Timing Analysis of Keystrokes and Timing Attacks on SSH*

Dawn Xiaodong Song David Wagner Xuqing Tian
University of California, Berkeley

Abstract

SSH is designed to provide a secure channel between two hosts. Despite the encryption and authentication mechanisms it uses, SSH has two weakness: First, the transmitted packets are padded only to an eight-byte boundary (if a block cipher is in use), which reveals the approximate size of the original data. Second, in interactive mode, every individual keystroke that a user types is sent to the remote machine in a separate IP packet immediately after the key is pressed, which leaks the inter-keystroke timing information of users' typing. In this paper, we show how these seemingly minor weaknesses result in serious security risks.

First we show that even very simple statistical techniques suffice to reveal sensitive information such as the length of users' passwords or even root passwords. More importantly, we further show that by using more advanced statistical techniques on timing information collected from the network, the eavesdropper can learn significant information about what users type in SSH sessions. In particular, we perform a statistical study of users' typing patterns and show that these patterns reveal information about the keys typed. By developing a Hidden Markov Model and our key sequence prediction algorithm, we can predict key sequences from the inter-keystroke timings. We further develop an attacker system, *Herbivore* , which tries to learn users' passwords by monitoring SSH sessions. By collecting timing information on the network, Herbivore can speed up exhaustive search for passwords by a factor of 50. We also propose some countermeasures.

In general our results apply not only to SSH, but also to a general class of protocols for encrypting interactive traffic. We show that timing leaks open a new set of security risks, and hence caution must be taken when designing this type of protocol.

*This research was supported in part by the Defense Advanced Research Projects Agency under DARPA contract N6601-99-28913 (under supervision of the Space and Naval Warfare Systems Center San Diego) and by the National Science foundation under grants FD99-79852 and CCR-0093337.

1 Introduction

Just a few years ago, people commonly used astonishingly insecure networking applications such as `telnet`, `rlogin`, or `ftp`, which simply pass all confidential information, including users' passwords, in the clear over the network. This situation was aggravated through broadcast-based networks that were commonly used (e.g., Ethernet) which allowed a malicious user to eavesdrop on the network and to collect all communicated information [CB94, GS96].

Fortunately, many users and system administrators have become aware of this issue and have taken countermeasures. To curb eavesdroppers, security researchers designed the Secure Shell (SSH), which offers an encrypted channel between the two hosts and strong authentication of both the remote host and the user [Ylö96, SSL01, YKS+00b]. Today, SSH is quite popular, and it has largely replaced `telnet` and `rlogin`.

Many users believe that they are secure against eavesdroppers if they use SSH. Unfortunately, in this paper we show that despite state-of-the-art encryption techniques and advanced password authentication protocols [YKS+00a], SSH connections can still leak significant information about sensitive data such as users' passwords. This problem is particularly serious because it means users may have a false confidence of security when they use SSH.

In particular we identify that two seemingly minor weaknesses of SSH lead to serious security risks. First, the transmitted packets are padded only to an eight-byte boundary (if a block cipher is in use). Therefore an eavesdropper can easily learn the approximate length of the original data. Second, in interactive mode, every individual keystroke that a user types is sent to the remote machine in a separate IP packet immediately after the key is pressed (except for some meta keys such `Shift` or `Ctrl`). We show in the paper that this property can enable the eavesdropper to learn the exact length of users' passwords. More importantly, as we have verified, the time it takes the operating system to send out the packet after the key press is in general negligible comparing to the inter-keystroke timing. Hence an eaves-

dropper can learn the precise inter-keystroke timings of users' typing from the arrival times of packets.

Experience shows that users' typing follows stable patterns[1]. Many researchers have proposed to use the duration of key strokes and latencies between key strokes as a biometric for user authentication [GLPS80, UW85, LW88, LWU89, JG90, BSH90, MR97, RLCM98, MRW99]. A more challenging question which has not yet been addressed in the literature is whether we can use timing information about key strokes to infer the key sequences being typed. If we can, can we estimate quantitatively how many bits of information are revealed by the timing information? Experience seems to indicate that the timing information of keystrokes reveals some information about the key sequences being typed. For example, we might have all experienced that the elapsed time between typing the two letters "er" can be much smaller than between typing "qz". This observation is particularly relevant to security. Since as we show the attacker can get precise inter-keystroke timings of users' typing in a SSH session by recording the packet arrival times, if the attacker can infer what users type from the inter-keystroke timings, then he could learn what users type in a SSH session from the packet arrival times.

In this paper we study users' keyboard dynamics and show that the timing information of keystrokes does leak information about the key sequences typed. Through more detailed analysis we show that the timing information leaks about 1 bit of information about the content per keystroke pair. Because the entropy of passwords is only 4–8 bits per character, this 1 bit per keystroke pair information can reveal significant information about the content typed. In order to use inter-keystroke timings to infer keystroke sequences, we build a Hidden Markov Model and develop a *n*-Viterbi algorithm for the keystroke sequence inference. To evaluate the effectiveness of the attack, we further build an attacker system, Herbivore, which monitors the network and collects timing information about keystrokes of users' passwords. Herbivore then uses our key sequence prediction algorithm for password prediction. Our experiments show that, for passwords that are chosen uniformly at random with length of 7 to 8 characters, Herbivore can reduce the cost of password cracking by a factor of 50 and hence speed up exhaustive search dramatically. We also propose some countermeasures to mitigate the problem.

We emphasize that the attacks described in this paper are a general issue for any protocol that encrypts interactive traffic. For concreteness, we study primarily SSH, but these issues affect not only SSH 1 and SSH 2, but also any other protocol for encrypting typed data.

The outline of this paper is as follows. In Section 2 we discuss in more details about the vulnerabilities of SSH and various simple techniques an attacker can use to learn sensitive information such as the length of users' passwords and the inter-keystroke timings of users' passwords typed. In Section 3 we present our statistical study on users' typing patterns and show that inter-keystroke timings reveal about 1 bit of information per keystroke pair. In Section 4 we describe how we can infer key sequences using a Hidden Markov Model and a *n*-Viterbi algorithm. In Section 5 we describe the design, development and evaluation of an attacker system, Herbivore, which learns users' passwords by monitoring SSH sessions. We propose countermeasures to prevent these attacks in Section 7, and conclude in Section 8.

2 Eavesdropping SSH

The Secure Shell SSH [SSL01, YKS+00b] is used to encrypt the communication link between a local host and a remote machine. Despite the use of strong cryptographic algorithms, SSH still leaks information in two ways:

- First, the transmitted packets are padded only to an eight-byte boundary (if a block cipher is in use), which leaks the approximate size of the original data.
- Second, in interactive mode, every individual keystroke that a user types is sent to the remote machine in a separate IP packet immediately after the key is pressed (except for some meta keys such `Shift` or `Ctrl`). Because the time it takes the operating system to send out the packet after the key press is in general negligible comparing to the inter-keystroke timing (as we have verified), this also enables an eavesdropper to learn the precise inter-keystroke timings of users' typing from the arrival times of packets.

The first weakness poses some obvious security risks. For example, when one logs into a remote site *R* in SSH, all the characters of the initial login password are batched up, padded to an eight-byte boundary if a block cipher is in use, encrypted, and transmitted to *R*. Due to the way padding is done, an eavesdropper can learn one bit of information on the initial login password, namely, whether it is at least 7 characters long or not. The second weakness can lead to some potential anonymity risks since, as many researchers have found previously, inter-keystroke timings can reveal the iden-

[1] In this paper we only consider users who are familiar with keyboard typing and use touch typing.

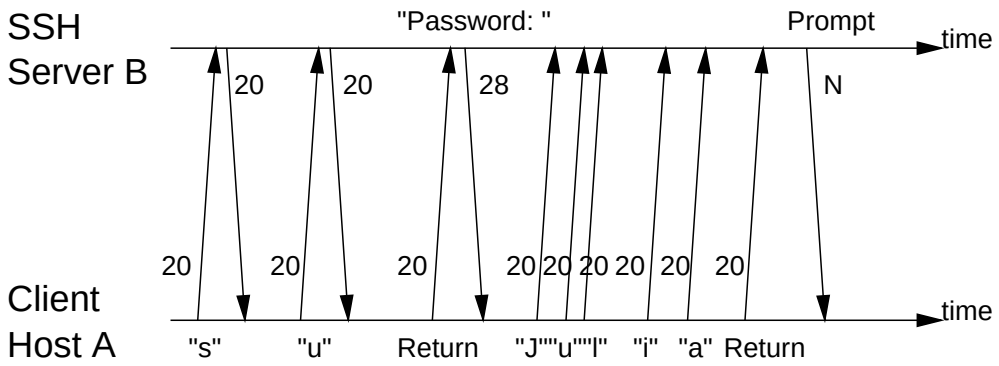

Figure 1: The traffic signature associated with running SU in a SSH session. The numbers in the figure are the size (in bytes) of the corresponding packet payloads.

tity of the user [GLPS80, UW85, LW88, LWU89, JG90, BSH90, MR97, RLCM98, MRW99].

In this section, we show that several simple and practical attacks exploiting these two weaknesses. In particular, an attacker can identify which transmitted packets correspond to keystrokes of sensitive data such as passwords in a SSH session. Using this information, the attacker can easily find out the exact length of users' passwords and even the precise inter-keystroke timings of the typed passwords. Learning the exact length of users' passwords allows eavesdroppers to target users with short passwords. Learning the inter-keystroke timing information of the typed passwords allows eavesdroppers to infer the content of the passwords as we will show in Section 3 and 4.

Traffic Signature Attack We can often exploit properties of applications to identify which packets correspond to the typing of a password. Consider, for instance, the SU command. Assume the user has already established a SSH connection from local host *A* to remote host *B*. When the user types the command SU in the established SSH connection $A \leftrightarrow B$, we obtain a peculiar traffic signature as shown in Figure 1. If the SSH session uses `SSH 1.x`[2] and a block cipher such as DES for the encryption [NBS77, NIS99], as is common, then the local host *A* sends three 20-byte packets: "s", "u", "Return". The remote host *B* echoes the "s" and "u" in two 20-byte packets and sends a 28-byte packet for the "Password: " prompt. Then *A* sends 20-byte packets, one for each of the password characters, without receiving any echo data packets. *B* then sends some final packets containing the root prompt if SU succeeds, otherwise some failure messages. Thus by checking the traffic against this "`su`" signature, the attacker can identify when the user issues the SU command and hence learn which packets correspond to the password keystrokes. Note that similar techniques can be used to identify when users type passwords to authenticate to other applications such as PGP [Zim95] in a SSH session.

Multi-User Attack Even more powerful attacks exist when the attacker also has an account on the remote machine where the user is logging into through SSH. For example, the process status command `ps` can list all the processes running on a system. This allows the attacker to observe each command that any user is running. Again, if the user is running any command that requires a password input (such as `su` or `pgp`) the attacker can identify the packets corresponding to the password keystrokes.

Nested SSH Attack Assume the user has already established a SSH session between the local host *A* and remote host *B*. Then the user wants to open another SSH session from *B* to another remote host *C* as shown in Figure 2. In this case, the user's password for *C* is transmitted, one keystroke at a time, across the SSH-encrypted link $A \leftrightarrow B$ from the user to *B*, even though the SSH client on machine *B* patiently waits for all characters of the password before it sends them all in one packet to host C for authentication (as designed in the SSH protocol [YKS+00a]). It is easy to identify such a nested SSH connection using techniques developed by Zhang and Paxson [ZP00b, ZP00a]. Hence in this case the eavesdropper can easily identify the packets corresponding to the user's password on link $A \leftrightarrow B$, and from this learn the length and the inter-keystroke timings of the users' password on host *C*.

[2] The attack also works when `ssh 2.x` is in use. Only the packet sizes are slightly different.

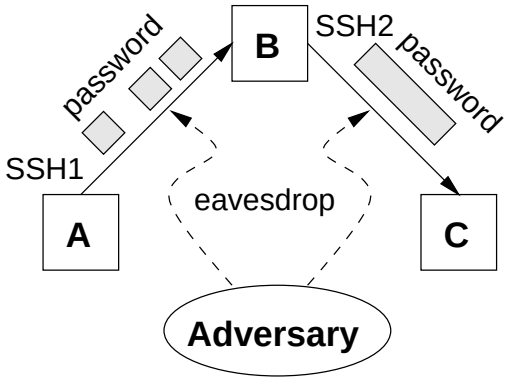

Figure 2: The nested SSH attack.

3 Statistical Analysis of Inter-keystroke Timings

As a first study towards inferring key sequences from timing information, we develop techniques for statistical analysis of the inter-keystroke timings. In this section, we first describe how we collect training data and show some simple timing characteristics of character pairs. We then show how we model the inter-keystroke timing of a given character pair as a Gaussian distribution. We then describe how to estimate quantitatively the amount of information about the character pair that one can learn using the inter-keystroke timing information. Denote the set of character pairs of interest as Q, and let Q denote the cardinality of the set Q.

3.1 Data Collection

The two keystrokes of a pair of characters (k_a, k_b) generates four events: the press of k_a, the release of k_a, the press of k_b, and the release of k_b. However, because only key presses (not key releases) trigger packet transmission, an eavesdropper can only learn timing information about the key-press events. Since the main focus of our study is in the scenario where an adversary learns timing information on keystrokes by simply monitoring the network, we focus only on key-press events. The time difference between two key presses is called the *latency* between the two keystrokes. We also use the term *inter-keystroke timing* to refer to the latency between two keystrokes.

In order to characterize how much information is leaked by inter-keystroke timings, we have performed a number of empirical tests to measure the typing patterns of real users. Because passwords are probably the most sensitive data that a user will ever type, we focus only on information revealed about passwords (rather than other forms of interactive traffic).

Our focus on passwords creates many challenges. Passwords are entered very differently from other text: passwords are typed frequently enough that, for many users, the keystroke pattern is memorized and often typed almost without conscious thought. Furthermore, well-chosen passwords should be random and have little or no structure (for instance, they should not be based on dictionary words). As a consequence, naive measurements of keystroke timings will not be representative of how users type passwords unless great care is taken in the design of the experimental methodology.

Our experimental methodology is carefully designed to address these issues. Due to security and privacy considerations, we chose not to gather data on real passwords; therefore, we have chosen a data collection procedure intended to mimic how users type real passwords. A conservative method is to pick a random password for the user (where each character of the password is chosen uniformly at random from a set of 10 letter keys and 5 number keys, independently of all other characters in the password), have the user practice typing this password many times without collecting any measurements, and then measure inter-keystroke timing information on this password once the user has had a chance to practice it at length.

However, we found that, when the goal is to try to identify potentially relevant timing properties (rather than verify conjectured properties), this conservative approach is inefficient. In particular, users typically type passwords in groups of 3–4 characters, with fairly long pauses between each group. This distorts the digraph statistics for the pair of characters that spans the group boundary and artificially inflates the variance of our measurements. As a result we would need to collect a great deal of data for many random passwords before this effect would average out. In addition, it takes quite a while for users to become familiar with long random passwords. This makes the conservative approach a rather blunt tool for understanding inter-keystroke statistics.

Fortunately, there is a less costly way to gather inter-keystroke timing statistics: we gather training data on each pair of characters (k_a, k_b) as typed in isolation. We pick a character pair and ask the user to type this pair 30–40 times, returning to the home row each time between repetitions. For each user, we repeat this for many possible pairs (142 pairs, in our experiments) and we gather data on inter-keystroke timings for each such pair. We collected the latency of each character pair measurement and computed the mean value and the standard deviation. In our experience, this gives better results.

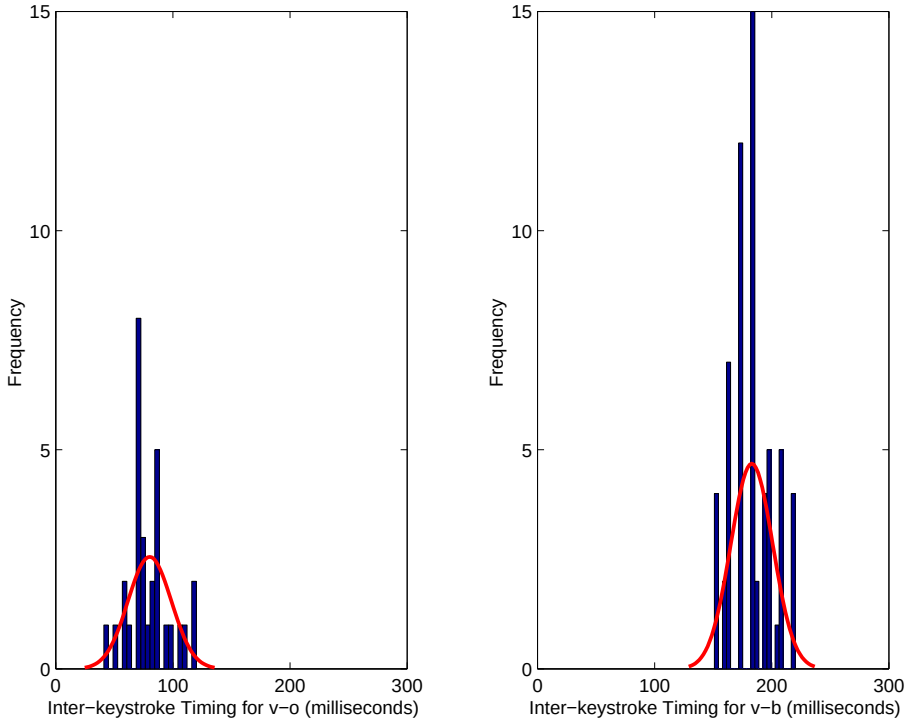

Figure 3: The distribution of inter-keystroke timings for two sample character pairs.

As an example, Figure 3 shows the latency histogram of two sample character pairs. The left model corresponds to the latency between the pair (v, o), and the right model corresponds to (v, b). We can see that the latency between (v, o) is clearly shorter than the latency between (v, b), and the latency distributions of these two sample character pairs are almost entirely non-overlapping.

The optimized data collection approach gives us a more efficient way to study fine-grained details of inter-keystroke statistics without requiring collecting an enormous amount of data. We used data collected in this way to quickly identify plausible conjectures, develop potential attacks, and to train our attack models. As far as we are aware, collecting data on keystroke pairs in isolation does not seem to bias the data in any obvious way. Nonetheless, we also validate all our results using the conservative measurement method (see Section 5).

3.2 Simple Timing Characteristics

Next, we divide the test character pairs into five categories, based on whether they are typed using the same hand, the same finger, and whether they involve a number key:

- Two letter keys typed with alternating hands, i.e., one with left hand and one with right hand;
- Two characters containing one letter key and one number key typed with alternating hands;
- Two letter keys, both typed with the same hand but with two different fingers;
- Two letter keys typed with the same finger of the same hand;
- Two characters containing one letter key and one number key, both typed with the same hand.

Figure 4 shows the histogram of latency distribution of character pairs for each category. We split the whole latency range into six bins as shown in the *x*-axis. Within each category, we put each character pair into the corresponding bin if its mean latency value is within the range of the bin. Each bar in the histogram of a category represents the ratio of the number of character pairs in the associated bin over the total number of character pairs in the category.[3] We can see that all the character pairs that are typed using two different hands take less than 150 milliseconds, while pairs typed using the same hand and particularly the same finger take substantially longer. Character pairs that alternate between one letter key and one number key, but are typed using the same

[3]Hence the sum of all bars within one category is 1.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.