years when it was demonstrated that keystrokes on a teletype terminal emitted a characteristic electromagnetic spike. They have since been demonstrated for a wide range of modalities such as acoustics, seismic activity, and hand motion [5]. These attacks generally fall into two different categories: spatial attacks, which utilize a channel that leaks information about

DOI 10.1109/SPW.2019.00047

where a key is located on the keyboard, and temporal attacks, which utilize a channel that leaks only the times of key press and release events.

Two main problems arise when trying to determine keystrokes from a side channel. The first is keystroke detection, a binary classification problem. In a packet trace, this involves deciding whether each packet contains a keystroke or not. The second problem is key identification: given that a keystroke occurred, determine which key it was. This is a multi-class classification problem.

Temporal keylogging attacks attempt to recognize which keys a user typed based only on the key press and release times. This attack may utilize timings sensed through acoustics, spikes in CPU load, or network traffic. In this regard, websites that emit network traffic in response to keyboard events, such as SSH in interactive mode or a search engine with autocomplete, may unintentionally leak information about keyboard events even when traffic is encrypted.

Temporal keylogging attacks are enabled by the similarity with which different people type on a keyboard. The typist is generally quicker to press keys that are far apart compared to keys that are close together, a consequence of having to reposition the hand or finger to strike neighboring keys [9]. This inverse scaling between key-distance and key-press latency is common among touch typists [10], enabling general inferences to be made about which keys were pressed based only on the time interval between key-presses.

The feasibility of a temporal keylogging attack is measured by the mutual information between timings and keys. Mutual information is given by

$$I\left[k;y\right] = H_0\left[k\right] - H_1\left[k|y\right]$$

where H_0 is the intrinsic entropy of symbol k and H_1 is the entropy of k conditioned on observation y. In this work, k is an ordered key pair, or *bigram*, such as "TH" and y is a time interval between keyboard events, such as the time from pressing "T" to pressing "H". The mutual information measures the ability to predict keys from timings.

B. Web search autocomplete

Many search engines have autocomplete functionality. With this feature, a list of suggested queries is presented to the user as the query input field changes. When the user presses a key on the keyboard, the client makes a request to the server and the server responds with a list of suggested search queries [11]. This results in a series of HTTP requests following keyboard events, such as those shown in Figure 1. In this example, an HTTP GET request is made upon each key press that results in a visible change to the query input field, i.e., modifier keys are ignored. The server response contains a list of suggestions aimed to anticipate the user's complete query.

Previously, it has been demonstrated that the size of the server response can leak a considerable amount of information about the query [12]. For each request, the attacker must only enumerate each possible key until a response with the same size is found, a search space that grows linearly with query size. However, this kind of attack is application dependent (among different search engines), time dependent (since suggestions change over time according to trending searches [7]), and user dependent (since suggestions generally depend on a user's search history [7]).

Using both outgoing and incoming traffic over Tor, traffic patterns could enable fingerprinting a limited set of keywords contained in the query [13]. This technique is also application dependent and requires a dictionary of target queries to be built. Unlike these works, we consider unrestricted queries and only examine the traffic emitted by the client.

The keystroke timings leaked by "on the fly" web apps, such as Google autocomplete, were modeled in [14]. This work investigated whether the latency distributions of particular bigrams could be recovered from network traffic generated by an autocomplete service. These may then be used in a keystroke biometric imitation attack. However, this work assumed that the traffic was unencrypted and measured only whether the distribution of time intervals associated with a particular bigram could be recovered. In the current study, we assume that the traffic is encrypted and measure the attacker's ability to identify the query.

III. Data collection methodology

We assume a remote passive attacker that can observe encrypted network traffic from a victim who types a query on a search engine with autocomplete. We built a system that captures network traffic while previously recorded keystrokes are replayed in the browser as they would be typed by a human subject. Since we aim to characterize only search engine behavior, and do not consider other network effects such as routing and buffering delays, we capture on the interface of the victim's machine and assume there is no background traffic. In future work, we plan to relax these assumptions.

The measurement setup consisted of: a keystroke dataset previously collected from human subjects, browser automation with Selenium WebDriver, and a low latency system to replay the keystrokes. Our data collection methodology assumes the victim types a query without corrections or selecting a search suggestion before the complete query is entered. We used a subset of a publicly available keystroke dataset collected from over 100k users typing excerpts from the Enron email corpus and English gigaword newswire corpus [15]. The timestamps in this dataset have millisecond resolution. From this dataset, we randomly selected 1000 phrases that contain between 5 and 50 printable characters, i.e., we do not consider sequences containing deletions or keys that may cause the cursor to change position, such as arrow keys. This selection contains a wide variety of typing speeds, ranging from 1.5 to 22 keys per second (KPS, averaged over each sample).

Each capture proceeded as follows. The web browser was opened and cookies cleared before starting the capture process (tshark). One second after the capture began, the website was loaded using the web driver. There was then a two second delay before replaying the keystrokes. The keystroke sequence was replayed by writing the sequence of key events to the

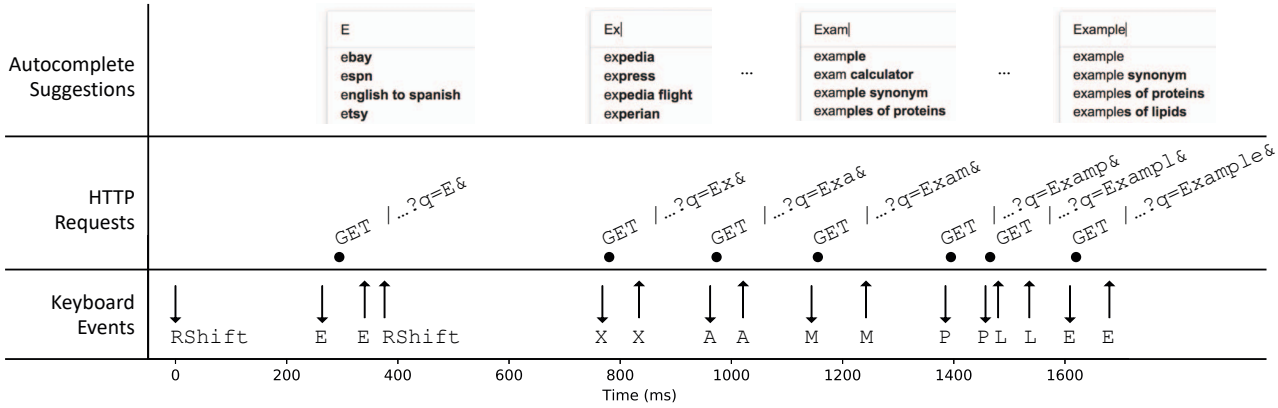

Fig. 1. Search autocomplete example. As a search query is entered, the client submits a request with the partially completed query and the server responds with a list of suggested queries. A request is only made upon visible changes to the query box, such the key press of a printable character (↓=press, ↑=release).

`uinput` device with delays between each event that correspond to the original keystroke sequence. The data collection was performed on an Ubuntu Linux desktop machine with kernel version 4.15 compiled with the CONFIG_NO_HZ=y option, which omits clock ticks when the CPU is idle. This ensured keyboard event times were replayed with high fidelity and not quantized due to the presence of a global system timer. Traffic was decrypted by setting the SSLKEYLOGFILE environment variable before each capture, which specifies a file to record the TLS session keys. Ground truth for the purpose of measuring keystroke detection accuracy and mutual information was obtained using the decrypted packets.

We collected 1000 queries on each of five different search engines: Google, Bing, DuckDuckGo (DDG), Baidu, and Yandex, all of which implement autocomplete albeit in different ways. To understand how the browser itself might affect network timings, the data collect was performed in both Chrome (v.71) and Firefox (v.64). All the search engines considered except Baidu currently support HTTP/2.

The dataset we collected contains a total of 10k queries (1000 keystroke sequences $\times$ 5 search engines $\times$ 2 web browsers), obtained over approximately 5 days. During this time, we did not experience any rate limiting. However, some captures did either miss some of the outgoing traffic or fail to completely decrypt (approximately 1%). These unsuccessful captures were repeated until the decrypted queries matched the original keystroke sequence.

IV. Analysis of website behavior

We characterize several aspects of search engine autocomplete behavior that, to our knowledge, have not been examined previously. We then evaluate the ability to detect keystrokes based on packet size and the ability to identify keystrokes from packet inter-arrival times.

A. Packet size

The problem of keystroke detection involves deciding whether each captured packet contains a keystroke event or not. As the user types, the client repeatedly sends HTTP GET requests that contain the partially completed query. The size of each request increases over the previous one since it contains the cumulative text that the user has typed. As a result, the sequence of packet sizes increases by about 1 byte with each request. All of the search engines we considered followed this behavior with some exceptions noted below.

We characterize the sequence of packet sizes emitted by each website by normalizing to the size of the first packet. That is, let s_i be the size in bytes of the ith packet and s_0 the size of the first packet. Relative packet sizes are given by $s_1 - s_0, s_2 - s_0, \ldots$. This sequence characterizes packet size as a function of query length, invariant to the size of the initial request which may vary across hosts due to different sized identifiers, authentication tokens, user agent string, etc. The relative packet sizes are shown in Figure 2.

We observed some variations to this general behavior. To initialize the autocomplete service, a website may include some additional parameters in the first request. These initialization parameters are then removed in subsequent requests. As a result, packet size initially decreases and then increases linearly thereafter. Both Bing and DuckDuckGo exhibit this behavior, noticeable by the sharp decline in packet size after the first request. In contrast, we found that Google includes additional parameters after a threshold is reached: after about 12 characters, an additional "gs_mss" parameter with the partially completed query at that point is included in each additional request. This results in a sudden increase of about 20 bytes (8 bytes for "&gs_mss=" and 12 bytes for the query), again increasing by about 1 byte per character thereafter.

Unlike other search engines, the autocomplete packet sizes of Baidu increase at about 2 bytes per character. This is due to the previous partially completed query being included in each request. For example, if the user types "cat", the request after pressing "t" will contain "?wd=cat&pwd=ca" where "pwd" refers to the previous partial query and "wd" to the current.

From these observations, keystroke detection may be accomplished by finding an increasing sub-sequence of packet sizes within the complete trace. Assuming that the time of

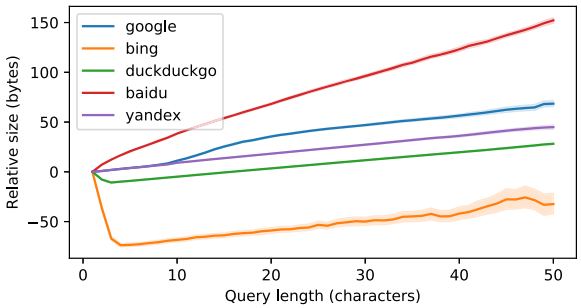

Fig. 2. Relative packet sizes of each search engine.

page load is known by the attacker and keystroke detection is applied only thereafter, we detect keystrokes by finding the longest increasing sub-sequence (LIS) of packet sizes, a problem that can be solved efficiently by dynamic programming. The LIS achieves near-perfect keystroke detection accuracy (F-score > 99%) for every website except Bing and DuckDuckGo. Taking into account the website specific behaviors above, such as the initial decrease in packet size, would further improve accuracy. The F-scores are summarized in Table I.

B. Event processing model

The event processing model describes the way in which the website detects and processes input events before making a request for autocomplete suggestions. Among the websites we examined, we found there are generally two kinds of event processing models: callback and polling.

In a *callback model*, an input event triggers a callback function responsible for sanitizing the query text and making an HTTP request to retrieve the list of autocomplete suggestions. The delay from the time of the input event to the time of request depends primarily on the execution time of the callback function. If this execution time does not vary between successive events, the time intervals between events is faithfully preserved in packet inter-arrival times.

In a *polling model*, the contents of the query input field are periodically checked at fixed intervals according to a timer cycle. When a change in the query field is detected, an HTTP request for autocomplete suggestions is made. Thus, the delay from input event to request depends on where in the cycle the event occurred. The packet inter-arrival times are closely aligned to some multiple of the timer period.

The event processing model introduces perturbations to the timing of keyboard events as seen in packet inter-arrival times. Perturbations may be caused by sampling noise, low frequency polling, or the presence of background processes (in the browser or on the host) with higher priority. We consider the spectral coherence to measure how much noise the event processing model introduces to the event times in each website. The spectral coherence measures the fractional part of the power spectral density of the keyboard event times that is preserved in packet timings. This reveals not only how much the keyboard event times are perturbed, but at what frequencies. The spectral coherence of each website is

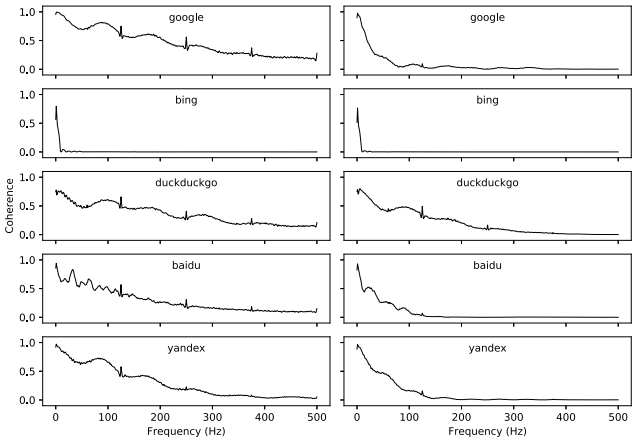

Fig. 3. Spectral coherence between keystroke events and autocomplete network traffic in Chrome (left) and Firefox (right).

shown in Figure 3. From this figure, we can make several observations:

- There is a gradual decay in coherence for Google, DuckDuckGo, Baidu, and Yandex. This indicates that these websites use a callback model, which faithfully preserves lower frequencies.
- That decay is generally steeper in Firefox than in Chrome. This indicates higher-frequency variations in the timing of input events in Firefox compared to Chrome.
- The spectral coherence quickly drops to 0 after 10 Hz for Bing in both browsers. This indicates a polling mechanism with clock rate around 100 ms, which cuts off frequencies above that range.
- There are apparent peaks at multiples of 125 Hz. We verified that these artifacts are present in the original keystroke dataset and were not introduced by our measurement setup. They are likely due to USB polling which is 125 Hz by default for low speed devices.

We examined the web page source code to verify the event processing model of each search engine. This involved beautifying the obfuscated JavaScript, setting breakpoints at places where XMLHttpRequest objects are created, and toggling callbacks to `keydown` and `keyup` input events. This revealed that Google, Baidu, and Yandex use callbacks on `input` events, DuckDuckGo uses a callback on `keyup` events, and Bing uses a polling mechanism with 100 ms timer. Note that `input` events are triggered immediately following `keydown` events [16]. Baidu additionally has a polling mechanism with 200 ms timer, seemingly as a fallback mechanism for when autocomplete requests are not triggered by other input events. Results are summarized in Table I.

C. Censoring

A temporal keylogging attack depends on the ability to observe keyboard event times as a victim types. The network traffic generated by a search engine autocomplete service reveals these timings as the client makes HTTP requests

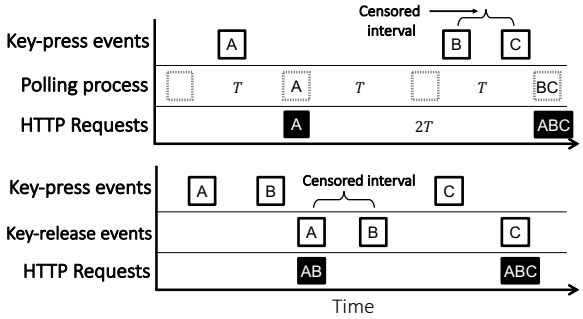

Fig. 4. Keyboard event time censoring. Polling censoring (top) occurs when typing speed exceeds polling rate in a polling event model. Rollover censoring (bottom) occurs when keystrokes overlap in a key-release callback model.

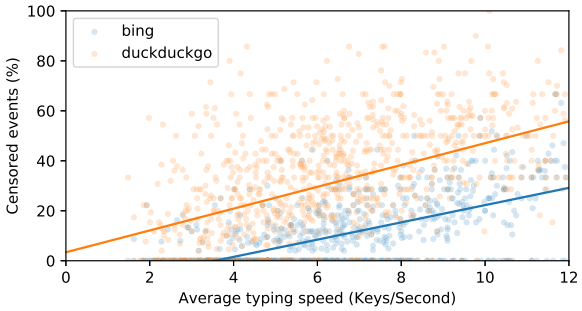

Fig. 5. Censoring rates of Bing and DuckDuckGo as a function of average typing speed. Each point is a keystroke sequence.

upon user input events. However, for some websites, if typing occurs too fast, characters could be merged into the same request and a single packet is generated for multiple keyboard events. When this occurs, some of the keyboard event times become *censored*[1] at the network level since they cannot be observed. The ability to perform a temporal keylogging attack is diminished.

We identified two ways in which keyboard event times may be censored. The first is through polling, when the typing speed of the victim exceeds the polling rate. This situation is shown in Figure 4 (top). The query input field is monitored for changes with polling interval T. When two key presses occur within a single polling window, the first event becomes censored since the following autocomplete request contains two additional characters instead of one.

Censoring can also occur in a callback model with hooks registered to `keyup` events. In such a model, censoring occurs when two consecutive keystrokes overlap, a typing phenomenon known as *key rollover*. An example is shown in Figure 4 (bottom). Key "B" is pressed after key "A" is pressed but before "A" is released. When key "A" is released, the query input field already contains "ab" since characters appear immediately following the `keydown` events. When the autocomplete request is made after releasing key "A", it contains the partial query "ab". Following this, key "B" is released; however this results in no visible changes to the contents of the query input field, so no request is made. As a result, the release of key "B" is censored.

Typing speed can affect both kinds of censoring. In a polling model, events become censored when two key press events occur within the same polling window, which is more likely to occur as typing speed increases. In a callback model triggered by `keyup` events, events become censored due to key rollover, a phenomenon characteristic of faster typists [15]. We measured the censoring rate (proportion of censored events) of Bing, which implements a polling model, and DuckDuckGo, which implements a `keyup` callback model. The positive relationships between censoring rates and typing speed are shown in Figure 5, an indication that faster typists are less prone to attack in both models. The queries in the keystroke dataset we used had an average 27% rollover ratio, calculated by the proportion of overlapping to total number of keystrokes.

[1]We borrow this term from survival analysis, a branch of statistics. In survival analysis, censoring occurs when the time of an event is not known.

D. Information gain

A temporal keylogging attack exploits the mutual information between time intervals and keyboard keys. In a remote attack, mutual information is diminished as noise is introduced to the packet inter-arrival times. This may occur through variations in *latency*. The latency is the delay from the input event on the victim's host to the observed packet arrival time.

Keyboard events are temporally buffered on the client (either implicitly or explicitly) from the time the event occurs until an HTTP request is made. In a callback model, this depends primarily on the time to sanitize the query and construct the request. In a polling model, this depends primarily on where in the polling window the event occurred. In both models, latency may also depend on request number: the delay of the first event is generally greater than the following events due to the time to setup the autocomplete service or to ready the user interface for search predictions [14].

As the latency varies, this introduces noise to the packet inter-arrival times. This form of *obfuscation* was previously proposed as a defense against temporal keylogging attacks [5], [17]. As more noise is introduced, the mutual information between time intervals and keys decreases. Note that obfuscation occurs only through *variations* in latency; if the latency remains constant, the time intervals between successive input events are preserved in the packet inter-arrival times.

The mean and standard deviation latency is shown for each website in Table I. Websites that implement a callback model have noticeably lower latency deviations than Bing, which implements a polling model. And while latencies are generally larger in Firefox than in Chrome, there is no consistent relationship between variation in latency and browser. The correlation between time intervals on the host and packet inter-arrival times is also shown in Table I.

We use the relative loss of mutual information to measure how much each website mitigates the possibility of attack. That is, let y_{pre} be the time intervals on the host and y_{post} be the packet inter-arrival times. The relative loss in mutual

TABLE I
RESULTS SUMMARY. CENSORING MEASURES PROPORTION OF CENSORED EVENTS. DETECTION MEASURES ABILITY TO DETECT UNCENSORED KEYSTROKES. MI LOSS IS THE REDUCTION IN MI FROM HOST TO NETWORK AND MEASURES THE INCREASED DIFFICULTY OF KEY IDENTIFICATION.

Website	Proto.	Event model	Censoring	Detection (F-score %)		Latency (ms)		Interval correlation		MI loss (%)	
				Chrome	Firefox	Chrome	Firefox	Chrome	Firefox	Chrome	Firefox
Google	HTTP2	Callback (keydown)	None	99.8	99.7	6.2±3.3	14±10	0.99	0.99	0.0	0.7
Bing	HTTP2	Polling (10 Hz)	11%	90.7	90.3	52±29	60±31	0.98	0.98	7.1	6.3
DDG	HTTP2	Callback (keyup)	32%	96.5	96.5	5.7±10.9	6.7±6.9	0.99	0.99	0.4	0.1
Baidu	HTTP	Callback (keydown)	None	99.4	99.9	15±20	20±25	0.99	0.99	0.3	0.5
Yandex	HTTP2	Callback (keydown)	None	99.9	99.0	10±12	13±10	0.99	0.99	0.6	0.5

information is given by $1 - I\left[k; y_{\text{post}}\right] / I\left[k; y_{\text{pre}}\right]$. A loss of 0% indicates that the mutual information on the network is completed preserved to what it was on the host, while a loss of 100% indicates that there is no shared information between keyboard keys and the time intervals on the network. These results are summarized in Table I, considering only uncensored intervals. With the exception of Bing, the mutual information on the network is within 1% of what it is on the host.

V. DISCUSSION AND CONCLUSIONS

Search engines that implement autocomplete suggestions using a `keydown`-based callback model (Google, Yandex, and Baidu) are susceptible to the same kind of keylogging side channel attack as those on the host [8], [18]. 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. Little information (< 1%) is lost compared to what would be available on the host. Note, this result indicates only that *if* a keystroke timing attack were successful on the host, then it would succeed on the network. The success of the attack on the host itself varies among users [5], and potentially other factors such as hardware, language, and keyboard layout, which have not yet been studied.

Among the search engines we considered, two seem to provide mitigation against a remote keylogging attack. Bing, which uses a polling model with 100 ms timer, has approximately a 10% censoring rate for a user typing at a speed of 6 keys per second. Not only does the polling model quantize event times, but it makes it practically infeasible to predict the keys that become censored. Reducing the polling rate would further increase censoring, although at the cost of increased latency to retrieve the autocomplete suggestions.

DuckDuckGo, which uses a `keyup` callback model, has a 30% censoring rate at a typing speed of 6 keys per second. While both Bing and DuckDuckGo are resilient to attack due to the presence of censored events, the `keyup` callback model seems to be an effective self-regulating mitigation strategy. Prior work has shown timing attacks to be more effective for faster touch typists compared to slower "hunt-and-peck" typists [5]. The faster typists also exhibit greater rollover [15], which in a `keyup` callback model has the effect of censoring the keyboard event times. Thus, as typing speed increases, rollover generally increases and so does censoring rate.

Future work will consider the performance of an actual attack and examine how the relatively low entropy of natural language could be leveraged to increase attack success.

REFERENCES

[1] C. Herley and P. van Oorschot, "Sok: Science, security and the elusive goal of security as a scientific pursuit," in *Proc. IEEE Symp. on Security & Privacy (SP)*. IEEE, 2017, pp. 99–120.

[2] Q. Ge, Y. Yarom, D. Cock, and G. Heiser, "A survey of microarchitectural timing attacks and countermeasures on contemporary hardware," *Journal of Cryptographic Engineering*, pp. 1–27, 2016.

[3] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology — CRYPTO' 99*. Springer, 1999, pp. 388–397.

[4] D. Genkin, A. Shamir, and E. Tromer, "RSA key extraction via low-bandwidth acoustic cryptanalysis," in *Advances in Cryptology – CRYPTO 2014*. Springer Berlin Heidelberg, 2014, pp. 444–461.

[5] J. V. Monaco, "Sok: Keylogging side channels," in *Proc. IEEE Symp. on Security & Privacy (SP)*. IEEE, 2018.

[6] 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*. IEEE, may 2011.

[7] "Search using autocomplete," https://support.google.com/websearch/ answer/106230, Accessed: 2018-12-17.

[8] D. X. Song, D. Wagner, and X. Tian, "Timing analysis of keystrokes and timing attacks on ssh." in *Proc. Usenix Security Symp.*, 2001.

[9] T. A. Salthouse, "Perceptual, cognitive, and motoric aspects of transcription typing." *Psychological bulletin*, vol. 99, no. 3, p. 303, 1986.

[10] J. V. Monaco, M. L. Ali, and C. C. Tappert, "Spoofing key-press latencies with a generative keystroke dynamics model," in *Proc. IEEE 7th Intl. Conf. on Biometrics Theory, Applications and Systems (BTAS)*. IEEE, 2015, pp. 1–8.

[11] S. D. Kamvar *et al.*, "Anticipated query generation and processing in a search engine," U.S. Patent 7,836,044, 2004.

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

[13] 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, oct 2017.

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

[15] 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 - CHI 18*. ACM Press, 2018.

[16] "UI Events," https://www.w3.org/TR/uievents, Accessed: 2018-12-17.

[17] J. V. Monaco and C. C. Tappert, "Obfuscating keystroke time intervals to avoid identification and impersonation," in *Proc. Intl. Conf. on Biometrics (ICB)*. IEEE, 2016.

[18] M. Schwarz, C. Maurice, D. Gruss, and S. Mangard, "Fantastic timers and where to find them: High-resolution microarchitectural attacks in javascript," in *Proc. 21st Intl. Conf. on Financial Cryptography and Data Security (FC)*. IFCA, 2017, p. 11.