

Countermeasure against Timing Attack on SSH Using Random Delay

Arief Karfianto <karfi@ictlab.org>, Yan Adikusuma <adiku20@yahoo.com>

Abstract

The SSH protocol is used for secure remote access to a remote server. However, the protocol as widely deployed is vulnerable to timing attack, where an adversary gets information from inter-keystroke time. This danger particularly threatens a user that execute su command in SSH session. By analyzing the packet transmitted, eavesdropper can get some information about the password typed by user.

We show that random delays between packets sending in SSH will destroy the timing statistics of keystrokes and cause difficulties for timing attack. The result shows that the standard deviation of inter-keystroke time increases about 14% by adding random delay in SSH.

Keywords : SSH, timing attack, network security, protocol.

1 Introduction

Protocol is needed to run some process in computer network, such as log into one account remotely from another, or transmit commands to a remote computer for execution. Various programs exist for these purposes, such as `ftp` for file transfers, `telnet` for remote logins, and `rsh` for remote execution of commands. Unfortunately, many of these network-related programs have security problems. Broadcast-based networks that were commonly used (e.g., Ethernet) allow a malicious user to eavesdrop on the network and to collect all communicated information. Therefore, SSH was designed to replace the insecure application above.

Secure Shell (SSH) was invented by Tatu Ylönen in 1995. The Secure Shell (SSH) is a public-key based authentication protocol suite which enables a user to securely login onto a remote server host machine from a client machine through an insecure network, to securely execute commands in the remote host, and to securely move files from one host to another [Mao,2004].

Nowadays, SSH is vulnerable to timing attack. SSH still leaks information in two ways [Song,2001]. 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.

Some people are familiar with touch typing. They have stable pattern when press the keyboard. It is possible to use the timing information to infer the key sequence being typed. In other word, the elapsed time between typing a pair of letter can be smaller than between another pairs. This becomes a security problem. If an attacker can get inter-keystroke timing information from user, then later he may learn and determine what user type from the packet arrival time.

There are several countermeasures against timing analysis implemented in the latest version of SSH. OpenSSH sends blank packets back to the user to complicate timing attack. But the attacker still can measure the Round-Trip-Times (RTT) to both host he eavesdrops. If these times are exact enough, he can reveal the even presented timing information by subtracting the RTT's from the eaves-dropped timings. So the attacker can distinguish, whether the echo-mode is activated or not.

It is possible that the blank packets are sent later and not immediately. Another fix could be applied to the SSH client. In interactive mode, the client can time pad the keystrokes that are sent to server which would destroy all keystroke timing statistics.

We make some contributions in this paper. We show how our approach makes inter-keystroke timing will have random latency and make timing attack complicated. We also have built the modified version of OpenSSH as the implementation for our approach.

2 Motivations

Implementation errors are the big security problem in real-world system. The motivation of this research is trying to implement a countermeasure of timing attack on SSH. The wide use of SSH is another reason why it is become necessary to study about countermeasure of the problems. Improving a cryptosystem is interesting, but it must be implemented correctly.

3 Related Works

[DS,2001] have pointed out several weaknesses in implementations of SSH (Secure Shell) protocols. When exploited, they let the attacker obtain sensitive information by passively monitoring encrypted SSH sessions. The information can later be used to speed up brute-force attacks on passwords, including the initial login password and other passwords appearing in interactive SSH sessions, such as those used with `su` command.

Timing analysis has been described in detail and implemented using Hidden Markov Model and solved by n-viterbi algorithm in [Song,2001]. They reveal that inter-keystroke timings leak about 1 bit of information per character pair, and describe an attacking system, herbivore, which tries to learn users' passwords by monitoring SSH sessions. Song also proposed some countermeasure of timing attack such as adding random delay and sending traffic in constant rate.

Another researcher, Andreas Noack then improved the timing attack by using second degree Hidden Markov Model assuming that a keystroke timing between two keys depends not only on these two keys [Noack,2007].

We expect to analyze that random delay is an effective countermeasure against timing attack on SSH.

4 Problem Statement and Approach Overview

Problem Statement

In [Song,2001], the authors claim that SSH leak password information in two ways. First, SSH transmits password packets that are padded to an eight-byte boundary during the login. From this, an eavesdropper can learn the approximate size of the password by looking at the number of packets that are sent. If only one packet is sent, then the eavesdropper knows the password is at most 7 characters long. Secondly, when SSH is in interactive mode, it is useful for eavesdroppers to monitor the time intervals between sent packets. SSH automatically goes into interactive mode after the initial login. During this mode, eavesdroppers can determine which packets contain password characters by carefully monitoring traffic between the client and the server.

Passwords are sent during interactive mode on two occasions: when a switch user command is executed and when the client starts a nested SSH session. The client sends the server a `su` command (`su` stands for “substitute user”), followed by a return, prior to the transmission of a password. After the `su` command is acknowledged, the next packets sent contain the password characters. These packets are not echoed by the server, thus indicating to the eavesdropper that these packets contain the characters of the password. Figure 1 shows an SSH packet transfer between the client and server for an `su` command execution. These transfers form a “signature” for recognizing the `su` command. The numbers indicate the packet size in bytes. Notice how the characters of the password “Julia” are not echoed by the server, indicating that those packets contain password characters.

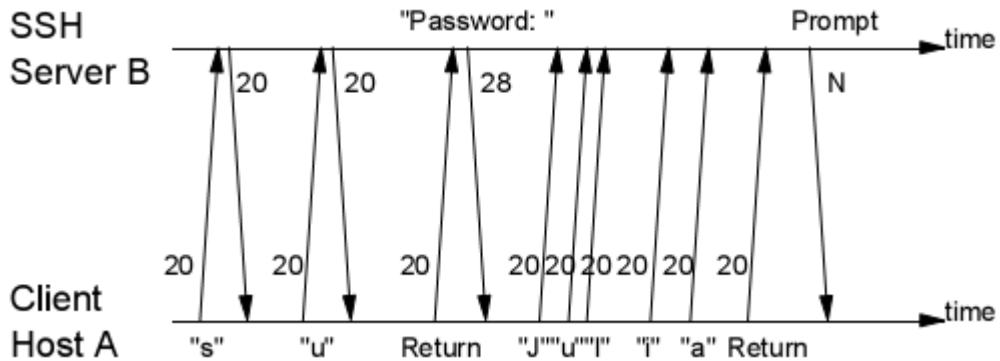


Figure 1. The illustration of timing attack scheme when user execute the `su` command.

Once the password character packets have been identified, keystroke timing analysis can be performed on the time intervals to aid in the cracking of the passwords. These time intervals can reveal information about the password typed.

In summary, timing attack applied with several steps. First is data collection. In this step, the attacker eavesdrop the traffic and records the time for packets transmitted between client and server. Second, examining keystroke characteristics. By doing this step, attacker can see that different character pairs have different latencies. Inter-keystroke timing can be modeled as Gaussian distribution. The model can be built by gathering statistics, and calculating the mean and variance of the keystroke timing. Third, inferring character sequence from inter-keystroke timing information if attacker has a sequence that supposed to be the password timing sequence. The inter-keystroke timing is modeled as a Hidden Markov Model and then it solved by n-viterbi algorithm. The output of this algorithm is n most likely candidate character sequences.

Approach Overview

In this paper, we focus to solve the timing attack by destroying the keystroke timing statistics.

Inferring the character sequence from keystroke timing information will be complicated when the keystroke characteristics destroyed. Destroyed here means the distributions have a large value of variance, so that attacker cannot determine the keystroke time pattern of each pairs pressed in SSH session. This is the basic idea of the countermeasure against timing attack.

Our approach evaluated by observing the descriptive statistics including the mean value and the standard deviation of keystroke time. The mean or expected value of a random variable X is a special importance in statistics because it describes where the probability distribution is centered. By itself, however, the mean does not give adequate description of the shape of the distribution. We need to characterize the variability in the distributions.

Here a sample the histogram of two discrete probability distributions with the same mean $\mu=2$ that differ considerably in the variability or dispersion of their observations about the mean.

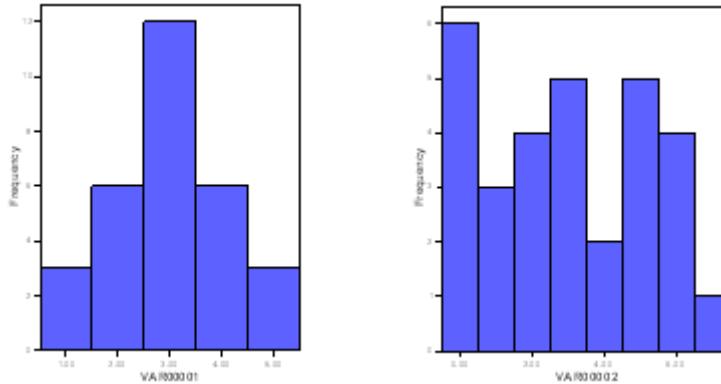


Figure 2. Distribution with equal means and different dispersions.

Here the descriptive statistics table of the histograms above :

	N	Minimum	Maximum	Mean	Standard Deviation
Variable 1	30	1,00	5,00	3,0000	1,11417
Variable 2	30	0,00	7,00	3,0000	2,22834
Valid N	30				

Figure 3. The statistics of distribution with equal mean and different dispersion.

Keystroke characteristics can be destroyed by adding random delay after each keystroke. For example, if the latency of two-keystroke from the experiment is between 0 to η milliseconds, random delay can be added up to η milliseconds so that it prevents the leakage of timing information. We expected that the statistics of random delayed keystroke will have bigger standard deviation and describe the large value of variance of the keystroke time.

5 Implementation

This technique can be applied in any SSH implementation. The random delay would be applied by creating such random delay function in C language program, that will be added to the OpenSSH. The random delay function will be called after user press the keystroke but before the SSH packet sent. The implementation including creating the random delay generator scheme, searching the code in OpenSSH package containing the function that handles the sending of packet, and then adding the random delay code on the OpenSSH.

Adding the random delay code

We found that sending packet process in OpenSSH handled by function `packet_send()` in C source file `packet.c`. The file `packet.c` containing two basic function for packet sending, they are `packet_send1()` and `packet_send2()`. That two function bundled in function `packet_send()` that allow process to choose the basic function will be used. Random delay function, `usleep(void)` in this case, added in function `packet_send()` before `if` statement contained in file `packet.c`.

Here the part of code in `packet.c` :

```
#include <sys/time.h>
.
.
void
packet_send(void)
{
    usleep(rand()%300000);
    if (compat20)
        packet_send2();
    else
        packet_send1();
    DBG(debug("packet_send done"));
}
```

Then add some declaration in main program of SSH client. The main program is coded in file `ssh.c`. The delay function `usleep()` got the input from function `rand()` and the function `rand()` got its seed from function `srand()` in file `ssh.c`. The computer's current time become the input of function `srand()`.

Here the part of code in **ssh.c** :

```
#include <sys/time.h>
#include <unistd.h>

/*
 * Main program for the ssh client.
 */
int
main(int ac, char **av)
{
    /* Getting rand() Seed from current time */
    time_t the_time;
    the_time=time((time_t *)0);
    srand(the_time);
}
```

6 Evaluations

In this research, we have an attack model applied to both OpenSSH software, before and after addition of random delay. We define following assumptions:

1. The considered users are familiar with keyboard typing and use touch typing.
2. The statistics are based on a fixed text with length of 16 characters. The demand on the typed text was, that the latency of two-character pairs pointed out their patterns.
3. Timing information collected by asking user to type a text 30-40 times and picks some character pairs from the text.
4. The network latency can be ignored to model the worst case.
5. The program for collecting the keystroke time of users placed on the SSH server

Users with original SSH client made a connection to SSH server and was asked to type the given text. Keystroke time from users was collected and stored in SSH server. Then, we tried another experiment by collecting timing information with addition of simple random delay in range 0-300 milliseconds. We choose this range due to the maximum time from previous experiment is about 300 milliseconds.

Here the result in histogram using Gaussian distribution.

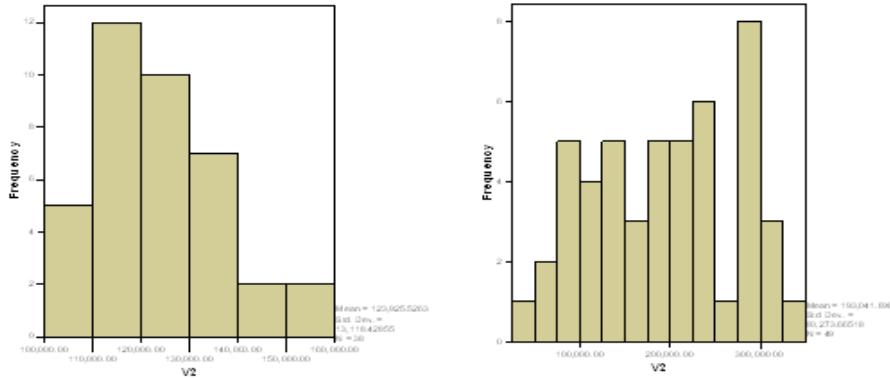


Figure 4. The distribution of inter-keystroke timings for pair t-h before and after addition of random delay.

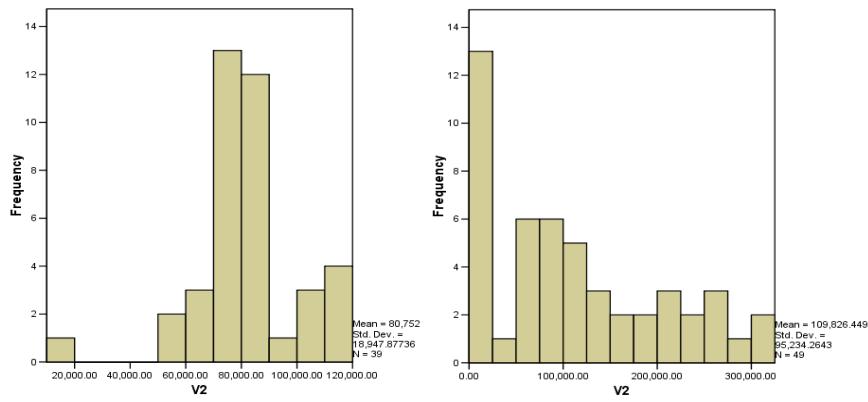


Figure 5. The distribution of inter-keystroke timings for pair u-i before and after addition of random delay.

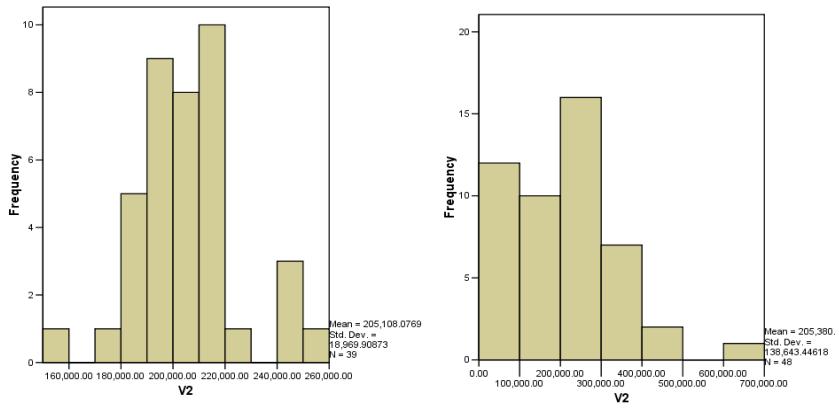


Figure 6. The distribution of inter-keystroke timings for pair *i-c* before and after addition of random delay.

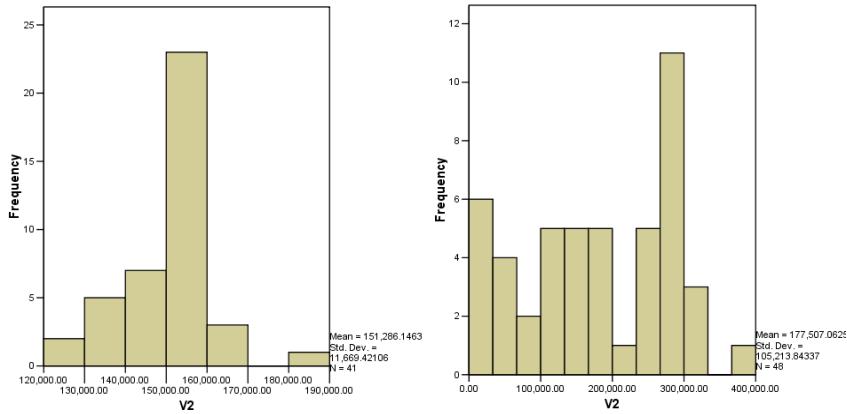


Figure 7. The distribution of inter-keystroke timings for pair *c-k* before and after addition of random delay.

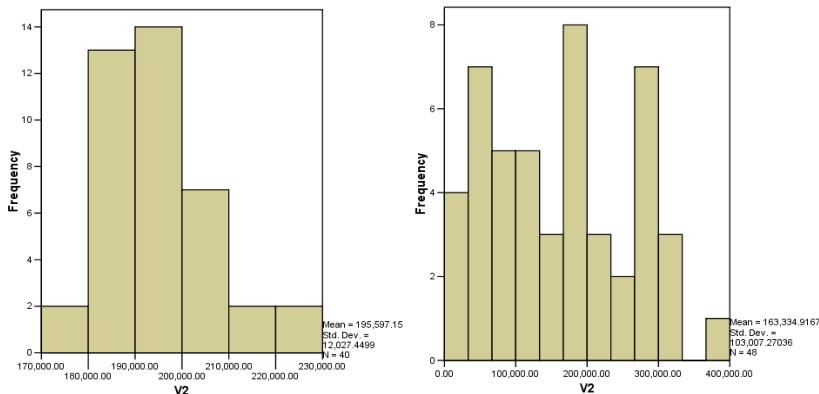


Figure 8. The distribution of inter-keystroke timings for pair *k-b* before and after addition of random delay.

Comparison table of the experiment result.

Pairs	Before		After	
	Mean	Standard Deviation	Mean	Standard Deviation
t-h	123,925	13,118	193,041	80,273
u-i	80,752	18,947	109,826	95,234
i-c	205,108	18,969	205,380	138,643
c-k	151,286	11,669	117,507	105,213
k-b	195,597	12,027	163,334	103,007

Figure 9. The distribution of inter-keystroke timings for pair some pairs of characters before and after addition of random delay.

Analysis

From the experiment above, we can see that the distribution of latency of each character pairs have a significant changes. The Figure 9 shows that the standard deviation of inter-keystroke time after addition of random delay increases about 14%.

Some keystroke time with little standard deviation value have a big probability to exploited by timing attack. For example, using original version of OpenSSH (without random delay), when we have time sequence of 80 milliseconds got from eavesdropping the SSH session, it is a big probability that the character pair pressed by user is u-i. Adding the random delay in packet sending on SSH session would randomize the timing information of the keystroke. It has a little chance that the attacker can reduce the effectiveness of the randomized noise, because:

1. The randomization of the delay is unpredictable.
2. The randomization range and the natural keystroke time are overlapping.
3. If the attack system is on a network with any significant traffic, the network latency would obscure the keystroke timing information.

7 Conclusions

In this paper, we point out that SSH is vulnerable to timing attack. Some people are familiar with touch typing. They have stable pattern when press the keyboard. It is possible to use the timing information to infer the key sequence being typed. This becomes a security problem. If an attacker can get inter-keystroke timing information from user, then later he may learn and determine what user type from the packet arrival time.

Therefore, we describe a countermeasure against timing attack on SSH by adding the random delay. We prove that adding the random delay in packet sending on SSH session would randomize the timing information of the keystroke and destroying the keystroke timing statistics so that attacker cannot determine the keystroke time pattern of each pairs pressed in SSH session.

References

- [Song,2001] Dawn Xiaodong Song, David Wagner and Xuqing Tian. 2001. *Timing Analysis of Keystrokes and Timing Attacks on SSH*. 10th USENIX Security Symposium.
- [Noack,2007] Noack, Andreas. 2007. *Timing Analysis of Keystrokes and Timing Attacks on SSH Revisited*. Chair of Network and Data Security Seminar.
- [Dai,2002] Dai, W. 2002. *Attack Against SSH2 Protocol*. Mail list at ietf-ssh@netbsd.org.
- [Hogye,2001] Michael Augustus Hogye, Christopher Thaddeus Hughes, Joshua Michael Sarfatyand Joseph David Wolf. *Analysis of the Feasibility of Keystroke Timing Attacks Over SSH Connections*. Research Project at University of Virginia, December 2001.
- [Barrett,2005] Daniel J. Barrett, Robert G. Byrnes, Richard E. Silverman. 2005. *SSH: The Secure Shell, The Definitive Guide, Second Edition*. O'Reilly.
- [Schechter,2002] Stuart E. Schechter, Jaeyeon Jung, Will Stockwell, Cynthia Mc Lain. 2002. *Inoculating SSH Against Address-Harvesting Worms*.
- [Mao,2004] Mao, Wenbo. 2004. *Modern Cryptography: Theory and Practice*. New Jersey: Prentice Hall.
- [Stahnke,2006] Stahnke, Michael. 2006. *Pro OpenSSH*. New York: Springer Verlag.
- [Snader,2005] Snader, Jon C. 2006. *VPNs Illustrated: Tunnels, VPN, and IPSec*. USA : Addison Wesley.
- [DS,2001] Solar Designer and Dug Song. *Passive Analysis of SSH (Secure Shell) Traffic*. Openwall Advisory OW-003, August 2001.
- [RFC4251] *The Secure Shell (SSH) Protocol Architecture*
- [RFC4252] *The Secure Shell (SSH) Authentication Protocol*
- [RFC4253] *The Secure Shell (SSH) Transport Layer Protocol*
- [RFC4254] *The Secure Shell (SSH) Connection Protocol*