Improving Timing-Based Keylogging Attacks Using a Neural Network

Khokhar, Arsh
khokharm@myumanitoba.ca

Taylor-Quiring, Duncan
taylorqd@myumanitoba.ca

16th April 2021

Abstract

Recent research has shown that the autocomplete functionality of search engines is vulnerable to timing-based keylogging attacks because of the series of `GET` requests they make. Even on an encrypted network, these requests can leak keystroke information from packet size and timing. Previous studies on SSH have also looked at the viability of keylogging attacks against the protocol, but these attacks have not been taken to be a serious threat. In this paper we survey a new attack on search engine autocomplete, which exploits the incrementally increasing packet sizes of autocomplete `GET` requests and uses neural networks and a language model to make accurate predictions about user input. We then revisit SSH to determine if techniques such as neural networks and language models can be deployed against SSH as well for practical keylogging. We also summarize some of the potential countermeasures of the search engine autocomplete attack, and investigate which ones can be applied to prevent the enhanced keylogging attacks against SSH.

1 Introduction

Keylogging attacks are a particular kind of side channel attack in which an attacker attempts to deduce from various sources information about what a user has typed. An attack of this kind can be carried out in two different ways [1]. One approach is to recover location information of keystrokes on the keyboard as they are typed, which will serve as an indicator of which keys were input. The second approach, which this paper focuses on, uses

timing information between keystrokes to determine what keys are being typed by a user. Such timing information can be obtained using a variety of different methods depending on the context. For example, it has been shown that this information is leaked through observation of hardware interrupt timing [2]. The method that will be explored here is based on observing timings between packets sent over a network between a client and server pair.

One environment in which the opportunity of using this kind of timing attack arises is search engines. Virtually any search engine in widespread use today comes with an auto-complete feature that aims to predict user queries. As the user types, requests are sent to the search engine's server that contain information on what the user has typed thus far, and the server returns its best guess as to how the user wants to complete their query. This functionality is provided in the hopes of providing a better user experience, but the research by Monaco [1] that we explore in this paper has shown that it also opens the door for timing-based keylogging attacks that use neural networks for query prediction. Given the sensitive nature of many queries provided to search engines, a keylogging attack has the potential to cause a great deal of harm by leaking private user information. Thus, anyone developing a search engine today must be aware that such an attack is possible in principle, and should act to mitigate it.

Another environment potentially susceptible to this kind of attack is Secure Shell (SSH). When being executed in interactive mode, SSH sends data packets from the remote host containing information about what key was typed to the server. While such attacks are possible in principle, it has been asserted that this kind of attack is unlikely in practice [3]. As will be shown however, the techniques for attacking search engines developed by Monaco that use a neural network to predict words typed with good probability may make such an attack on SSH more viable.

This paper will begin by briefly outlining the history of timing-based keylogging attacks on SSH, together with their limitations. Next, we will outline the new form of a keylogging attack applicable to search engines that was presented in [1]. We will conclude by analyzing the applicability of this new attack to SSH. We will also provide some ways to mitigate this type of attack on search engines and SSH.

2 Attacks on SSH

The SSH protocol provides a secure way to communicate with and control a remote host over a network [4]. Song et al. [5] showed that it is possible in principle to execute a timing-based keylogging attack on SSH. The method used relied on two weaknesses in the protocol. First, packets are padded to an eight-byte boundary, which can reveal whether or not a password sent over the connection is short (eight bytes or less long) or not. This information can be used to determine which users are most vulnerable to a password-guessing attack. Second, and more relevant to our discussion, while in interactive mode the protocol transfers packets that contain individual characters to the remote host. The timing between when these characters were typed is mostly preserved over the network. Song et al. collected data on timing information between pairs of keys in isolation as typed by a user, and used this data to build a Hidden Markov Model that can guess what sequence of keys were typed based solely on timings between the keystrokes. They used this model to guess short passwords, and found that they were able to increase their success rate up to 50 times versus that of an exhaustive search of the password space.

However, there are problems with this approach. First, it cannot be assumed that looking at timing between key pairs in isolation can provide an accurate model of the way users type in actuality. For example, it is likely that some key pairs are typed more quickly than they otherwise would be when they succeed a particular sequence of keys, due to differences in hand position caused by the preceding letters [6]. We also note that Song et al. state in their paper that these key timings do not come from users typing real (or even simulated) passwords. They tried this method initially, but found that having users learn to type passwords many times was too time-consuming of a process.

It has also been noted that if there is no limit to the character set used for a password (even a relatively short one), there is a combinatorial explosion of possible key pairs that need to be analysed [3]. While many people do not bother to use passwords that contain special characters, enough people do for this to be a serious hurdle to executing the attack successfully. Thus, extending the data collection method of Song et al. to include more complicated passwords is untenable.

However, with the development of new techniques by Monaco for performing timing attacks on autocomplete, we believe that it would be worthwhile to re-examine the feasibility of a timing attack on SSH in this new light to see if the methods used have the possibility of improving the accuracy of password-guessing. We will now outline the The Keystroke Recognition and Entropy Elimination Program (KREEP) attack presented in [1].

3 The KREEP attack

3.1 Threat model

In the KREEP attack, the adversary can observe packets sent from the web browser to the search engine's server. The packets may be encrypted. It is assumed that the user types their queries into the web browser using only alphabetic keys together with the space key, and that all words typed are English words that can be found in the attacker's dictionary. We see that this model is not too far off from what we may see in an SSH session in which a password is being typed; in this situation, the shift key would be likely have to be included in the list of keys to allow for uppercase letters and special characters such as '%' and '$', and the constraint that words must be in the dictionary would be relaxed. The search query given by the user must contain words that are included in the attacker's dictionary. No prior knowledge of the user's typing habits is needed, as the neural network used to identify words based on timing of keystrokes is trained on independent data.

3.2 Attack execution workflow

The attack proceeds in five stages. First, from the noise of network activity generated by web activity (connecting to a site, loading page assets, etc.) the packets that are generated by user keystrokes must be identified. After identification, the keystrokes are analyzed to find word boundaries and generate word tokens. With each word boundary identified, the lengths of the packets constituting a single word are analyzed for clues as to what the word is. This work proceeds using a predefined dictionary to filter out words that do not match a certain profile.

The last two steps make use of neural networks. A neural network trained on differences in key press timings is first used to make a guess about what words were actually typed in the query. Lastly, a natural language model is applied to generate the most likely queries given the above data collected. We now proceed to outline each step of the attack in greater detail.

3.2.1 Keystroke detection

The first step needed to perform the attack is to differentiate between packets created when the user types a character into the search bar and packets created from other requests that the browser sends to the server for pages or other assets. To do this, the attack leverages the fact that the GET requests sent as the user types contain subsequences of the complete query that monotonically increase in length. Table 1 shows an example of such a sequence of requests.

Packet id	Packet size	`GET` request URL
1	163	`?q=t&cp=1&...`
2	164	`?q=th&cp=2&...`
3	164	`?q=the&cp=3&...`
4	166	`?q=the%20&cp=4&...`

Table 1: Sequence of HPACK-compressed autocomplete packets taken from [1]

As can be seen, the size of the packets that contain subqueries gradually increases as the length of the subqueries increase. The subsequence of packets corresponding to key presses can be detected by applying a dynamic programming algorithm to find the longest increasing subsequence (LIS) of packets lengths. However, as noticed for packets 2 and 3 in Table 1, LIS fails to capture that the size of the packets can sometimes stay the same even when a character is appended. This can occur if the search engine uses the HTTP2 header compression known as HPACK. To deal with this, Monaco proposes a generalization of LIS that can be modelled by a search engine-specific deterministic finite automaton (DFA) that accepts such subsequences while rejecting the ones which fail to show the required increase. The tran-

sition conditions of this DFA are defined by using experimental data related to the packet sizes.

3.2.2 Tokenization

Once the keystroke packets are identified, the next step is to determine how to divide the packets into tokens that represent words. As described in the threat model, it is assumed that each word in the search query is an English word in a dictionary known to the attacker and that individual words are separated by a space. Since the space character is the only delimiter, tokenization becomes a binary classification problem where each packet is labelled as either a part of currently running word or as a delimiter.

While any alphabetic character in the user input is put as-is into the request URL, spaces are URL encoded as '%20'. As a result, the packet size increases by a larger amount when the packet corresponds to an added space versus when it contains a letter. This is true whether or not HTTP2 header compression is used, although the exact difference in packet sizes differs depending on the way in which the search engine parameterizes the query and whether or not it uses header compression. Thus, to determine whether a packet contains a letter or a space, it suffices to compare packet sizes and divide the packets accordingly.

3.2.3 Dictionary Pruning

The next step is only applicable to search engines that use HPACK for header compression. This compression format uses a static Huffman encoding for string literals [7]. A Huffman code of a symbol (i.e. character) is a bit string, and a string literal $\mathcal{S} = c_1, c_2, ...c_k$ is encoded by concatenating the codes of each individual symbol $c_1, c_2, ...c_k$. Under this particular Huffman code, a character will be encoded as a bit string with a length that depends on the frequency of that character in a large sample of HTTP headers. In other words, the encoded form of some characters will be longer than that of others. As such, the attacker can observe the incremental changes in size of the encoded subqueries and obtain information about the string that was encoded.

For each word in their dictionary, the attacker calculates the incremental

changes in length that would be observed if the word were compressed with HPACK. These are combined with varying amounts of padding to match what would be added in a real packet, yielding several signatures for each word. These signatures are compared with the pattern of incremental changes observed from the packets. In this way, words that do not have a signature that matches the pattern can be pruned from the list of potential matching words.

This portion of the attack cannot be carried out by search engines that do not use HPACK header compression, since then every new character appended to a subquery will have the same size as any other. In this case, the length of the word is the only information that can be gained.

3.2.4 Word identification using timing data

All steps up to this point have exploited the sizes of sequential packets in order gain information. This next step is the first that deals with timings between packets. A recurrent neural network (RNN) is trained on the timing data generated by data of thousands of users typing English phrases from a variety of sources. For a sequence of timing differences τ of length $n+1$ (the first timing difference is that between the space preceding a word and the word's first character), the RNN generates for each alphabetic character c the probability $P(c_i)$ that the i-th timing difference ($1 \leq i \leq n$) corresponds to that character. Then the probability that a word w corresponds to those timings is calculated as the joint probability:

$$P(w|\tau) = \prod_{c_i \in w} P(c_i) \tag{1}$$

3.2.5 Language model and beam search

In the last step of the attack, a list of possible queries is generated using probabilities of individual words calculated using Equation 1 combined with a natural language model. To achieve this, [1] notes the fact that certain words are more likely to occur together than others depending upon the context of the running sentence. For example, if a sentence starts with the words *"A barking"*, the next word is more likely to be *"dog"* than *"cat"*. The likelihood can be modelled as the conditional probability of a word given its

predecessor words. Formally, for any word w_i at position i in the sentence, the language model encodes the probability $P(w_i|w_1, w_2, ..., w_{i-1})$.

The probability of a query $Q = [w_1, w_2, ..., w_n]$ is:

$$P(Q) = \prod_{w_i \in Q} P(w_i|\tau)P(w_i|w_1, w_2, ...w_{i-1})^{\alpha} \tag{2}$$

In this definition α is the weight parameter of the language model, which can be lowered or increased to change the influence of the language model on the final result. This equation gives us the means to determine the likelihood of a query, but first a list of possible queries must be generated before any probabilities can be calculated. To generate this list, Monaco uses beam search, a greedy search algorithm. This algorithm proceeds by maintaining a running list of subqueries to which new words are appended that maximize Equation 2. When the best match for all current subqueries in the list are found, the subqueries plus their best-matching words become the new subquery list. This continues until the length of the subqueries match the length of the actual query.

4 Applicability to SSH

It is noted in [1] that while each individual component of this attack has limited effectiveness when executed in isolation, together they form a feasible attack to recover search queries from timing information alone. The next question we want to investigate is which techniques developed here, if any, can be applied to the problem of deducing passwords from SSH network traffic.

First, consider the problem of detecting network packets that are relevant to each attack. In the case of the attack on autocomplete, the problem is to isolate packets that correspond to keystrokes. Since there could be a great deal of network noise, the technique for finding the largest increasing monotonic subsequence with respect to packet length had to be developed. However, each keystroke over SSH will yield a packet of the same size as any other keystroke, since each packet does not contain the previous letters that were typed as is the case with autocomplete. Thus, no analogue to a longest increasing subsequence of packet sizes exists for SSH, and so other methods for

keystroke detection must be deployed. In the treatment given by Song et al., they note that a client entering a password over SSH generates an identifiable pattern in packet sizes that could be exploited to determine which packets correspond to keystrokes. Other research has shown that classification algorithms can be developed to detect keystrokes over SSH [8]. Keystroke detection for SSH is therefore a problem that can be solved, but the methods given by Monaco are not directly applicable in this context.

Similarly, the problem of tokenization is not relevant to SSH, since the attack is focused on one group of characters (the password) and not multiple groups separated by a space. Likewise, the dictionary pruning algorithm of Section 3.2.3 cannot be applied, since it relies on examining packets that contain more than one character.

With that in mind, it's clear that if any improvement to the SSH attack can be gained from an analysis of the autocomplete attack, it is to be found in the use of neural networks and natural language models. Since passwords are only one word long, we believe that a natural language model would be of limited use to any attack that guesses passwords. More useful would be the neural network used in Section 3.2.4 to guess words based on timings between keystrokes. The neural network presented there was trained on a data set generated from many people typing English sentences and phrases and cannot be directly applied to a context where other symbols, such as numbers and special characters, may appear. If such a data set existed, however, we believe that an attack based on a neural network would be a vast improvement over the method used by Song et al.

5 Countermeasures

There were a number of countermeasures considered in [1] to mitigate the attack on autocomplete. The first suggestion was to pad packets with random data to obscure the true length of the data. This is useful for the autocomplete attack, where information is leaked through packet size, but is not helpful for the attack against SSH. The other two measures suggested both serve to obfuscate timing information and so have the potential to mitigate SSH timing attacks. We briefly examine the two methods below.

Dummy packets

One option would be to introduce meaningless packets into the connection between client and server (for example, characters that do not affect the input) [6]. As long as these packets are of the same length as those that carry real character data, an attacker would have no way of differentiating them. All timing data that would be observed would no longer reflect the typing of the user, and so the RNN of section 3.2.4 would not be able to make any accurate guess as to which characters constitute the password.

Packet merging

Another option would be to merge packets that contain character data into larger packets. Timing could still be measured between these larger packets, but a great deal of information regarding typing patterns is lost. This method could, however, affect usability of the interface considerably depending on how often keystrokes are batched together [6].

6 Conclusion and Future Work

The KREEP attack, if successfully executed, presents a real threat to privacy. Our focus has been on determining what aspects of this attack can be carried over to a packet-sniffing attack on SSH, and we have found that at least one component, the neural network that guesses words based on timing between packets, has the potential to improve upon known attacks. While these attacks can be mitigated by obscuring the timing between keystrokes, it would be beneficial to first establish if a neural network could indeed be built that can accurately guess characters of a password.

A first step would be producing a large data set of timings between typed symbols of many passwords. The passwords would have to either be a sample of real-world passwords, or a collection of procedurally generated passwords that mimic the patterns often found in real passwords. After this data is acquired, a neural network trained on this data would be applied to packets collected from an SSH session to see how accurately a transmitted password can be guessed. We believe that such a study could do much to improve confidence in the security of SSH, or identify flaws that should be corrected.

References

[1] John V. Monaco. What Are You Searching For? A Remote Keylogging Attack on Search Engine Autocomplete. In *Proc. 28th USENIX Security Symposium (USENIX Security 19)*, pages 959–976, Santa Clara, CA, August 2019. USENIX Association.

[2] Moritz Lipp, Daniel Gruss, Michael Schwarz, David Bidner, Clémentine Maurice, and Stefan Mangard. Practical Keystroke Timing Attacks in Sandboxed JavaScript. In *Proc. 22nd European Symposium on Research in Computer Security*, 2017.

[3] Timing analysis is not a real-life threat to SSH secure shell users. http://web.archive.org/web/20010831024537/http://www.ssh.com/products/ssh/timing_analy Accessed: 2021-04-15.

[4] SSH (Secure Shell) Home Page. https://www.ssh.com/academy/ssh. Accessed: 2021-04-16.

[5] Dawn Xiaodong Song, David A Wagner, and Xuqing Tian. Timing Analysis of Keystrokes and Timing Attacks on SSH. In *Proc. USENIX Security Symposium*, 2001.

[6] Andreas Noack. Timing Analysis of Keystrokes and Timing Attacks on SSH - revisited. 2007.

[7] HPACK: Header Compression for HTTP/2. https://tools.ietf.org/html/rfc7541. Accessed: 2021-04-15.

[8] Saptarshi Guha, Paul Kidwell, Ashrith Barthur, William S. Cleveland, John Gerth, and Carter Bullard. A Streaming Statistical Algorithm for Detection of SSH Keystroke Packets in TCP Connections. In *Proc. 12th INFORMS Computing Society Conference*, 2011.