Detecting Stepping-Stone Intruders with Long Connection Chains

Wei Ding[1], Matthew J. Hausknecht[2], Shou-Hsuan Stephen Huang[1], Zach Riggle[3]

[1] Department of Computer Science, University of Houston, Houston, TX, USA
[2] Department of Computer Science, Emory University, Atlanta, GA, USA
[3] Department of Computer Science, Michigan State University, East Lansing, MI, USA
shuang@cs.uh.edu

Abstract—*It is generally agreed that there is no valid reason to use a long connection chain for remote login such as SSH connection. Most of the stepping-stone detection algorithms installed on a host were designed to protect the victim of a third party downstream from where the algorithm is running. It is much more important for a host to protect itself from being a victim. This project uses an approximated round-trip time to distinguish a long connection chain from a short one. Several measures were studied to distinguish long chains from short ones. An estimated roundtrip time was defined to measure the chain length. Preliminary result suggests shows that the proposed algorithm can distinguish long connection chains from short ones with relatively low false rate.*

Keywords: *Intrusion Detection, Stepping-Stone, Security, Connection Chain.*

I. Introduction

In order for intruders to steal information from a host, it is necessary for the intruders to remotely login to the host. To avoid being detected, most of intruders use long connection chains of *stepping-stones* to reach their destination host (called the victim host). Existing malicious stepping-stone chain detection research has been concentrated on detecting intermediate hosts [1-8]. Detecting malicious connection chain is much more challenging from a victim's perspective than at stepping-stones. This is due to the fact that a stepping-stone can perform timing and correlation analysis with all of the information sent between the attacker and victim. The packets from the intermediate host to the victim and back form a closed loop. Previous research approaches were able to detect a growing downstream connection chain in real time.

However, these stepping stone detection methods have several drawbacks. First, most of the benefits go to the host at the end of the chain, most likely an unknown third party. Secondly, the stepping stone is only able to gauge the maliciousness of a connection by the number of downstream hops it detects [1]. If the stepping stone is very near the victim in the connection chain, it may not be unable to distinguish a malicious chain from a benign connection.

Victim-based detection, attempting to address the issues outlined above, has many difficulties of its own. First, there is no straightforward method of estimating the full round-trip time (RTT) for the length of the connection chain. This is due primarily due to the nature of tunneled SSH connections, and the fact that SSH is an interactive terminal session. This means that over the course of an SSH session, there is no point in time at which the server sends data to the client and the client's machine automatically sends a reply back to the server. Even if such a feature did exist, it is likely that the client-response would come from the nearest host in the connection chain. Stepping Stone detection, in contrast, relied upon the time difference between the attacker sending data downstream, and a response from the server passing back upstream ("reply echo time").

II. Our Approach

While it may not be possible to determine the upstream full round-trip time with the same certainty as the downstream, it is possible to find the time difference between the server sending a reply to the client and the client sending the next packet to the server with relative certainty. This time difference usually represents the full round trip time plus the time it takes the user to generate the next packet (via keystroke).

$$\begin{aligned} \textit{Time Diff} = \; & \textit{Time to send an Echo packet} + \\ & \textit{User Delay Time} + \\ & \textit{Time to send the next packet}, \\ = \; & \textit{Full Round Trip Time} + \\ & \textit{User Delay Time}. \end{aligned}$$

Here, the Echo Time of the previous echo and Send Time of the next packets combined to form an approximate round-trip time.

Therefore, if the time it takes for the user to press the next key, i. e., the user delay time, is subtracted from the time difference, the full round trip time remains. Our approach seeks to estimate the user delay time in order to find the full round trip time. More specifically, our algorithm finds the time difference between the client's enter keystroke to submit a command and the client's next keystroke to start a new command. It then analyzes this time gap based on several other features of the connection and attempts to determine the full round trip time and ultimately the length of the client's connection chain.

It is not an easy task to estimate the user delay since the time gap between typing varies significantly even for the same user. We chose to analyze not all but some special gaps. We selected the time gap between the end of a command and the beginning of the next command because of the expectation that a user will often need to see the command's output before starting to type the next command. For example, it is often necessary to see the file listing returned by a directory listing ("`ls`") command before selecting which directory to change into ("`cd`"). Other possible time gaps such as the gap between individual keystrokes would have been equally valid for analysis; however, our algorithm is designed to ignore these gaps because of the possibility that clients, when using a long connection chain, may not wait for each character to appear on their terminal before typing the next. This would result in an unusually short delay, often shorter than the actual full round trip time. This is probably the most difficult part of the analysis. Imagine that a packet is echoed back from one end of the chain while a packet from the originating host is already on the way. The two packets may cross each other somewhere in the middle of the connection chain. At the receiving end, the time difference of these two packets is very small.

While it might seem that user delay times would be dramatically larger than full round trip times, we found that for long connection chains, the full round trip times can be on par with the user delay times that we selected. For example, using a ten hop chain of reasonably fast hosts (40 milliseconds round trip time), there would be a 400 millisecond delay which is approximately equal to the delay of certain pairs of commands for many of our recorded users.

As mentioned earlier, our algorithm seeks to estimate the length of a connection chain by finding the full round trip time. This is found by subtracting the estimated user delay time from the time gap between the enter key-press and the next keystroke. The user delay time is estimated to be the client's average typing speed. While this estimation does not account for the time the user might spend reading/thinking before starting to type, our approach seeks to target those pairs of commands which require minimal cognitive delay. By subtracting the estimated user delay from the total gap time, we are left with the estimated full round trip time.

Unavoidably some users will wait long periods of time between certain pairs of commands. For this reason, estimated full round trip times greater than a threshold value of two seconds were discarded.

In order to accurately detect measures such as the user typing speed, and the occurrence of a new command, it was necessary to examine certain characteristics of an SSH session. Despite the fact that all of the data in an SSH session is encrypted, much information can still be gathered simply by monitoring the quantity, characteristics, and timing of packets. Session characteristics which allowed the detection of keystrokes, new commands, and nearest hop round trip time will all be discussed. Note that all the data about user delay is taken at the victim host, not at the source of the chain.

Keystroke Detection. The TCP header of all packets in TCP sessions (SSH packets included) holds information about, among other things, the source port, destination port, sequence number, and acknowledgment number of the packet among other pieces of information. The latter two numbers were used extensively to detect nearly all of the desired SSH session characteristics.

In a normal flow of information at the start of a TCP session, the client will connect to the server with a SYN packet, telling the server its sequence number. The server will then respond with a SYN ACK packet, providing its own sequence number and setting its acknowledgment equal to the client's next sequence number. The client will then (a) send a reply (echo) packet together with an acknowledgement (ACK) of the received packet, or (b) respond with an ACK packet if no response is ready within a certain period of time. The sequence number of the sender is used as the acknowledgement number of the echoing host.

After the three way TCP handshake has been completed, the fourth and final packet demonstrates the client sending another packet to the server. In the context of an SSH connection each character typed by the client generates a series of packets, similar to the ones above. The next packet would indicate that the client has typed another character. This can most reliably be detected by the fact that the last two packets have identical sequence or acknowledgment numbers.

In this way, our algorithm was able to detect when the client had entered the first keystroke after finishing a command. Additionally, by analyzing the time gaps between individual keystrokes, it is possible to find a user's average typing speed. Because people will generally pause while typing, gaps which were far larger than current average typing speed were discarded. Similarly, time gaps which were far smaller than the current average were also eliminated.

Command Detection. New command detection is slightly more difficult and less reliable than keystroke detection. The intuition behind command detection is that after the client enters a command, the amount of data that is sent back will be large enough to exceed one block size for the encryption algorithm being used. Note that the DES block size is 64 bytes, so all data less than 64 bytes in size will be padded to take up 64 bytes, such as an individual character. Data larger than the block size, such as a directory listing, will be padded to the next block size, and will thus be reported as being a packet with a larger payload.

Additionally, these post-command packets exhibit the characteristic that each packet's sequence number is the same as the last packet's acknowledgment number. Because this, our algorithm is able to distinguish the series of packets denoting the return of a command from the series of packets generated has the user continued typing new characters. However, the number of packets generated in response to a new command can vary greatly. For example, "`ls /usr/bin/`" generates 290 post command packets while simply pressing enter at an empty prompt only generates five. Network latency and user typing speed can also cause the number of post command packets to vary. Our approach flagged a new command after observing four post command packets. In practice we found this to be quite accurate and reliable.

Near Hop Round Trip Time (nRTT) Detection. The last necessary piece of information to our algorithm was the round trip time to the nearest hop in the connection chain. This information was rather easy to gather due to the nature of the TCP session. After every data packet sent by the client to the server, the server replies with an acknowledgment to the last host in the connection chain. The last host then automatically sends an acknowledgment back to the server, completing the sequence of packets. The round trip time to the nearest host can then be found by examining the time difference between the server sending the acknowledgment to the last host in the chain and the client replying with its own acknowledgment. Since the nearest host upstream from the victim machine is communicating with the victim, the IP address can be found in the packet header. Thus this is the only host on the chain that one can access from the victim host.

Estimated RTT. We define a measure, called *estimated roundtrip time*, to distinguish a long chain from a short one. We start out with the time gap between an echo packet and the next send packet received at the victim's host. We then subtract the average user delay (as defined earlier) from this quantity. Out objective is to see if the "estimated round-trip time" (eRTT) can be used to differentiate the chain length. This definition will be used for the rest of the paper. More work is needed to find a better measure.

III. EXPERIMENTAL METHODOLOGY

A total of seven computers were used to build chains and collect data. The computers ran various varieties of Linux and all were connected to the Internet via high speed connections. Two computers located at the University of Houston were used to listen for incoming connections (victim) and to initiate connection chains (attacker). The other five computers were located in various regions of the US and primarily served as intermediate hosts in the building of connection chains.

SSH session data was collected on a machine running the Slackware 12.1 Linux distribution. A total of 61 SSH sessions contributed by four different users were recorded and analyzed. These sessions consisted of the user logging into an SSH tunnel containing from zero to eight intermediate hops and executing a series of commands. Five different routes or chains of hosts were used. Each route employed a unique combination of hosts.

To simulate natural typing, users were given "objectives" to accomplish rather than a list of commands to type. Some example objectives were writing a short program, searching for text in a group of files, and creating a "tar" archive. Sessions generally tended to last around five minutes although some took up to fifteen. Clearly the users were slow at the beginning since they may be unclear of the intention of the instruction. After a few attempts, their speed stabilized. We discarded the first few data collections since they don't represent a true user behavior.

Our initial effort was to compare the eRRT and the nRTT to see if the ratio can be used as an indication of how long the chain is. Theoretically, if our eRTT is close to RTT, and the nRTT is a good representation of the network, the ratio should be a good estimator of the chain length. However, the result of this approach does not give us good result. Work is continuing on better estimating the chain length. It turns out that eRTT itself is a better measure to use.

Graph of the eRTT of chains with one intermediate hop to chains with eight intermediate hops is shown in Figure 1. Each marker represents a new command. The eRTT of each chain is determined by our algorithm described above. In each experiment, the number of packets collected varies from 31 to 100. We took the last 20 packets from each

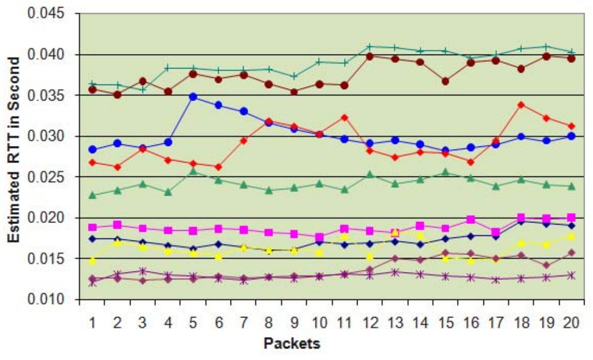

Figure 1: Estimated RTT between length-1 and length-8 Chains

experiment since they are more stable than the beginning of the packet stream. The top five series in Figure 1 represent chains of length 8 and the bottom five represent chains of length 1. On average, the eRTT rating of a length-8 chain is about 200% higher than that of length-1 chains.

Rank Checking. It is very difficult for us to correctly determine the length of a chain. It is, however, mush easier to determine which one of many chains is a long chain. In real situation, the chance of have two intruders attacking the same host is low. The purpose of the rank checking is to see if our algorithm can correctly identify the longer chain when given chains of various lengths. Rank checking was first employed upon the sessions. The objective of this analysis was to compare two sessions of different length and correctly rank the longer chain over the shorter. This ranking was accomplished by finding a representative value for each of the sessions, higher values indicating more suspicious connections. The median of the eRTT of each session is used to represent the session. Using this approach, each of the user sessions was compared to all other sessions of all other lengths. Each column in Table 1 below represents the number of intermediate hosts in the connection chain. Each cell contains the percentage of correctly ranked sessions. For example, the cell at Row 2 and Column 5 represent the success rate (84%) when a chain of 2 hops is compare with a chain of 5 hops. Each cell is the average of at least 5 experiments.

It might be noted that each of the zero chains had a 100% success rate. This is because all of the zero chains (those with no intermediate stepping-stone hosts) were direct connections over the local area network. The local connection is mush faster than a chain connected to a host off campus. For this reason they are omitted from Figure 2 below. Also note that a chain of *i*-hops consists of (*i+1*) SSH logins. Here we average the success rate of detection based on the difference of the chain of 1 to 7 hops. The figure indicates that the success rate for a difference of 3 is about 84%. Even if the difference is 1, the success rate is about 67%.

Ranking analysis fails to indicate the degree of separation between two different incoming SSH sessions. In order to truly determine the success of our approach, it is

Table 1: Correct Ranking Percentage of Sessions

	1	2	3	4	5	6	7	8
0	100	100	100	100	100	100	100	100
1	-	84	72	92	100	100	84	100
2	-	-	44	64	84	92	72	88
3	-	-	-	64	84	92	80	96
4	-	-	-	-	68	88	68	88
5	-	-	-	-	-	76	56	68
6	-	-	-	-	-	-	44	48
7	-	-	-	-	-	-	-	56

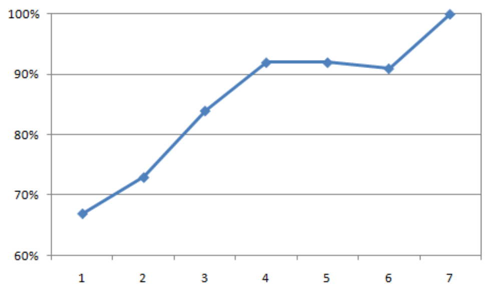

Figure 2: Percentage Correctly Ranked with different chain length difference

necessary that the long chains be substantially separated (easily differentiable) from the short ones. Short chains were (somewhat arbitrarily) defined as those connections containing from zero to two intermediate hops and long chains were defined as connections containing from six and eight intermediate hops. Representative points for all long chain sessions were plotted in decreasing order while representative points for all short chains were plotted in increasing order. The result is summarized in Figure 3. In this Figure, we collected data from 15 long chains and 15 short chains. The data are arranged in decreasing order for the long chains and increasing order for the short chains.

If an absolute threshold (doted line) was selected at the intersection point of the long and short chains, our algorithm would have correctly identified 13 of 15 long chains and 13 of 15 short chains, yielding a false positive rate of 13% and a false negative rate of 13%.

However, the consequences are higher for every false negative than for each false positive detected. More harm is done in preventing one legitimate user from accessing the system than from allowing one illegitimate user access. For this reason, if a threshold was set just above all of the short chains (~.0195), the algorithm would have correctly identified 8 of 15 long chains while keeping all short chains safe. This would be associated with a 47% false negative rate and a 0% false positive rate. Each host probably has to conduct experiments to determine the best threshold value to separate long and short chains. The system administrator may have to determine the tolerable level of false positive

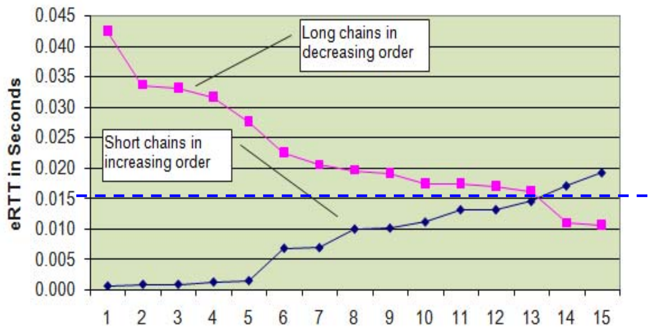

Figure 3: Comparison of eRTT of short vs. long chains

and false negative.

IV. Conclusion

We have proposed a new approach to detect long SSH connection chains at the victim host. Our method of detection centers around analyzing the delay between the time a user presses enter to finish a command and the time that the user types the next character. Taking into account the user's typing speed, it is possible to estimate if the user is connected through a long or a short chain. Preliminary results show that 86% of long chains can be correctly identified with a false positive rate of 13%.

Our approach differs from previous work in that previous methods could only detect long downstream connections from the perspective of a stepping stone. All benefits of detecting such suspicious connections would go to the end hosts, who are likely unaffiliated with the implementer of the software.

There are however several limitations to our approach. First, while 61 total sessions were recorded and analyzed, they were only contributed by four different users. A much broader sampling of users would have been desirable. Also, because our algorithm hinges on the assumption that a human is ultimately typing into the terminal, none of our chain length predictions would remain valid if a computer or script were entering commands. Additionally, our approach assumes that the user is typing normally. It is quite possible that an intruder, being aware of the specifics of our detection algorithm, would be able to intentionally alter his typing speed or command delay in order to hinder accurate detection. Lastly, our approach flags all incoming connections from the LAN as legitimate which would present a weakness if an attacker was able to successfully compromise another computer on the LAN and then attack our machine. Some future work might investigate the effects of X11 or other graphical sessions on our algorithm.

Acknowledgement

The research is supported in part by a Research Experience for Undergraduates (REU) Grant from the National Science Foundation (CNS-0755500) awarded to the University of Houston.

References

[1] J. Yang and S.-H. S. Huang, "A real-time algorithm to detect long connection chains of interactive terminal sessions," in *Proceedings of the 3rd international conference on Information security* Shanghai, China: ACM, 2004.

[2] D. L. Donoho, A. G. Flesia, U. Shankar, V. Paxson, J. Coit, and S. Staniford, "Multiscale Stepping-Stone Detection: Detecting Pairs of Jittered Interactive Streams by Exploiting Maximum Tolerable Delay " in *5th International Symposium on Recent Advances in Intrusion Detection*. vol. LNCS 2516: Springer, 2002, pp. 17-35.

[3] Y. Kuo and S.-H. S. Huang, "An Algorithm to Detect Stepping-Stones in the Presence of Chaff Packets," in *International Conference on Parallel and Distributed Systems (to appear)*, 2008.

[4] Y.-W. Kuo and S.-H. S. Huang, "Stepping-stone detection algorithm based on order preserving mapping," in *International Conference on Parallel and Distributed Systems*, Hsinchu, Taiwan, 2007, pp. 1-8.

[5] Y. Zhang and V. Paxson, "Detecting stepping stones," in *Proceedings of the 9th conference on USENIX Security Symposium - Volume 9*, Denver, Colorado, 2000, pp. 171-184.

[6] L. Zhang, A. G. Persaud, A. Johnson, and Y. Guan, "Detection of stepping stone attack under delay and chaff perturbations," in *25th IEEE International Conference on Performance, Computing, and Communications Conference (IPCCC)*, 2006, p. 10 pp.

[7] T. He and L. Tong, "A Signal Processing Perspective to Stepping-stone Detection," in *CISS*, Princeton, NJ, 2006, pp. 687-692.

[8] T. He and L. Tong, "Detecting encrypted stepping-stone connections," *IEEE Transactions on Signal Processing,* vol. 55, pp. 1612-1623, May 2007.

[9] K. H. Yung, "Detecting Long Connection Chains of Interactive Terminal Sessions" in *Recent Advances in Intrusion Detection*, Lecture Notes in Computer Science 2516, 2002, pp. 1-16.