Detecting Network Intruders by Examining Packet Crossovers in Connections

---

A Thesis Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Hongyang Zhang

May 2014

Detecting Network Intruders by Examining Packet Crossovers in Connections

---

Hongyang Zhang

APPROVED:

Dr. Shou-Hsuan Stephen Huang
Department of Computer Science

Dr. Weidong Shi
Department of Computer Science

Dr. Fatima Merchant
College of Technology

Dean, College of Natural Sciences and Mathematics

Acknowledgements

I would like to thank my advisor, Dr. Stephen Huang, who guided my research throughout whole graduate student life with his kindness and patience. I would also like to thank the whole research group for helping and supporting me during my study here. My appreciation goes to Dr. Larry Shi and Dr. Fatima Merchant for serving as my committee members and providing their comments. I would also like to thank my friends, who supported me to finish my experiments by providing their own machines, Chenlei Zhang from the University of Alberta, Lingjia Deng from the University of Pittsburg, and Dr. Jianhua Yang from Columbus State University. Last but not least, my sincerest appreciation goes to my family, especially my wife Han Zhang, for constantly supporting me. I love you all.

Detecting Network Intruders by Examining Packet Crossovers in Connections

---

An Abstract of a Thesis

Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Hongyang Zhang

May 2014

Abstract

Routing packet traffic through a chain of hosts is a common technique for hackers to attack a victim machine without exposing themselves. Generally, a long connection chain formed is an indication of the presence of an intruder. Previous work has mostly focused on detecting stepping-stone hosts. Few researchers have addressed the issue of long connection chains (especially downstream detection). A challenging issue in this area is to detect users connecting to a server using a long connection chain with only the information at the end of the chain. This thesis presents a solution to the problem of detecting upstream long connection chains. We first observe that the longer a connection chain is, the more packet crossovers are generated. Thus we reduce the problem of detecting long chains to that of detecting unusually large number of packet crossovers along the chain between requests and responses at server side. However, the approach requires the packet information along the whole chain. Since we cannot directly measure the number of crossovers on intermediate nodes, we are forced to study the consequences of large number of crossovers. A detection algorithm has been designed based on the distribution of packet gaps. We validated our algorithm using test data generated on the Internet. The result shows a high detection rate of long connection chains from short ones without too many false positives.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Information security has becoming a more and more significant domain recently in the whole world, since everything has been digitalized to enjoy the convenience it brings. In particular, the Internet is regard as one of the most important inventions in the 20th century. Security across the Internet is essentially required in order to stop hackers from stealing information and damaging our networks. Even though a lot of measures have been deployed, such as firewall, IDS, IPS, etc., to prevent hackers from intruding the systems, there are still many intrusion cases happened recently. For example, servers at LinkedIn were compromised and 6.5 million users' passwords were stolen in 2012 [1]. In 2011, Sony PlayStation suffered from 24 days of network outage due to intrusions, which cost Sony $171 million loss. The attack occurred for 3 days and Sony had to turn off the PlayStation Network for maintenance [2] [3].

There are many different ways of attacking. Since network communication is divided into a 7-layer OSI model [17], each layer has a lot of corresponding methods to perform attacking. Our project focuses on the layer of Application, where Secure Shell protocol works. In the layer of Application, a great number of attacks can be performed, such as Denial of Service, SQL Injection, Man-in-the-middle attack, etc. In our case, we aim to detect hackers who are trying to tamper with systems. Normally, when a hacker tried to compromise a victim host, a secure connection needs to be set up for communication between hacker and victim. With the purpose of not being detected and

caught, most intruders use long chains of stepping-stones before login to victim hosts. Figure 1.1 below indicates the process of logging in a host using stepping-stones.

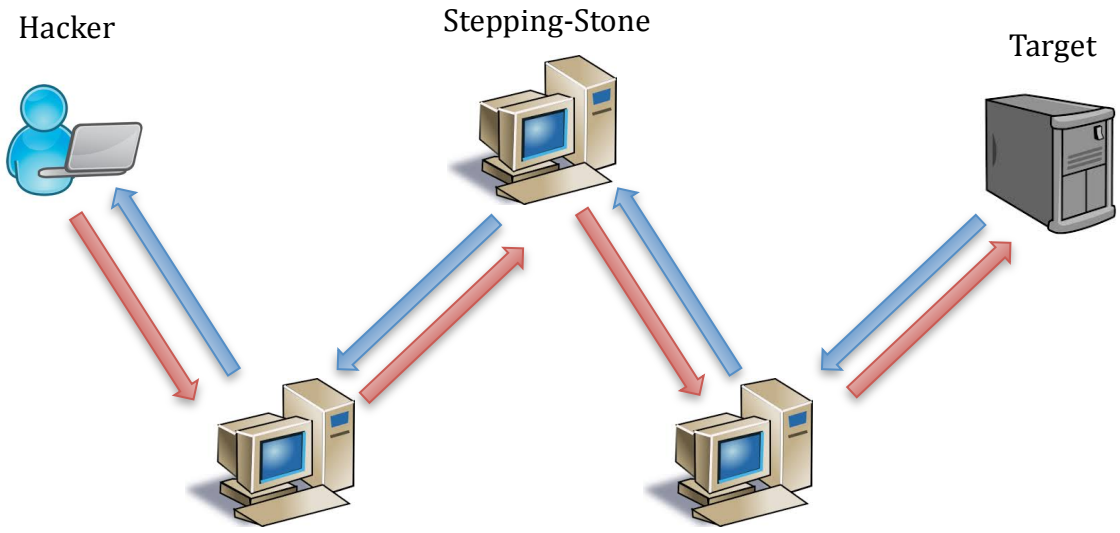


Figure 1.1: An illustration of a hacker attacking a target server using three stepping-stone hosts

This stepping-stone technique is widely used by hackers to perform intrusion because they would prefer to be anonymous and not being traced back. The intermediate nodes, called stepping-stones, along the long chain are usually previously compromised by hackers. Before conducting an intrusion to a target host, the hacker connects to a series of machines he controlled and carries out the final attack via the last intermediate node. The controlled hosts are usually picked by intruders and may be in different countries. Therefore it would be extremely difficult to trace back to the original attacker without the assistance of the system administrators of these intermediate hosts in real-time. Thus it is important to detect stepping-stones while the hackers are connected [4]. There have been many publications focusing on this area during the past two decades. Staniford-Chen and Heberlein [5] wrote the first paper addressing this issue. Without bothering the intermediate hosts, the paper proposed a method that helps to flag

suspicious activities, maintaining logs in case of an intrusion to be detected as starting from local sites, setting up their connections through external nodes. It also helps to enforce policies in terms of cross-traffic and to detect insecure combinations of legitimate connections.

Previous research primarily concentrated on intermediate host based stepping-stone detections, i. e., detecting stepping-stones by correlating streams of packet traffics. From the view of an intermediate host, the data generated by the attacker are sent downstream and a response from the server (target host) is passed backward upstream to the attacker. The request/response packets form a closed loop between an intermediate host and the target server. Meanwhile, the time difference between request and response is captured for detection. Yang and Huang [6] use time gaps in the closed loop to detect a downstream connection chain in real time. They are trying to stand between the attacker and victim, protecting the target host, which is usually an unknown third party connected by a long chain from a hacker. However, most of the times we do not have privilege on intermediate hosts on a long chain. It will be extremely helpful if we could detect intruders and avoid being compromised at the target host. There is no straightforward method that is able to measure the full Round Trip Time (RTT) from the target to the hacker. Since SSH is an interactive terminal session, the client's machine will not automatically send a reply back to the server for us to compute RTT. The responses that the server could receive are only generated from the closest node on the connection chain. As presented in Figure 1.2, the neighbor hosts are the only machines visible to the victim, which are directly connected to it. The rest hosts on the chain are beyond the vision of victim.

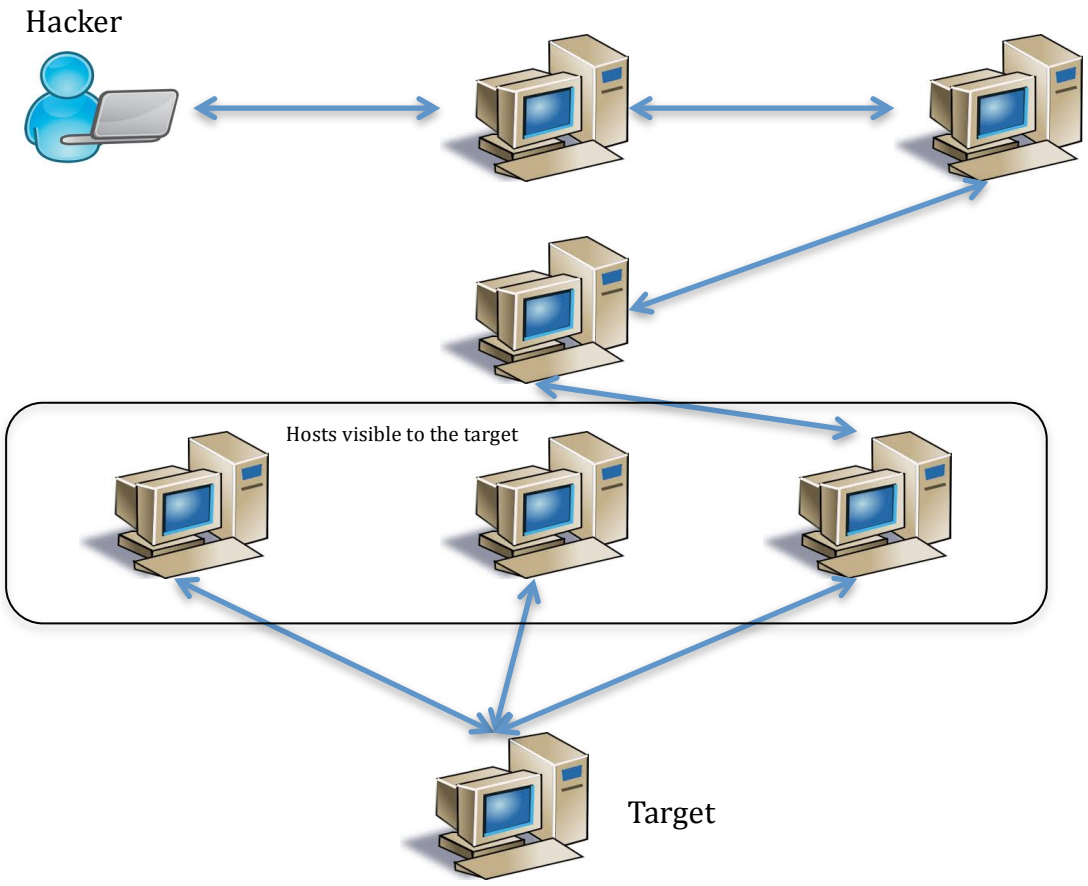


Figure 1.2: One of the three incoming connection chains is a stepping-stone attack using a long connection chain

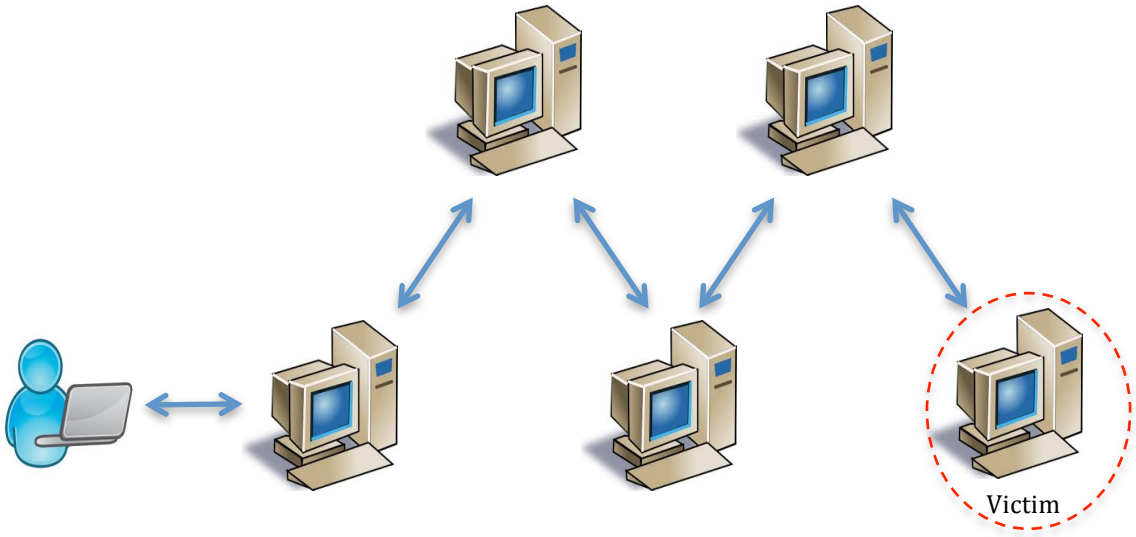


Figure 1.3: Detecting intruders at the end of a long connection chain

Ding et al. [7] propose an intrusion detection algorithms based on the victim host, which is at the end of the connection chain shown in Figure 1.3. Their goal is the same as ours, which is to detect and prevent suspicious intrusion connection chains by distinguishing long connection chains from short ones. To our knowledge, that is the only work done in this area.

Their approach is to compute the difference of the distribution of the packet gaps. Their hypothesis is the longer the chain is, the larger the gaps are. In their experiments to validate their hypothesis, connection chains of 4-hop and 6-hop are compared to short connection chains of length one. The result shows that the algorithm works, as 86% of long connection chains could be identified, with a false positive rate of around 13% in the case of 6-hop chains. The accuracy for the 4-hop chains is less impressive.

Our project's goal is to improve the accuracy rate of detecting long connection chains based on Ding's work. To their credit, they discovered the packet crossovers in long connection chains, even though they did not take advantage of that to detect long connection chain. It was also suggested that longer connection chains have more packet crossovers. Our work is based on these findings.

We collected the cross-traffic packets on each intermediate node and analyzed them to verify the hypothesis, that there could be many crossovers generated between machines when they are used as a stepping-stone. Moreover, the closer an intermediate node is to the hacker on a connection chain, the more crossovers it will generate. With such a conclusion, we develop a new method of detecting long connection chains by examining the gap differences between selected request/response packets, which only

requires data collected at the end of the connection chain. As a result, we are able to achieve a detection rate of 85% at a false positive of 5%. Moreover, the experiments were designed to identify connection chains of length of 3 hops, which is more realistic in real world.

The rest of thesis is organized into four chapters. Chapter 2 gives definitions about network security, intrusion detection, stepping-stones, and some related work. Chapter 3 studies the issue of packet crossover and its relationship with the length of the connection chain. We present an algorithm to estimate the number of packet crossovers as well as techniques used to analyze them. Experiments are conducted which validated our hypothesis on the relationship between packet crossover and the length of the connection chain. Chapter 4 shows a detection algorithm to identify long connection chains from short ones. We are able to detect long connection chains by examining data packets collected at the end of the chain only. The conclusion chapter summarizes our work and also gives some further prospect on stepping-stones detection.

Chapter 2

Literature Survey and Definitions

In this chapter we briefly review the numerous advances in the area of stepping-stone intrusion detection. Terminologies and assumptions are properly defined.

2.1 Definitions and Assumptions

Network security is concerned with a variety of security issues that happen on networks, especially on the Internet. Network security has become a hot topic of research from over the past few decades, as well as intrusion detection [7][8][9][10]. Intrusion detection provides the network as well as resources that are stored in the network, accessible devices protection from intruders stealing information from them. Authentication could be deployed as the first step defense of network security. People could use their username and password to be authorized into the system. There are also some further authentication mechanisms such as token, fingerprint, and retina recognition. Sometimes information security is also regarded as one aspect of network security, which focuses on protecting data from unauthorized access or being misused by legitimate users inside an organization.

Secure Shell (SSH) is a cryptographic network protocol for securing data communication, remote command-line login, remote command execution, and other secure network services between two networked computers. It connects via a secure channel over an insecure network, a server and a client running SSH server and SSH

client programs, respectively [16]. Although the data field of SSH packet is encrypted, we are still able to get the header of packet in plain text. With information contained in the header, we could be aware of the type of packet, source/destination IP, and size of packet.

More and more network attacks and security breaches happen with the growth of the Internet. It is necessary and inevitable for intrusion detection to come to stage. Intrusion detection plays a critical role in securing networks and systems since various attacks have caused so much trouble. It is a process of monitoring the network and system activities via auditing techniques to detect malicious behaviors or policy violations, which attempt to destroy CIA triad (confidentiality, integrity, and availability). Intrusion protection techniques are usually used to handle detected security events pairing with intrusion detection.

Stepping-stones are always involved when the hacker tries to hide their existence, which makes it relatively difficult to track the intruder. Thus it will be helpful if we can detect the stepping-stones that lead to the occurrence of intrusion. A hacker may log in a computer, start a new connection to another one from this machine and do the same thing thereafter. Those intermediate computers are called “stepping-stones”, which form a connection chain. Anytime such long chains are detected, we are convinced that someone may try to intrude the system and hide his existence.

During the connection of secure shell, each client’s keystroke generates a request packet (called “send packet”). Then the server will send one or more response packets (called “echo packet”) back to the client after receiving a request so that the content of

responses will show up in client's terminal window.

Normally, the client host is able to receive the response of previous request before sending out the next request. However, in some circumstances, the client host sends out the request packets before receiving responses due to large round trip time caused by long connection chain. In those cases, the request packet will "meet" the coming response packet halfway. We call this situation "crossover". Every time a request packet meets a response packet, one crossover is counted.

In summary, the type of intrusion that we discuss in this paper has the following characteristics: (1) the intruder must login into the target host in order to steal information or cause other damages to the system and (2) the data content of the connection may be encrypted and not visible to the detection algorithm. Denial of service attack, for example, does not fall under this category.

2.2 Previous Work

There have been quite a few studies focused on stepping-stones detection, which using various techniques to discover those hosts used by the hacker to escape tracing. For an intermediate host, there are a great amount of incoming and outgoing connections. If we are able to pair those connections, the host will be highly suspicious to be a stepping-stone. Figure 2.1 shows how this technique works.

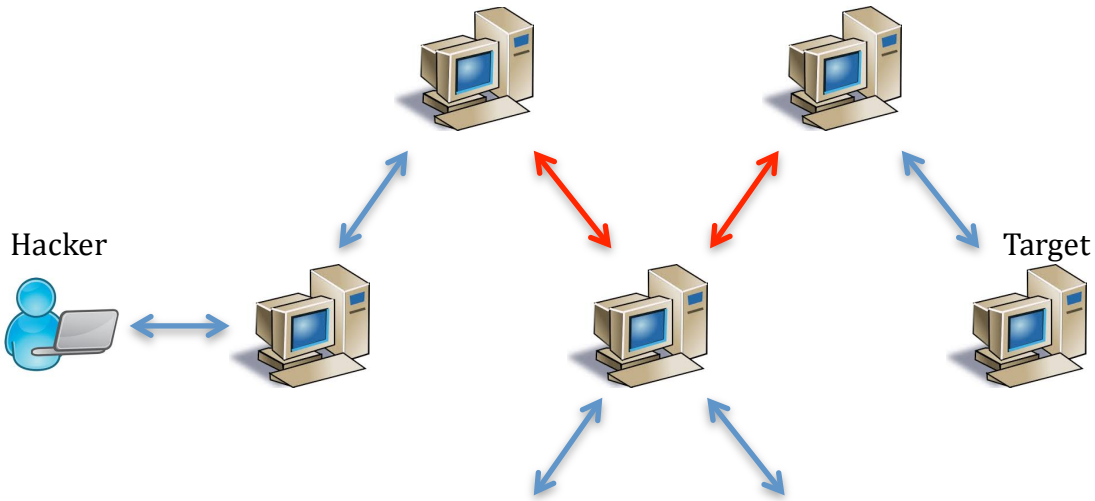


Figure 2.1: A host can be identified as a stepping-stone by correlating its incoming packet streams with its outgoing streams

An algorithm can be used to match those incoming and outgoing connections based on time gaps or other flags. If the host is detected as a stepping-stone, the connection from this host will be paid extra attention in case it is originated from the hacker.

Staniford-Chen and Heberlein [5] are the pioneers in this area. Their early research was based on the content of the traffic, which they captured. Thumbprints are short summary of the connection packet captured in cross-traffic. They compared these thumbprints to determine whether those bi-directional packets contain same content. If so, the host is likely to be an intermediate node of a long connection chain. However, this method is only able to detect stepping-stones using unencrypted messages, such as Telnet connection. It becomes useless when it deals with encrypted connections like SSH, whose content of packets are unreadable to people except for users at both ends of the connection. The thumbprint method is no longer valid.

Wang and Reeves [11] proposed a novel intrusion framework called “Sleepy

Watermark Tracing (SWT)". It integrates a sleepy intrusion response scheme, a watermark correlation technique, and an active tracing protocol. The "sleepy" infers that no overhead is introduced when no intrusion is detected. But it will become active when detecting an intrusion. The target host will inject a watermark into the backward connection of intrusion and then wake up to collaborate with routers along the chain. According to their results, SWT could achieve a high efficient and accurate source tracing when the hackers use telnet or rlogin to intrude the target, which are not encrypted during communication comparing to SSH. Wang and Reeves' work is quite inspiring, since they makes it possible to trace back to the original source node in real time when the hacker uses long chain to cover his presence in order to achieve interactive intrusion. This method is efficient and scalable to detected intrusion using unencrypted connection tools, which is a big milestone in the era of Telnet.

With the purpose of privacy, SSH is gradually becoming the most useful remote connection tool over Telnet. Thus we need some new techniques to detect intrusions because the previous methods do not work any more due to the encryption of connections. In order to adapt new ways of connection, Zhang and Paxson [4] proposed a method using timing correlation of ON/OFF periods of different connections to detect stepping-stone attacks. Relying on the properties of interactive traffic, such as packet sizes and idle periods, they do not need to look at the contents of packets of the connections to detect intrusions. Their algorithm provides a good accuracy and less workload of capturing packets by only keeps packet headers. But there is still a weakness in their approach that it cannot distinguish the legitimate stepping-stones from malicious ones.

Yung's research [12] focused on estimating the RTT for out-going connections

using request and response pairs between the client and downstream server. Yung's approach monitors an outgoing connection to estimate two metrics. The first metric is the time gap between requests from client and acknowledgements from the downstream server, which is used to estimate RTT between the client and the server. The second one is the time gap between client's request and server's response used to estimate how far away the targeted server is. These two metrics are also used to estimate how many hops there are on the connection chain to the victim. This method performs well in identifying connections with more than two downstream hops and can be used on interactive sessions, such as Telnet and SSH. However, the time gaps are greatly influenced by the network environment and the machine at the end, since any delay in network traffic and server machine could impact the size of RTT.

Yang and Huang [6] proposed an algorithm to estimate the length of downstream connection chain by monitoring outgoing and incoming packets. The algorithm computes the RTT in a request and response pair. By capturing the changes in these RTTs, the number of nodes in downstream chain could be estimated. With this approach, the user could find if his machine is used as a stepping-stone when the hacker is connected to a target server and take measures to stop the intrusion. The method is able to detect intrusion in real-time, which is significant in real world. It also delivers an accurate result of estimating the length of downstream connection chain. Meanwhile, it still has a restriction, like other approaches, that the algorithm can only estimate the length of downstream chain instead of the whole connection chain.

After Yang and Huang's work [6], they proposed two additional algorithms [13] of matching TCP Send and Echo packets. One algorithm is relatively conservative, and

can accurately match a smaller number of packets, whereas the other one is heuristic, and can match a larger number of packets with less accuracy. The authors justified the correctness of their conservative algorithm. By applying these two algorithms in the experiments, the result shows that two algorithms could achieve the same performance. If the conservative algorithm failed to generate enough data, the heuristic one will be used as a supplement. The combined algorithm can detect intruders in real-time. It can also estimate the encrypted connection chain length accurately even with fluctuated network traffic. The major weakness of this approach is that it requires capturing the packets throughout a connection session.

Since the previous research [13] was not able to capture enough send packets using a conservative algorithm, Yang and Huang proposed a new clustering partitioning algorithm [14] to extract the timestamps from send and echo packets of a connection chain.

After that, Ding et al. [7] continue detection work from another perspective by proposing an intrusion detection algorithm based on the victim host. They analyzed the delay between the time a user presses enter button to finish a command and the time that the user types the next character, and used an approximated upstream round-trip time to separate a long connection chain from short ones. Their result showed that this proposed algorithm was able to distinguish long connection chains from short ones with relatively low false rate.

Chapter 3

Packet Crossovers and their Relationship to the Length of Connection Chain

Ding et al. [7] discovered the “crossover” issue in their examination of the packets in a long connection chain. In this chapter we shall study the relationship between the number of crossovers and the length of the connection chain. We first formally define packet crossovers and state our hypothesis about the crossovers in Section 3.1. In Section 3.2 we describe our setup for the experiments to validate the hypothesis. Finally results from the experiments are summarized in Section 3.3 to validate our hypothesis.

3.1 Packet Crossovers and our Hypothesis

Since we need to find a feature that could separate long connection chains with short ones, let us review the process of SSH connection and the characteristics of protocol.

When TCP/IP Services is started on an SSH server host, the auxiliary server creates a listening socket for SSH. The SSH server is now ready to accept a remote connection request. When the client execute an SSH command on a remote client host, the SSH client is initiated. The client reads the configuration file and initiates a TCP connection to a server host using the specified destination port. On an SSH server host, the auxiliary server creates a copy of the server process, which reads the server's

configuration file. The SSH client and server exchange information about supported protocol versions. During the connection the SSH server runs in a loop, accepting request messages from the client, performing required actions, and returning respond messages to the client. For a stepping-stone type of connection, there is a new packet between every pair of successive hosts in the chain. For example, there are four separate packets used as vehicles to send the contents to the target machine in Figure 3.2 below. Each packet may be of different sizes and must be agreed upon the two hosts involved.

Each time a user on the client-side pressed a key, the client host generates a request packet with the character (padded and encrypted) as the content to the server. The server will send a response packet back to the client after processing the request packet. If the server is an intermediate stepping-stone host, it will turn around and send the request to the next sever and wait for its response. However, if the server is the intended target host, then the host processes the request, which includes sending a respond packet back with the same character in it. Occasionally, if a command has been received at the server side, it will send one or more packets back with the reply. A server also sends an acknowledgement packet if no respond was ready to be sent. When the user closes the connection, the server process terminates. The auxiliary server continues to listen for new SSH connection requests [15].

In TCP/IP protocol, a client is allowed to send a limited number of packets to the server without having to wait for the response. What was shown in Figure 3.1 with three request/response nicely separated is highly unlikely. The Round-Trip Time (RTT) is the difference of the time stamps between sending a packet and receiving its response. The RTT for different request packets will be different. It can be modeled as a random

variable depending on the network traffic and the availability of the hosts involved. Normally, request and response packets stay in the same sequence at both ends of the connection. Also round-trip times between two consecutive hosts are much shorter than intervals between keystrokes as showed in Figures 3.1 and 3.2.

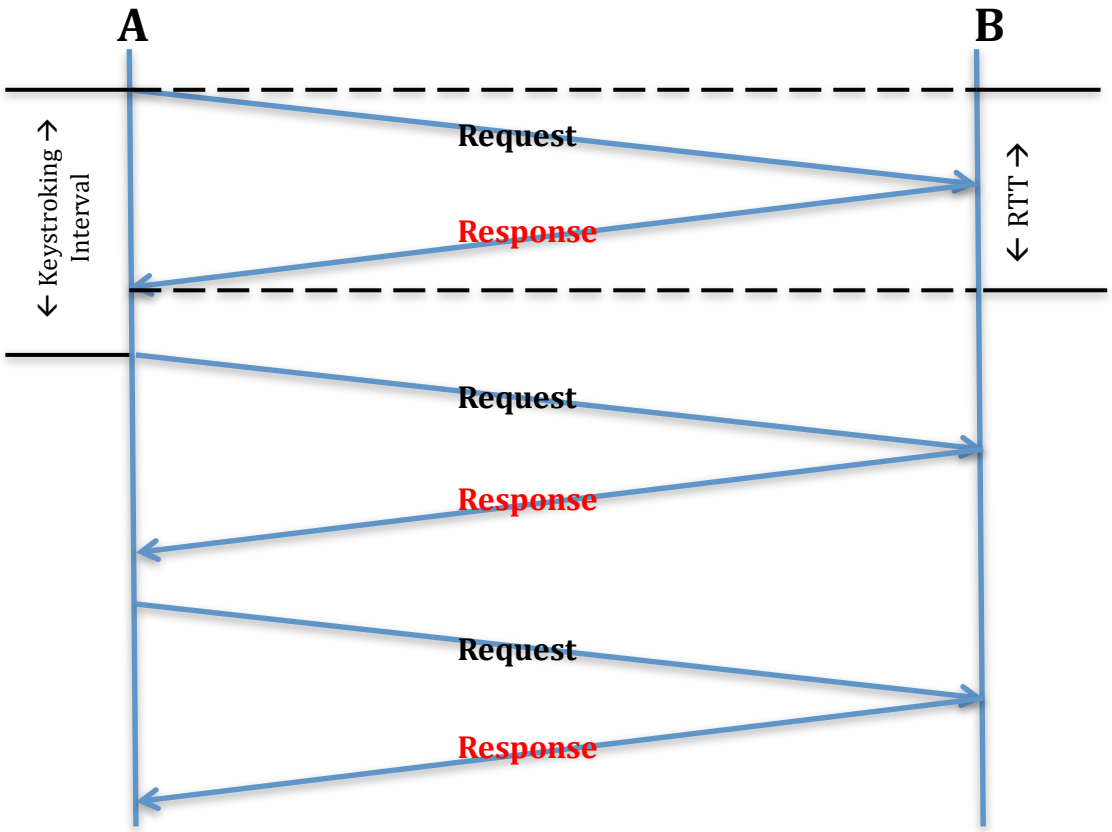


Figure 3.1: An Illustration of packet exchanges between two hosts A and B

However, in some cases, the connection chain is long enough that round-trip times are longer than intervals between keystrokes. For data transfer, the client is allowed to send further messages without waiting for the response to the request [16]. Therefore, if the client's keystroke intervals are longer than round-trip time, response packet will arrive at client's machine before another request is sent out. On the other hand, if the client's keystroke intervals are shorter than the round-trip time of the previous packet,

there will be two or more consecutive request packets sent before a response packet arrives. Figure 3.2 shows some packets with overlapping round-trip time. Intuitively, the round-trip time is proportion to the length of connection chain. Thus, it is reasonable to assume that the client's intervals are likely to be shorter than round-trip times in a long connection chain. When a response packet arrives at the client's machine later than another request packet is sent out, this response packet will "meet" the coming request packet halfway before arrival, which is called "crossover".

Crossover of the request and response packets deserves more explanation. When a "crossover" happens, the downstream node stays in the normal request/response packets order, whereas the upstream node sends out the next request packet before receiving a response packet for the previous one. Then in upstream node, there will be two consecutive request packets whose sequence is different from the downstream node. So if there is a flip over of request/response packets, a crossover is generated. Figure 3.2 shows this procedure.

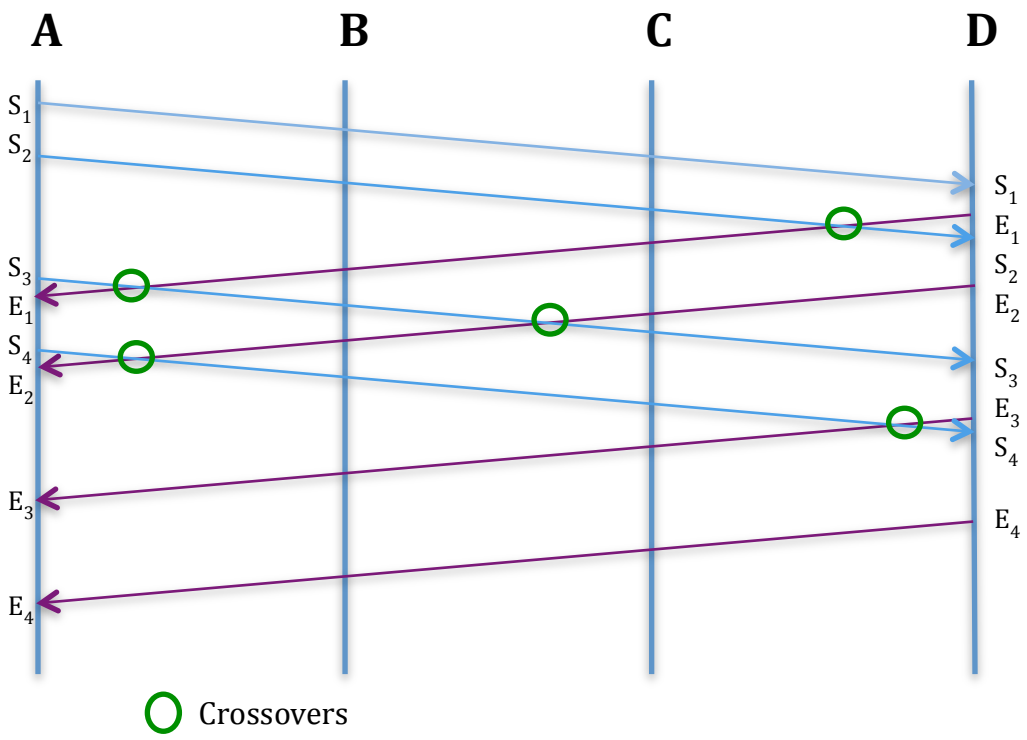


Figure 3.2 Crossovers in long connection chain

Based on the analysis above, we proposed a hypothesis that there are more crossovers generated in a long connection chain than in a short one. In other words, the longer a connection chain is, the more crossovers there are, which means the number of packet crossovers is positively correlated to the length of the chain.

3.2 Validation by Experiments

We designed experiments of real connection chains routing through four hosts on the Internet. Network packets were collected along every step of the chain. Collecting data during one-time connection ensures data not being influenced by network environmental differences and fluctuation.

3.2.1 Configuration

In our experiments, four different machines are used to act as client, server, and stepping-stones. Operating systems are Ubuntu and OS X with Secure Shell client/server and Wireshark installed. To simulate intrusions in real world, some of the machines are in different cities across the country. Two of hosts are located in Houston here on campus. One of the hosts is in Pittsburg, PA and another one stays in Columbus, GA. Our scenario is to connect the host one by one using SSH. With two nodes off campus, we are able to make sure that each connection is across the country. Therefore, the connection chain is long enough just like how hackers build. Figure 3.3 shows how our experiment is set up. Since CSU is simulated as the target host that a hacker is trying to attack, data collected from UH-2, Pittsburg, and UH-1 are regarded as packets that travels through 1 hop, 2 hops, and 3 hops respectively. We will see data collected from each different length chains later in next section.

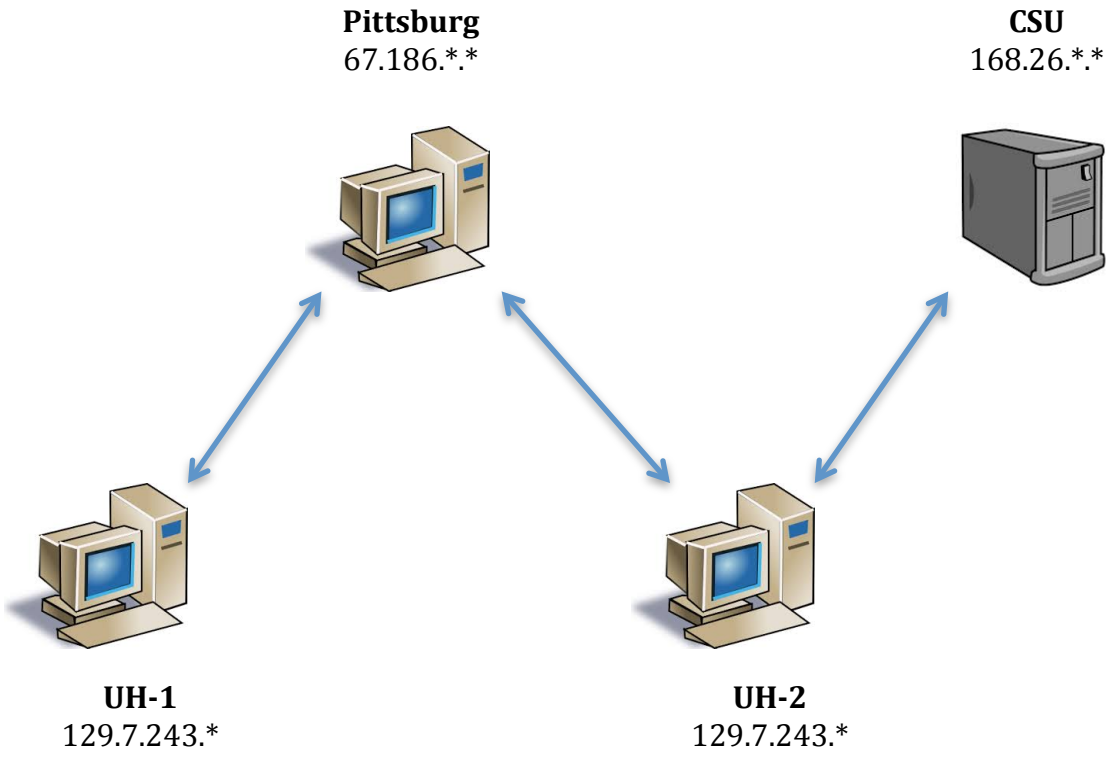


Figure 3.3 Connection chain setup in the experiment with two local hosts and two remote hosts

3.2.2 Data collection and analysis

To collect every Secure Shell packet, we start tshark (command line version of Wireshark) before the connection is set up and stop it after the connection is terminated. During the connection, we simulate hackers' actions such as executing system commands and editing files. We also tried to maintain the same and normal typing speed.

Since Secure Shell packets are the only packets we focus on, we eliminate those irrelevant packets such as TCP Ack, UDP, DHCP, and ARP etc. We conducted this experiment 20 times to guarantee our result's accuracy. All data packets are saved in text

files for further analysis.

As displayed in Figure 3.3 above, we can see that request packets start from UH-1 node and route through Pittsburg and UH-2, arrive at the server host, CSU. Response packets travel backward from CSU to UH-1. From UH-1 to CSU, there are three hops, two for Pittsburg and one for UH-2. So we collected SSH packets between two consecutive hosts to compute the number of packet crossovers. For example, we collected all SSH communication packets between UH-1 and Pittsburg at both hosts. Same collections also happened between Pittsburg – UH-2 and UH-2 – CSU.

After collection, we proposed an algorithm to compute the number of packet crossovers between two consecutive nodes. The algorithm works as follows:

1) Extract request/response packets from Host A and label them according to timestamps individually.
2) Extract request/response packets from Host B and label them according to timestamps individually.
3) Mix request and response packets respectively in time order.
4) Compare two sequences of label and compute how many flip overs occur, which equals the number of packet crossovers.

The number of packet crossovers for 3 hops, 2 hops, and 1 hop can be computed by applying the algorithm above to the packet data between UH-1 and Pittsburg, Pittsburg and UH-2, and UH-2 and CSU respectively.

3.2.3 Results

We perform the experiments for 20 times in order to get the most accurate results. Each experiment delivers one dataset consisting of data packets collected at four nodes. Then we could get data from three different lengths of connection chain. The result of experiments shows as follows in Table 3.1 and Figure 3.4:

Table 3.1 Percentage of crossover among packets

3-hop			2-hop			1-hop		
Packets #	Crossover #	Ratio	Packets #	Crossover #	Ratio	Packets #	Crossover #	Ratio
746	283	37.94%	729	110	15.09%	717	17	2.37%
778	325	41.77%	758	88	11.61%	746	2	0.27%
778	285	36.63%	760	79	10.39%	747	3	0.40%
778	303	38.95%	761	103	13.53%	749	2	0.27%
797	348	43.66%	779	165	21.18%	763	14	1.83%
789	311	39.42%	770	129	16.75%	750	11	1.47%
757	298	39.37%	739	69	9.34%	727	3	0.41%
766	333	43.47%	748	96	12.83%	736	8	1.09%
776	314	40.46%	753	111	14.74%	732	5	0.68%
758	304	40.11%	741	136	18.35%	729	2	0.27%
755	317	41.99%	738	110	14.91%	725	4	0.55%
756	292	38.62%	714	127	17.79%	680	27	3.97%
772	323	41.84%	750	114	15.20%	738	9	1.22%
760	311	40.92%	715	117	16.36%	698	12	1.72%
762	298	39.11%	703	99	14.08%	683	7	1.02%
788	321	40.74%	770	134	17.40%	755	18	2.38%
766	306	39.95%	729	129	17.70%	699	8	1.14%
743	311	41.86%	726	115	15.84%	711	8	1.13%
753	369	49.00%	741	115	15.52%	711	9	1.27%
770	413	53.64%	756	113	14.95%	733	7	0.95%

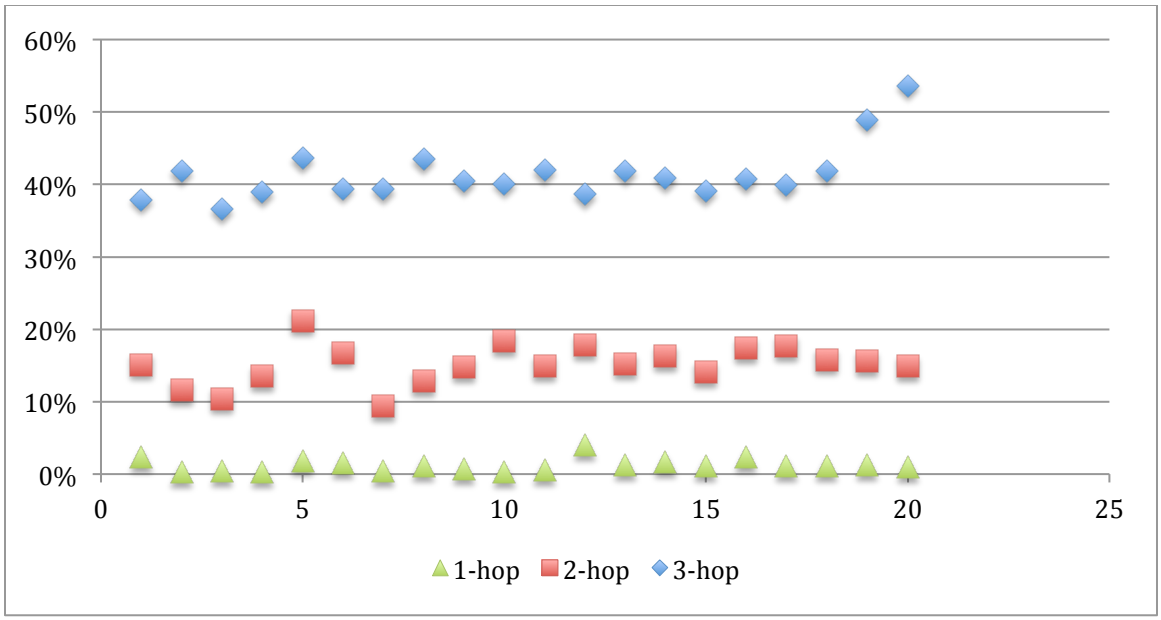


Figure 3.4 Ratio of crossovers for different lengths of connection chain

From the figure above, we could find that there are clear margins of ratio between different numbers of hops, which can be used to easily distinguish from each other. Thus if we can collect data at each host along the connection chain, it is possible to determine which part of chain a host stays. In summary, we have validated that there are more crossovers generated in a long connection chain than in a short one. The number of crossovers is related positively to the length of the chain.

Chapter 4

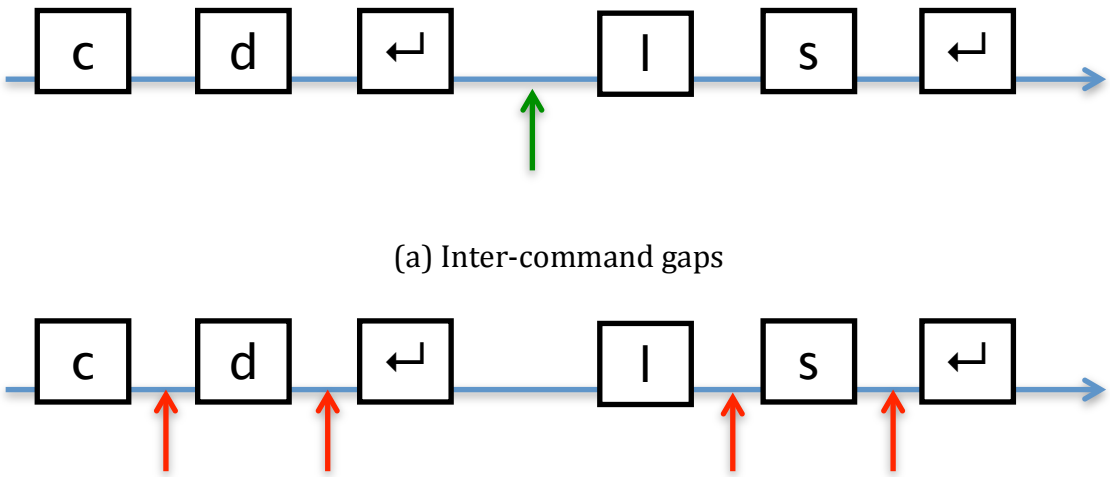
Detection of Long Connection Chains

The goal of validating the relationship between crossovers and connection chain is to help us to detect long connection chains. Although we have already concluded that a high number of crossovers implies a long connection chain, we cannot use that to identify long connection chains. The reason is due to the availability of the packet information. We do not have packet information along the chain except for the last host where the monitoring algorithm resides. However, the large number of crossover packets will alter the distribution of the packet gaps. This chapter proposes an algorithm to capture those gaps variances resulted by the large number of packet crossovers. With this algorithm, it is possible to identify long connection chains from shorter chains.

First of all, we would like to introduce the definition of "Upstream Round Trip Time". We take a response packet sent back and the next packet received as an estimated RTT, which is called Upstream Round Trip Time (uRTT). This is the time gap we will use later in our research, since we cannot get a real RTT due to lack of control over hosts except for our server machine.

Then we distinguish two types of gaps between packets: "Inter-command Gap" and "Intra-command Gap" used in Ding et al. [7]. Each command is usually followed by an "Enter" button. Inter-command Gap refers to the time gaps between a "return" character and the first character of the next Unix command by the user. Intra-command

Gap refers to the time gaps between two keystrokes within a single command, i. e., no return or end-of-line characters. Examples of several gaps of each type are shown in Figure 4.1 below.



(a) Inter-command gaps

(b) Intra-command gaps

Figure 4.1 Inter-command gaps and Intra-command gaps

The reason for separating inter- and intra-command gaps is to filter out some of the packet gaps that are not contributing to our detection algorithms. Figure 4.2(a) shows all the gaps between successive packets of a long connection chain (shown in blue) and that of a short one (shown in red). The gaps are sorted in increasing order in Figure 4.2. The two curves are difficult to separate from each other. Intra-command gaps essentially measure the typing speed of a user and do not depend on the length of the chain. On the other hand, inter-command gaps do depend on the chain length, because the user may have to see the result from the prior command to determine what to do next.

Thus we need an algorithm to separate inter-command gaps and intra-command gaps from each other. The strategy behind the separation method is that after a client enters a command, the amount of returned data exceeds the size of MTU (Maximum Transmission Unit) that Secure Shell protocol defines is typically fairly large. So the server will usually return more than one response packets. Even an empty "Enter" stroke may get two response packets returned. Therefore, at any moment there are more than one response packets consecutively returned, it indicates an inter-command gap. This method is probably not 100% accurate but our observation of the data suggest that it is fairly accurate. With that, we are able to separate inter-command gaps from intra-command gaps. Figure 4.2(b) and (c) shows the two sets of gaps sorted in increasing order of the gaps.

From previous experiments, we conclude that long connection chain will generate a great amount of packet crossovers. These crossovers lead to some relatively small inter-command gaps, which are even smaller than round-trip times. So if there exists some unusual small inter-command gaps, we will have a suspicion that it is possibly a long connection chain, which an intruder may use for attack. Figure 4.2 (a)-(c) indicate the differences of three gap types between long connection chains and short ones. As the figures show, we cannot separate long chains from short ones via first two charts. While in the third chart, the marked line of long chain has a huge jump at first several data points while the short chain's line increases smoothly.

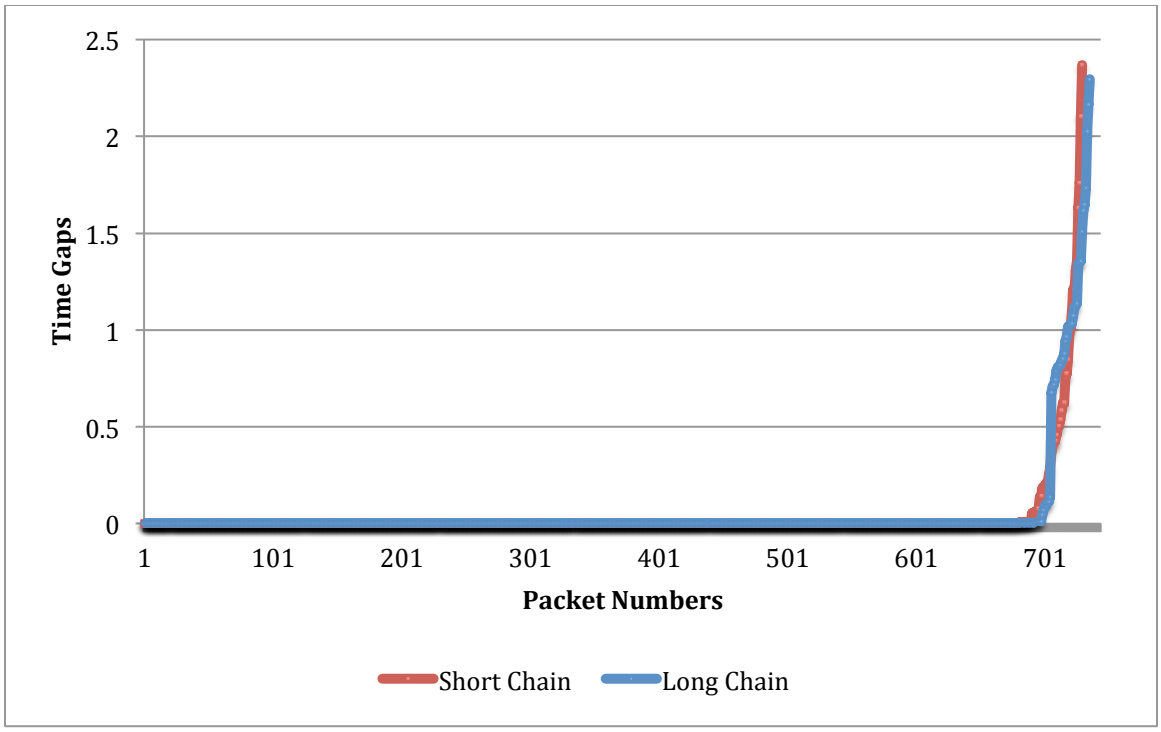


Figure 4.2(a) Mixed Gaps: Long chains vs. short chains

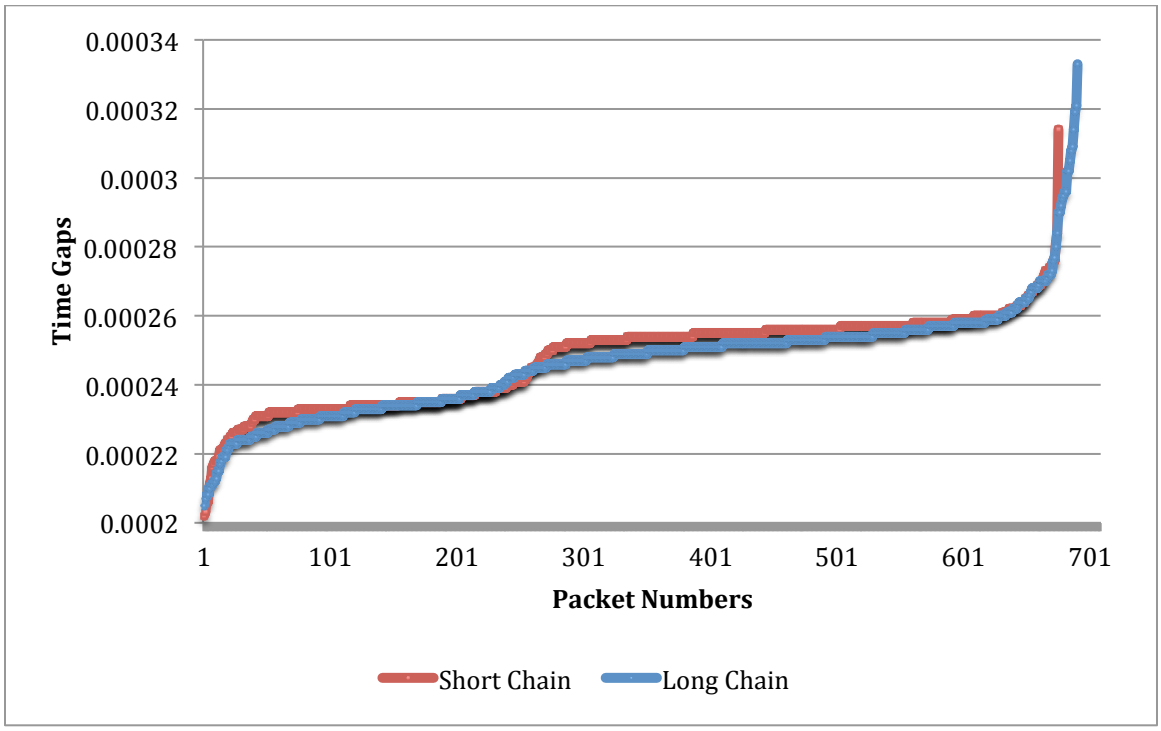


Figure 4.2(b) Intra-Gap: Long chains vs. short chains

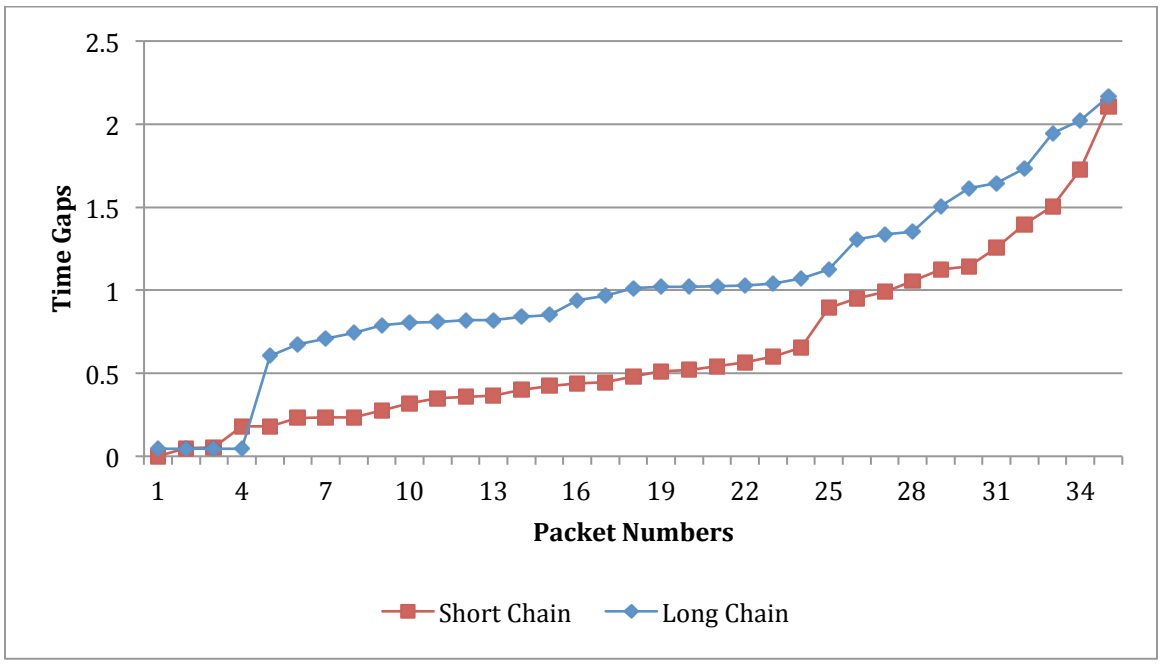


Figure 4.2(c) Inter-Gap: Long chains vs. short chains

Since the target server is the only host on the connection chain that we are able to monitor the packets, we need to analyze the packets captured at the server machine in the previous experiments. Here is how our algorithm works:

1) Extract request/response SSH packets from data collected at the targeted server host of the selected connection.
2) Compute uRTT gaps of successive packets and sort them in ascending order.
3) Filter out the intra-command gaps and keep only the inter-command gaps G[i] sorted in ascending order.
4) Compute the ratios of successive gaps over their previous ones, R[i] = G[i-1]/G[i].
5) Find the maximum gap ratio *mgr* = $\max\{R[i] | i = 2, \ldots, n\}$ and if this maximum

ratio *mgr* is greater than a predetermined threshold *t*, it is considered as a long connection chain. Otherwise, it is a short one..

Table 4.1 Three cases of ratios of consecutive gaps among twenty long connection chains collected in our experiments (chain length = 3 hops). Only the first 20 gaps are shown here in the table.

	Case 1		Case 2		Case 3	
	Gaps (s)	*Ratios*	*Gaps (s)*	*Ratios*	*Gaps (s)*	*Ratios*
1	0.0447		0.0198		0.0432	
2	0.0456	1.0217	0.0434	2.1899	0.0445	1.0296
3	0.0457	1.0013	0.0443	1.0212	0.0447	1.0032
4	0.0466	1.0207	0.0452	1.0196	0.0447	1.0012
5	0.5117	10.9721	0.0454	1.0050	0.0693	1.5497
6	0.5203	1.0168	0.5513	12.1381	0.0730	1.0539
7	0.5231	1.0054	0.5533	1.0038	0.1901	2.6035
8	0.5506	1.0525	0.6865	1.2406	0.3542	1.8630
9	0.5906	1.0727	0.6899	1.0050	0.4588	1.2954
10	0.6512	1.1027	0.7240	1.0494	0.4733	1.0314
11	0.6547	1.0052	0.7342	1.0141	0.5273	1.1141
12	0.6603	1.0087	0.7648	1.0417	0.5539	1.0504
13	0.6715	1.0169	0.7976	1.0428	0.6134	1.1075
14	0.6978	1.0392	0.8509	1.0669	0.6372	1.0388
15	0.6996	1.0025	0.8546	1.0044	0.6604	1.0364
16	0.7311	1.0451	0.8616	1.0082	0.6905	1.0456
17	0.7836	1.0718	0.9165	1.0637	0.7494	1.0853
18	0.8229	1.0501	0.9276	1.0121	0.7709	1.0286
19	0.8316	1.0106	0.9414	1.0149	0.7729	1.0026
20	0.9312	1.1198	0.9562	1.0157	0.7820	1.0118

The above algorithm did not specify what the threshold value to use. There is an obvious trade-off between the accuracy of our ability to detect long connection chains (lower t value) and the false positive rate. We will address that later.

Table 4.1 gives us three of the twenty long-connection cases tested. As one can

see, there is one ration that stands out among all of the ratios. Case Three has the lowest *mgr* value among all twenty test cases.

We did a similar analysis for the short connection chains as showed in Table 4.2. The ratios for the short chains are supposed to be low, but we do see some outlier values, as showed in Case 3, in the twenty cases of short connection chains. However, for most of the cases, the *mgr* for the short chains is smaller than those of the long chains. We can use this as a feature to separate the two cases.

Table 4.2 Three cases of ratios of consecutive gaps among twenty short connection chains collected in our experiments (chain length = 1 hop). Only the first 20 gaps are shown here in the table.

	Case 1		**Case 2**		**Case 3**	
	Gaps (s)	*Ratios*	*Gaps (s)*	*Ratios*	*Gaps (s)*	*Ratios*
1	0.0505		0.0517		0.0505	
2	0.0517	1.0222	0.0518	1.0004	0.0522	1.0346
3	0.0519	1.0038	0.0548	1.0593	0.4992	9.5575
4	0.1267	2.4431	0.1266	2.3085	0.5127	1.0271
5	0.1949	1.5381	0.2411	1.9046	0.5191	1.0124
6	0.1961	1.0062	0.2597	1.0771	0.5261	1.0135
7	0.2040	1.0406	0.2829	1.0897	0.5827	1.1076
8	0.2874	1.4089	0.3098	1.0950	0.5960	1.0229
9	0.3055	1.0629	0.3390	1.0940	0.5980	1.0032
10	0.3633	1.1891	0.3499	1.0324	0.6142	1.0271
11	0.3973	1.0935	0.3725	1.0643	0.6304	1.0264
12	0.4287	1.0791	0.3882	1.0424	0.6782	1.0759
13	0.4306	1.0044	0.4691	1.2084	0.7006	1.0330
14	0.4418	1.0260	0.5105	1.0881	0.7025	1.0027
15	0.4703	1.0646	0.5329	1.0439	0.7063	1.0054
16	0.5374	1.1426	0.5439	1.0206	0.7420	1.0506
17	0.5976	1.1121	0.5758	1.0586	0.7481	1.0082
18	0.5996	1.0033	0.5790	1.0055	1.0134	1.3546
19	0.6046	1.0083	0.5920	1.0225	1.0476	1.0338
20	0.6861	1.1348	0.6054	1.0227	1.2036	1.1489

In the rest of this chapter we will determine what is a reasonable threshold value and evaluate the effectiveness of our detection algorithm. We pick the highest *mgr* value from each experiment for both long and short chains. The result for all 40 cases (20 long and 20 short) is summarized in Table 4.3 below. We notice that there are some overlaps in value between short chains and long chains. However, the result is still enough for use to separate the two cases (showed in Table 4.3).

Table 4.3 Maximum gap ratios (*mgr*) of 20 long chains and 20 short chains

	Long	Short
1	13.8608	9.5575
2	7.9976	4.7240
3	10.4442	3.0517
4	6.2593	3.4897
5	10.2412	3.6067
6	13.0579	1.7436
7	4.9606	1.9759
8	6.6623	3.5048
9	10.7373	2.4431
10	8.7353	4.4025
11	8.3042	2.8133
12	12.1381	2.3085
13	6.6999	3.3191
14	4.4003	2.2266
15	7.1686	2.0160
16	7.9714	2.9693
17	2.6035	2.2142
18	8.3235	2.9597
19	8.6655	4.2507
20	10.9721	5.4760

Take the forty numbers from Table 4.3 and mix them together and sort them in ascending order. As showed in Figure 4.3, most short chain (blue circles) and long chain (red squares) data points stay in lower part and upper part of the chart respectively. Figure 4.3 may be used to evaluate the accuracy of our algorithm. For example, if we select the threshold t to be 6.0, then there is one short chain misclassified (the only blue circle above 6.0), a false positive. At the same time, there are three long connection chains that we will be unable to detect, three false negative cases.

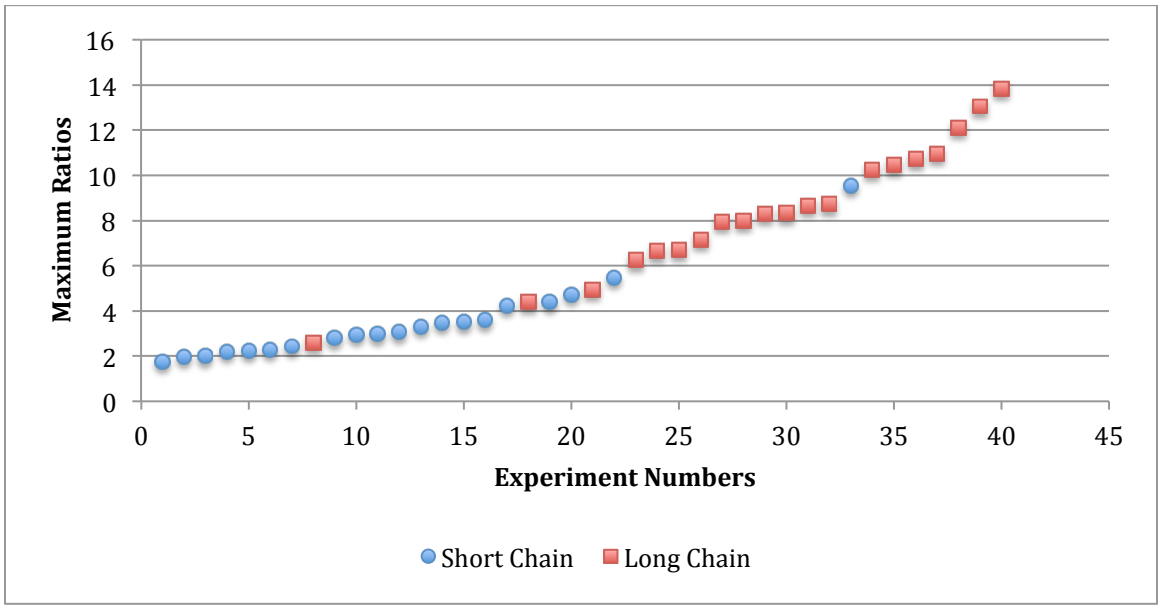


Figure 4.3 Maximum ratios of 20 short chains (blue circles) and 20 long chains (red squares)

In order to better examine the detection rate of this method, we set the ratio threshold at different levels to distinguish long connection chains from short ones. For example, if we set ratio threshold as 6, there is only one data point falling in false positive, which means a false positive ratio of 5%. Meanwhile, 17 of 20 long connection chains

are correctly detected with a true positive ratio of 85%. In this way, we generate a ROC (Receiver Operating Characteristic) curve with different pairs of false/true positive ratios. ROC curve gives a simple visual evaluation of the accuracy and false alarm rate of our method in long connection chain detection. Depending on the amount of false positive one is willing to tolerate, one can estimate the accuracy of the detection algorithm. ROC curve is presented in Figure 4.4, which turns out to be an excellent result comparing with previous research performances.

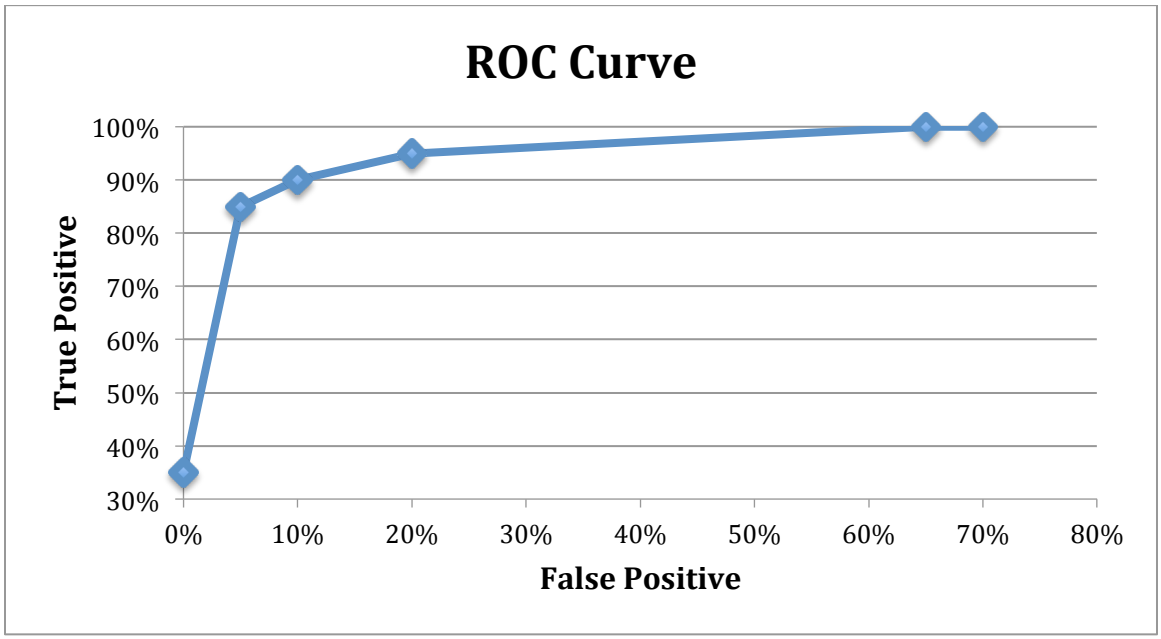


Figure 4.4 ROC Curve showing the accuracy of the detection algorithm vs. false positive rate.

Chapter 5

Conclusion

Cyber attack through stepping-stones has been widely used by hackers to perform an intrusion to servers anonymously. It can effectively prevent from being traced back to the source. In this research, we first establish that the number of packet crossovers is proportional to the length of the connection chain. This was done by collecting packets along the whole connection. A "flip over" in two successive nodes indicated a packet crossover. Our experiments validated our hypothesis.

Unfortunately we cannot use the result directly to help us identify the long connection chains. To protect a server from being attacked via a long connection chain, we can only monitor packets entering and leaving the server (at the end of the chain). We designed an algorithm to do exactly that in Chapter 4.

Our algorithm is based on the prior work done by Ding et al. [7], which investigated the upstream RRT of the target server. They also separated the intra-command gaps from the inter-command gaps so that they can focus on the difference of long and short chains. Their result shows modest success in detecting long connection chains of lengths 4 and 6.

We propose a new approach to detect long connection chain based on the hypothesis that we already validated. We follow the approach of Ding and observe that there is a obvious sharp increase in uRTT for the long connection chain. This is a direct

result of the crossover packets. For the short chains, the (sorted) uRTTs are distributed fairly smoothly (cf. Figure 4.2c). For the long chains, some of the gaps were significantly reduced because of the packet crossovers. These artificially created RTTs dragged the uRRTs down significantly, thus opening up a gap between the crossover uRTTs and the non-crossover uRTTs (see the fourth and the fifth value of the long chain). Since we don't know the number of crossovers, we look for the sharp jump in the uRTTs caused by the crossover instead. Thus our approach is to find the maximum inter-command gaps ratio from collected uRTT data, which achieves a true positive of 85% with false positive as low as 5%. Our approach is able to detect connection chain of three hops with a better accuracy than the previous research.

Despite that our experiment results are already acceptable, we still have several improvements to make. Firstly, we only conducted the experiment 40 times in total for both short and long chains. And, each experiment consists around 35 commands input, which could be increased in order to collect more data. Secondly, 5-hop connection chain or even longer is not conducted due to shortage of experimental machines nationwide. We believe our approach can achieve an even better detection rate for longer chain detection.

Our research will help servers to stay safe by detecting long connection chains, which imply the existence of intruders. Also it only requires the data at the end of chains, so it is easy to implement. Last but not least, the false positive is acceptable along with high detection rate. More research is needed on intrusion detection in the future to guarantee a safer network.

Bibliography

[1] LinedIn Corp. Linedin Passwords Compromise. URL: *http://blog.linkedin.com/20-12/06/06/linkedin-member-passwords-compromised/*, accessed April 8, 2014.

[2] Sony Corp. Sony PlayStation Network/Qriocity Services Outage. URL: *http://bl-og.us.playstation.com/2011/04/22/update-on-playstation-network-qriocity-services/*, accessed April 8, 2014.

[3] Wikimedia Foundation. Sony PlayStation Network Outage. URL: *http://en.wiki-pedia.org/wi-ki/PlayStation_Network_outage/*, accessed April 8, 2014.

[4] Y. Zhang and V. Paxson. Detecting stepping stones. In *Proceedings of the 9th Conference on USENIX Security Symposium*, volume 9, pages 171-184, 2000.

[5] S. Staniford-Chen and L. Heberlein. Holding intruders accountable on the Internet. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 39-49, May 1995.

[6] J. Yang and 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*, pages 198-203, 2004.

[7] W. Ding, M.J. Hausknecht, S. Huang and Z. Riggle. Detecting stepping-stone intruders with long connection chains. In *Proceedings of the 5th International Conference on Assurance and Security*, volume 2, pages 665-669, 2009.

[8] Peter G. Neumann and Phillip A. Porras. Experience with emerald to date. In *1st USENIX Workshop on Intrusion Detection and Network Monitoring*, pages 73-80, April 1999.

[9] Stefan Axelsson. Intrusion detection systems: A survey and taxonomy. In *Technical Report 99-15, Department of Computer Engineering*, *Chalmers University*, March 2000.

[10] Richard P. Lippmann. Evaluating intrusion detection systems: The 1998 DARPA off-line intrusion detection evaluation. In *Proceedings of DARPA Information Survivability Conference and Exposition*, pages 12-26, 2000.

[11] X. Wang and D. Reeves. Sleepy watermark tracing: An active network-based intrusion response framework. In *Proceedings of the 16th International Information Security Conference*, pages 369-384, 2001.

[12] K. H. Yung. Detecting long connection chains of interactive terminal sessions. In *Recent Advances in Intrusion Detection, Lecture Notes in Computer Science*, pages

1-16, 2002.

[13] J. Yang and S. Huang. Matching TCP packets and its application to the detection of long connection chains on the Internet. In *19th International Conference on Advanced Information Networking and Applications*, volume 1, pages 1005-1010, 2005.

[14] J. Yang and S. Huang. A clustering-partitioning algorithm to find TCP packet round-trip time for intrusion detection. In *20th International Conference on Advanced Information Networking and Applications*, volume 1, pages 231-236, 2006.

[15] Hewlett-Packard. How the SSH Client and Server Communicate. URL: *http://h71000.www7.hp.com/doc/83final/ba548_90007/ch01s04.html*, accessed April 8, 2014.

[16] T. Ylonen and C. Lonvick, The Secure Shell (SSH) Connection Protocol, IETF RFC 4254, January 2006; *http://www.ietf.org/rfc/rfc4254.txt*.

[17] Wikimedia Foundation. OSI model. URL: *http://en.wikipedia.org/wiki/OSI_model*, accessed April 8, 2014.

Appendix

1. Computing the number of crossovers from collected data stored in spreadsheet.

```
import sys
import xlrd
import string

data = xlrd.open_workbook(sys.argv[1])

for i in range(1,21):
    table = data.sheet_by_name("Sheet"+str(i))
    nrows = table.nrows
    count = 0
    crossover = 0
    false = 0
    for r in range(nrows):
        if table.cell_value(r,23) == 'TRUE':
            crossover += count%2
            count = 0
            continue
        if table.cell_value(r,23) == 'FALSE':
            count += 1
            false +=1

    print "Row # of Sheet" + str(i) + ": " + str(nrows)
    print "Crossover # of Sheet" + str(i) + ": " +
          str(false/2 + crossover)
```

2. Computing Inter-command and Intra-command gaps.

```
import re
import sys
import string

count = 0

f1 = open(sys.argv[1],"r")
f2 = open(sys.argv[1].rstrip('.txt')+'-IntraGap.txt', 'w')
f3 = open(sys.argv[1].rstrip('.txt')+'-InterGap.txt', 'w')

for line in f1:
    Packet = line.split()
#    print Packet[0], Packet[7]
    if Packet[7] == 'request' and count == 0:
        t1 = string.atof(Packet[0])
    elif Packet[7] == 'request' and count == 1:
        IntraGap = str(t2 - t1)
        t1 = string.atof(Packet[0])
        count = 0
        f2.write(IntraGap+'\n')
        #        print 'IntraGap = '
        #        print IntraGap
    elif Packet[7] == 'request' and count > 1:
        t2 = string.atof(Packet[0])
        InterGap = str(t2 - t1)
        count = 0
        t1 = string.atof(Packet[0])
        f3.write(InterGap+'\n')
        #        print 'InterGap = '
        #        print InterGap
    elif Packet[7] == 'response':
        t2 = string.atof(Packet[0])
        count+=1

f1.close()
f2.close()
f3.close()
```