DETECTING AND PREVENTING STEPPING-STONE INTRUSION BY MONITORING NETWORK PACKETS

A Dissertation

Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

By

Jianhua Yang

August 2006

DETECTING AND PREVENTING STEPPING-STONE INTRUSION BY MONITORING NETWORK PACKETS

Jianhua Yang

APPROVED:

Dr. Stephen S. H. Huang, Chairman

Dr. Christoph F. Eick

Dr. Robert Y. Ling
Flight Software Branch, NASA JSC

Dr. Jehan-François Pâris

Dr. Rong Zheng

Dean, College of Natural Sciences and Mathematics

ACKNOWLEDGEMENTS

The work on this dissertation has been extensive and trying, but most of all exciting, instructive, and fun. Without the help, support, and encouragement from several persons, I would never have been able to finish this work.

First of all, I would like to give my sincere thanks to my honored supervisor, Dr. Shou-Hsuan Stephen Huang, for his support, patience, and encouragement throughout my graduate studies. It is not often that one finds an advisor and colleague that always finds the time for listening to the little problems and roadblocks that unavoidably crop up in the course of performing research. His technical and editorial advice was essential to the completion of this dissertation and he has given me innumerable lessons and insights into the workings of academic research in general. I have learned a lot from him: how to be a researcher, a professor, an advisor, and most of all, how to be a straight person. Without his help and guidance, I could not have finished this dissertation successfully.

My thanks also go to the members of my graduate committee: Dr. Christoph F. Eick, Dr. Robert Ling, Dr. Jehan-François Pâris, and Dr. Rong Zheng. I appreciate their time and efforts. I would also like to thank the rest of the faculty and staff of the department. In particular, I extend my gratitude to these colleagues and friends: Mr. Ming D. Wan, Mr. Bingshan Li, Dr. Jesús Ubaldo Quevedo, and Mr. Chandler Wilkerson.

I am very grateful to my mother, and to my mother-in-law. Their understanding and love encouraged me to work hard and to continue pursuing a Ph.D. abroad. Their firm and kind-hearted personalities have shown me how to be steadfast and never bend to

difficulties. They always let me know that they are proud of me. My mother-in-law's hard time due to cancer encouraged me to work harder and do my best.

I am extremely grateful to my wife, Peng Zhao, and my son, Songpei Yang, for their understanding and love during the past few years. My wife's support and encouragement was in the end what made this dissertation possible. They form the backbone and source of my happiness. For these reasons, and for their loving kindness, this dissertation is dedicated to them.

DETECTING AND PREVENTING STEPPING-STONE INTRUSION BY MONITORING NETWORK PACKETS

An Abstract of a Dissertation

Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

By

Jianhua Yang

August 2006

ABSTRACT

Most computer intruders chain several previously compromised computers so as to hide themselves before launching attacks on a target computer. One way to stop such intruders is to detect the intrusions and prevent them from using compromised computers; another way is to trace the intrusions back. The former is called stepping-stone detection, and the latter prevention or connection traceback.

We propose here three approaches to detect stepping-stone intrusion: Request-Response, Step-Function, Network Fluctuation; we also propose two approaches to trace intrusions back: Temporal Thumbprint and Round-trip Time Thumbprint. The experimental results and theoretical analysis show that these can perform better than existing stepping-stone intrusion detection and prevention approaches in terms of false positive rate, false negative rate, and the resistibility to intruder evasions, such as time jittering and chaff perturbation.

Matching TCP/IP packets is critical to both stepping-stone intrusion detection and connection traceback. We formally model and study this problem and propose three approaches to match the TCP/IP packets of an interactive session: TCP/IP protocol-based matching, clustering-partitioning matching, and standard deviation-based matching.

The information embedded in the network traffic can be used to detect and prevent stepping-stone intrusion. We verify this statement through the technologies to detect stepping-stone, and to trace intrusions back. By monitoring, matching, mining, and statistical analyzing the TCP/IP packets of an interactive session, this dissertation offers a

view of computer security breaches not from their origin or intended effect, but from their manifestation in the network traffic trace.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

In this chapter we first present the basic concepts of security for computers, networks and information, and show the need for securing computer, network and information systems. Second, we discuss the role of intrusion detection and connection traceback. Some definitions and notations used in this dissertation are given in the third part of this chapter. Finally, the statement and the outline of the whole dissertation are described.

1.1 Security for Computers, Networks and Information

Computers are everywhere; they impact almost every aspect of modern life to one degree or another. People store their information on computers and access it through networks. The act of placing this information in computerized systems is an act of trust [Sol05]. The information is protected from unauthorized access or modification and is available to a legitimate user. The question is how to make information secure when it is stored and processed in a computerized system, and how to make information secure when it is propagated through a network or the Internet. Once this information is illegally accessed, modified, or intercepted, can this event be detected and prevented? Before discussing these issues, we first present the basic concepts of security for computers, networks and information.

Security is a continuous process of protecting an object from attack [Kiz05]. Here "object" has a broad meaning. For a computer system, it means physical hardware components, such as printers, CPU, monitors, memories, and so on; it also means soft

resources, such as data and information that need to be protected. For a network or distributed computer system, it means all the hard and soft resources that make up the network, including communication channels, connectors, and the files stored on the servers of the network. Security involves three goals: confidentiality, integrity, and availability [Sol05, Mai03, Gol06, Bis02].

Confidentiality is defined as preventing unauthorized disclosure of information to third parties. It includes the disclosure of information about resources. Integrity is defined as preventing unauthorized modification of resources. Availability is the prevention of unauthorized withholding of system resources from those who need them when they need them. Malicious individuals have corresponding mechanisms to defeat these three goals of security: disclosure, alteration, and denial. Disclosure occurs when unauthorized individuals gain access to confidential information. Some examples are when a hacker gains access to a system and reads confidential information; when an insider disseminates confidential information to unauthorized third parties; and when a programming failure causes customers to access the critical information of a web site. Alteration occurs when security mechanisms fail to ensure the integrity of computer resources. Denial occurs when events take place and prevent authorized users from accessing a system for legitimate reasons. DoS (Denial of Service) attack is one typical example of this mechanism. The methods proposed in this dissertation are mainly focused on dealing with disclosure and alteration.

It is very difficult to exactly define security for computers, networks, and information. From different viewpoints, we have different definitions. In this dissertation, we use the

definition adopted by Gollmann [Gol06]: "Computer security deals with the prevention and detection of unauthorized actions by users of a computer system; a computer is secure if one can depend on it and its software to behave as expected." Different computer security levels have been defined by the U. S. Department of Defense in the Standard 5200.28, the Trusted Computing System Evaluation Criteria (TCSEC), also known as the Orange Book [TCS85]. The Orange Book defines computer systems in terms of security according to the following scale: 'D' means minimal protection or unrated; 'C1' means discretionary security protection; 'C2' means controlled access protection; 'B1' means labeled security protection; 'B2' means structured protection; 'B3' means security domains; 'A1' means verified design. Most current computer systems have the security level up to C2 (included); very few of them were certified to the A1 level.

In a network, more interactions are possible, including more unwelcome interactions and malicious actions. We may wish to control which users on a network can access the system, as well as which users on a system can access the network, and thus protect the data which travels through the network. Therefore, in addition to access control and intrusion detection, network security also includes cryptography, though we do not discuss this here. Generally, network security can be defined as the process through which a network is secured against internal and external threats of various forms.

Computer security is necessary to control access on a computer system, and network security is needed to control the security of local and wide area networks. All of these provide information security [Mal03]. To judge if computers, networks, and information

(CNI) are at risk, we refer to the assets and vulnerabilities CNI has, and the threats CNI faces. Assets include hardware, software, data and information. Vulnerabilities are weaknesses of a system that could be accidentally or intentionally exploited to damage assets. Some typical vulnerabilities of CNI system are: 1) accounts with system privileges; 2) programs with unnecessary privileges; 3) programs with known flaws; 4) weak access control settings on resources; and 4) weak firewall configurations. Threats are actions by adversaries who try to exploit vulnerabilities to damage assets. They can be classified into four categories by the damage done to assets: access, modification, denial of service (DoS), and repudiation [Mai03].

An access attack is an attempt to gain information that the attacker is not authorized to see. This attack can occur wherever the information resides or may exist during transmission. It is against the confidentiality of the information. Some typical access attacks include snooping, eavesdropping, and interception. A modification attack is an attempt to modify information that an attacker is not authorized to modify. This attack can occur wherever the information resides. It is against the integrity of information. Such an attack typically would include changes, insertions, and deletions. DoS attacks are attacks that deny the use of resources to legitimate users of the system, information, or capabilities. DoS attacks generally do not allow the attackers to access or modify information on the computer system. Typical DoS attacks include 1) denial of access (DoA) to information; 2) DoA to applications; 3) DoA to systems; and 4) DoA to communications. Repudiation is an attack against the accountability of the information. It is an attempt to give false information or to deny that a real event or transaction has occurred. Typical repudiation attacks are masquerading, and denying an event.

There are numerous approaches that have been developed to deal with these threats. Our work focuses on detecting and tracing malicious activities, such as access and modification attacks through interactive connections. We propose several methods to detect such attacks. We also develop two thumbprint methods to prevent access and modification attacks by tracing intruders back.

1.2 Stepping-Stone Intrusion Detection

Intrusion is a violation of the security policy of the system. An intrusion is defined as a set of actions that attempts to compromise the integrity, confidentiality, or availability of computer or network resources [Hea90]. Intrusion detection is the art of detecting and responding to computer misuse [Pro01]. It is the process of detecting attempts to gain unauthorized access to a network or to create network degradation.

The need for intrusion detection is based on the increasing number of threats faced by computer networks in today's world. Research has shown that various attacks through the Internet have increased greatly since 2000. Malik [Mal03] lists some fundamental reasons why so many attacks are being launched on computer networks: 1) computer networks are used to carry information that is valuable to the intended receiver; 2) the World Wide Web has become a very common way of delivering information to the masses; 3) tools to launch network attacks are readily available; 4) Internet-based attacks give relative anonymity to attackers; 5) access to networks is easy and widespread; and 6) the amount of traffic often carried by networks today makes it difficult to use traditional intrusion detection techniques to provide a meaningful deterrent in many cases.

Intrusion detection can be divided into host-based and network-based detection [Mar 01]. Host-based technologies examine events like what files were accessed and what applications were executed. Network-based technologies examine events as packets of information exchanged between computers. Host-Based Intrusion detection technologies can be categorized into misuse detection, and anomaly detection [Kru05]. Misuse detection is to detect activity that matches explicit patterns of misuse. It can detect intrusion efficiently and accurately, but it cannot detect new or updated attacks. Anomaly detection is to compute deviations from acceptable behavior profiles to determine an intrusion; it can detect new attacks. The stepping-stone intrusion detection and connection traceback technologies can also be categorized into active and passive approaches. All the methods we propose to detect and traceback intrusion are network-based, passive approaches.

A significant part of our work is concentrated on stepping-stone detection and stepping-stone intrusion detection. We assume an intruder accesses a host through a long interactive TCP session. The basic way to detect if a host has been used as a stepping-stone is to find a stepping-stone pair by comparing the incoming connections with the outgoing connections of the host. A way to detect stepping-stone intrusion is to estimate the downstream length from the sensor (defined in Section 1.4) to the victim site and to see if that is a long connection chain. We make use of the idea that if the downstream connection chain is more than n connections long (such as $n >= 3$), there is a high probability that there is an intrusion. Of course, that host is also used as a stepping-stone. But we cannot draw a conclusion that there is an intrusion simply because a host has been used as a stepping-stone. The bad news is that occasionally even if the downstream

length is not more than three connections, it is still possible that the chain is being used by an intruder. The reason that we use '3' as a key number is that legitimate users or applications still use stepping-stones; based on our observation, most of them seldom use more than three compromised hosts. If we can combine both downstream and upstream connection length, our detection rate will significantly improves. In the following, we give a brief introduction to the approaches we propose to detect stepping-stone and intrusion.

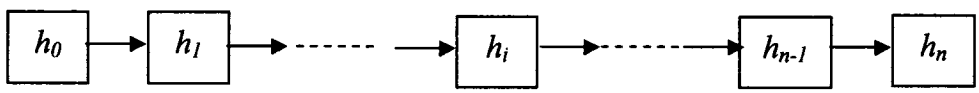


Figure 1.1 An interactive connection chain

1.2.1 Step-Function Method for Intrusion Detection

If we monitor an outgoing connection chain at a sensor from the beginning to the end, we are able to collect all the Send and Echo packets of the connection. As Figure 1.1 shows, we assume the sensor is host h_i, and monitor the connection chain at h_i. At the time when an intruder connects to h_{i+1} from h_i, which means that the connection chain has only one connection in the downstream; we monitor all the Sends and Echoes, and compute the gap between each send and its matched echo packet; these gaps are the round-trip times (RTT) of the Sends captured while the downstream chain has only one connection. Because of network traffic fluctuation problem, we cannot expect the RTTs to be the same. However, they are related to the same connection chain, so they vary slightly. These RTTs have a clear lower bound called one level. We monitor this chain

continuously until it connects to the victim site h_n and capture all the Sends and Echoes. Each time one more connection is extended, we obtain one more RTT level by matching the packets and compute the RTTs. Therefore, estimating the downstream length reduces to counting the number of RTT levels. If we put all the RTTs in increasing order of the Send timestamp, the levels look like steps. This is the reason this detection approach is called the Step-Function method [Yan04].

With this method, we can estimate the length of a connection chain efficiently and accurately, thus detect stepping-stone intrusion. One problem in the method above is how to match the Sends and the Echoes. In this dissertation, we first propose two methods to match TCP packets: the Conservative and the Greedy algorithms [Yan05]. Then we propose the clustering-partitioning algorithm [Yan06] which has a better performance in matching TCP packets than the previous two. Another problem of the Step-Function method is that we need to monitor a connection chain all the time. Once we miss some packets, it is possible to miss one or more RTT levels, and this would result in the inaccurate estimation of the downstream length. For such a case, we propose the Network Fluctuation-based detection method.

1.2.2 Fluctuation-based Method for Intrusion Detection

Through observation of the network traffic in terms of RTTs, we found that the higher the levels in the RTTs, the larger the difference among the RTTs within one level. The difference in the RTTs can be used to represent network fluctuation. That is to say, the longer a connection, the larger the network fluctuation. If we can measure a network fluctuation, we can use it to estimate the relative length of a connection chain [Yan05-1].

We assume that h_i is our sensor, and a connection chain from h_i to h_n has been set up (see Figure 1.1). We monitor this chain and capture Sends, Echoes, and Acks. We match the Sends and the Echoes and get the RTTs, which are denoted echo-RTTs. Similarly, we match the Sends and Acks and get RTTs, which are denoted ack-RTTs. We compute the standard deviations of ack-RTTs and echo-RTTs, respectively, and compute the ratio between them. Even though we cannot exactly know the number of connections the chain has, at least we are able to estimate relatively how long the chain is. The closer to zero the ratio is, the higher the probability an intrusion has occurred.

1.2.3 Stepping-Stone Pair Detection Method

The difference Δ between the number of Echoes on an incoming connection and the number of Sends on an outgoing connection can be modeled as a random walk process. We monitor an outgoing connection and record the number of send packets in a period of time. However, for an upstream connection, we filter out the larger size packets and only record the number of echo packets each of which corresponds to a single keystroke packets in the same period of time. If the two connections are relayed and not manipulated, the difference Δ should be around zero practically. We assume a packet captured might be a Send with probability q, and an Echo with probability p, where most probably $p = q$. When one more packet is captured, the difference walks a step toward negative direction with probability q and positive direction with probability p.

If the connection chain is manipulated, such as time and chaff perturbation, the probability that a packet might be a Send with q, and an Echo is with p, where we are

not sure which one is bigger. If more packets were introduced into the request stream of an outgoing connection, q would be bigger than p, and the difference Δ would walk toward more negative direction; otherwise, it would walk more positive direction. If we assume that an intruders' ability in manipulation is bounded, the difference Δ should be bounded for some ranges with a certain time interval when the two connections are relayed. We assume the range is $[-\Omega_l, \Omega_u]$, where $-\Omega_l$ is the lower boundary, and Ω_u is the upper boundary. Anyway, if the two connections are relayed, there exist $-\Omega_l$ and Ω_u such that the difference is inside the range $[-\Omega_l, \Omega_u]$ with high probability. We have compared the performance of this method with the best method [Blu04] so far, and found our method to be better.

However, knowing there is an intrusion is insufficient. We need to trace intrusions back to the source and deal with intruders in an effective manner. Because many attacks use spoofed IP addresses or are sourced from compromised devices, dealing with attacks is a nontrivial issue. In the next section, we focus on discussing the methods we propose to trace intruders.

1.3 Connection Traceback

Intrusion prevention has a broad meaning. It is defined as a class of security mechanisms that contains ways of preventing or defending against certain attacks before they can actually reach and affect the target [Kru05]. Tracing intruders back is one of the intrusion prevention ways to stop the attacks. By comparing the thumbprints of an incoming connection and an outgoing connection, we can use thumbprint to trace back

the intruder along a connection chain step by step. We propose two kinds of thumbprint to trace back intruders: one is called temporal thumbprint (T-thumbprint) and the other is called RTT-thumbprint.

1.3.1 T-Thumbprint

The idea behind T-thumbprint [Yan05-3] is very simple: use the feature that each person's keystrokes have their own distributions that can be represented by the time interval between two consecutive keystrokes. Even though the occurrence of keystroke intervals follows a Passion distribution, a specific person's keystroke intervals are discernible and this feature does not change much with propagation on a connection chain. It can be used to identify a connection chain, trace intruders and has the advantages of efficiency, secrecy, and immediateness. The experimental results showed us a surprising accuracy in identifying a connection chain.

One problem of using T-thumbprint to trace intrusions back is correlating two T-thumbprints between incoming and outgoing connections in a host. We propose an algorithm to find the largest similarity between two T-thumbprints by using dynamic programming [Wan06]. However, the T-thumbprint is vulnerable to intruders' manipulation. To resist intruders' evasion, we propose RTT-thumbprint [Yan05-04] to trace back intrusions.

1.3.2 RTT-Thumbprint

The original design of the TCP/IP protocol makes it difficult to reliably trace back to original intruders if they obscure their identities by logging through a chain of multiple

hosts. However, once a connection chain has been established between an intruder's host and the victim's host, it is shown that every packet sent from the intruder's host is going to be decrypted and then encrypted in each host in between and forwarded to the victim's site at the end of the chain; a responding packet is echoed from the victim and propagated to the intruder's side in a similar way but in a reverse direction. If we monitor an outgoing connection of each host, we can observe the send packet from this host and one echo packet from the adjacent host downstream. This fact prompts us to consider these pairs (each pair includes a send and its matching echo packet) as a unique characteristic which identifies a connection. Thus, we can use a sequence of packet RTT timestamp pairs to characterize an encrypted connection. This motivated us to propose RTT-thumbprint.

Instead of using timestamps of sending packets or contents in a connection, the RTT-thumbprint is a sequence of timestamp pairs of a send packet and its corresponding echoed packets. Each pair of timestamps represents a RTT of a send packet. Besides the advantages of efficiency, secrecy, and robustness, RTT-thumbprint has the ability of resisting intruder's time and chaff perturbation to a certain degree. An exhaustive and a heuristic algorithm are proposed to correlate RTT-thumbprints.

1.4 Definitions and Notations

Network intruders tend to comprise more computers (hosts, or networks), which are called stepping-stones [Zha00], to launch their attacks in order to reduce the risk of being discovered. As Figure 1.1 showed, we assume that an intruder launches his attack from

host h_0 to host h_n through hosts h_1, h_2, …, h_{n-1} by using OpenSSH [Ylo04-1, Ylo04-2], where h_0 is the intruder's site, and h_n is the victim's site.

The flow $h_i \rightarrow h_{i+1}$ is called a connection, denoted C_i. The sequence of connections $<C_0, C_1, \ldots, C_i, \ldots, C_{n-1}>$ is called an interactive TCP connection chain, or a chain, or a session. A connection to a host is called an incoming connection of the host, while a connection out of a host is called an outgoing connection of the host, i.e., connection C_i is an incoming connection of the host h_{i+1}, as well as the outgoing connection of the host h_i. A host where our detecting program resides is called a sensor. The direction toward the victim site from the sensor is called downstream; the connection chain from the sensor to the victim site is called the downstream connection chain of the sensor; the length in terms of the number of connections is called downstream length. Similarly, the direction toward the intruder's site from the sensor is called upstream, and the connection chain from the intruder's site to the sensor is called an upstream connection chain of the sensor; the length is called upstream length.

For each TCP connection, we are interested in request, response, and acknowledgement packets. They are defined as Send, Echo and Ack, respectively, as the following:

Send: A TCP packet is defined as a Send if it propagates downstream and has either flags 'Push (P)' and 'Acknowledgement (A)' or only flag 'P' [USC81];

Echo: A TCP packet is defined as an Echo if it propagates upstream and has either flags 'Push (P)' and 'Acknowledgement (A)' or flag only 'P';

Ack: An Ack is defined as a TCP packet moving in either upstream or downstream and has 'Acknowledgement (A)' flag only.

A packet stream is defined as the sequence of successive Sends, Echoes and Acks. When we capture a packet, we use the local host clock as the timestamp of the packet. We use the symbol P to represent a packet stream, $P = \{p_1, p_2, \ldots, p_n\}$, where p_i represents the ith packet, as well as the timestamp of the ith packet. When a request is sent to a host, eventually there will be one or more response packets generated. We call the packet representing the request and the first response packet of the Echoes a matched pair. If more Sends are responded to by one Echo, the matched pair is defined as the last Send and the Echo. The action to find matched pair in a packet stream is called packet-matching. The method to find the matched pair in a packet stream is called the packet-matching algorithm.

Each connection has two streams, request and response. The request stream is composed of Sends, and the response stream is composed of Echoes. As Figure 1.2 shows, we use C_i^k to represent i^{th} incoming or outgoing connection of the sensor h_i, where k=1 represents an incoming connection, and k=2 represents an outgoing connection. We use the symbol $S_i^{(k)}$ to denote the request stream of the connection C_i^k, while $E_i^{(k)}$ denotes the response stream.

As Figure 1.2 shows, to determine if the host h_i is used as a stepping-stone is called stepping-stone detection. The main idea to detect a stepping-stone is to compare the incoming connections with the outgoing connections to see if there is one pair (C_i^1, C_i^2) in which two connections are relayed; if yes, this pair is called a stepping-stone pair;

otherwise, it is called a normal pair. So the statement of detecting stepping-stone reduces to finding a stepping-stone pair.

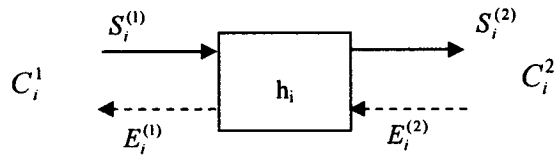


Figure 1.2 Illustration of connections and streams of a host

As Figure 1.1 showed, instead of comparing incoming connections with outgoing connections to detect stepping-stones, we determine if a chain is used by an intruder through estimating the downstream length in terms of number of connections. Usually, if the downstream length is more than n connections (we selected n=3 in this dissertation), it is highly suspicious that the chain is being used by an intruder. To determine if a chain is being used by an intruder is called stepping-stone intrusion detection. We must be aware that being used as a stepping-stone does not necessarily mean an intrusion has occurred. So the methods used to determine stepping-stones have high false positives in detecting intrusion. Stepping-stone prevention is the technology to prevent stepping-stone intrusion. In this dissertation, we mainly focus on tracing back intruders to prevent their attacks.

1.5 Statement and Outline

This dissertation provides an answer to the following question: Can we successfully and effectively detect and trace stepping-stone intrusions back by monitoring TCP/IP packets? A more complete statement of this dissertation is:

It is possible to detect and prevent stepping-stone intrusions by monitoring network traffic.

This dissertation focuses on checking and modeling network traffic to detect and prevent stepping-stone intrusion. It offers a view of computer security breaches not from their origin or intended effect, but from their manifestation in the network traffic trace. By being aware of the nature and statistics of TCP/IP packets flowing at different stages in network traffic, we are able to optimize and model their occurrence to determine or trace intrusions back. Even if under an intruders' manipulation, it is still possible to detect and prevent intruders' evasion by checking and modeling network traffic.

This dissertation is arranged as follows. In Chapter 2, we review and summarize the literature related to stepping-stone intrusion and connection traceback. In Chapter 3, we discuss a novel method to detect stepping-stones. In Chapter 4, we discuss the Step-Function method to detect intrusion, as well as the Network Fluctuation-based method. In Chapter 5, we discuss three methods to match TCP/IP packets: the TCP/IP protocol-based packet-matching approach, the clustering-partitioning packet-matching approach, and the standard deviation-based packet-matching approach. In Chapter 6, we discuss two approaches to trace back intrusions: temporal thumbprint and RTT-thumbprint. Finally, in Chapter 7, we summarize the whole dissertation and present future work.

1.6 Summary

Network security attacks occur despite the protection offered. They can be internal or external and can involve data theft. Network intrusion detection provides a suitable mechanism for detecting and preventing these attacks. It usually forms the last line of

defense against security threats. Detection and prevention are indispensable parts of effective security. This dissertation proposes a novel scheme to detect and trace back stepping-stone intrusions by monitoring and modeling network traffic and applying packet-matching techniques.

2. RELATED WORK

This chapter summarizes other researchers' work related to stepping-stone intrusion detection and connection traceback. The two issues are related in a sense that once systems intrusion has been detected, connection traceback technologies would be used actively or passively to prevent the intrusion attempts. In this dissertation, when we talk about intrusion prevention, we always mean intrusion traceback or connection traceback. The essential problem of intrusion traceback is a stepping-stone detection problem. We do not distinguish whether a method is passive or active, but what we distinguish is if a method can be used to trace intrusion back or detect stepping-stones. Even so, most stepping-stone detection methods can be used to trace back intrusion, and vice versa. There are lots of methods which have been proposed to detect stepping-stone intrusion [Che95, Zha00, Yod00, Don02, Blu04] and trace back intrusions [Sna91, Jun93, KFG93, Sav01, Sno02, Wan02, Wan03, Wan04, Pen05]. In this section, we first summarize briefly some passive methods to detect stepping-stones, and then conclude with some methods proposed to trace back intrusions.

2.1 Stepping-stone Detection

Most stepping-stone detection technologies are passive approaches, which are techniques to detect stepping-stones depending on the information gathered from hosts and networks. Their main advantage is they do not need to interfere with the sessions. The disadvantage of passive approaches is that they find a stepping-stone pair among all

the incoming and outgoing connections of a host, and thus have more computations than active methods.

2.1.1 Content-based Thumbprint

Staniford-Chen and Heberlein proposed a method to identify intruders by comparing different sessions for similarities suggestive of connection chains [Che95]. Thumbprint, which is short summaries of the contents of a connection chain, was proposed. The basic idea of thumbprint was originally proposed in [Heb92]. The ideal is a function of the connection which uniquely distinguishes a given connection from all other unrelated connections, but has the same value over two connections which are related by being links in the same connection chain. Intrusion using stepping-stones can be detected by comparing one thumbprint of the connection coming to a host to that of one leaving the host.

The thumbprint requires a 24 byte-per-minute per connection, and thus is efficient and has a low probability of leaking the information contents. A methodology from multivariate statistics called principal component analysis is used to infer the best choice of thumbprint parameters from data. An algorithm was developed to compare the thumbprints which allow for the possibility that data may leak from one time interval to the next.

This approach copes with the problems of clock skews, propagation delays, loss of characters, and packetization variation. The concept experiments demonstrated that the thumbprints are additive and tolerant of noise. The experimental results showed that in the total of 302 thumbprint comparisons, 98.3% could give expected results, while 1.7%

did not work. The results indicate that this method can clearly identify the connections despite the noise.

The fatal problem of this method is that it cannot be applied to encrypted sessions because their contents are not available and thus are unable to make thumbprint. Another problem is that it is vulnerable to intruders' manipulation. Related connections could be maneuvered to be unrelated. Therefore, misdetections are increased. Unrelated connections may be manipulated to seem related and thus introduce false alarms. Besides, this method also assumes the availability of thumbprints from all hosts involved, and this is practical only within a local area network.

2.1.2 Time-based Approach

The time-based approach was proposed by Zhang and Paxson [Zha00]; this is the first time the concept of stepping-stone was formally proposed. Unlike the above, this method can be used to detect stepping-stones or to trace intrusion even if a session is encrypted. In this approach, a flow is considered to be in an 'OFF' period when there is no data traffic on a flow for more than a predefined time threshold; otherwise, it is in an 'ON' period. By correlating the 'ON/OFF' signatures of incoming and outgoing connections, stepping-stones could be detected.

Zhang and Paxson further pointed out that tracing traffic at multiple points could potentially provide more information about traffic characteristics. Doing so complicates the problem of comparing the traffic traces. Therefore, they confine their discussion to the single measurement point case, and assume that the measurement point is on the access link between a site and the rest of the Internet. They describe direct and indirect

stepping-stones, discuss the real-time and off-line detecting, and compare passive monitoring and active perturbation.

To achieve real-time stepping-stone detection, Zhang and Paxson also introduced a filtering technique to only capture small packets. It can significantly reduce the packet capture load. The reason is that for an interactive session, keystroke packets are quite small. Even when entire lines of input are transferred using "line mode" [Bor90], packet payloads tend to be much smaller than those used for bulk-transfer protocols.

Another technique to make their approach [Zha00] more real-time is to minimize state for connection pairs because it is often unfeasible to keep the stepping-stone state for all possible pairs of connections due to the N^2 memory requirements, where N is the number of incoming or outgoing connections. Stepping-state size is largely minimized by removing connection pairs sharing the same port on the same host, removing connection pairs with inconsistent directions, and removing connection pairs with inconsistent timing. This paper [Zha00] also discusses the accuracy of the time-based approach to detect stepping-stone in terms of false positives and false negatives, as well as its responsiveness; this means how soon a stepping-stone is detected after the connection starts. The experimental results showed for the case that has 23 stepping-stones in a total of 3831 connections, 21 stepping stones are reported by the time-based detection algorithm with zero false positive and 2 false negatives.

There are some problems with the time-based approach to detect stepping-stones. First, the fatal problem is that this method is vulnerable to intruders' manipulation. An intruder can easily evade time-based detection by 1) avoiding the introduction of any idle times to

correlate; 2) introducing spurious idle times on one of the connections not reflected in the other connection; 3) stretching out the latency lag between the two connections; or 4) inserting some meaningless packets in one connection. Second, time-based correlation requires that the packets of connections have precise, synchronized timestamps, in order to be able to correlate them. This makes correlations of measurements taken at different points in the network difficult or impractical. Third, Zhang and Paxson [Zha00] also were aware that a large number of legitimate stepping-stone users routinely traverse a network for a variety of reasons. Their observation results showed that one large site (the University of California at Berkeley) has more than 100 such stepping-stones each day.

2.1.3 Deviation-based Approach

The deviation-based approach proposed by Yoda and Etoh [Yod00] is a network-based correlation scheme. It defines the deviation as the minimum average delay gap between the packet streams of two TCP connections. This comes from the observation that the deviation for two unrelated connections is large enough to be distinguished from the deviation of connections in the same connection chain. This approach considers both the packet timing characteristics and their TCP sequence numbers. It does not require clock synchronization and is able to correlate connections observed at different points of a network. It is robust against retransmission variations.

Yoda and Etoh [Yod00] introduce deviation to measure the similarity of two packet streams. If the value is small, one stream is likely to be in the same chain with another. Otherwise, they are probably unrelated. If we plot a packet stream on a connection with the sequence numbers of the packets on the Y-axis and the time stamps when the packets

are captured on the X-axis, they found that the data point should move down and to the right when a retransmission occurs. The points are monotonically increasing if the upper bound of the sequence numbers for each of the time stamps is considered. For an interactive session, the packet streams of the connections must show characteristic patterns for each intrusion. So it is possible that packet streams of different connections will be similar if the proper parts of the streams from the same chain are compared to each other.

Deviation between two streams is a very important concept in this approach. We assume that there are two streams, A, and B, with n, and m packets, respectively; the sequence number of the last data byte in the ith packets of stream A, B are a_i, and b_i , and the data sizes are a_i-a_{i-1}, b_i-b_{i-1}, respectively. We also assume that a_0 is the initial sequence number of stream A and b_0 is the initial sequence number of stream B. Let $t(s)$ ($a_0<s\leqslant a_n$) represent the timestamps of the packets of stream A with sequence number s, and $u(r)$ ($b_0<r\leqslant b_m$) represent the timestamps of the packets of stream B with sequence number r. The deviation for B from A is defined as

$$\frac{1}{d}\min_{0\le k\le m'}\left\{\left|\sum_{h=1}^{d}\left(T(h,k)-\min_{1\le h\le d}\{T(h,k)\}\right)\right|,\left|\sum_{h=1}^{d}\left(T(h,k)-\max_{1\le h\le d}\{T(h,k)\}\right)\right|\right\},$$

where $T(h,k)=u(b_k+h)-t(a_0+h), d=a_n-a_0$, and $m'=\max\{i\mid b_i+d\le b_m\}$.

The experimental results showed that in the distribution of the deviations (18733 deviations in total), a deviation less than three seconds is extremely rare. This indicates that if the deviation of a packet stream is in this range, it is highly likely that the packet stream is in the same connection chain with the given one. Therefore, it can be found that

a packet stream on a connection is in the same connection chain with the given one; this can be achieved by looking for connections whose average propagation delay minus minimum propagation delay is at most three seconds between the beginning and the end of the chain. Generally, this upper bound of the average propagation delay minus the minimum propagation delay of a connection chain gets larger as the time period of a given connection is longer and more data bytes are available.

Besides the same problems as the time-based approach, the deviation-based approach has other problems: 1) deviation computation is not efficient; 2) it is not available for a compressed session because it depends on packet size; 3) it can only correlate TCP connections that have one-to-one correspondences in their TCP sequence numbers, and thus is not able to correlate connections where padding is added to the payload; and 4) it defines the correlation metrics over the entire duration of the connections to be correlated and thus makes correlation applicable to post-attack traces only. The last point also exists in the time-based approach.

2.1.4 Round-trip Time Approach

The round-trip time (RTT) approach proposed by Yung [Yun02] is a method to detect stepping-stones, as well as intrusion, by estimating the downstream length through the gap between a request and its response, and the gap between the request and its acknowledgement. Comparing with the methods determining an intrusion depending on if a host is being used as a stepping-stone, the method estimating the downstream length to determine an intrusion can largely decrease false positives. The RTT approach mainly focuses on detecting a stepping-stone intrusion, rather than purely a stepping-stone.

Yung computes the gaps in which each gap is between each send and its consecutive echo packet. To decrease false negatives, Yung takes the minimum of the gaps to represent the RTT of the session, denoted as echo-RTT. The gaps between each send and its corresponding acknowledgement packet are similarly computed. In order to decrease false positives, the maximum of the gaps is used to represent the length of the connection to the downstream closest host, denoted as ack-RTT. The ratio between ack-RTT and echo-RTT can be used to estimate the relative length of a downstream connection chain. The smaller the ratio is, the longer the downstream length and the higher probability an intrusion has been detected. The experimental result showed that this method has fewer false positives, but false negatives are introduced. The reason is that estimating the length of a downstream session through ack-RTT is not reasonable; this is called the yardstick problem [Yan05-1]. On the other hand, the precision of estimating ack-RTT and echo-RTT still affects the detecting result; this is called packet-matching problem.

In general, the RTT approach is the first one proposed to detect stepping-stone intrusion by estimating the downstream length and thus largely decrease false positives; however, it still has some problems besides the two mentioned above: 1) it does not exactly compute the downstream length in terms of the number of connections; 2) there is no reason to multiply ack-RTT by factor '2' when the ratio is computed; and 3) the upstream length is not taken into consideration and thus false negatives are somehow increased. We must mention that all the methods we propose still have the third problem. Currently, we are still working on estimating the upstream length in order to decrease the false negatives in detecting stepping-stone intrusion.

2.1.5 Packet Number Difference-based Approach

The packet number difference-based (PND-based) approach proposed by Blum [Blu04] is a method to detect stepping-stones by checking the difference of Send packet numbers between two connections. The basic idea is that if the two connections are relayed, the difference should be bounded; otherwise, it should not. This method has the ability to resist intruders' evasions, such as time jittering and chaff perturbation.

Donoho *et al.* [Don02] are the first to show that there are theoretical limits on the ability of attackers to disguise their traffic using evasions during an interactive long session. These evasions consist of local jittering of packet arrival times and the addition of superfluous packets. It is assumed that intruders have a maximum delay tolerance. They proved by using wavelet and multi-scale methods that detecting stepping-stones is still possible by monitoring a session long time enough, even under the assumption that a session is jittered by time and chaff perturbation. However, Donoho *et al.* never pointed out how long time a session needs to be monitored in order to detect a stepping-stone, and never showed the effectiveness against chaff.

Blum continued Donoho's work and proposed the PND-based algorithm for stepping-stone detection using ideas from Computational Learning Theory and from the analysis of the random walk process [Blu04]. Blum achieved provable upper bounds on the number of packets needing to be monitored in an interactive session in order to get given confidence. He solved the problem of the minimum time a session is monitored to detect a stepping-stone. He also examined the consequences when attackers insert chaff into the stepping-stone traffic, and provided a lower bound on the amount of chaff that an attacker

would have to send to evade his detection. The problem with the PND-based approach is that its upper bound on the number packets required is large, while the lower bound on the amount of chaff needed for an attacker to evade his detection is small. For example, an intruder can evade Blum's chaff detection by inserting four packets in every 200 packets, based on Blum's Theory 7 [Blu04]. Thus, Blum's method is very weak in resisting intruders' chaff evasion.

2.2 Connection Traceback

Connection traceback technologies can be divided into two categories: active and passive approaches. Active approaches are technologies to detect stepping-stones (or to trace back intrusions) without depending on the information gathered from hosts and networks. Instead, they actively inject or embed some information into the sessions. They need to interfere with hosts and networks and thus cost some time at this point. But they do not need to compare all the incoming connections with all the outgoing connections of a host to determine a stepping-stone pair. Instead, what they need to do is to compare the injected incoming connection with all the outgoing connections. In this point, active approaches are more efficient than passive ones. In addition to the passive technologies discussed in Section 2.1, other techniques, including the distributed intrusion detection system, caller identification, and caller ID approach, can also be used to trace intrusion back. We first discuss these three technologies, and then summarize the watermark-based approaches.

2.2.1 Distributed Intrusion Detection System

The Distributed Intrusion Detection System (DIDS) [Sna91] is a host-based, passive approach to trace intrusions back. DIDS has a centralized analysis server, which is responsible for tracking all the users in the network and their accounts and for collecting audit trails for each user in each host monitored. Audit abstracts are generated based on the audit trials collected and are sent to the centralized DIDS analysis server. This server determines if there have been intrusions based on the audit abstracts, which are very small and impossible to leak any information of a host. Tracing intruders back is trivial in a kind of framework like this.

Collecting and transferring audit abstracts do not cost too much CPU time and do not result in too much network traffic for a local area network. It is a fact that this approach cannot be applied to a large network because of its centralized analysis of the users' activities. Another problem is that it failed to trace intruders back if one host was unavailable to provide the user's audit log, as this method is still a host-based approach. The third problem is that once the server was attacked by something like DoS, it would be a disaster for the whole DIDS.

2.2.2 Caller Identification System

The Caller Identification System (CIS) was proposed with the goal of applying this system to the Internet [Jun93]. This method eliminates the centralized analysis server used in DIDS [Sna91]; this was a weakness because the DIDS server would be overloaded to prevent CIS from being applied to a large area network, such as the Internet. CIS works by asking each host along the chain to provide the information that

records the previous login situation. Suppose the current chain contains n-1 hosts; there is one user to login to host h_2 from host h_1, and so on, until there is a login to host h_{n-1} from host h_{n-2}. Once the user tries to login to host h_3 from host h_2, host h_2 must remember its current chain is from h_1 to h_2, and host h_3 will query host h_2 to report its view of the current chain. If the user iterates this step for every login, the consequence is that once a connection chain is established from host h_1 to host h_{n-1}, host h_{n-1} will keep the whole view of this connection chain. If host h_n is a victim, as long as the user tries to login to host h_n, it will get the whole connection chain information from host h_{n-1}, unless host h_{n-1} has failed to login to host h_n. It is trivial to trace the original attack position back with the help of the connection information holding on each compromised host.

The problem with CIS is that the network load will be increased because the query incurs additional communication when the user tries to login to one host from another. Another problem is that this method breaks the integrity of privacy because connection information should be private for each host. The most important issue is that no intruder would be likely to select such a tool to attack others. What is being used currently are some open tools with encryption function, such as OpenSSH.

2.2.3 Caller ID Approach

Another approach that was actually used by the United States Air Force to track an intruder is Caller ID [Che95], which is also host-based, but is an active method. Its basic idea is based on the belief that if an intruder could hop through compromised hosts prior to launching an attack, there must be vulnerabilities in those hosts. We could break back into those hosts in reverse order with the help of the knowledge of the same attacks by

using these vulnerabilities and thus eventually identify the original attacking position. This method largely reduces network load compared to the previous approaches, but the drawbacks of this tracing technique include the following [Che95]: the possibility that one could not break back into one of the compromised hosts; one must perform the tracing while the intruder is still active; one runs the risk of accidentally damaging intermediate systems; and in some countries, and in many states in the United States, many states' laws prohibit breaking into other hosts or networks.

2.2.4 Watermark-based Traceback

2.2.4.1 Sleepy watermark

Sleepy watermark (SW) proposed by Wang [Wan01] focuses mainly on tracing intrusions. It is sleepy because it does not introduce any overhead when no intrusion is detected; it is active because when an intrusion is detected, the target will inject a watermark into the backward connection of the intrusion, and then wake up and collaborate with intermediate routers along the intrusion path. By combining a sleepy intrusion response scheme, a watermark correlation technique, and an active tracing protocol, SW provides a highly efficient and accurate source tracing on interactive intrusions. The prototype shows that SW is able to trace back to the trustworthy SW guardian gateway that is closest to the source of the intrusion chain, within a single keystroke of the intruder. Even if an intruder is silent, the intrusion is still traceable by actively injecting a watermark back to the intrusion connection. The experiment shows that SW's own impact on a gateway's processing delay is only around 50 microseconds.

Wang [Wan01] claimed that SW outperforms other existing intrusion tracing approaches, and has many potential advantages, as follows: 1) SW does not require any node other than the intrusion target to have intrusion detection capability; 2) it is not necessary to record all the concurrent incoming and outgoing connections at any node; 3) it does not require correlating each of the incoming connections with each of the outgoing connections; 4) it does not require clock synchronization, and is robust against retransmission variation; 5) it can be implemented efficiently; 6) it does not introduce any noticeable overhead to routers; and 7) it only requires a few network server applications at the intrusion target host to be modified to inject a watermark.

If the delay perturbation is not very large, the watermark information will remain along the connection chain. SW's fatal problem is that it is not guaranteed that the watermark may not be destroyed when delay and chaff perturbations exist simultaneously in a connection.

2.2.4.2 Inter packet delay

Inter packet delay (IPD) proposed by Wang *et al.* [Wan02] is to trace encrypted connections through stepping-stones. Of course, it can be used to detect stepping-stones as well. Wang *et al.* define their correlation metric over the IPDs in a sliding window of packets of the connections to be correlated. They show that the IPD characteristics may be preserved across many router hops and stepping-stones. The effectiveness of IPD for correlation purposes also requires that timing characteristics be distinctive enough to identify connections. Experiments showed that the number of packets needed to correlate two correlations correctly is quite modest. The IPD correlation scheme could be evaded

by only highly sophisticated intruders, and thus it deters network-based intrusion. The experiments also indicate that correlation detection is significantly dependent on the uniqueness of flows. Normal interactive connections such as telnet, SSH and rlogin are almost always unique enough to be differentiated from the connections not in the same chain.

IPDs are invariant across routers and stepping-stones, are not affected by encryption and decryption, and are unique to each connection chain. Therefore, IPD is the best candidate to be used to correlate two connections, compared with other potential candidates, such as packet header information and packet size. To support the true real-time correlation, an IPD correlation of two connections involves the two-step process of first generating correlation points, and second, obtaining correlation value. A correlation point generated from IPDs within the window reflects some localized similarity between the two flows; the correlation value obtained from all the correlation points indicates the overall similarity of the two flows. We assume for a flow n packets are captured, and use t_i to represent the timestamp of the ith packet. The ith IPD is defined as $d_i = t_{i+1} - t_i$, and the IPD vector is defined as $< d_1, d_2, ..., d_n >$.

Intruders could thwart an IPD-based correlation by deliberately changing the inter-packet delays of a connection in a chain. Other countermeasures include injecting "padding" packets that can be removed by the application, and segmenting one flow into multiple flows and reassembling them. In addition, an IPD-based correlation is not applicable to non-interactive traffic.

2.2.4.3 Active timing-based correlation

The active timing-based approach (ATB) proposed by Wang *et al.* [Wan03] is a method to detect stepping-stone intrusion and evasion by using injected watermarks. It is also called watermark-based approach. The approach was designed to meet the challenge caused by time perturbation. A watermark is introduced by slightly adjusting the timing of selected packets of the flow. The tradeoff between timing perturbation characteristics and achievable correlation effectiveness is also identified. Experiments showed that this method performs significantly better than existing, passive, timing-based correlation in the presence of random packet timing perturbations. It can achieve a detection rate arbitrarily close to 100%, while a watermark collision (false positive) rate is arbitrarily close to 0% at the same time.

Digital watermarking [Cox02] involves the selection of a watermark carrier domain and the design of two complementary processes: embedding and decoding. The embedding process embeds the watermark bits into the carrier signal by a slight modification of some property of the watermark carrier. The decoding process extracts any watermark bits from the carrier signal. In [Wan03], the IPD is selected as the watermarking carrier domain. To make ATB more robust against timing attacks, the watermark is embedded only over selected IPDs. The selection of IPDs requires randomly choosing the set of packets and randomly pairing those chosen packets to get IPDs. It should be difficult for intruders to detect, extract, or corrupt the embedded watermark because the IPD's selection function is unknown to the intruders. Wang and Reeves [Wan03] show that if a perturbation of an IPD is within the tolerable perturbation

range, the embedded watermark bit is guaranteed not to be changed by the timing jittering attack. The larger the value of the tolerable perturbation range, the more robust the embedded watermark bit will be, but resulting in more delay to the selected packets.

In theory, this watermark scheme is effective and robust against random delays that are independent and identically distributed over the set of watermarked packets. This is also one problem of this approach because an intruder may use non-independent, non-random delays to perturb a session. An extreme case would be when an intruder knows exactly which packets have been delayed and by how much, making it much easier to corrupt the embedded watermark bits. Another problem of this approach is that if the perturbation of the IPD is outside the tolerable perturbation range, the embedded watermark bit may be altered by the intruder. It is not trivial to determine the tolerable perturbation range of this scheme. Another problem is that this watermark scheme does not work against intruders' chaff perturbation.

The more challenging thing is to develop a stepping-stone detection scheme which can be against intruders' chaff and time perturbation simultaneously. Peng, *et al.* [Pen05] modified the watermark scheme above to make it suitable to correlate perturbed traffic flows with chaff packets. Their basic idea is to embed timing-based watermarks into attack flows, and then use a packet matching method to find all possible corresponding packets in suspicious flows. Correlation results are decided by decoding the watermarks closest to the original ones from all packet combinations. There are four algorithms proposed with different tradeoffs among detection rate, false positive rate and computation cost: Brute Force Algorithm (BFA); Greedy Algorithm (GA); Greedy Plus

Algorithm (GPA); and Greedy Star Algorithm (GSA). Among those algorithms, BFA always has the highest detection rate but the highest computation cost; GA has the lowest computation cost and a potentially high false positive rate; GPA reduces the false positive rate at a slightly decreased detection rate compared with GA; and GSA is a simplified GPA with much better efficiency but with the cost of a slightly decreased detection rate compared with GPA. In general, GPA has the overall best tradeoff among detection rates, false positive rates and computation cost, while GSA suffers from a high computation cost, especially when it fails to find a correlation.

The experimental results showed that this scheme has better overall performance than the best existing approach when comparing for detection rate, false positive rate and computation cost. But there are still some problems with this approach. First, this scheme may not always return the desired result against packet loss or re-packetization which is common when packets arrive too closely or system load is high. Second, the performance of this scheme is evaluated under the assumption that the chaff packets obey a Poisson distribution. Actually, an intruder could insert packets with any distribution he/she wanted, not necessarily obeying a specific distribution.

3. STEPPING-STONE DETECTION

There are lots of proposed approaches to detect stepping-stones. We have discussed the advantages and disadvantages of those methods in Chapter 2. The state of the art is Blum's approach [Blu04]. We propose a novel approach which exploits the number of TCP requests and responses to detect stepping-stones. The difference between the number of requests and that of the responses can be modeled by a random walk process. The theoretical analysis shows that the performance of this approach is better than Blum's approach in terms of the number of packets monitored necessarily under the same confidence with the assumption that the session is manipulated by time jittering or chaff perturbation.

3.1 Problem Statement

The basic idea of detecting a host or a network is being used as a stepping-stone is to compare the incoming connections with the outgoing connections to see if there is a stepping-stone pair. As Figure 1.2 showed, host h_i has one incoming connection C_i^1 and one outgoing connection C_i^2, while each connection has one request stream and one response stream. If we make the following assumptions, then in a period of time, the number of packets monitored in each connection should be close to equal if the two connections are relayed:

1) Each packet appearing in one connection must appear in its relayed one;

The condition above means there are no packet drops, combinations, or decomposing. These conditions guarantee that the number of the packets in an incoming connection must be greater than or equal to the number of the packets in the relayed outgoing connection. If two connections are relayed, we can at least find a relationship between the number of the requests of the outgoing connection and the number of responses of the incoming connection. So the problem of detecting stepping-stones becomes the problem of finding a correlation between the number of requests and the number of responses. Before discussing this relationship, we make the following two additional assumptions:

2) An intruder could hold any packet at any place, but the holding time has an upper bound;

3) An intruder could insert any packet at any time, but the inserting rate is bounded.

Assumption 2) comes from the fact that each user has a time tolerance of using an interactive session; and the last assumption indicates that the ability of a user to insert packets into an interactive session is bounded.

Based on the discussions above and the four assumptions, if two connections are relayed, the relationship between the number of requests and that of responses should exist, and we can use the existence of this relationship to determine if two connections are in the same chain. Our statement is that it is possible to detect stepping-stone by comparing the number of Sends in an outgoing connection with the number of Echoes in an upstream connection.

3.2 Stepping-stone Detection Algorithm

3.2.1 Motivation

Under the assumption that connections are not manipulated, what is the relationship between the number of requests of an outgoing connection and the number of responses of the corresponding incoming connection? If the connections are not relayed, then what is the relationship? To answer these questions, we first need to analyze the TCP/IP protocol to understand the mechanism of request and response, and the relationship between a request and its corresponding response.

TCP/IP protocol regulates that a request will be replied to in most cases. But there are some except special cases, (such as the requests of keep-alive, key-exchange, ignore, or some other special purpose requests,) which are not replied to. Among all the packets, the number of special purpose requests without replies is very small. So in our analysis, we ignore this part and assume that each request will be responded to or echoed at the destination host. It does not mean there is a one-one mapping between the requests and the responses. It is possible that many requests are replied to by one response, or one request is replied to by many responses.

However, in our observation, the case that multiple requests are echoed by one packet seldom happens, unless the requests have very close timestamps, which can be caused by a very fast keystroke speed. The case in which one request is echoed by more packets happens frequently. For example, if a user inputs the command "*ls*" in UNIX environment, and then inputs an 'Enter' request, the user will get response packets '*l*', '*s*', and then one or more responses for 'Enter'. In general, we know that if we monitor a

connection for a period of time, the number of responses will be more than the number of requests; the difference between them (number of Echoes minus number of Send) should be positive when two connections are relayed. If the two connections are not relayed, sometimes the difference may be positive, sometimes may be negative, but not always positive. If two connections are relayed, there is a strong correlation between them. If they are not relayed, the correlation is relatively weak.

3.2.2 Basic Idea to Detect Stepping-stone

In an interactive session, the user will input a command by typing some letters, and then execute the command at the server side. The execution result would return to the client side in terms of packets. In general, when a user type one letter (keystroke), the letter would be echoed by one-letter packet. We call them single letter Send and Echo, respectively. If we can filter out the non-single letter Echo and only keep the single letter Sends and Echoes, the number of the Sends in an outgoing connection should be close to the number of the Echoes in an incoming connection if the two connections are relayed.

As Figure 1.2 showed, we use $N_{(i,s)}^{(2)}$ to denote the number of requests of the outgoing connection, and use $N_{(i,e)}^{(1)}$ to denote the number of responses of the incoming connection, and use Δ to denote $N_{(i,e)}^{(1)} - N_{(i,s)}^{(2)}$. For relayed connections, Δ should vary around zero. There are two reasons why Δ may not be exactly zero. One is that either Sends or Echoes may be combined to one packet during propagation. Due to the TCP/IP protocol, we may not be able to identify all single letter packets. We cannot completely remove the packets

of command execution result by checking packet size. However, if the two connections are relayed, the difference should be around zero with a high probability.

If two relayed connections are manipulated, based on assumptions 2) and 3), the difference Δ should be bounded within a range $[-\Omega_l, \Omega_u]$. In time jittering evasion, if a packet is held, we assume that the holding time cannot be larger than H and, at most Ω_H packets can be held in each connection. In chaff perturbation, we assume that at most r packets are introduced in a unit time for each connection. Assuming that we collect the packets in Φ units of time, the difference Δ should be bounded within a range $[-\Omega_\Phi, \Omega_\Phi]$ for two relayed connections where $\Omega_\Phi = \Phi * r$. Detecting a stepping-stone pair is reduced to judging if the differences of the number of single letter packets between two connections are bounded. That is for a stepping-stone pair, the follow relationships should be held:

$$-\Omega_\Phi < \Delta < \Omega_\Phi \tag{3.1}$$

3.2.3 Stepping-stone Detection Algorithm

To reduce the false alarms, and misdetections in detecting stepping-stone pair, we check Equations (3.1) every time a packet is received. If we monitor a total of w packets, the equation will be checked w times. We propose the following algorithm to detect stepping-stones. We call this algorithm detecting stepping-stone evasion (DSE).

DSA $(S_i^{(2)}, E_i^{(1)}, \Omega_\Phi, w)$

$$
\begin{aligned}
& N_{(i,e)}^{(1)} = N_{(i,s)}^{(2)} = 0; \\
& for \quad j = 1 : w \\
& \qquad if \quad p_j \in S_i^{(2)} \quad N_{(i,s)}^{(2)} ++; \\
& \qquad if \quad p_j \in E_i^{(1)} \quad N_{(i,e)}^{(1)} ++; \\
& \qquad \Delta = N_{(i,e)}^{(1)} - N_{(i,s)}^{(2)}; \\
& \qquad if \quad \Delta < -\Omega_\Phi \quad or \quad \Delta > \Omega_\Phi \\
& \qquad\qquad return \quad Normal \\
& Endfor \\
& return \quad Stepping - Stone
\end{aligned}
$$

End

In this algorithm, we capture and check up to w packets on the two connections to see if (3.1) is satisfied. If there is one time that (3.1) is not satisfied, then we conclude that there is no stepping-stone pair. The final conclusion that if there is a stepping-stone should be made only after all the connections are checked. If (3.1) is satisfied within w times check, then we say with a very high probability there is a stepping-stone. It is not necessary to check if (3.1) holds for all the connections. The larger the w, the higher the confidence of DSE. For a given confidence, which is also called false positive probability, how many packets are minimally needed to be monitored on the two connections?

3.2.4 Performance Analysis

We assume that a packet is a Send with probability q, and an Echo with probability p. The difference Δ between the number of the Sends of a stream and that of the Echoes of another stream can be modeled as a random walk process with independent jumps Z_1, $Z_2, \ldots, Z_i, \ldots$, where i is a positive integer. If a packet captured is a Send, the difference Δ will make a jump Z_i=-1, otherwise, a jump Z_i=1; there is no other choice. So we have:

$$\left.\begin{array}{l} prob(Z_i = -1) = q \\ prob(Z_i = 1) = p \\ p + q = 1 \end{array}\right\}. \tag{3.2}$$

We evaluate the performance of DSE by computing the smallest w for a given false positive detection probability or false negative detection probability. For a given w, we compute the false positive probability and false negative probability to evaluate the algorithm DSE. A false negative probability indicates the possibility that Equation (3.1) holds even if the two connections are in the same chain. A false positive probability indicates how possible it is for the difference Δ to be within the bounds even if the two connections are not in the same chain. For convenience, in the rest of this chapter, we use the notations in Table 3.1.

Table 3.1 Notations used in the analysis of random walks

p_{NC}	False negative probability
p_{PC}	False positive probability
δ_N	A given false negative probability
δ_P	A given false positive probability

3.2.4.1 False negative probability

False negative probability p_{NC} of DSE is actually the sum of the probabilities that the random walk process Δ hits the lower bound $-\Omega_\Phi$ or the upper bound Ω_Φ. Based on the results of the random walk process from [Cox65], we have:

$$p_{NC} = f_{0\Omega_\Phi}^{(w)} + f_{0\Omega_\Phi}^{(w)} \le \frac{1}{2}(\frac{p}{q})^{\frac{1}{2}\Omega_\Phi} \frac{1}{s_1^{w-1}} + \frac{1}{2}(\frac{q}{p})^{\frac{1}{2}\Omega_\Phi} \frac{1}{s_1^{w-1}} = \frac{1}{2s_1^{w-1}}((\frac{p}{q})^{\frac{1}{2}\Omega_\Phi} + (\frac{q}{p})^{\frac{1}{2}\Omega_\Phi}), \quad (3.3)$$

where $s_1 = \dfrac{1}{1-p-q+2(pq)^{\frac{1}{2}}\cos\frac{\pi}{2\Omega_\Phi}} = \dfrac{1}{2(pq)^{\frac{1}{2}}\cos\frac{\pi}{2\Omega_\Phi}}$.

One special case is the following, when p=q:

$$p_{NC} = f_{0\Omega_\Phi}^{(w)} + f_{0\Omega_\Phi}^{(w)} \le \frac{1}{s_1^{w-1}} = \cos^{w-1}\frac{\pi}{2\Omega_\Phi}, \quad (3.4)$$

where $s_1 = \dfrac{1}{2(pq)^{\frac{1}{2}}\cos\frac{\pi}{2\Omega_\Phi}} = \dfrac{1}{\cos\frac{\pi}{2\Omega_\Phi}}$.

From equation (3.4), we get the least packet number needed to monitor with given false negative probability δ_N when p=q:

$$w \ge \frac{\log\delta_N}{\log(\cos\frac{\pi}{2\Omega_\Phi})} + 1. \quad (3.5)$$

3.2.4.2 False positive probability

False positive probability p_{PC} of DSE is the probability that the difference Δ could walk within the range $[-\Omega_\Phi, \Omega_\Phi]$ in all w times checked even though the two connections are not relayed. From the results in [Cox65], we get the following:

$$p_{PC} = \sum_{k=w+1}^{\infty}(f_{0\Omega_\Phi}^{(k)} + f_{0\Omega_\Phi}^{(k)}) = \sum_{k=w+1}^{\infty}\frac{1}{2s_1^{k-1}}((\frac{p}{q})^{\frac{\Omega_\Phi}{2}} + (\frac{q}{p})^{\frac{\Omega_\Phi}{2}}) = \frac{1}{2}((\frac{p}{q})^{\frac{\Omega_\Phi}{2}} + (\frac{q}{p})^{\frac{\Omega_\Phi}{2}})\sum_{k=w+1}^{\infty}(\frac{1}{s_1})^{k-1},$$

$$= \frac{1}{2}((\frac{p}{q})^{\frac{\Omega_\Phi}{2}} + (\frac{q}{p})^{\frac{\Omega_\Phi}{2}})\frac{1}{s_1^{w} - s_1^{w-1}} \tag{3.6}$$

where $s_1 = \dfrac{1}{1-p-q+2(pq)^{\frac{1}{2}}\cos\frac{\pi}{2\Omega_\Phi}} = \dfrac{1}{2(pq)^{\frac{1}{2}}\cos\frac{\pi}{2\Omega_\Phi}}$.

When p=q, we have the following simplified results,

$$p_{PC} = \frac{\cos^{w}\frac{\pi}{2\Omega_\Phi}}{1-\cos\frac{\pi}{2\Omega_\Phi}}. \tag{3.7}$$

Similarly, we get w from (3.7) with given δ_P when p=q as the following:

$$w = \frac{\log[\delta_P(1-\cos\frac{\pi}{2\Omega_\Phi})]}{\log(\cos\frac{\pi}{2\Omega_\Phi})}. \tag{3.8}$$

3.3 Comparison

The best way to justify an algorithm is to compare its performance with the best existing algorithm. So far, Blum's approach has been considered to be the best way to detect stepping-stones. One of the algorithms he proposed can resist intruders' time perturbation and chaff evasion. Here we compare DSE with Blum's methods to see which one can perform better. The method by which we compare them is by seeing which

method needs to monitor fewer packets under the same given false positive rate δ_P. We must mention that Blum did not give false negative analysis in his paper [Blu04]. So we only compare the performance between the two approaches in terms of false positive probability. We discuss two cases: one is without consideration of chaff; another is with consideration of chaff.

3.3.1 Comparison between DSE and the Best Existing Algorithm without Chaff

In order to compare with Blum's Detect-Attacks-Chaff (δ_P, p_Δ) stepping-stone detection algorithm, called DAC (δ_P, p_Δ), we assume the following condition $p_\Delta = \Omega_\Phi$.

With Blum's method, in order to get a given false positive probability δ_P, at least w_B packets are needed to be monitored, while our approach DSE, under the same condition, needs to capture at least w packets. Our purpose is to compare which one is larger. The larger this number, the worse the performance of the algorithm. The numbers w_B and w can be computed by the following formula:

$$w_B = 2(p_\Delta + 1)^2 \log\frac{1}{\delta_P}, \text{ and} \tag{3.9}$$

$$w = \frac{\log[\delta_P(1-\cos\frac{\pi}{2p_\Delta})]}{\log\cos\frac{\pi}{2p_\Delta}}. \tag{3.10}$$

We cannot compare the two numbers directly by using Equations (3.9) and (3.10) because there is no guarantee which one is absolutely bigger than the other one. Figures

3.1 to 3.4 show the comparison results between w_B and w with varying p_Δ under each fixed δ_P, where the Y axis uses the logarithmic scale.

From Figure 3.1, we know DSE has better performance than DAC when p_Δ is under 8. If p_Δ is more than 8, DSE has worse performance than DAC. But when δ_P is under 0.01, obviously, DSE has better performance than DAC up to p_Δ is 50. From Figure 3.4, we know when δ_P becomes smaller, our algorithm can show better performance than Blum's. Our conclusion is that under a high confidence (low false positive probability) and no chaff, DSE always has better performance than DAC because DSE needs to monitor a smaller number of packets.

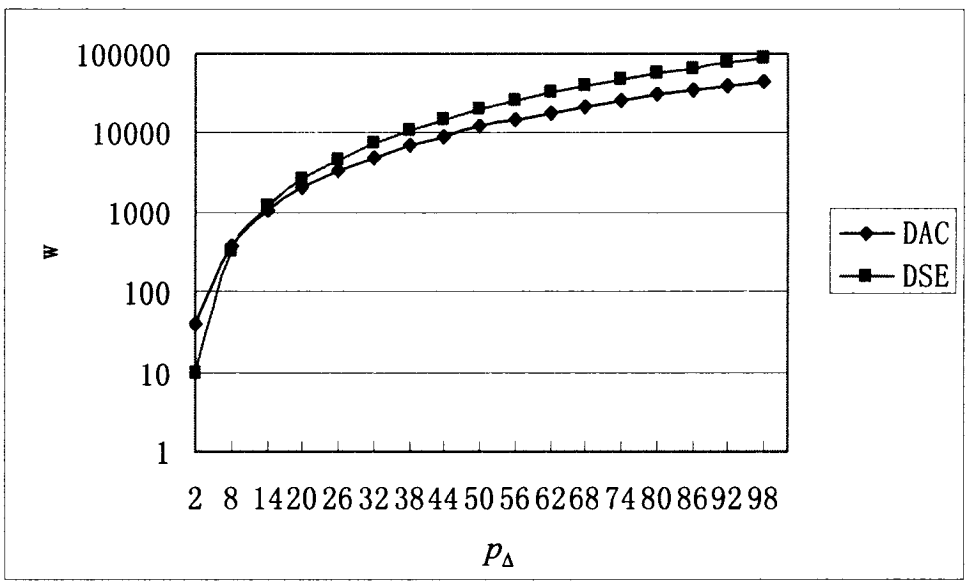


Figure 3.1 Comparison of number of packets monitored with Blum's method under $\delta_P = 0.1$

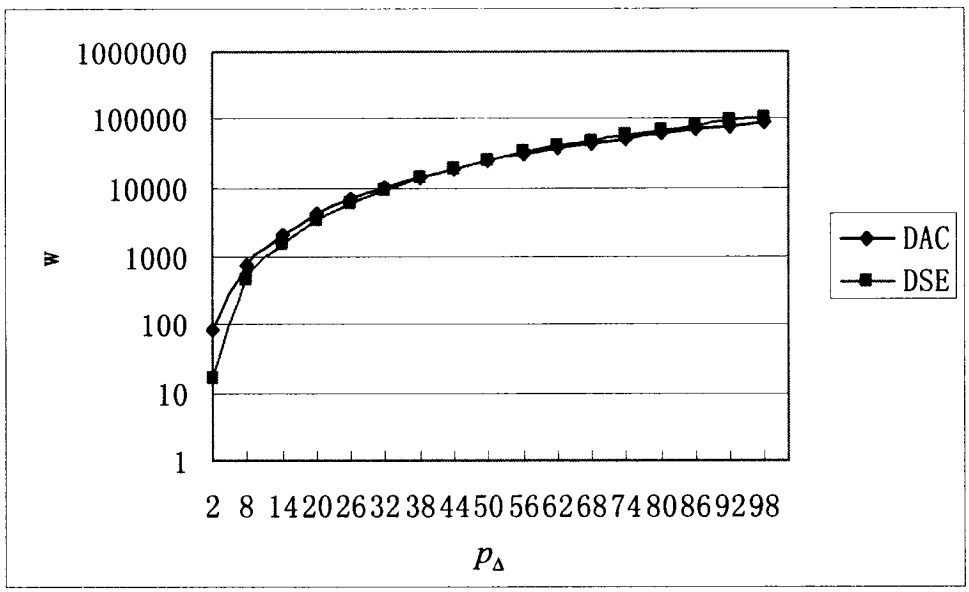


Figure 3.2 Comparison of number of packets monitored with Blum's method under $\delta_P = 0.01$

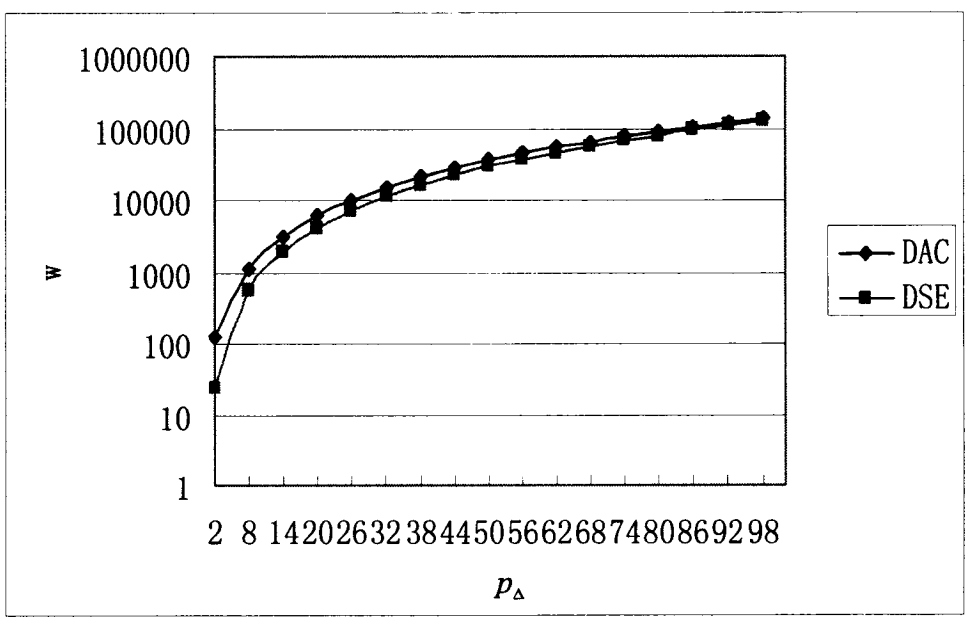


Figure 3.3 Comparison of number of packets monitored with Blum's method under $\delta_P = 0.001$

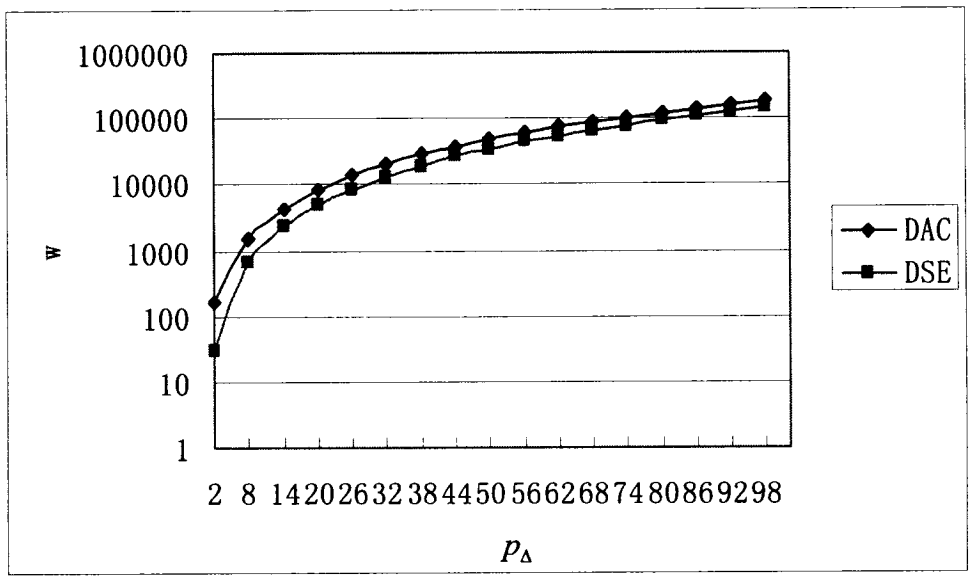


Figure 3.4 Comparison of number of packets monitored with Blum's method under $\delta_p = 0.0001$

3.3.2 Comparison between DSE and the Best Existing Algorithm under Chaff

When a session is manipulated with a chaff perturbation, Blum claimed that his method can still detect stepping-stone, but with a restriction that is no more than x packets are inserted for every $8(x+1)^2$ packets. Otherwise, his method does not work. We evaluate the performance of our method by comparing with Blum's approach. We assume we insert x packets into to a send stream for every x send and approximate x echo packets. This means $p/q = x/(x + x) = 1/2$ and the inserting rate is approximately 50%, which is much bigger than Blum's inserting rate. From Equation (3.6), we have the least number of packets w monitored by DSE with a given δ_p.

$$w = \frac{\log[2\delta_P(1-0.943\cos\frac{\pi}{2p_\Delta}) \Big/ (pow(0.5, p_\Delta/2) + pow(2, p_\Delta/2))]}{\log(0.943\cos\frac{\pi}{2p_\Delta})} \tag{3.11}$$

From [Blu04], the least number of packets w_B monitored by DAC with a given δ_P is as Equation (3.12) shows:

$$w_B = 8(p_\Delta + 1)^2 \log\frac{1}{\delta_P} \tag{3.12}$$

Figures 3.5 to 3.8 show the comparison results between DSE and DAC. Figure 3.5 shows DSE has better performance than DAC when the detection boundary is less than 50 with given δ_P is 0.1. When false positive probability δ_P is decreased to 0.01, 0.001, 0.0001, respectively, the comparison results are shown at Figures 3.6, 3.7, and 3.8, respectively. The lower the false positive probability, the better performance the algorithm DSE. Under chaff perturbation, our algorithm still has better performance than Blum's approach.

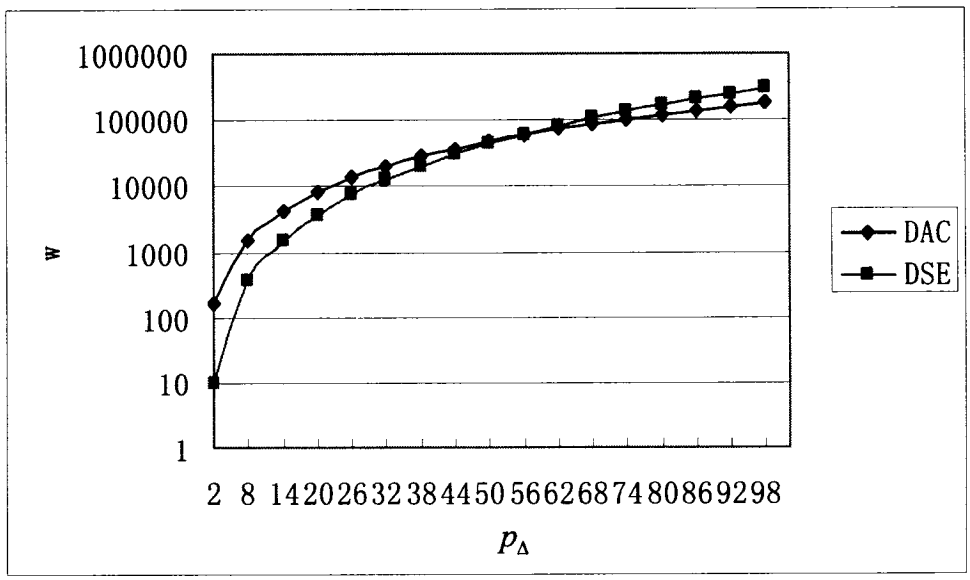


Figure 3.5 Comparison of number of packets monitored with Blum's method under chaff and $\delta_P = 0.1$

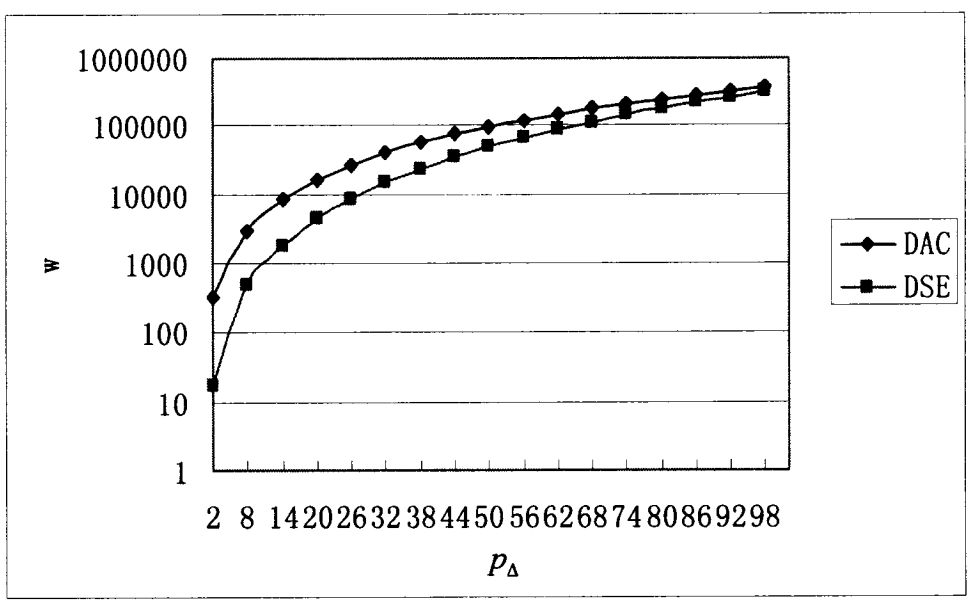


Figure 3.6 Comparison of number of packets monitored with Blum's method under chaff and $\delta_P = 0.01$

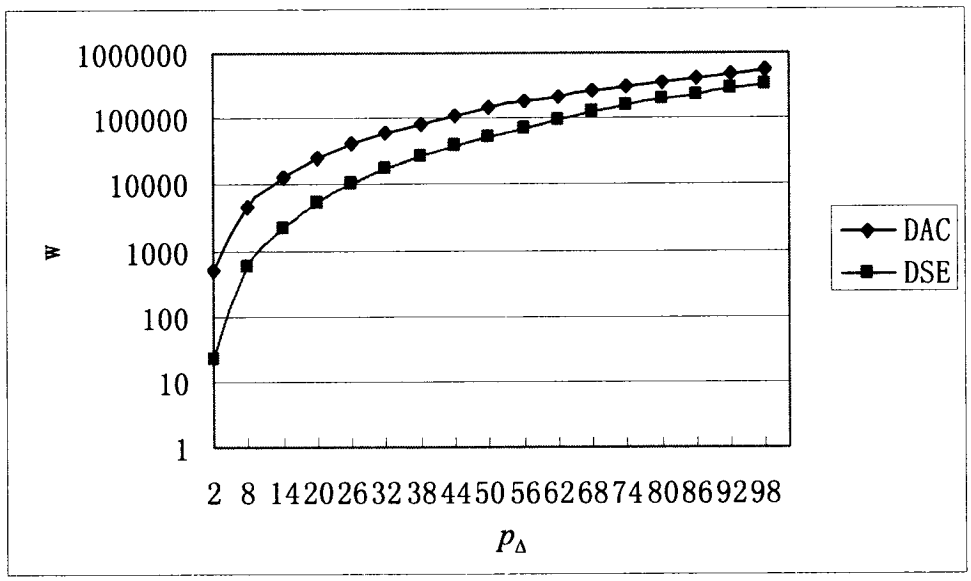


Figure 3.7 Comparison of number of packets monitored with Blum's method under chaff and $\delta_P = 0.001$

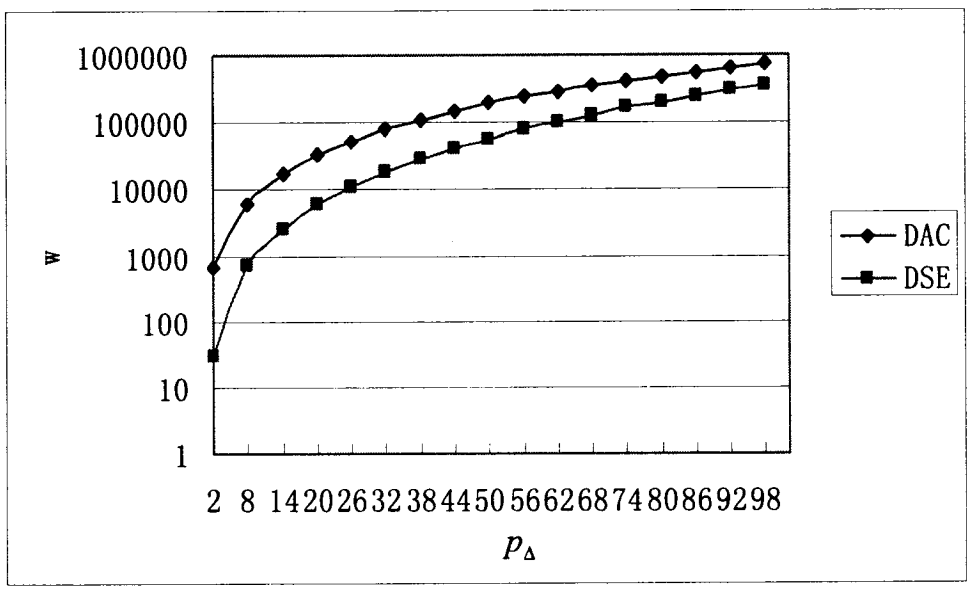


Figure 3.8 Comparison of number of packets monitored with Blum's method under chaff and $\delta_P = 0.0001$

4. STEPPING-STONE INTRUSION DETECTION

In this chapter, we propose several approaches to detect stepping-stone intrusion. They include the step-function, and the fluctuation-based approaches. In Chapter 3, we proposed a method to detect stepping-stones. We must be aware that detecting stepping-stones is not equal to detecting intrusion. If we applied a detecting stepping-stone approach to detecting stepping-stone intrusion, it would introduce false positive errors. The reason is that some legitimate users or applications also use stepping-stones. So purely depending on whether something is being used as a stepping-stone to decide if it is an intrusion would cause some false alarms. But most of the time, being used as a stepping-stone indicates an intrusion.

In order to detect stepping-stone intrusion, it is inadequate to compare the incoming and the outgoing connections of a host (or a network). Yung proposed an approach to detect stepping-stone intrusion, rather than only stepping-stones [Yun02]. His main idea is first to estimate the downstream length, and then determine if there is an intrusion depending on the downstream length estimated. If the length is relatively longer than the length from the host to the next closest one, it is highly suspicious that the session is an intrusion. Otherwise, it is not. Yung uses the gap between a Send and its corresponding Ack to represent the connection length to the next host, called a-RTT, and uses the gap between a Send and its corresponding Echo to represent the downstream length to the destination host, called e-RTT. The ratio between them can indicate the possibility that a session is an intrusion. The lower the ratio, the higher the probability an intrusion exists.

There are still some problems with Yung's approach in detecting stepping-stone intrusion. First, his method to estimate e-RTT and a-RTT cannot work well, especially when Send-Echo pairs are deeply overlapped. Second, it is not reasonable to use a-RTT to measure e-RTT. There are two extreme cases which make Yung's approach work incorrectly. One is if a-RTT is very small, which will make the ratio very small even though the e-RTT is not long; another case is if a-RTT is very long, it will make the ratio very big even though the e-RTT is not short. The first case will cause false alarms, and the second case will make more misdetections. We call this phenomenon the yardstick problem [Yan05-1].

We propose two approaches to solve the yardstick problem and the problem of estimating the downstream length inaccurately. The former is the step-function approach, and the latter is the network fluctuation-based approach.

4.1 Step-function Approach

4.1.1 Motivation and the Basic Idea

The question of detecting stepping-stone intrusion becomes estimating the downstream length in terms of connections. The more connections a downstream length has, the higher the probability an intrusion is being detected. A normal user or application would not like to access the target machine through a long session (more connections) if there is a direct path to access the machine. Observation and analysis show that if a legitimate user or application needs to use stepping-stones, most probably the downstream length is limited within two connections. It is seldom that the downstream

length is more than three connections. From the viewpoint of intruders, if one stepping-stone (meaning two downstream connections) is used, it is still not safe. Of course, the more the stepping-stones there are, the safer but less efficient an intruder feels. Most intruders need to trade off between safety and efficiency. In this dissertation, we select the downstream length threshold to be three. This means if the downstream length of a session is more than 3, an intrusion is detected with a high probability. The problem is how to estimate the downstream length of a connection chain accurately. Step-function is a method to estimate this downstream length. This method is simple and intuitive, but accurate and efficient.

Figure 4.1 shows a session established using OpenSSH, in which a session coming from another host (or network) goes through h_i and connects to h_{i+1} from h_i, and eventually connects to h_n, which is the destination host of this session.

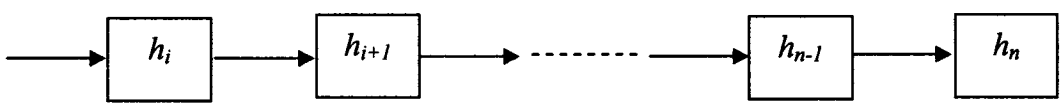


Figure 4.1 Sample of a connection chain using OpenSSH

We assume the host h_i is our monitor point (MP) or the sensor where our detecting algorithm resides. Our purpose is to estimate the number of hosts compromised from the sensor (included) to the destination host h_n (excluded). The number of stepping-stones is the same as the number of connections of the downstream chain from the sensor.

TCP/IP is a protocol between two directly connected hosts. This means host h_i can only know host h_{i+1} is connected with it. Host h_i has no idea about connecting situation

after h_{i+1} in the downstream of the session. If we monitor the outgoing connection of the host h_i, what we could know is the TCP/IP packets coming from and going to host h_{i+1}, rather than any other hosts, such as h_{i+2}, ...h_{n-1}, and h_n. But if there is a session between h_i and h_n, each packet sent from h_i must be acknowledged by h_{i+1} first, and then forwarded to the following hosts of the chain until the destination host h_n. Even though the Echo of a Send of h_i comes from host h_{i+1} from the point of h_i, the reality is this packet is echoed at the destination host, rather than its directly connected host unless there is only one connection in the downstream chain. This motivates in us the idea that if we compute the gap between each Send and its corresponding Echo coming from h_{i+1}, the gaps should vary depending on the number of hosts connected after h_{i+1}. The gaps are increased while more connections are extended along a session.

As Figure 4.1 showed, we monitor the host h_i, capture all the Sends and Echoes from the time that the session is only extended to host h_{i+1}, and compute all the gaps between each Send and its corresponding Echo. We find that those gaps are different, but vary slightly. They are bounded within a range, called one level, denoted as L_1. If we monitor this host and capture all the Sends and Echoes continuously, when one more host is connected, we are supposed to get L_2. In level 2, even though the RTTs are different, on average they are bigger than the RTTs in L_1. When more and more hosts are connected one by one, more and more levels are obtained. As long as we monitor this session from the beginning to the end continuously and capture all the Sends and Echoes, we are able to determine the number of levels we get. In terms of connections, the number of the levels is exactly the same as the downstream length of a connection chain. If we call each

level a step, this method to detect stepping-stone intrusion is called the step-function approach. The related detailed idea was published in the paper [Yan04]. In order to count the number of RTT steps, we designed a step-counting algorithm.

4.1.2 Step-counting Algorithm

We put the RTTs computed into an array. We designed the algorithm in Figure 4.2 to count the "jumps" (steps) in an array. The algorithm can be used in real-time as the values in the array are filled. It only examines the last 2*w elements in order to determine whether there is a jump in the RTTs. Intuitively, we split the 2*w elements into two windows (left and right) of size w each. Within the windows, we select the minimum of the w values. This is to eliminate the network fluctuations. If the difference between the two minima exceeds a threshold, we declare there is a jump between the left

```
// The simplified algorithm below finds only one (the first) "jump"
// Given: Packet roundtrip time array rtt[]
//
threshold = average of some randomly selected values in array rtt[];
for (each element artt, i>=5) {
   minLeft = min(rtt[i-5], rtt[i-4], rtt[i-3]);
   minRight = min(rtt[i-2], rtt[i-1], rtt[i]);
   diff = minRight - minLeft;
   if (diff > 0) {
      jumpDir="up";     // login on to a new host
   } else {
      diff *= -1;
      jumpDir = "down";    // logout from host
   }
   if (diff > threshold) {
      if (jumpDir == "down") {
         there is a jump down between rtt[i-3] and rtt[i-2];
      } else {
         there is a jump up between rtt[i-3] and rtt[i-2];
      }
   }
}
```

Figure 4.2 Step-counting algorithm

and right windows.

This algorithm uses a window of size 6 (w = 3 on each side) that works reasonably well in our experiments. The larger the window size, the better the algorithm. Currently, w = 3 works well in a local area network. More extensive tests on different network conditions are needed to confirm the ideal window size.

This algorithm was simplified to find only one jump. It can be generalized to find all the jumps in an array. We claim that the algorithm is real-time because we have to look ahead w packets of data in order to determine that a jump occurred. If we accumulate all the jumps, we can easily determine the number of intermediate hosts.

4.1.3 Experimental Justification

We designed an experiment to demonstrate our conjecture that RTTs can form steps, and that we can estimate the downstream length by counting steps. All machines in our experiment run the Red Hat 8.0 distribution of Linux. Our test software is installed on one of the machines on which we have root access. We created a connection of seven hosts. Starting from machine host h_1, we login to host h_2, and then from host h_2 login to host h_3, and so on. Host h_7 is the targeted victim in this experiment, and our detecting program was running on host h_4. In every machine we login, we performed a simple task: listing the current directory. For the victim, we did a little bit more: listing current directory, creating a new directory, copying some files from the other directory to the current directory, and then logging out from host h_7 in reverse order. We collected the network packets from the outgoing connection of host h_4 and got a round-trip time array *rtt*[].

A sample result is shown in Figure 4.3, where the Y axis represents the values of RTTs in microseconds, and the X axis represents the packet index number. From this experiment, we have three hosts downstream, and Figure 4.3 shows that we have three steps up. Every step up corresponds to one more host connected. On the right side of Figure 4.3, we have three steps down, showing our exits from each machine. One interesting observation we made is the length of each level. The case in Figure 4.3 is one sample data set, but it is a good representation of the data we collected. Each level consists of about 40-50 packets between the host and the victim. By visually examining Figure 4.3, we can clearly see the forming of several levels (or steps) in the RTTs. This confirms our conjecture that the number of hosts can be determined if we observe the packets from the beginning to the end of a session. We implemented our step-counting algorithm to detect this number automatically.

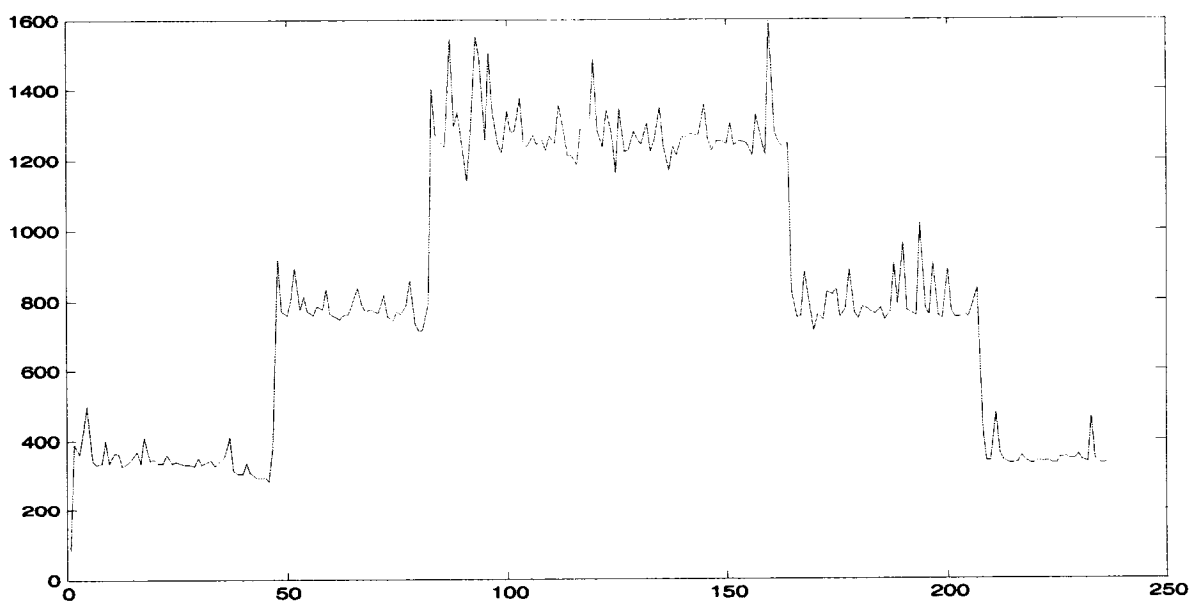


Figure 4.3 Demonstration of the steps of round-trip times

4.2 Network Fluctuation-based Approach

Another method we proposed is to detect stepping-stone intrusion by exploiting network fluctuation, which is the variation of RTTs. The hypothesis of this method comes from the sense that the longer a connection chain, the larger the network fluctuation. We monitor a connection chain as long as the chain is active and compute the RTTs for each send packet. The RTTs are supposed to be uniform without considering the network traffic. The fact is that the RTTs vary to some degree because of router and host processing, queuing delay and network load fluctuation. We use the standard deviation of the RTTs in a connection chain to characterize the network fluctuation. We analyze and verify by experiments that the network fluctuation is mainly determined by the hosts connected and the routers in between. The bigger the network fluctuation, the more connections a connection chain has. We use the fluctuation to the downstream host as a measuring yardstick to estimate the network fluctuation. We call this method the fluctuation-based stepping-stone intrusion detection approach. This approach can overcome the shortcoming of the step-function approach, which needs to monitor a connection chain all the time. Another advantage of this approach is that it can solve the step aggregation problem [Yan05-1] to a certain degree, and the biased yardstick problem.

4.2.1 Network Propagation and Delay

4.2.1.1 Network propagation analysis

A connection chain of more stepping-stones is shown in Figure 1.1, where h_1 is assumed to be a host used by an intruder, h_2, ..., h_{n-1} are hosts used as stepping-stones,

and h_n is a host used as a victim; each connection is built by using tool OpenSSH. We also assume that our stepping-stone intrusion detection program is running on sensor h_i. Actually, there are many routers between two consecutive hosts, such as host h_i and h_{i+1}. We model the routers in between into one as shown in Figure 4.4 for simplifying the network fluctuation analysis, where h_i and h_{i+1} are two consecutive hosts, and $r_{(i,\ i+1)}$ represents all the routers in between.

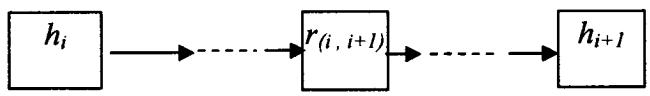


Figure 4.4 Part of a connection chain with routers

Figure 4.4 shows one connection in a connection chain. A typical OSI [For02], [RFC81] communication scenario of the connection shown in Figure 4.4 can be sketched as in Figure 4.5, where L7 to L1 represent application, presentation, session, transport, network, data link, and physical layers, respectively. Most probably a host in this connection works through seven layers, but the router only works through the lowest three layers. Assume h_i is the client side and h_{i+1} is the server side, and a request sent from h_i to h_{i+1} via $r_{(i,\ i+1)}$ is echoed at h_{i+1}. Any packet sent from the client side first is processed in h_i and then forwarded to the server side by the routers in between, just as

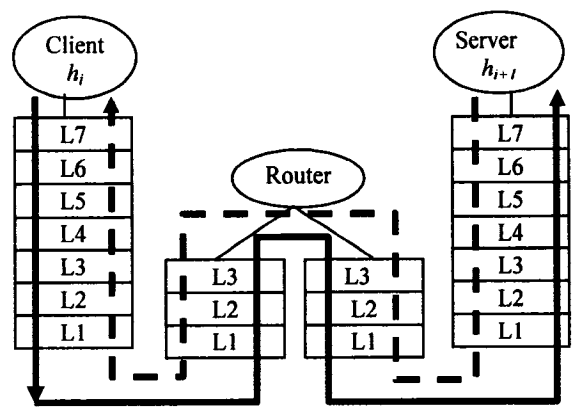


Figure 4.5 OSI model of a connection chain

shown as the solid line in Figure 4.5. The echoed packet sent from the server side experiences the same career but with reverse direction, as the dash line shows.

In a client-server communication, h_i executes the client side program and attempts to send packets to the server h_{i+1}. Before any packet is sent, h_i must first establish a connection to the application layer on itself. The client program uses an A-Associate request function call to start the communication session. Once the application layer has initialized, it contacts the presentation layer, sending a P-Connect request primitive. This establishes the format of the data to be used and the data formats which are to be supported by the system on the network. This information is sent to the session layer with an S-Connect request primitive. The session layer firstly allocates a session identifier, and secondly selects appropriate protocol options to support the services requested by the applications layer. The session layer also identifies the intended recipient of the communication. The session layer proceeds to request a transport layer connection to the remote system using a T-Connect request. The transport request identifies the remote service required and the type of transport protocol to be used. The transport layer requests the network layer to establish a connection to the remote system. The network layer service will normally have established a link layer connection and a physical layer connection to the nearest router. The router receives the packet via the physical layer, and the data link layer and finally processes this packet at the network layer by identifying the destination IP and selects an appropriate path to the next router, eventually reaching the destination host h_{i+1}. Similar processing occurs at the server side h_{i+1} but propagates in reverse direction.

4.2.1.2 Basic idea

The RTT for each 'send' packet is determined by several factors based on the packet propagation process discussed above: (1) the physical distance L (either wired or wireless) between h_i and h_{i+1}; (2) processing and queue time t_2 in h_i to encrypt/send a packet; (3) processing and queuing time t_3 in router $r_{(i,\, i+1)}$; and (4) processing and queuing time t_4 in h_{i+1} to receive/decrypt the echo packet. We use RTT_i to represent the RTT of a send packet, and we have

$$RTT_i = 2(t_1(L_i) + t_2(h_i) + t_3(r_i) + t_4(h_{i+1})). \tag{4.1}$$

We use t_1 to represent the time that a packet propagates the physical distance L with approximate light speed. If we do not consider the bandwidth problem, t_1 is relatively stable for different packets because $t_1 = L_i/c$; here c is the light propagation speed in vacuum space. The time t_2 varies with different packets because it is determined by the queuing time in h_i and the processing time cost on the seven layers, as well as the time t_4. The variation of the time t_3 reflects the scenario of processing, transmitting, and queuing in all the routers. The variation ΔRTT_i of RTT_i is determined by the variation time on the two hosts and the routers. We have the following:

$$\Delta RTT_i \approx 2(\Delta t_2 + \Delta t_3 + \Delta t_4). \tag{4.2}$$

If we have n hosts in between, we should have

$$\Delta RTT = \sum_{i=1}^{n} \Delta RTT_i \cdot \tag{4.3}$$

Here RTT represents the round-trip time for a whole connection chain. Similarly we can compute ΔRTT_1, which is the round-trip time to the downstream nearest server. The ratio between ΔRTT_1 and ΔRTT is supposed to be close to one if the session only has one connection. Otherwise, this ratio is close to zero if the chain is long enough. So the basic idea is that we can determine if there exists an intrusion by estimating the ratio between the variation of RTTs to the downstream nearest server and that of a whole connection chain.

4.2.2 Fluctuation Estimation and Detecting Algorithm

4.2.2.1 Characterizing network fluctuation

Network fluctuation can be described as the variation of RTTs. The RTTs for a specific connection chain are supposed to be the same under ideal network traffic, which is a network without queuing, processing, forwarding delay, but with propagation delay; these are called ideal-RTTs. Actually, there is no way to get an ideal-RTT for a connection chain because we cannot make such a connection chain under ideal network traffic in a real world application. Each packet sent must meet more or less traffic during its propagation time, making different send packets have different RTTs. The lighter the network traffic, the closer to the ideal-RTT for each send packet RTT. We use statistical knowledge to process RTTs, and we found that the occurrence of RTTs approximately follow a Poisson distribution [Kao96], [Pax95], [Fro94]. There are some gaps that are close to the minimum value, and there are very few gaps that are close to a very high value, while most gaps occur around average network traffic. The Poisson distribution

phenomenon motivated us to think about using the standard deviation of RTTs to describe the network fluctuation.

The standard deviation of RTTs can represent network fluctuation to a certain degree. The standard deviation is computed based on the mean value of a data set. The mean of RTTs indicates that the values around it occur with higher probability. Concerning the network traffic, the mean of RTTs represents the average network traffic. If an RTT is less than the mean, it represents lighter network traffic. Otherwise, if an RTT is higher than the mean, it represents heavier network traffic. So a network fluctuation can be presented by using the standard deviation of the packet RTTs.

However, we still have the problem of using the standard deviation of RTTs to estimate the length of a connection chain. The problem is that we need a unit bar to measure the standard deviation. The standard deviation of RTTs measures the network traffic of a whole chain. How heavy the traffic is depends on the unit bar selected. The best unit is the standard deviation of RTTs to a downstream neighbour host. This is impossible because the echo packets occur at the end of a chain, rather than at the next host. In this dissertation we select the standard deviation of the acknowledgement round-trip times (ack-RTT) to the next host as the unit bar to measure the traffic of a whole chain. This will incur false positive error because the ack-RTT is usually smaller than the echo-RTT for the same host.

Network fluctuation has the characteristics to overcome the biased yardstick problem of Yung's method [Yun02], as well as the step aggregation problem of Yang's step-function method [Yan04]. The biased yardstick problem of Yung's method comes from

the unit used to measure the network traffic. The step aggregation problem is caused by the combination of one long distance connection and one short distance connection, and thus Yang's step-function method is not available because the step of a short distance connection is drowned by the big step of a long distance connection. The reason that the fluctuation-based method can overcome the two problems above is that the variation of RTTs is mainly determined by the connected hosts and the routers in between were as shown by Equation (4.2) whether a connection is long or short.

4.2.2.2 Detecting algorithm

We assume that a connection chain has reached the victim machine and the chain remains the same throughout the data collection phase. The following algorithm can be used to detect a long interactive session.

Fluctuation-based stepping-stone intrusion detection algorithm: Given an outgoing connection,

1) Monitor the connection for a period of time and capture all the TCP/IP send and echo packets and put them into a data set, which is called the data set E.
2) Collect send and ack packets to the downstream neighboring host concurrently with Step 1 and put the packets into the data set A.
3) Match the send and echo packets of E by applying a packet-matching method and compute the echo-RTTs.
4) Compute the ack-RTTs of the send and ack packets in data set A similar to Step 3.
5) Compute the standard deviations σ_E of echo-RTTs and σ_A of ack-RTTs.
6) Output the ratio $R = \sigma_A/\sigma_E$.

The ratio R gives us an indication of how long the connection chain might be. If R is close to one, it is likely that the length is probably just one. On the other hand, if the value is much less than one, the connection chain is probably long and thus an indication of stepping-stone intrusion. It should also be noted that the $1/R$ is an approximation of the length of the connection chain in some degree. We compare our fluctuation-based result with Yung's method based on the average behaviour of the RTTs.

4.2.3 Experimental Results and Comparison

We established the connection chains on the Internet to test the algorithm above. A program was made using Libpcap [LBNL04], [DNWS04] to monitor the connection chains, sniff TCP/IP send, ack and echo packets, match them, and compute the standard deviations of the ack-RTTs and the echo-RTTs, respectively. We designed three experiments, each one concentrating on one objective. The first experiment with a chain containing only one downstream connection was for exploring the error caused by replacing echo-RTTs with ack-RTTs. The second one with a chain composed of several connections was for verifying the fluctuation-based algorithm. The third one with a chain containing one long connection and one short connection was for verifying the ability of this algorithm to solve the step aggregation and the biased yardstick problems.

4.2.3.1 Round-trip time: Echo vs. Ack

Ideally, we should compare the RTT to the victim computer with the RTT to the next closest host. However, there is no echo RTT to the next closest host. Thus, in the algorithm we use the acknowledgement RTT as the approximate value. The first question

we must answer is if this is a reasonable approximation. We set up the connections using OpenSSH on the Internet, as shown in Figure 4.6, where h_1 is a host used to simulate an intruder's host on our campus, Acl08 is a another host on the campus that is monitored, and Mex is a host located in Mexico. The connection between Acl08 and Mex is a long connection so we can be sure of generating some acknowledgement packets. We monitored this connection chain and captured the ack, send and echo packets in Acl08. We computed the standard deviations of ack-RTTs and echo-RTTs, respectively, compared them and found that on average the ratio between them is approximately 0.8~0.99. The result means that it is reasonable to use the standard deviation of the ack-RTTs to replace that of the echo-RTTs.

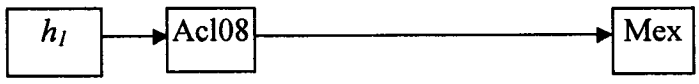


Figure 4.6 A connection chain composing of one short and one long connections

4.2.3.2 The Performance of the algorithm

The main idea of the fluctuation-based stepping-stone intrusion detection algorithm is to estimate the length of a connection chain by computing the ratio between the standard deviation of the ack-RTTs to the next nearest host and that of the echo-RTTs of the whole connection chain. We can test the effectiveness of the algorithm by the following experiment. We collected testing data as we added up to six hosts from our originating machine, as shown in Figure 4.7(a) to (f), where Tms and Bay are two hosts located on campus, and Ela is a host located in California. For each configuration shown, we

repeated the experiments 30 times, and the average results are summarized in Table 4.1. The first column represents the known number of hosts chained.

From the results shown in Table 4.1, we know that we cannot estimate the number of hosts chained by using the algorithm exactly, but we can predict the length of the chain by using the ratio R. The ratio R decreases with an increase in the number of hosts chained. It should be noted that the distances among the hosts vary from very short (campus LAN) to very long (cross-country).

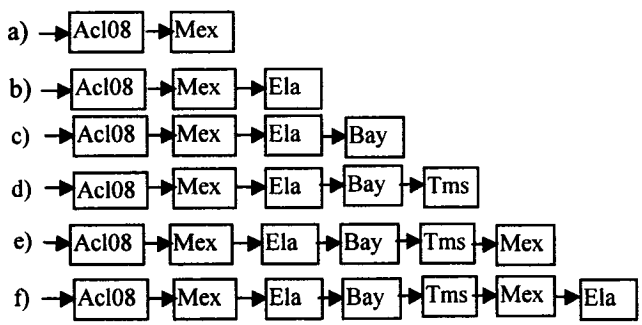


Figure 4.7 Connection chains with one to six hosts compromised downstream from monitor point Acl08

Table 4.1 Network fluctuation estimation with different lengths of a connection chain

Item / Host N	σ_A	σ_E	R
1	2975.82	3011.75	0.99
2	3045.31	4342.18	0.70
3	3153.97	6881.43	0.46
4	2955.33	8477.83	0.35
5	5887.01	28054.68	0.21
6	6739.52	31327.87	0.21

We can compare our result with Yung's result [Yun02] even though the two results were obtained in different experimental conditions. Our method is more monotonic and provides a wider range than Yung's method. When there is one connection in a chain, the two methods almost get the same results. But when the chain contains 6 connections, the result given by Yung's method is 0.57, and by our method this is down to 0.21. Figure 4.8 shows the comparison results, where X axis represents number of hosts, and Y axis represents the ratio for our method, quan(E, 2α) for Yung's method.

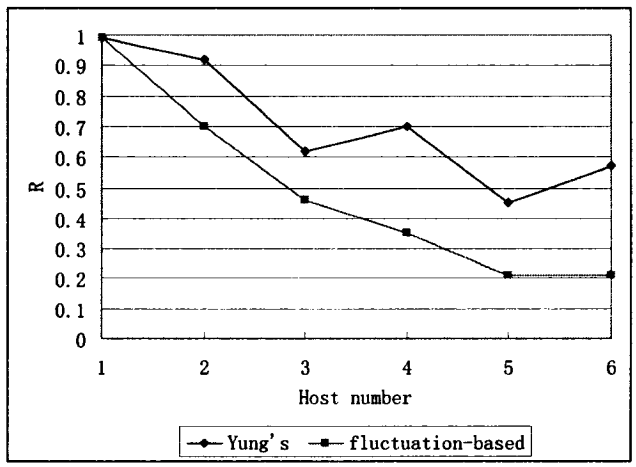


Figure 4.8 Yung's method vs. fluctuation-based algorithm

4.2.3.3 The step aggregation and biased yardstick problems

This experiment was designed to verify if the algorithm can handle the step aggregation and biased yardstick problems. First we built a connection chain on the Internet which contained two long connections as shown in Figure 4.9 a); then we

increased the chain by one more short connection, as shown in Figure 4.9 b), which is the connection between Bay and Tms.

The experimental results in Table 4.2 show that the step aggregation problem can be handled by the fluctuation-based algorithm to a certain degree. When there are two long connections, as in Case 1, the ratio R is 0.66. The only difference Case 2 made is to extend the chain by a short connection. The small step corresponding to the short connection cannot be detected by the step-function [Yan04] method, but can be detected by the fluctuation-based algorithm with a changed σ_E or R. However, this does not mean we have solved the problem, because only comparing with the ratio of the original two long connections, we know R is changed. The fact is that there is nothing to refer to when we get a ratio R during a practical detection. The result tells us that it is possible to solve the step aggregation problem by applying the fluctuation-based algorithm.

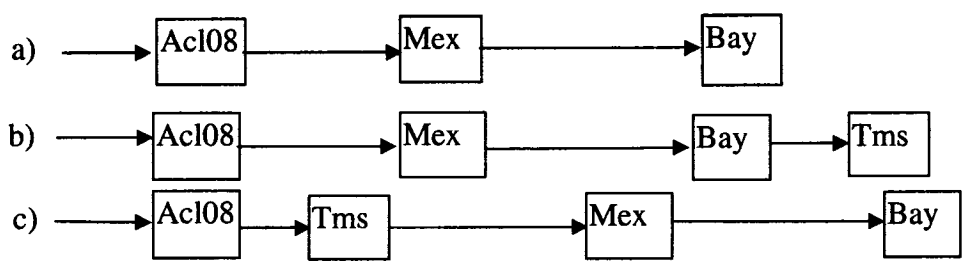


Figure 4.9 Connection chains used in step aggregation and biased yardstick problems

Since we use the next connection as the basis of comparison (yardstick), it is very important that we find out how dependable R is on the distance to the next host. The biased yardstick problem is not as serious with the fluctuation-based algorithm, as the

results show in Table 4.2. We set up an experiment as shown in Figure 4.9 c) to verify the possibility that the biased yardstick problem can be solved by the fluctuation-based algorithm. In Case 3, we built a connection chain on the Internet containing three downstream connections from the monitoring point. The first connection between Acl08 and Tms was short because the two hosts were all located on the campus, and the other two connections were relatively long. The experimental result shows in the 'short + long + long (S-L-L)' case that R is 0.40 which is enough to drive us to predict that the chain is probably longer than one or two.

Table 4.2 Network fluctuation estimation with a chain containing long and short connections

Cases \ Item	σ_A	σ_E	R
1. L-L	2323.48	3531.69	0.66
2. L-L-S	2235.75	5319.30	0.42
3. S-L-L	2765.97	6823.10	0.40

4.3 Issues on TCP Packet Matching

Suppose a user logs in from host h_1, and connects to host h_n through host h_2, ... , and host h_{n-1}, where host h_i is the node where we can put our program to capture the packets going through it, one port of host h_i, which connects to port 22 (OpenSSH) of host h_{i+1}, is monitored. If there is a packet sent from host h_i to host h_{i+1}, most probably this packet is going to be acknowledged by host h_{i+1} first, and then echoed by host h_{i+1}, so we are going to have Send, Echo, and Ack packets, but how to match them? In a simple case with only these Send-Ack-Echo packets, the Send packet and the Echo packet would form a pair.

Host h_{i+1} cannot send back its Echo Packet until it receives the Echo packet from its following host, such as host h_{i+2}. The difference of the timestamps between matched Send and Echo packets thus represents the round-trip time between host h_i and destination host h_n. If there are no other packets between the Send and the Echo packets, it is very easy to match them. Unfortunately, most cases are much more complicated.

In [Yan02], Send-Ack-Echo triples do not intersect with each other because the experiment was done in a local area network with very small RTTs. When the RTT increased to a certain level, the Send-Ack-Echo started to intersect with each other because the TCP protocol allows multiple packets to be sent before they are acknowledged.

OpenSSH is a tool that uses SSH as a protocol for secured remote login and other secure network services over an insecure network [Ylo04-1]. Most hackers like to use OpenSSH to connect several hosts to form a chain. There is three-stage process of data transport: synchronization, data transfer, and connection closed; we are interested in the phase of data transfer. Flow control and congestion control are used to guarantee that the data transfer is reliable, but there are still several issues making data transfer complicated: (1) lost packet retransmission, (2) packet cumulative acknowledgement and echo, (3) session transmit window, (4) packet communication between adjacent hosts (such as ignore packets, keep-alive packets sent from client side, and key re-exchange; these data are not intended for the target machine), and (5) multiple Echo packets from server side [Yl096], [Ylo04-1], [Cla03], [Ylo04-2].

Any lost packets during transmission are retransmitted either automatically by the sending client having not received an acknowledgement or on request of the receiving server. Retransmission of the same packet continues until either an acknowledgement is received or the connection timeout expires. So we are faced with the situation that one Echo packet could match two or more Send packets.

Not every TCP packet is individually acknowledged; sometimes, cumulative acknowledgement may take place. This has several advantages: one of the most important of these is to reduce the number of Ack messages, thereby reducing the possibility of network congestion. This network control mechanism benefits the network traffic, but makes one-to-one packet matching impossible; a similar problem occurs on Echo packets as well.

For data flow control and congestion control, TCP maintains a transmit window, the size of which determines how many unacknowledged octets of data the transmitter is allowed to send before it must cease transmission and wait for acknowledgement. Thus, if this size is set to one, it means that each packet is sent if and only if the previous Send packet has been acknowledged or echoed. Most probably this size is not one, so more packets can be allowed to send continuously without receiving any Ack packet, and more Send-Ack-Echo triples overlap each other, making packet matching difficult.

An Ignore packet is a very special type of packet. The only time this kind of packet is used is as an additional protection measure against advanced traffic analysis techniques [Cla03], [Ylo04-2]. If the server side receives an Ignore packet, it only acknowledges

this packet without any other action, so if we do not process the ignore packet well, it would affect all the packets matching after the ignore packet.

Keep-alive and key re-exchange are packets that are different from the previous cases. They do not affect packet matching, but the packets matched in these two cases are not what we expect because the packets are only sent to the neighbor host of the connection chain, and not the destination host. For most of a session, the key used for encryption is not changed, but it might be changed during the data transfer. It is recommended [Ylo04-2] that the key might be changed and exchanged after each gigabyte of transmitted data or after each hour of connect time, whichever comes sooner. In this case, there would be an extra packet sent to the neighbor host, rather than the destination host. Thus, there is an echoed packet coming from the downstream neighbor host, and also there is a packet pair in which the packets are matched. This matched packet pair is not what we expect because it only indicates the RTT to the nearest neighboring host, rather than a RTT to the target machine. It is easy to remove them by setting a filter or a clustering packet matching approach. When a command is executed at the target host, the result may be sent back in more packets. Consequently, this further complicates the packet matching.

4.4 Summary

We have discussed the approaches to detect stepping-stone intrusion. Unlike the methods to detect stepping-stones, the basic idea of the approaches to detect stepping-stone intrusion is to estimate the downstream length. We have proposed two methods to estimate the downstream length: the step-function and the network fluctuation-based approaches. The common issue with these approaches is to match each Send with an

Echo. In other words, all these approaches need to compute the RTTs for all the Sends captured. The accuracy of matching TCP packets or computing RTTs directly affects the accuracy of detecting intrusion. So developing efficient and accurate methods to match TCP packets (computing RTTs) is significant for detecting intrusion.

As we discussed in section 4.1.1, matching TCP/IP packets, especially Sends and Echoes, is not trivial. We have developed three approaches to match packets or compute RTTs: 1) TCP/IP protocol-based matching approach; 2) Clustering-partitioning matching approach; and 3) Standard deviation-based matching approach. In Chapter 5, we discuss these in detail.

5. TCP PACKET MATCHING

In this chapter, we focus on discussing TCP packet matching, which is a very important issue in stepping-stone intrusion detection and connection traceback. First we talk about the method to match packets based on TCP/IP protocol, the details of which have been published [Yan05]. Second, we discuss an interesting method which exploits data mining technology to match TCP/IP packets. The detailed results have also been published [Yan06]. Third, we present a standard deviation-based method to match TCP/IP packets. These research results have also been published [Yan05-1].

5.1 TCP/IP Protocol-based Matching Approach

The problems of packet matching are inherited from the fact that the Send and Echo packets may be a many-to-many relationship, not one-to-one. It is impossible to match them deterministically even with a complete log. Therefore, it is much more difficult to match in real–time. To be noted is that if we make a mistake in packet matching at one point of a packet stream, the mistake would affect all the packet matches after that point. To prevent this kind of mistake from occurring, our policy is to limit the possible mistakes in a certain range. By dividing a packet stream into sub-streams, which is the scope in which we match the packets. Thus, if we make a mistake, it would only affect the packet matching within that sub-stream. Another benefit of dividing a packet stream into sub-streams is that we are relatively confident that in each sub-stream, there is one and at least one matched packet pair. The only thing left is how to divide a packet stream into sub-streams online. Here we use the fact that an intruder would need to think about

and pause prior to each step; consequently, there exists a gap, which is more than a predefined threshold, between the keystroke right before the thinking-pause-point and the keystroke right after that point. The Conservative matching algorithm only matches the packets that we are relatively sure about their matching correctness in each sub-stream, while it discards the others. Otherwise, once we are not sure about the packet matching correctness, we use the chronicle, sequence number, and size of a packet in a stream to decide packet matching, using the Greedy matching algorithm.

5.1.1 The Conservative Algorithm

Each (Send or Echo) packet propagated between two hosts A and B carries a sequence number and an acknowledgement (receive sequence) number. The initial send sequence number (ISS) is chosen by the data sending TCP, and the initial receive sequence number (IRS) is obtained during the connection establishing procedure [RFC81].

In data communication, the sender of data keeps track of the next sequence number to use in the variable SND.NXT. The receiver of data keeps track of the next sequence number to expect in the variable RCV.NXT. The sender of data keeps track of the oldest unacknowledged sequence number in the variable SND.UNA. If the data flow is momentarily idle and all data sent have been acknowledged, then the three variables will be equal. When the sender creates a segment and transmits it the sender advances SND.NXT. When the receiver accepts a segment it advances RCV.NXT and sends an acknowledgment. When the data sender receives an acknowledgment it advances SND.UNA. The extent to which the values of these variables differ is a measure of the delay in the communication [RFC81].

Though we stated many challenges in matching all TCP packets, we do not have to match 100% of these packets in order to detect a new connection chain. If we can match a significant portion of the packets, it is sufficient for the purpose of intrusion detecting. There are two choices: (1) match only those for which we are confident of correctness; or (2) include some where we are not completely sure of their correctness. In the first algorithm, we collect only the matches that we are truly confident about their correctness and sacrifice on the matching-rate.

During an interactive terminal session, send packets are normally partitioned into several segments with some time gaps between the segments, corresponding to the user's pause in reading and responding to the previous operation. These gaps, caused by human interaction, are normally measured in seconds, and are considerably larger than a typical network round-trip time. It is safe to assume that no echo packet will match a send packet across segment gaps. If an interval between two consecutive send packets is more than *TG* (a time gap threshold), we will assume the existence of a gap. In our experiments, *TG* is set to 1 second, which works well.

Suppose a group of Send packets corresponds to a group of Echo packets, and if there is only one outstanding Send when an Echo is received, it is easy to match them. However, if there is more than one unmatched Send packet when an Echo packet is received, we can only match the Echo packet with the first Send, and conditions (5.1a) and (5.1b) are used to ensure that only the first Send is matched:

$$\text{Send.Ack} = \text{Echo.Seq} \tag{5.1a}$$

$$\text{Send.Seq} < \text{Echo.Ack} \tag{5.1b}$$

Condition (5.1a) indicates that the receiver's sequence number does not change, i. e., the Echo is the first echo packet since the first Send packet, and condition (5.1b) means that this Echo packet also acknowledges or echoes the Send packets, the sequence number of which is less than the acknowledgement number of this Echo packet.

```
Initialize a SendQ queue;
CorrectMatch = true;   //Clear match flag
while (there are more packets) {
   Capture the next packet P;
   if P is a Send packet {
      Compute Time Gaps TG since last Send;
      if(TG > Threshold){
       Reset the SendQ;
       CorrectMatch = true;
    } else {add P to SendQ;}
   } else if P is an Ack packet{// Ignore it
   } else if P is an Echo packet{
      Q = dequeue (SendQ);
      if ((Q.ack# = P.seq#) and (Q.seq# < P.ack#)
            and (CorrectMatch)){
         Packets P and Q are matched;
         Compute round-trip time between P and Q;
      } else { // No match, set confusing match flag
         CorrectMatch = false;
      }
   }
}
```

Figure 5.1 Conservative matching algorithm

We designed the algorithm shown in Figure 5.1 to match TCP packets precisely based on segment gap and conditions (5.1a) and (5.1b). The problem with this algorithm is that its matching-rate is low because some packets that are supposed to be matched are discarded due to the strict matching conditions. Therefore, this method is called the Conservative algorithm. Once a proper matching is not determined, the queue of Send packets (by using the Boolean variable *CorrectMatch* in the algorithm) will be cleared and packets are discarded. The experimental results showed that the closer to victim, the

more packets are discarded, and the lower the matching-rate, but the matching-accuracy is high. There is a trade off between the matching-rate and the matching-accuracy.

5.1.2 The Greedy Algorithm

The algorithm shown in Figure 5.2 can increase the matching-rate with a little cost of sacrificing packet matching-accuracy. This algorithm is called the Greedy algorithm because it is not necessary to get high packet matching-accuracy in terms of detecting intrusions. We fully understand that the Greedy algorithm may include a few matches which may not be correct; however, whenever we had doubts we erred on the high end, i. e., a higher RTT. Many-to-many mode is the key part that makes packet matching-rate low, so we are going to start the discussion here.

```
Initialize a SendQ queue;
while (there are more packets) {
   Capture the next packet P;
   if P is a Send packet {
      Compute Time Gap TG;
      If (TG > Threshold){ Reset the SendQ;}
      else {add P to SendQ;}
   } else if P is an Ack packet{
        // Ignore it
   } else if P is an Echo packet{
      Q = dequeue (SendQ);
      if ((Q.ack# = P.seq#) and (Q.seq# < P.ack#)){
         Packets P and Q are matched;
         Compute round-trip time between P and Q;
      } else if(((Q.ack# =< P.seq#)
            and (Q.seq# < P.ack#)){
         Packets P and Q are matched;
         Compute RTT between P and Q;
      } else {//No match;}
   } else {Return;}
}
```

Figure 5.2 Greedy matching algorithm

To illustrate the difficulties of many-to-many matching of Send and Echo packets, we use the following example:

```
Send P1
Ack P1
Send P2
Ack P2
Send P3
Ack P3
Echo Q1
Ack Q1
Echo Q2
Ack Q2
```

There are three packets sent and two Echo packets received. The first Echo contains some of the characters in the first Send, but we are not sure whether the second Echo should be matched with the first Send. In this case, the matching-rate would be 33% with Algorithm 1. There are several matching possibilities in this case:

1) Q1 matches P1, P2, P3, Q2 matches nothing;
2) Q1 matches P1, P2, Q2 matches P3; and
3) Q1 matches P1, Q2 matches P2, P3.

From these three possibilities, we are confident that Q1 must match P1; Q2 has four choices, matching nothing, P2, P3, or P2 and P3 together. Our greedy policy is to select Q2 to match the very first one that is P2, and if the fact is that Q2 matches P3, the only problem with this greedy policy is that we get a slightly higher value of RTT, which is

not harmful to the purpose of detecting stepping-stone intrusions. The Greedy matching algorithm can get the matching-rate to 66% or 100% for the case above. The major difference between the Conservative and the Greedy matching algorithms is the removal of the Boolean variable *CorrectMatch*. The policy of matching the second Echo with the first remaining Send will result in the general increasing of RTT.

5.1.3 Comparison of the Two Algorithms

It has been demonstrated that the matching results of the Conservative matching algorithm are correct, but the same thing cannot be claimed for the Greedy algorithm. The Conservative algorithm has a high matching-accuracy, but suffers a low matching-rate. In a case for which we would not collect enough data points, we should use the Greedy algorithm. In this section, we will show that for the purpose of determining the connection chain, the two algorithms will produce the same result. To test the two algorithms, we designed an experiment with both programs running on the same host concurrently.

In this experiment, our connection chain spans from Houston to Mexico and California UH1→Mex→UH2→Epic→UH3→Mex, where UH1, UH2, UH3 are hosts located in Houston, Texas, Mex is a host located in Mexico, and Epic is a host located in California. From the experiment, we confirm that (1) the Greedy algorithm produces significantly more RTTs, and (2) all pairs matched in the Conservative algorithm are also matched in the Greedy algorithm. The matching-rate (*MR*, the percentage of Send packets matched) depends on the number of hosts and the RTT, and so is the matching-accuracy (*MA*), which is defined as the percentage of correct matches in those matched packets. Table 5.1

shows the *MR*s and *MA*s of a typical experiment with five connections in the chain. As the RTT increase, the *MR* dropped from 100% to about 21% for the Conservative algorithm, but for the Greedy algorithm it dropped a little. A similar conclusion can be made for matching-accuracy.

Table 5.1 Comparison between the Conservative and the Greedy algorithms in terms of matching-rate and matching-accuracy

	RTT (microsecond)	The Conservative *MR* (%)/*MA* (%)	The Greedy *MR* (%)/*MA* (%)
1	61,000	100.0/100.0	100.0/98.0
2	120,000	70.0/70.0	100.0/94.1
3	172,000	38.1/38.1	100.0/90.3
4	222,000	27.5/27.5	98.6/85.1
5	282,000	21.6/21.6	96.0/79.5

5.2 Clustering-partitioning Matching Approach

Either the Conservative algorithm or the Greedy algorithm has a tradeoff between the packet matching-rate and matching-accuracy. They cannot get both high matching-rate and high matching-accuracy. The problem is that they always search for a 'candidate' Echo packet locally, rather than globally, when they try to match a Send packet.

Unlike the policy used in the Conservative and the Greedy algorithms, the novel approach is to use a data clustering and partitioning technique to match TCP/IP packets and to find the RTTs of the packets of a connection chain. Previous methods to match send and echo packets have worked locally, i. e., compared one packet at a time. Our new approach is a global approach in which we look at all the packets together to determine TCP packet matches. We capture all the send and echo packets of a connection

chain in a certain time interval and compute the difference between each send packet and all echo packets received after it. We can be sure that the correct round-trip time will be among these differences. Based on this observation, our approach is to find the subset that truly represents the round-trip times.

We also know that these round-trip times will cluster around several levels. We use the maximum-minimum distance clustering algorithm (MMD) to find the real RTTs information and determine the number of connections in a chain. The experimental result showed that this algorithm can match TCP/IP packets with both high matching-rate and high matching-accuracy. The mechanism to match packets of the algorithm is to use the distribution of RTTs. Before we discuss the detailed packet matching algorithm, we discuss the distribution of RTTs.

5.2.1 Distribution of TCP/IP Packet Round-trip Time

A packet RTT for a connection is the sum of processing delay, queuing delay, transmission delay and propagation delay of the connection [Li98]. We assume there is one connection established by using software such as OpenSSH [Ylo04-1], [Ylo04-2] between host h_i and h_{i+1}, where h_i is the client side and h_{i+1} is the server side. Even though there may be many hops in between, we can model the delay as a queue for simplicity. Letting $T(t)$ represent the RTT of the connection, it can be expressed as the following form:

$$T(t) = T_0 + \Delta T(t), \tag{5.2}$$

where T_0 is the constant part, and $\Delta T(t)$ is the varying part. Most of the contributions of T_0 come from propagation delay, and most of the contributions of $\Delta T(t)$ mainly come from the queuing delay [Li98]. If */M/M/1* queuing model is applied to the queue [Ber92], the distribution of $\Delta T(t)$ can be modeled as

$$\begin{aligned} P(\Delta T > x) &= \lim_{t\to\infty} P(\Delta T(t) > x) \\ &= e^{(-\gamma_i x)}, \quad \gamma_i = \mu_i(1-\rho_i) \end{aligned} \quad , \tag{5.3}$$

where μ_i is the service rate on link *i*, and ρ_i is the corresponding utilization factor.

The research above indicates that the process of the variance of RTT for one connection can be approximately simulated by an exponential distribution, even though some researchers, such as [Li98], have pointed that ΔT decays more slowly than an exponential rate. The variation ΔRTT of the RTT for a whole connection chain is the sum of ΔT for each connection. Due to the delay of all hosts in between, the variance of RTT for one chain cannot be simulated by exponential distribution. Through experiments, we found that the occurrences of ΔRTT for a connection chain are more like a Poisson distribution [Pax95], [Kao96], or the intervals ΔRTT can be simulated as a Gamma distribution.

It does make sense to use the Poisson distribution to model the variance of RTTs for a connection chain. From the analysis above we know if there is only one packet propagating on the Internet, the RTT of this packet is an ideal one (i-RTT), which is the one in which ΔRTT is zero. The fact is that ΔRTT cannot be zero in reality. But if the Internet traffic were lighter, the RTT would be closer to i-RTT. On the other hand, if the Internet traffic is heavy, the RTT should be far away from i-RTT. On average the RTTs

of a connection chain are supposed to be around an average value, which is the expectation of a Poisson process. From Equation (5.2), we know that the RTTs of a connection chain share the same distribution as its variation ΔRTT. Figure 5.3 shows one of the experimental results of RTTs distribution for a connection chain with four hosts, in which the Y axis stands for the probability that each RTT occurred, and the X axis stands for the RTT value in microseconds (μs). From this experiment we know that most RTTs concentrate around RTT=138,500 μs, with more than 95% of the RTTs being between 137,000 μs and 141,000 μs.

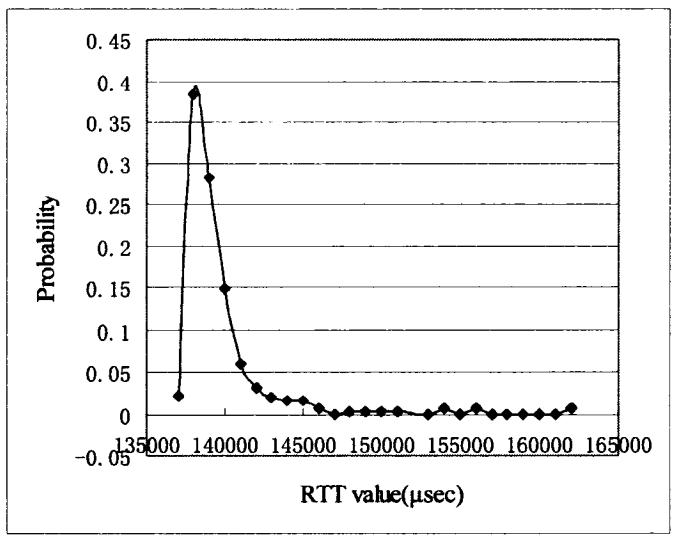


Figure 5.3 Occurrence distribution of RTTs for a connection chain

5.2.2 Clustering-partitioning Algorithm

Suppose we monitor and capture the TCP packets of a chain from the time the chain is being established to the eventual time when the chain has four connections. At the time when the chain has only one connection, based on the analysis of the distribution of the

RTTs of TCP packets in Section 5.2.1, most of its RTTs should be around RTT_1, which is the average value of the RTTs of the chain. Similarly, if the chain is extended incrementally until it has four connections, we have RTTs concentrating around RTT_2, RTT_3, and RTT_4, respectively. This result was first observed in [Yan04]. This clustering of RTTs at different levels will be used to identify the RTTs.

The second observation that will help us with this algorithm is the disjointed partitioning of the RTTs at the different clusters. If we identify the RTTs by the indices of the "send" packets, we will see that these indices are partitioned into four subsets, one for each cluster. Furthermore, each subset is an interval of the form $[i, i+1, .., j]$ inclusively. By combining these two observations, clustering and partitioning, we can compute the RTTs of a connection chain.

There have been many data clustering methods proposed in recent years, such as distance-function classifiers, minimum-distance classifiers, statistical classifiers, fuzzy classifiers, syntactic classifiers, and neural nets classifiers [Fri99], [Mir96], [Jai88]. The special properties of the data set coming from send and echo packet timestamps determine that the maximum-minimum distance clustering (MMD) algorithm is most suitable for our purpose. We have no idea about the final clustering results and there is no *a priori* knowledge about the number of classes. The threshold in MMD is only determined if a new cluster is created, rather than if an element is added. Once a cluster center is fixed, it does not shift during the clustering process.

There are other properties that make the MMD work differently. First, it is easy to get the maximum value, which is the gap between the last echo packet and the first send

packet; the minimum value is the gap between the first send packet and the first echo packet after it. This makes our algorithm more efficient. Second, there are no two elements that come from the same send data set if we assume that each data set corresponds to one send packet. Third, if two elements in a cluster have the same echo packet order number, we will keep the one with the smaller send packet order number. Fourth, if we compute the range (defined later) of each cluster, the ranges of all the real RTT clusters are supposed to be disjoint and the union of the ranges is supposed to equal to the range of the send packet set.

Another important useful property is the consecution of the send packet in an RTT cluster. Most send packets in an RTT cluster are almost consecutive. Here 'almost' means that not one 100% of all real RTT elements are classified into one cluster because from the discussion above and the RTT distribution example we know that only 95% of the RTT elements are around its expectation value.

Given n consecutive send packets $S=\{s_1,s_2,\ldots,s_n\}$ and m consecutive echo packets $E=\{e_1,e_2,\ldots,e_m\}$, we assume that these packets are captured in one connection of a host at a certain time interval, and each packet in send set S is echoed by one or more packets in echo set E. We compute the difference between each send packet in S and all the echo packets in E. Since RTT must be positive, we can safely eliminate the negative values. We group these differences in sets according to each send packet in S, forming data sets $S_1, S_2, \ldots, S_n$ where $S_i=\{t(i,1), t(i,2), \ldots, t(i,m)\}$ for $i = 1, \ldots, n$, and $X = \cup S_i = \{t(i,j), 1 \le i \le n, 1 \le j \le m\}$ where $t(i,j) = e_j - s_i$ represents a potential RTT between the ith send packet and the jth echo packet.

Step 1 of our algorithm is simply a clustering algorithm with a predefined threshold *th*. The MMD algorithm can result in *v* clusters. We shall assume these clusters are sorted in the increasing order of the first index *i* which is the send packet index. The **range** of a cluster *C* is defined as the maximum *i*-index of the elements in *C* minus the minimum *i*-index plus one. In Step 2, we filter out duplicated elements either with the same send packet or with the same echo packet in each cluster. The preference is given to a smaller RTT.

In Step 3, we measure the likelihood of a cluster truly representing a level in the RTTs. A true RTT cluster should have elements representing consecutive send packets with very few exceptions. So we first define a subset of a cluster containing "connected" elements, i. e., elements having neighbors with a distance of *g*. A typical value for *g* is 2 (allowing one missing send packet). "Disconnected" elements are mostly not part of this cluster.

Steps 4 and 5 are used to select the clusters that have very high likelihood of true RTTs. Based on the Chebyshev inequality there are very few clusters outside two standard deviations of the mean. A true RTT cluster is a partition of all send packets satisfying the following two conditions: (1) all clusters are mutually disjoint, and (2) the union of all clusters is equal to the whole send packet set. In reality, condition (1) is easy to satisfy, but it is very difficult to make the union of the clusters exactly equal to the send packet set S. Thus, in our algorithm, we find the clusters whose union has the largest distribution in S.

Clustering-Partitioning Algorithm (X, th, g):

Begin

1. Call MMD(X, th), output clusters $C_1, C_2, \ldots, C_v$.
2. For each cluster C, (1) if $t(i,j)$, $t(i,k) \in C$, and $j < k$, delete $t(i,k)$, and (2) if $t(i,j)$, $t(k,j) \in C$, and $i < k$, delete $t(i,j)$.
3. For each cluster C, compute the clustering ratio $r_i = |C^n|/\text{range}(C)$, where $C^n = \{t(i,j) \in C \mid \exists\, t(p,q)\ \forall C$, and $|i-p| \leq g\}$.
4. Select clusters $D_1, \ldots, D_s$ from the clusters $C_1, C_2, \ldots, C_v$. They have significantly higher ratios among $R=\{r_1, \ldots, r_v\}$. We consider r_i to be significantly higher than the rest of the values if it is two standard deviations higher than the mean of R.
5. Find among $D_1, \ldots, D_u$, a maximum disjoint subset. If there is a tie, select the subset whose union is the largest.
6. Output the number of sets in the subset selected in the last step as the number of connections.

End

The detailed MMD algorithm can be found in [Fri99], [Mir96], and [Jai88]. It first finds all the cluster centers, and then adds all the elements to each cluster. Its cluster results depend on its first element, rather than on the input elements' order. If we assume $m=n$, the time complexity of MMD is $O(n^2v^2)$ with the worst case being $O(n^4)$. The MMD algorithm still has many computations. The main reason is that data set X has many elements which are $m*n$. In fact, we can reduce this number to make MMD more efficient.

It is not necessary to compute the gaps between each send and each of all the echo packets. If we assume that the correct echo packet should be returned within a window of w echo packets, we can compute only w difference of $t(i,j)$. Based on our experiment, a window of size $3*(m/n)$ works well. In Section 5.2.3.3, we verify experimentally whether the cluster results vary while w is changed under fixed threshold th.

5.2.3 The Performance of the Algorithm

We will verify three issues by experiments. The first is to verify if the number of clusters equals the number of the hosts (connections) and if the elements in each cluster represent the correct RTTs. As a test set in this experiment, we use one set of data for which we know the correct RTTs. The second experiment compares the levels estimated by the clustering-partitioning algorithm with the levels estimated by the previous algorithms. The experiments were repeated hundreds of times but only a selected (typical) result is presented below. The third experiment is to study the effect of the window size w on the cluster results by using the clustering-partitioning algorithm.

5.2.3.1 Effectiveness of the clustering-partitioning algorithm

The first experiment is to verify if the algorithm works in detecting the number of hosts in a connection chain. We built a connection chain using Open SSH that passed through host A to host M, and to hosts T, M, B and M; all these hosts were located in Texas except host M which was located in Mexico. We monitored the connection chain and captured the send and echo packets at host A from the time that host M was first connected to the time that the whole connection chain was built. In this experiment we

already knew that the chain had five connections, with each connection corresponding to one host. Our purpose was to verify whether we could use the clustering-partitioning algorithm to get five disjoint clusters. Figure 5.4 shows the clustering results, where the X axis represents each cluster center in milliseconds, and the Y axis represents the cluster ratio *r* of each cluster.

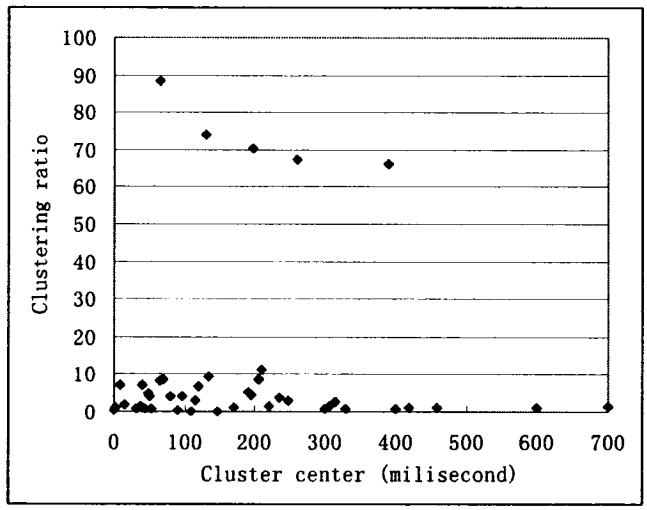


Figure 5.4 Clustering-partitioning results of the algorithm

There were about 720 send and 810 echo packets captured. Under the conditions w=5, g=1, th=0.05, the clustering algorithm got 82 clusters with 56 non-zero clusters. We found that most of the cluster's r values were less than 10 percent except five of them that were significantly higher. We used the clustering-partitioning algorithm to get the five clusters. They are the clusters with centers around 65 *ms* (r = 88.46%), 130 *ms* (r = 73.42%), 197 *ms* (r = 70.28%), 260 *ms* (r = 67.15%), and 392 *ms* (r = 66.74%), respectively. We could predict that those five clusters correspond to the five hosts compromised. This shows how the algorithm works in matching TCP/IP packets and detecting a long connection chain.

The second experiment was to verify whether the elements in each cluster from the algorithm are real RTTs. We built a connection chain that had four hosts connected after host A and monitored the outgoing connection at host *A*, capturing the send and echo packets only when the chain had been established. We controlled our typing speed to be as slow as possible so as to get the send-echo pairs without overlap. It is possible to compute the correct RTT by matching each send packet with its nearest echo packet. We also used the clustering-partitioning algorithm to process the same send and echo set to get the cluster results which are supposed to correspond the RTTs of the chain. We compared the cluster results with the correct RTTs to see how many send packets were missed by using the clustering-partitioning algorithm. We repeated this experiment ten times; on average, 1.37% send packets were incorrectly matched with the parameters set at w=5, g=1, and th=0.05. The results depend on the selection of the parameters used. If we tuned up the parameters, we would get a better result. From the comparison results we know that most of the real RTTs can be clustered by the clustering-partitioning algorithm.

5.2.3.2 Comparison with the previous methods

This third experiment was to compare the performance of the clustering-partitioning algorithm with the Conservative and the Greedy algorithms. We built a connection chain that was the same as that used in Section 5.2.3.1. We monitored the chain at host A using the Conservative and the Greedy algorithms [Yan05] from the time that the chain had only one connection to the time that the chain had four connections. In the meantime, we captured the send and echo packets of the chain at host *A* and used the clustering-partitioning algorithm to get four clusters. The experimental results are summarized in

Figure 5.5, where the X axis represents the send packet index number, and the Y axis represents the RTT value in *ms*.

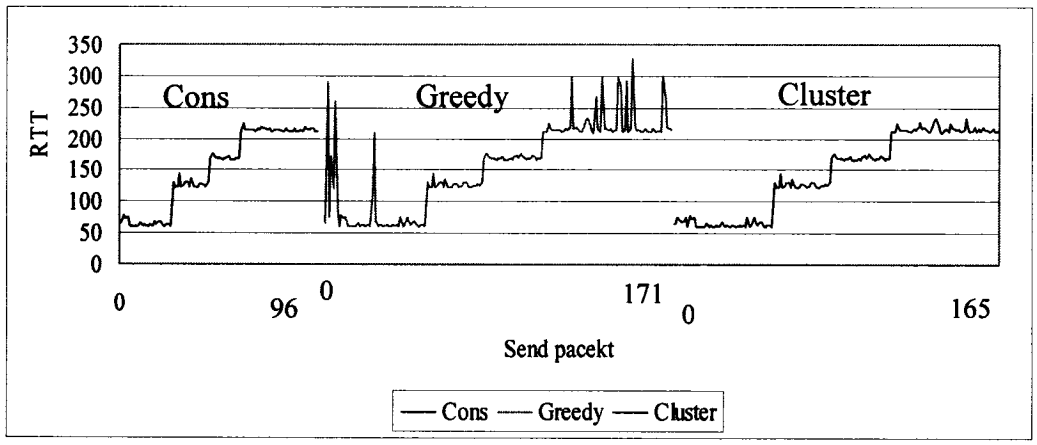


Figure 5.5 Performance comparison among Conserve, Greedy and Clustering algorithms

From Figure 5.5 we know that the three algorithms can derive the same chain length. However, the clustering-partitioning algorithms can get more RTTs than the Conservative algorithm, especially in the second and third levels. If we use the standard deviation of each level to represent its quality, the two algorithms have almost the same quality. The clustering-partitioning algorithm can almost match the same number of send packets as the Greedy algorithm, but it has better quality. In this experiment, we captured 171 send packets. The Conservative, Greedy, and clustering-partitioning algorithms match 96, 171, 165 send packets with standard deviations (4.90, 5.61, 2.52, 2.51), (48.59, 4.79, 2.79, 27.9), and (5.2, 4.79, 2.78, 5.15), respectively (the elements in each bracket correspond to the four levels of each algorithm). Obviously, the clustering-partitioning algorithm combines the advantages of the Conservative and the Greedy algorithms.

Comparing with Yung's method [Yun02], this method can estimate the length of a connection chain more accurately and thus decrease false negatives. This is because the method does not compare one RTT against that of the connection to the next host (a yardstick).

Comparing with Zhang and Paxson's method [Zha00], this method is more stable in resisting intruder's evasion. If a packet random delay were carried out along the chain before a monitoring point, it would be useless, because it does not change the RTTs of the chain from the monitoring point. If it were done after a monitoring point, intruders would have to manipulate the send and echo packets concurrently. This would increase the difficulty for intruders to manipulate a connection by using packet random delay. If a chaff manipulation is used to elude detection, the manipulation must be before a monitoring point. Otherwise, there would be no affection to the RTTs observed from the monitoring point. The chaff must be removed by intruders before it reaches the victim side to avoid affecting a command execution. If a chaff manipulation happened before a monitoring point, it would not affect the RTTs from that point because there are no echo packets for those chaffs but some additional send packets.

5.2.3.3 The effect of the window size

We built a connection chain from a computer on campus to a host M (the remote host), and then connected back to a different host B on campus (the destination host). The send and echo packets were captured at our original host from the time that the first connection was made. We clustered the send and echo packets to get two clusters which corresponded to the two hosts connected from the original host. Different window size w

was used to check its effect on the clustering results. The results are presented in Figure 5.6, where the X axis represents window size w and the Y axis represents the clustering ratio r. No matter what w is, the result that two levels are detected cannot be changed. The ratio r_i for different lengths of the connection chain changes with the varying w.

The two cluster centers are 65,920 μs and 133,352 μs, which correspond to the average RTTs to the remote host (Level-1) and destination host (Level-2) respectively. From Figure 5.6, we notice two points: (1) the clustering ratio is stable after w is two for the remote host, and w is three for the destination host; (2) the clustering ratio for the destination is generally smaller than that for the remote host. The reason that we observe the first point is that when the RTT is bigger, it is easier to generate send-echo pair overlaps, so we need to check more echo packets in the remote host than in the destination host. The reason for the second point is that under the same threshold, the bigger the average RTT, the bigger the standard deviation of the RTTs. So under the same w, the one with the bigger standard deviation has the smaller clustering ratio.

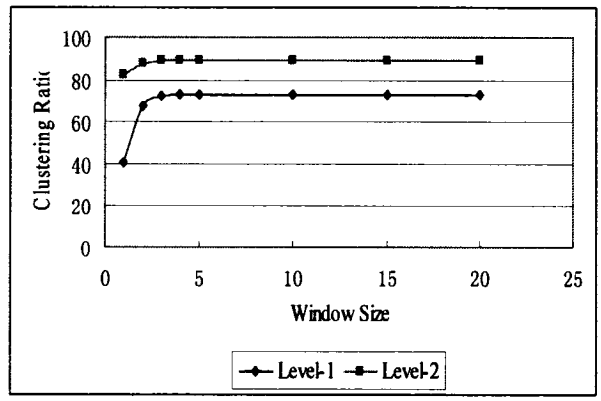


Figure 5.6 Window size vs. clustering ratio

Based on this experiment, we conclude that a window size w has little effect on the clustering ratio when w is greater than 3 or 4. We thus can modify to examine significantly fewer differences in the clustering-partitioning algorithm and make it more efficient.

5.3 Standard Deviation-based Matching Approach

This is the third method we propose to match TCP/IP packets. Unlike the previous one which gets different results with different selected parameters, this algorithm does not depend on any parameters and always gives a unique packet matching result. To understand the complexity of matching TCP/IP packets, we here present one example in which we simply assume that each send packet is only echoed by one responding packet to demonstrate this point. For example, it would be trivial to match each send and its corresponding echo in a packet sequence $\{s_1, e_1, s_2, e_2, s_3, e_3, s_4, e_4\}$ in which s_i (e_j) represents the timestamp of ith (j^{th}) send (echo) packet. Had the sequence become $\{s_1, s_2, s_3, e_1, e_2, s_4, e_3, e_4\}$, the matching result coming from the method in [Yun02] would have been pairs (s_3, e_1), and (s_4, e_3), rather than pairs (s_1, e_1), (s_2, e_2), (s_3, e_3), and (s_4, e_4). To compute RTTs is essentially to match send and echo packets. Yang and Huang [Yan05] proposed two algorithms to match TCP/IP send and echo packets: the Conservative and the Greedy algorithms. The Conservative algorithm can give accurate packet match results but with few matches. The Greedy algorithm could 'match' all the send packets but with some incorrect 'matches'. There is a tradeoff for packet matching-rate and matching-accuracy between these two algorithms.

Matching TCP Sends and Echoes is equivalent to computing TCP packet RTTs, which is challenging and significant in detecting stepping-stone intrusion. Only when we have matched send and echo packets precisely can we detect intrusion accurately, by using the method in [Yun02] or [Yan04]. Here we propose a novel algorithm, the standard deviation-based clustering matching approach (SDC), to compute TCP packet RTTs

more accurately than the methods in [Yan05]. We also evaluate the performance of SDC by estimating the probability of making a correct selection of RTT through Chebyshev inequality. SDC can match most send packets with the same correctness as the Conservative algorithm, and compete against the Greedy algorithm on the number of packets matched but with a higher matching-accuracy. The experimental results showed that SDC can get both high packet matching-rate and matching-accuracy. Compared to the clustering-partitioning packet matching approach, this method can be applied to different contexts.

5.3.1 Motivation

It is easy to match each send packet with its corresponding echo packet when the echo packet is always received before the next packet is sent. This is often the case in a local area network rather than on the Internet. Overlaps of send-echo pairs occur often on the Internet because of network delay and host burden. For efficiency, the TCP/IP protocol allows sending the next packet before the previous one is acknowledged [Ylo04-1], [Ylo04-2]. This might result in more send packets being echoed by one packet, and complicate the packet-matching. There is no marker available to identify each packet on the Internet. The available unencrypted stuff we can use to identify each packet is its size, timestamp, and sequential number. Obviously, a packet size cannot provide any information useful in identifying a packet because it depends on an encryption key. The Conservative and Greedy algorithms [Yan05], which take advantage of the sequential number to match TCP/IP send and echo packets, have the problem that they are not available to get both high matching-rate and matching-accuracy. When there are

unmatched send packets followed by some echo packets, there is no way to know which echo packet matches which send packet except in the case that the first send packet always matches the first echo packet, which is the strategy used by the Conservative algorithm. When there is confusion, the strategy used by the Greedy algorithm is to match each send with each echo with FIFO policy which incurs some errors because in some cases they are not one-one map.

The essential problem of the Conservative and Greedy algorithms is that they only search for the echo packets locally when they try to match each send packet. When checking the echo packets globally, rather than locally, we may have some additional information to identify packet-matching even under the scenario that there are overlaps of send-echo pair. Through long observation, we found that the RTTs of the send packets of a connection chain vary slightly. They are bounded within a narrow range. For a sequence of send and echo packets, if we compute all the gaps between each send and each echo packet, organize the gaps into data sets based on each send packet, and combine the data sets to form clusters by taking one element from each data set, the cluster representing the true RTTs should have the smallest fluctuation among all the clusters.

The strategy of this method is to take advantage of the distribution of RTTs to find packet-matching, or compute the RTTs. We capture $S=\{s_1, s_2, \ldots, s_n\}$, and $E=\{e_1, e_2, \ldots, e_m\}$ from one outgoing connection of a host for a period of time. For any send packet s_i, even though we are not sure which echo packet in E matches s_i, we do know that the matched packet of s_i must be in E. We assume each packet in E has the possibility to

match s_i, except the packets which appear earlier than s_i. We compute the gaps between s_i and each possible packet in E. If we only look at the gaps for s_i, we have no idea which gap represents the true RTT of s_i. But if we take a look at all the gaps for more send packets in S, it would be clear the gaps represent the true RTTs of the send packets. Let us demonstrate this point with an example. For the following two sequences which have four send and echo packets in microseconds, respectively,

S= $\{s_1, s_2, s_3, s_4\}$ = {1099702684, 1099772525, 1099909440, 1099928524},

E= $\{e_1, e_2, e_3, e_4\}$ = {1099828523, 1099898019, 1100036000, 1100058999},

we compute the following data sets based on each send packets in S:

S_1={*125839*, 195335, 333316, 356315} corresponding to s_1;

S_2={55998, *125494*, 263475, 286474} corresponding to s_2;

S_3={-80917, -11421, *126560*, 149559} corresponding to s_3; and

S_4={-100001, -30505, 107476, *130475*} corresponding to s_4.

Checking each data set alone, there is no way to know which element in each data set represents the true RTT except the negatives. If we check all the elements in the four data sets, it is not difficult to know that the italic number in each data set represents the true RTT of the corresponding send packet. Even though we do not know exactly the element in each data set representing the true RTT, we know it must be hidden in each data set. So we make a combination by taking one element from each data set. Obviously, we can get up to 256 combinations from all the elements of the four data sets above. Only the combination with the smallest fluctuation has the highest possibility to represent the true RTTs with the feature of RTTs distribution considered. In this example, we can verify

that the combination composed of the red numbers from each data set has the smallest standard deviation among all 256 combinations. Here we use standard deviation to measure the fluctuation of each combination. Is it always true that the combination with the smallest standard deviation is used to represent the true RTTs? Even though we can not prove this point strictly, we can estimate to what degree it is correct to use the combination with the smallest standard deviation to represent the true RTTs. The algorithm SDC is proposed based on the point above.

5.3.2 Matching Algorithm

Given two sequences $S=\{s_1, s_2, \ldots, s_n\}$ and $E=\{e_1, e_2, \ldots, e_m\}$, if a packet s_i in S is echoed by a packet e_j in E, we denote $s_i \propto e_j$. If all the packets in S and E are captured from an outgoing connection of a host at a specific period of time, the following conditions must be held:

(1) Any send packet in S must be echoed by one or more packets in E; similarly, any echo packet in E must echo one or more send packets in S;

(2) Packets both in S and E are stored in chronological order.

For any two packets s_i, s_j in S and e_p, e_q in E, if $s_i \propto e_p$, $s_j \propto e_q$, and $i < j$, then we must have $p \leq q$.

Condition (1) indicates that the relationship between a send and its corresponding echo packets may be one-one, many-one, or one-many. The RTT of a send packet can be defined as the gap between the timestamp of a send packet and that of its corresponding echo packet if the relationship between them is one-one. However, if the relationship is many-one or one-many, the gap is not unique. If there are k send packets from s_i to s_{i+k-1}

echoed by e_j, the RTT of these send packets is defined as the gap between s_{i+k-1} and e_j. Similarly, if a send packet s_i is echoed by k packets e_j to e_{j+k-1}, only s_i and e_j are involved in the RTT definition of s_i. Condition (2) guarantees that send packets must be replied to sequentially and an RTT must be a positive value. The two conditions above can be justified from the TCP/IP protocol [Ylo04-1], [Ylo04-2].

We computed the gaps between the timestamp of each echo packet in E and those of all the send packets in S. It is safe to eliminate the negative values, as RTT must be positive. We grouped these differences in sets according to each echo packet in E, forming data sets $E_1, E_2, \ldots, E_m$ corresponding to echo packets $e_1, e_2, \ldots, e_m$, respectively:

$E_1=\{s_1e_1, s_2e_1,\ldots, s_ne_1\}$,

$E_2=\{s_1e_2, s_2e_2,\ldots, s_ne_2\}$,

...

$E_m=\{s_1e_m, s_2e_m,\ldots, s_ne_m\}$.

Any element s_ie_j in E_j represents the gap e_j-s_i between timestamp of jth echo packet in E and that of ith send packet in S, $1\leq i\leq n$, $1\leq j\leq m$. For convenience of analysis, we created each data set based on each echo packet in E. They are eventually equivalent to the data sets created based on each send packet in S.

We know that each send packet can be echoed by one or more packets successfully at one time, which indicates that in each data set, such as E_j, there is at most one element to be able to represent the real RTT of the send packet. In the following algorithm, we form a cluster X_u, which is one candidate to be able to represent the true RTTs, by taking one element from each data set E_j ($1\leq j\leq m$). Each cluster X_u ($1\leq u\leq n^m$) has m elements while

some of them may share the same echo or send packet. As we have discussed, we remove the elements which share the same echo (send) packet but keep the one which prefers the smallest gap. Eventually we can get a cluster X_u ($1 \le u \le n^m$) containing different elements related to different send and echo packets. If we go through all the elements in all the data sets and enumerate all the possible combinations, there will be n^m clusters, but the true RTTs can only be represented by one of them. We select the cluster with the smallest standard deviation to represent the true RTTs of the send packets in S among all the clusters. The following is the algorithm SDC:

Algorithm SDC(S, E)

Begin:

(1) Generate data sets E_j ($1 \le j \le m$): $E_j = \{t(i,j) \mid t(i,j) = e_j - s_i,\ i=1,\ldots, n\ \&\ t(i,j)>0\}$

(2) Combine the elements in data sets E_j ($1 \le j \le m$) to form clusters X_u ($1 \le u \le n^m$): $X_u = \{t(i_j,j) \in E_j \mid \forall\, 1 \le j \le m\ \&\ i_1 < i_2 \ldots < i_m\}$

(3) For each cluster X: (a) if $x(i, j), x(i, k) \in X, j<k$, then delete $x(i, k)$, and (b) if $x(i, j), x(k, j) \in X, i<k$, then delete $x(i, j)$

(4) Out $R = \{r_1, r_2, \ldots, r_s\}$ ($1 \le s \le n$) which is the cluster with smallest standard deviation among all clusters X_u ($1 \le u \le n^m$).

End

Let us analyze the time and space complexity of this algorithm. Obviously, the time complexity of SDC in worst case is $O(n^m)$. This will cost much CPU time and make SDC inefficient. The space complexity is not a serious problem for SDC because it is not necessary to memorize all the combinations. The smaller m and n, the better the time

complexity is. There is a way to diminish m and n by dividing a long packet stream of a session into subsections: if the gap between two consecutive send packets, such as s_i, and s_{i+1}, is more than a predefined threshold (usually 1 second), it makes sense to assume all the echo packets after s_{i+1} only reply to the send packets after s_{i+1}. This can make m and n smaller, but it is difficult to define a threshold.

Another factor making SDC inefficient is its global computation, which indicates looking over all the possibilities of X to find the RTTs. Instead of generating all the possibilities of X, we only generate the cluster which would be a candidate of the RTTs. Taking one element of the first data set E_1 as the first element of candidate X, we then check the second data set E_2 to find one element from E_2 to make X have smallest standard deviation when this element is added to X. Similarly, we can select one element from each of the other data sets E_j ($3 \leq j \leq m$) and add them to X to make X hold the smallest standard deviation. Consequently, we generate n candidates (clusters) upon n elements of data set E_1. We select the one with the smallest standard deviation to represent the true RTTs among the n clusters. Suppose there are n elements in E_1 (the worst case) and m data sets; the complexity of this method is $O(n*(n+n)*(m-1))=O(m*n^2)$. Comparing the time complexity of SDC, this one is apparently more efficient. The problem of this method is that we cannot guarantee the correctness of its results because we do not go over all the possible combinations.

5.3.3 Probabilistic Analysis

5.3.3.1 Probabilistic analysis

SDC always selects the cluster with the smallest standard deviation to represent the true RTTs; it does not mean the selected cluster is the correct RTTs. But at least it is reasonable to say that the selected cluster can represent the correct RTTs with a certain probability. In this section, we will estimate the probability to represent the true RTTs by selecting the cluster with the smallest standard deviation. Each RTT is independent from the others. We only need to estimate the probability of making the correct choice of an RTT in cluster R. So we first estimate the probability of making an incorrect choice of an RTT, such as any element r_j in cluster R, which is a cluster with the smallest standard deviation. We assume the distribution of R is Z with mean μ_1 and standard deviation σ_1, and send packet inter-arrival distribution is Y with mean μ_2 and standard deviation σ_2. The element r_j must be selected from $E_j = \{s_1e_j, s_2e_j, \ldots, s_{k-1}e_j, s_ke_j, s_{k+1}e_j, \ldots, s_ne_j\}$ according to the algorithm SDC. We assume the correct selection should be s_ke_j, but another element in E_j is selected. To satisfy the condition that R has the smallest standard deviation, the element in E_j selected incorrectly must be closer to μ_1 than s_ke_j. Only $s_{k-1}e_j$ or $s_{k+1}e_j$ has the highest possibility to satisfy this condition because the elements in E_j are in descending order. Here we assume $s_{k+1}e_j$ is closer to μ_1 than $s_{k-1}e_j$, which means inequality (5.4) is satisfied:

$$\left|s_{k+1}e_j - \mu_1\right| < \left|s_ke_j - \mu_1\right|. \tag{5.4}$$

We assume the smallest inter-arrival of the send packets is L, so the interval between s_k and s_{k+1} should be bigger than L while the worst case is they are equal:

$$s_{k+1} - s_k \geq L = 2q\sigma_1 . \tag{5.5}$$

Here q is a real number that makes (5.5) satisfied. From equality (5.5), for any echo packet e_j, we have,

$$\begin{aligned}
&s_{k+1} - e_j + e_j - s_k \geq 2q\sigma_1 \\
&(e_j - s_k) - (e_j - s_{k+1}) \geq 2q\sigma_1 \\
&s_k e_j - s_{k+1} e_j \geq 2q\sigma_1 \\
&s_k e_j - \mu_1 + \mu_1 - s_{k+1} e_j \geq 2q\sigma_1 \\
&|s_{k+1} e_j - \mu_1| + |s_k e_j - \mu_1| \geq 2q\sigma_1 \ .
\end{aligned} \tag{5.6}$$

From equations (5.4) and (5.6), we have,

$$|s_k e_j - \mu_1| > q\sigma_1 \ .$$

We estimate the probability that r_j is selected incorrectly by using Chebyshev inequality [Kao96], [Fel68] as in the following:

$$\begin{aligned}
P(r_j \ is \ selected \ \ incorrectly) &= P(s_{k+1} e_j \ is \ selected) \\
&= P(|s_{k+1} e_j - \mu_1| < |s_k e_j - \mu_1|) \ . \\
&\leq P(|s_k e_j - \mu_1| > q\sigma_1) < \frac{1}{q^2}
\end{aligned}$$

The probability of making an incorrect selection of a packet RTT is bounded if we select the cluster with the smallest standard deviation among all clusters X_u ($1 \leq u \leq n^m$) to

represent the true RTTs. So it is easy to get the probability of making a correct selection of a packet RTT in cluster R by using inequality (5.7):

$$P(r_j \text{ is selected correctly}) \geq 1 - \frac{1}{q^2}. \tag{5.7}$$

It can be seen that the parameter q really depends on the inter-arrival distribution of the send packets and on the distribution of RTTs.

5.3.3.2 Estimation of parameter q

The parameter q is determined by the smallest inter-arrival L of the distribution Y. The probability of making an incorrect selection of a packet RTT is affected by L. If the inter-arrival between two consecutive send packets is the smallest one of Y, we get the highest probability of a wrong selection of an RTT. The problem is the probability that the inter-arrival between two consecutive send packets taking the smallest value of Y is very small. In practice, instead of using the smallest inter-arrival to estimate the parameter q, we usually use the inter-arrival L_p which makes the cumulative probability $p(x<L_p)$ in Y be 5%, where x represents an inter-arrival between any two consecutive send packets. We estimate L_p upon the assumption that Y follows a Gamma distribution with shape parameter β and scale parameter α as shown in (5.8). The L_p selected must satisfy the following equation:

$$\int_0^{L_p} \frac{(x/\alpha)^{\beta-1} e^{-\frac{x}{\alpha}}}{\alpha\Gamma(\beta)} dx = 0.05 , \tag{5.8}$$

where $\Gamma(\beta) = \int_0^{\infty} e^{-u} u^{\beta-1} du$.

We can compute L_p from Equation (5.8) if β and α are known. Parameters β and α vary upon the keystroke speed and the network environment. The most usual way to estimate L_p is to take a sample of inter-arrivals of send packets to estimate the parameters β and α by using MLE (maximum likelihood estimation) [Joh70], and then compute the L_p for the distribution Y with the parameters β and α. This way is appropriate for individual computation, but is not convenient for probabilistic analysis. Instead of estimating a specific value of β and α, we estimate the range of these two parameters, and thus compute the range of L_p. We use the lower bound of L_p to compute the probability that one element is selected correctly in SDC by using inequality (5.7) because the smaller the L_p, the lower probability one element in data sets E_j ($1 \leq j \leq m$) is selected correctly to represent a true RTT.

We did many experiments with different users and different environments (here it means to build the connection chains on different paths at different times) on the Internet, and we present some typical examples in Table 5.2, where the L_p and α is in microseconds. The experimental results show that the probable range of L_p is from 32000 to 52000 (μs) approximately. It might be impossible to predict the exact range of L_p by merely using some experimental results without further theoretical analysis. But in a sense, the results are significant. In Section 5.3.4.2, we use the lower bound of L_p to compute the probability of making a correct selection of an RTT to evaluate the performance of SDC in finding TCP packet RTTs.

Table 5.2 Send packets minimum inter-arrival estimation

Item / Sample	Size of each sample	β	α	L_p
1	1297	2.043	137280	51115
2	990	1.956	137480	46448
3	816	1.4434	212600	33733
4	900	1.809	143970	40541
5	176	1.426	280220	43016
6	800	1.629	172720	37617
7	412	1.364	242270	32874

5.3.4 Empirical Study

We have proposed that the inter-arrival distribution of TCP send packets of an interactive session can be represented by a Gamma distribution. In this section, we first verify its reasonability. The probability of making a correct selection of an RTT in SDC can be estimated by using inequality (5.7). It is a very important benchmark to evaluate the performance of SDC in finding the true RTTs of send packets of an interactive TCP session. In this section, we secondly compute the probability for some real world examples. To justify the high performance of SDC in finding true RTTs, thirdly, we compare SDC with the best existing algorithm for finding RTTs.

5.3.4.1 Verification of inter-arrival distribution of TCP Sends

We established a TCP interactive session, Acl09→Acl08→H1→H2→H3→H4, in which H1, H2, H3, and H4 are hosts located in the U.S. and Mexico, using OpenSSH. Acl08 and Acl09 are two local hosts. Host Acl08 is the monitoring point where our monitor program resides. We operated at host Acl09 by typing some commands freely

and regularly. We collected all the send packets on the outgoing connection of Acl08, computed the inter-arrivals of these send packets, and used the tools of Matlab to fit their distribution. Before fitting the distribution, we first drew the histogram of these data to see what kind of distribution they showed. We found they are more like a Gamma distribution with a shape parameter bigger than one. Then we used the Matlab distributing fit function to estimate its shape parameter β and scale parameter α.

With these two parameters, we had a theoretical distribution with shape parameter β and scale parameter α. We used the quantile-quantile function of Matlab to verify how well the Gamma distribution fit the example. Figure 5.7 shows the verification result of one typical example, where the X and Y axes have a scale factor 10^5 in microseconds.

In this example, the shape and scale parameters are estimated to be 2.0426, and 137280, respectively.

From Figure 5.7, we find that the points with inter-arrivals more than 400,000 (μs) are not well-fitted by a Gamma distribution where the dashed line indicates an ideal fitting. But the points

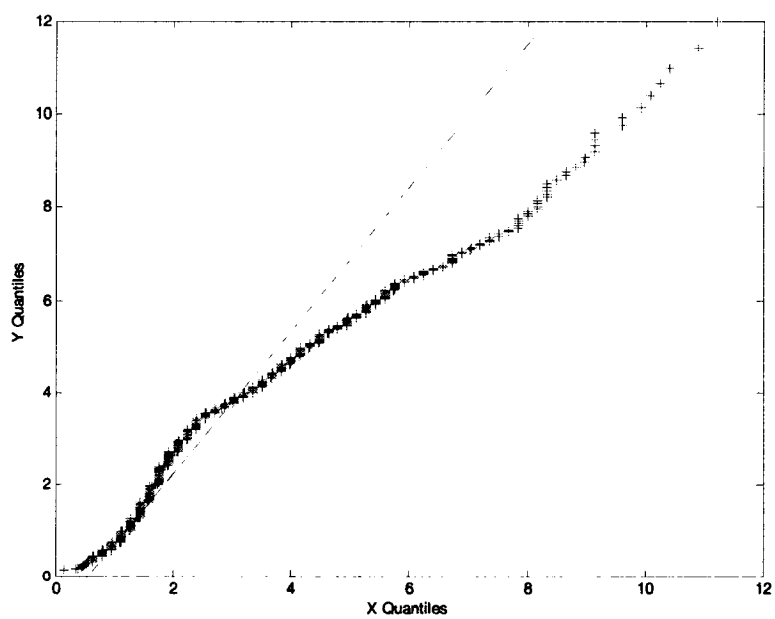


Figure 5.7 Verification of send packets inter-arrival distribution

with inter-arrivals less than 400,000 (μs) are simulated closely by the Gamma distribution with β=2.0426 and α=137280.

5.3.4.2 Probability estimation

The key idea of the algorithm SDC is to select a cluster with the smallest standard deviation to represent the true RTTs. The best way to justify this point is to compare the RTTs obtained from SDC with the correct RTTs to see if they are consistent. The problem is how to compute the correct RTTs. From [Yan05] we know that matching each send with its corresponding echo packet is trivial when there is no overlap of a send-echo pair. So in our first experiment we controlled the keystroke speed so as to generate the scenario without an overlap of a send-echo pair to make it easy to compute the correct RTTs, with which we compare the RTTs coming from SDC to ascertain whether SDC can compute the RTTs correctly. This is one way to justify the performance of SDC under a very special situation. The usual way to evaluate SDC when there are overlaps of send-echo pairs, which occurs often on the Internet where we do not have correct RTTs, is to justify its performance by computing the probability of making a correct selection of RTT.

We established a connection chain similar to the one in Section 5.3.4.1. The students were asked to control their keystroke speed. We collected all the send and echo packets for a period of time at Acl08. First we matched the send and echo packets to compute the correct RTTs, and then used the send and echo packet sequence as the inputs for SDC to find the RTTs. We repeated the experiment many times with one of the comparisons presented in Figure 5.8, where the Y axis represents an RTT value in microseconds and

the X axis represents the RTT index number. These experimental results show that the RTTs from SDC are exactly the same as the correct RTTs.

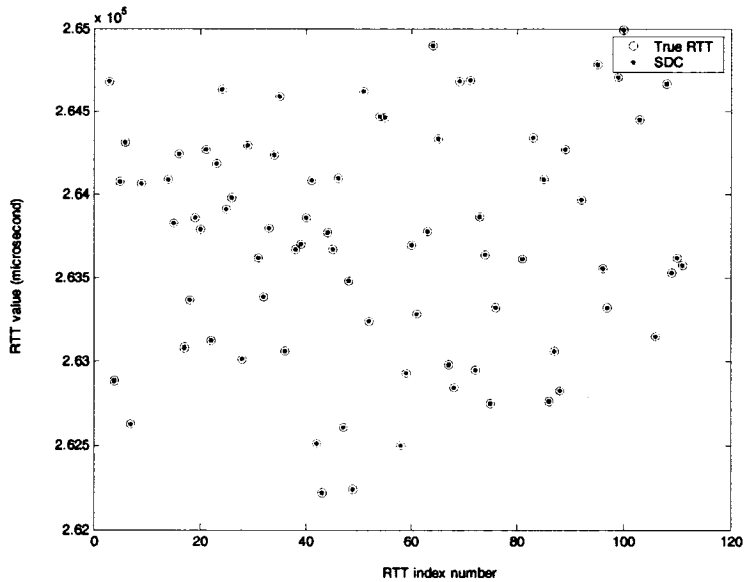


Figure 5.8 Verifying SDC under the situation without overlaps of send-echo

The second experiment was to evaluate the performance of SDC focusing on the situation in which there are overlaps of send-echo pairs. Letting some students type independently and freely, we captured all the send and echo packets for a period of time, and computed the RTTs from SDC. We take L_p=32874 and compute the lower bound of the probability of making a correct selection of RTT by using inequality (5.7). Three examples are presented in Table 5.3, where the second to the fifth columns are average values of the RTTs in microseconds, standard deviation, q number, and the lower bound of the probability, respectively. From the probability estimated, we are confident about the result from SDC because the probabilities in these three examples are all higher than 97%, respectively. Even though we cannot compare the result from SDC with the correct

RTTs when there are overlaps of send-echo pair, we can still evaluate SDC by estimating the probability of making a correct selection of RTT.

Table 5.3 Packet matching probability estimation

Items / Examples	μ	σ	q	p
1	264947.0	2810.708	11.695	0.9927
2	265756.3	5514.666	5.9612	0.9719
3	265727.2	5549.605	5.9237	0.9715

5.3.4.3 Comparison between SDC and the best algorithm for finding RTTs

In this section, we compare the performance between SDC and the best existing algorithm for finding RTTs. As we know, the best algorithm to find RTTs is the Conservative algorithm [Yan05] in terms of packet matching-accuracy, or the Greedy algorithm [Yan05] in terms of packet matching-rate. So far there is no algorithm that can obtain both high packet matching-accuracy and high packet matching-rate in finding RTTs. The test bed is the same as the one in Section 5.3.4.1, and the monitor host is Acl08.

First, we compared SDC with the Conservative and Greedy algorithms under the situation in which there is no overlap of send-echo pair. When we did the experiment we needed to control our keystroke speed to be as slow as possible so as to be sure there was no overlap of a send-echo pair. Three algorithms ran at Acl08 at the same period of time to monitor the same connection chain. The RTTs found by the three algorithms are partly shown in Figure 5.9, in which each point represents the RTT of one send packet. From the results shown in Figure 5.9, we know if there is no overlap of a send-echo pair, the

three algorithms have the same performance for finding the RTTs in terms of matching-accuracy and matching-rate.

Second, we compared the performance of the three algorithms under the situation that there are overlaps of send-echo pairs on the Internet. We still used the same test bed as the one used in Section 5.3.4.1, and Acl08 is the monitoring point. Instead of controlling the keystroke speed, we freely typed some commands at Acl09. We still ran the three algorithms on Acl08 and collected the send and echo packets. The three algorithms gave us three different RTTs. The comparison results are partly shown in Figures 5.10 and 5.11.

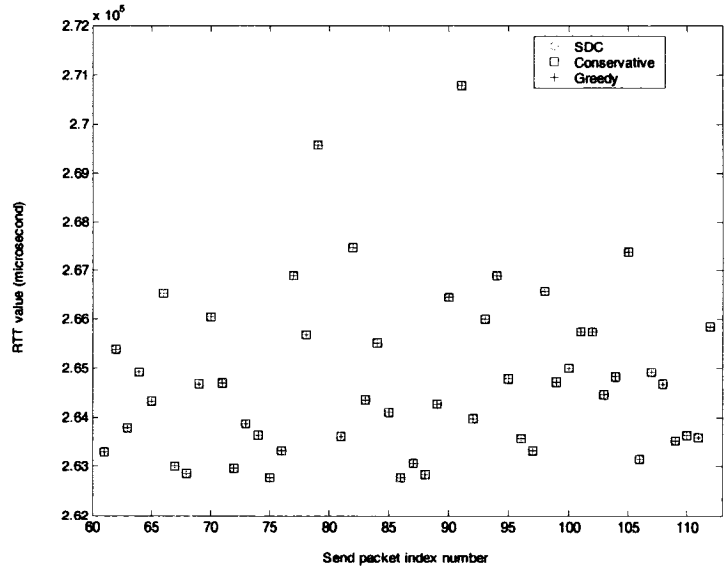


Figure 5.9 Packet-matching comparison among the Conservative, Greedy, and SDC without send-echo pair overlaps

Figure 5.10 shows us part of the RTTs comparison results between the Conservative algorithm and SDC. We collected 169 send packets, in which 44 send packets (in Figure 5.10 only 28 matches are displayed for clarity) are matched by the Conservative algorithm, and 169 send packets (in Figure 5.10 only 71 matches are displayed) are matched by SDC. The RTTs found by the Conservative algorithm are exactly included in the RTTs found by SDC. Even though we are not sure of the correctness of the rest of the

RTTs found by SDC, we still get a sense about the correctness of the RTTs computed by SDC from their distribution.

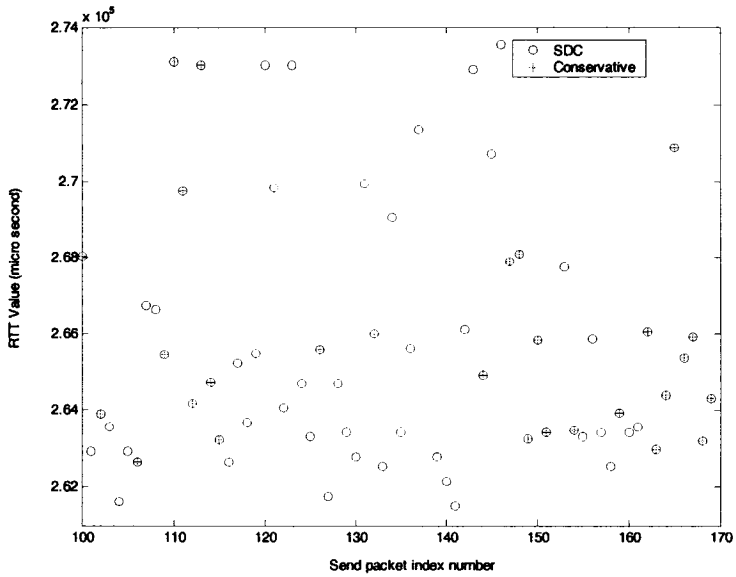


Figure 5.10 Packet-matching comparisons between the Conservative and SDC algorithms under overlaps of send-echo pair in finding RTTs

We verified the packet matching-rate of SDC by comparing it with the Greedy algorithm. Figure 5.11 shows only part of the RTTs comparison results between SDC and the Greedy algorithm. It shows that most of the RTTs between the two algorithms are consistent. In 169 RTTs, 157 RTTs found by the Greedy algorithm are included in the results of SDC. But we are still not sure about the correctness of the remaining 12 RTTs (for clarity only 7 of the 12 RTTs are displayed in Figure 5.11) found by SDC until we compare them with the results of the Conservative which are supposed to give us the correct results. We found 5 of the 12 RTTs (in Figure 5.11, the rest 7 RTTs are displayed) found by SDC are consistent with the RTTs found by the Conservative algorithm. Even though we cannot judge the correctness of the other 7 of 12 RTTs found by SDC, from their distribution, they might be correct. To summarize our comparison results among

SDC, the Conservative, and the Greedy algorithms, we found that the algorithm SDC is the only one that can get both high matching-accuracy and high matching-rate in finding RTTs.

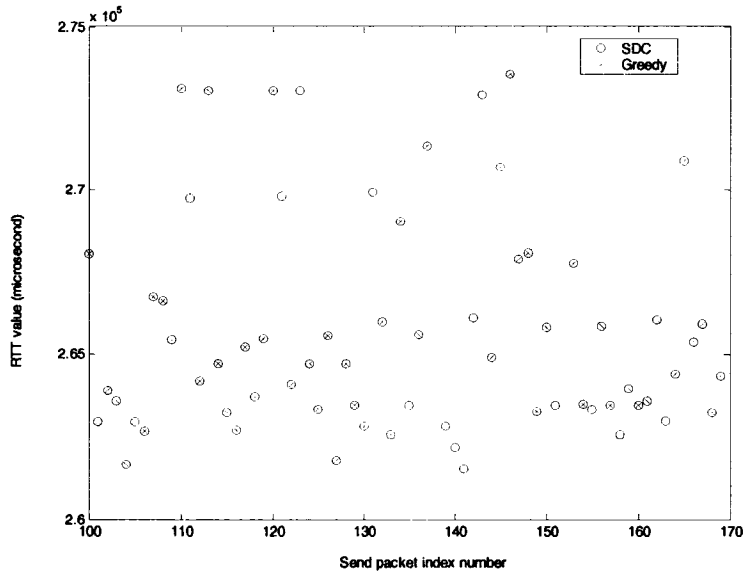


Figure 5.11 Packet-matching comparison between SDC and the Greedy algorithms under overlaps of send-echo pair in finding RTTs

6. THUMBPRINT AND CONNECTION TRACEBACK

In Chapters 3, and 4, we discussed the approaches to detect stepping-stones and stepping-stone intrusion. In this chapter, we propose two new technologies which combine with the packet matching techniques in Chapter 5 to trace intrusions back. One uses temporal thumbprints to do traceback; the other one uses RTT-thumbprints. This work appears in [Yan05-3] and [Yan05-4].

6.1 Using T-thumbprint to Trace Stepping-stone Intrusion Back

The design of the TCP/IP protocol makes it difficult to reliably trace back to the original attackers if they obscure their identities by logging on through a chain of multiple hosts. A thumbprint method based on connection content was proposed to trace back attackers; but this method is limited to non-encrypted sessions. Even though there are many other methods proposed to trace intrusions back, they have different kinds of problems (see Chapter 2).

To identify a connection, and to trace intrusions back, we propose a temporal thumbprint (T-thumbprint), which is based on packet time intervals. T-thumbprint is a sequence of time gaps between the consecutive TCP 'Send' packets of an interactive terminal session. An algorithm is presented to correlate two T-thumbprints to see if they belong to the same connection chain. We also discuss the possibility of defeating an attacker's manipulation over a connection chain by using a T-thumbprint to do traceback. T-thumbprint has the following advantages: (1) it can be applied to encrypt sessions; (2)

it does not require tightly synchronized clocks; (3) it can resist intruders' evasion to a certain extent; and (4) it is efficient, and could be used to trace back intruders in real time.

6.1.1 Temporal Thumbprints

Given the sequence of 'Send' packets $<p_1, p_2, \ldots, p_{n+1}>$ from Host 1 to Host 2, let $<t_1, t_2, \ldots, t_{n+1}>$ be the corresponding time stamps of $<p_1, p_2, \ldots, p_{n+1}>$. We define a T-thumbprint of the connection to be the sequence $<t_2-t_1, t_3-t_2, \ldots, t_{n+1}-t_n>$. Each element represents a time gap between two consecutive Sends. In the following sections, for convenience, we usually use array $[1, 2, \ldots, n]$ to represent a T-thumbprint.

The length n of a thumbprint depends on the number of send packets collected; we can certainly divide a thumbprint into subsequences for each time unit (such as a 1-minute interval). How to select n depends on the network. For a local area network, n is not necessarily large because of less network fluctuation, usually 8; but for a wide area network, n needs to be a little bit larger because of the serious network fluctuation, usually 32, 64, or 128. We define T-thumbprint for incoming connections as well. The latter is called the incoming T-thumbprint, denoted as iT-thumbprint, and the former is called the outgoing T-thumbprint, denoted as oT-thumbprint. For convenience, we usually use T-thumbprint to represent both incoming and outgoing temporal thumbprints.

The most important concern about T-thumbprint is how to determine whether one T-thumbprint is close to another so as to determine if the two connections, which are represented by two T-thumbprints, are in the same chain. For example, in some cases, we need to determine if an iT-thumbprint is close to an oT-thumbprint of the same host, or to determine if the oT-thumbprint of one host is close to the oT-thumbprint of another host

so as to decide if the two hosts are in the same connection chain. We use *MR* (matching-rate), which is the ratio between the number of matched elements between two T-thumbprints and the minimum of number of elements of the two T-thumbprints, to determine if the two T-thumbprints are close. We assume the two T-thumbprints are *a[1, 2, 3, ..., n]* and *b[1, 2, 3, ..., n]*, respectively, and use the following inequalities (6.1), (6.2) and (6.3) to determine if element *a[i]* matches element *b[j]*, where $1 =< i, j = < n$, and ε is a predefined value:

$$\frac{2|a[i]-b[j]|}{|a[i]+b[j]|} < \varepsilon, \tag{6.1}$$

$$\frac{2|a[i+1]-b[j+1]|}{|a[i+1]+b[j+1]|} < \varepsilon, \tag{6.2}$$

$$\frac{2|a[i+2]-b[j+2]|}{|a[i+2]+b[j+2]|} < \varepsilon. \tag{6.3}$$

6.1.2 Challenges for T-thumbprint Correlating Algorithm

We cannot always assume that elements between two T-thumbprints correspondingly match. We assume that host h_1, host h_2 and host h_3 are connected by one SSH connection chain. If all the Send packets from host h_1 to host h_2 were forwarded exactly to host h_3, correlating such T-thumbprints would be trivial; but for most cases, especially on the Internet, it is more complex. Even if in the same chain, the Send packets sent from host h_3 do not have a one-to-one relationship with the Send packets of host h_2, making the correlation of two T-thumbprints more complex.

To understand how a packet propagates on the Internet, we need to be clear how TCP and SSH protocols work [RFC81], [Ylo96]. We assume there is a packet sent from host h_1 to host h_2; this packet would be decrypted first, then encrypted at host h_2, and finally be forwarded to host h_3; this procedure is repeated until this packet reaches the end machine of the chain. There is a possibility in the packet delivery procedure that the packet would be divided into some sub-packets or merged into a big packet. There are several reasons that T-thumbprint correlating is a challenge.

The first reason is that the encryption key will be re-exchanged after a certain period of connection time, or after each gigabyte of transmitted data between two adjacent hosts. This communication happens only between two adjacent hosts; it is not forwarded to the downstream host. The second reason is lost packet retransmission. Suppose that a packet sent from host h_1 to host h_2 is lost during transmission for the reason that either host h_2 does not receive that packet or host h_1 does not receive any acknowledgement packet from host h_2; host h_1 would then resend that packet until host h_2 acknowledges it. The third one is that for security reasons, some ignore packets are randomly sent to the server side from the client; once the server side receives an ignore packet, it neither responds nor forwards, but acknowledges the client side. So ignore packet transmission would result in T-thumbprint correlation that is not one-to-one. The fourth reason is that the packet may be fragmented during delivery process on the Internet. The packet will be fragmented if either the packet size is more than the maximum size allowed on the Internet or if travels from IPv6 to IPv4. Packet fragmentation would result in a T-thumbprint correlation that was also not one-to-one. The fifth reason is that intruders may inject some characters (or add some random delay) to a chain in order to evade intrusion

traceback, such as the approach in [Zha00]. Intruders' manipulation on a connection chain would also result in difficulty to match the T-thumbprints. The algorithm proposed in this dissertation is able to tolerate some of these stated problems, and the experimental results show that if an attacker injects fewer than 35% of the characters, our algorithm is able to handle it. We discuss how this approach can resist intruders' evasion in Section 6.1.6.

6.1.3 T-thumbprint Correlating Algorithm

Suppose there are two sequences A: $a[1, 2, ..., n]$ and B: $b[1,2, ..., m]$, which represent two T-thumbprints, respectively. It is also assumed that element $a[1]$ matches element $b[1]$. We cannot claim that each element in B exactly matches the elements in A or vice versa because of the T-thumbprint's unsymmetrical feature caused by the reasons just discussed. We need to compute MR to determine if the two T-thumbprints are closed.

The first and easiest way to do this is to compare one element from B with each element in A, to see if this element in B matches any element in A; this is done by using Equation (6.1), and computing MR. The problem with this approach is that there may be several elements in A that match the same element in B; as well, its time complexity is $O(m*n)$. That is why we use Equations (6.1), (6.2), and (6.3) together to determine one matched pair. This will largely decrease false positives with the time complexity $O(m+n)$. We already know $a[1]$ matches $b[1]$, but this does not mean that $b[2]$ must match $a[2]$. The element $b[2]$ in sequence B probably matches $a[2]$ and $a[3]$ or more elements. Scanning twice is used in this algorithm. In the first scanning, one-to-one matched pairs are reached; in the second scanning, non-one-to-one matched pairs are reached. A one-to-

one matched pair is one in which one element comes from sequence *A* and another comes from sequence *B*.

In the first scanning, if *a[2]* does not match *b[2]*, instead of moving to match *b[3]* with *a[3]*, we continue to check if *b[2]* matches with a certain number of elements following *a[2]*, as well as to check the elements in *B*. If there is no match, one more element is slid down in each sequence; if there is a match, the current position for each sequence is set to the position next to the matching element in each sequence. The steps above are repeated until no more elements are left in *A* or *B*.

Unlike in the first scanning, in the second scanning, it only needs to check the elements in between. Suppose in the first scanning, we have got two one-one pairs; assume they are <*a[2], b[2]*>, and <*a[5], b[7]*>, respectively; the second scanning is just simply to check if the sum of *a[3]* and *a[4]* matches the sum of *b[3]*, *b[4]*, *b[5]*, and *b[6]*.

The details of the correlating algorithm are shown as the following. The output of the algorithm is *MR* between *A* and *B*, and if *A* is close to *B*, *A* is not close to *B*, or undecided.

program T-thumbprint Correlating Algorithm
function main(){

```
//A, and B are two T-thumbprints with elements a[1,…,n] and
//b[1,…,m] respectively
// Range is a range to check within, Th_Max, Th_Min are thresholds
Initialize current position CurrPA, CurrPB for A, and B
 //get all the one-to-one match pairs
 while (there are more elements in B and A) {
 //counter is the number of matching elements between A and B
 //NumA, NumB are the number of elements in A, B respectively
     lbA=CurrPA;        lbB=CurrPB;
     ubA=CurrPA+Range; ubB=CurrPB+ Range;
     Match=false; i=lbB;
     while(i<=ubB-2 && !Match){
       j=lbA;
       while(j<=ubA-2 && !Match){
        ra1=2*|b[i]-a[j]|/(a[j]+b[i]);
        ra2=2*|b[i+1]-a[j+1]|/(a[j+1]+b[i+1]);
```

```
            ra3=2*|b[i+2]-a[j+2]|/(a[j+2]+b[i+2]);
            if(ra1, ra2, ra3 are all less than ε ){
               MP[i] = j;  //save the matching in MP
               counter++;
               CurrPA=j+1; CurrPB=i+1;
               Match=true;
            }
            j++;
         }
         i++;
      }
      if (Match) {currPA=j; currPB=i;}
      else {currPA++; currPB++;}
   }
   //Merge the unmatched segments
   while(there are undefined elements in MP){
      Suppose MP[i1]=j1 and MP[i2]=j2 and all MP[i] are
      undefined for i1<i<i2, merge all a[i] (i1<i<i2) into one
      element s1 and a[j] (j1<j<j2) into one element s2.
      if (2*|s1-s2|/(s1+s2)<ε) counter++;
   }
   matching rate MR=counter/min(NumA , NumB);
   if(MR >Th_Max) A is close to B
   else if(MR<Th_Min) A is not close to B
   else undecided
}
end.
```

6.1.4 The Environment and Test Procedures

We have two hosts Acl08 and Acl09 under control, each running Linux Red Hat 9.0; they belong to the domain 'cs.uh.edu'. We need to monitor and capture all the packets going through the NIC of Acl08 and Acl09 with network speed 100M/s. The machines, host h_1, ... , host h_4, which are running the Linux operating system, are located in the Computer Science Lab.

We also have other three hosts out of control with some regular accounts to login: One is Mex, which is located in Mexico; the second is Epic, which is located in California; the third one is Bayou, which is located in Houston. With the hosts above and the accounts

that we collected, it is easy to form an Internet connection chain such as host h_i (i=1 to 4)→ Acl09 → Mex → Acl08 → Epic → Bayou by using SSH. In the chains above, Bayou is supposed to be the victim, while host h_1, … , host h_4 are attacking machines, and other four hosts in between are stepping-stones.

We implemented a program called Temporal Thumbprint Traceback (TTT), which only supports Ethernet for simplicity, with Libpcap to capture the T-thumbprint.

We established four connections, h_i→Acl09→Mex→Acl08→Epic→Bayou, where i=1 to 4, while the four connections share the same path. We ran the program TTT on Acl08 and Acl09 to capture the packets passing through them, and the T-thumbprint was collected in Acl08 and Acl09. Four persons operated h_1, h_2, h_3, and h_4 at the same time, each person inputting the same content at his/her own typing speed. We captured one T-thumbprint on each connection passing through Acl08 and Acl09. We verified that we can use the four oT-thumbprints to pick up the four connections; the results are shown in Section 6.1.5.

We must mention why we claim the connections we built share the same path, same location, same contents, and same time. Actually, in the real world, most probably, intruders on the Internet could not operate around the same location and input the same thing at the same time. We try to use the connections that we set up to simulate a worst case scenario. If a T-thumbprint could deal with such a case, of course it could deal with all kinds of cases.

6.1.5 Experimental Results and Analysis

We are going to show the following results: 1) we could identify the same chain among connections with given T-thumbprints; 2) the performance of the correlating algorithm; 3) the performance of the T-thumbprint.

We used the time interval that it takes for one packet to travel from one host to another to represent the distance between two hosts. Table 6.1 shows the correlating results between each oT-thumbprint on Acl08 and those on Acl09 with a distance 70 *ms* long; the values represent *MR* that are computed by the correlating algorithm, and each oT-thumbprint is at most 256 long. This result clearly tells us 1) if two connections are in the same chain, the *MR* is much bigger than when they are not; 2) if two connections are not in the same chain, the *MR* is much smaller than where they are.

Table 6.1 T-thumbprint correlating *MR* between two hosts on the Internet

	Connection at Acl08			
Connection at Acl09	C_0(%)	C_1(%)	C_2(%)	C_3(%)
C_0	92.37	0.00	0.57	0.52
C_1	–	84.00	0.57	0.00
C_2	–	–	87.42	0.62
C_3	–	–	–	89.00

How big and how small the *MR* is really depends on the distance between two hosts. Suppose two thresholds are selected: *Th_Max*, and *Th_Min*; if *MR* is bigger than *Th_Max*, we are relatively sure that the connections are in the same chain, and if *MR* is smaller than *Th_Min* we are also sure that the connections are not in the same chain; however, if *MR* is between *Th_Max* and *Th_Min* we are not sure if they are in the same chain.

Therefore, how to select these two parameters correctly is very important to the correlating algorithm; they are affected by many factors, but the important one is the distance between two hosts. The relation between *MR* and distance is discussed in the T-thumbprint performance part of this section.

Figures 6.1, 6.2, 6.3, and 6.4 show us the performance of the correlating algorithm, where in each figure the X axis stands for the T-thumbprint on Acl08 and the Y axis for Acl09 with unit in microseconds, Figure 6.1 shows the scenario of two T-thumbprints that come from the same chain before being processed with the correlating algorithm. Figure 6.2 shows the same thing but after being processed by the correlating algorithm. Figures 6.3, and 6.4 show similar scenarios but the two T-thumbprints come from different chains. By comparing Figure 6.1 with Figure 6.2, and Figure 6.3 with Figure 6.4, one can see it is easier to match the T-thumbprint with the help of the correlating algorithm than without.

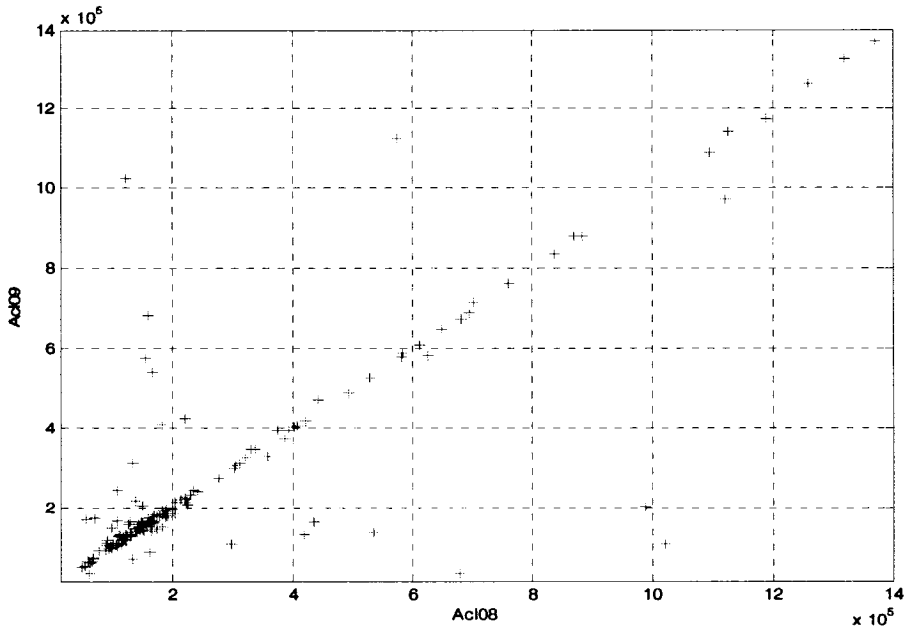


Figure 6.1 Matched T-thumbprints correlation before processing

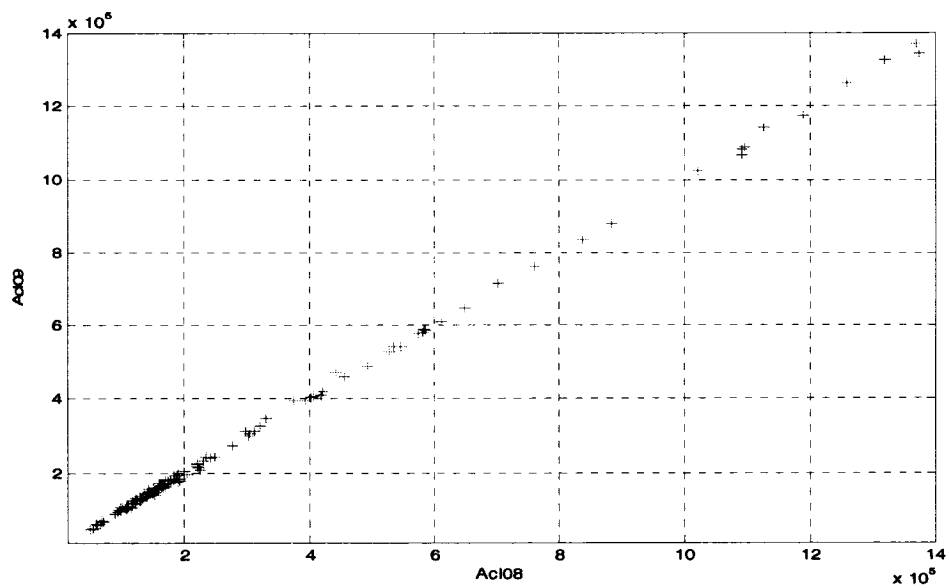


Figure 6.2 Matched T-thumbprints correlation after processing

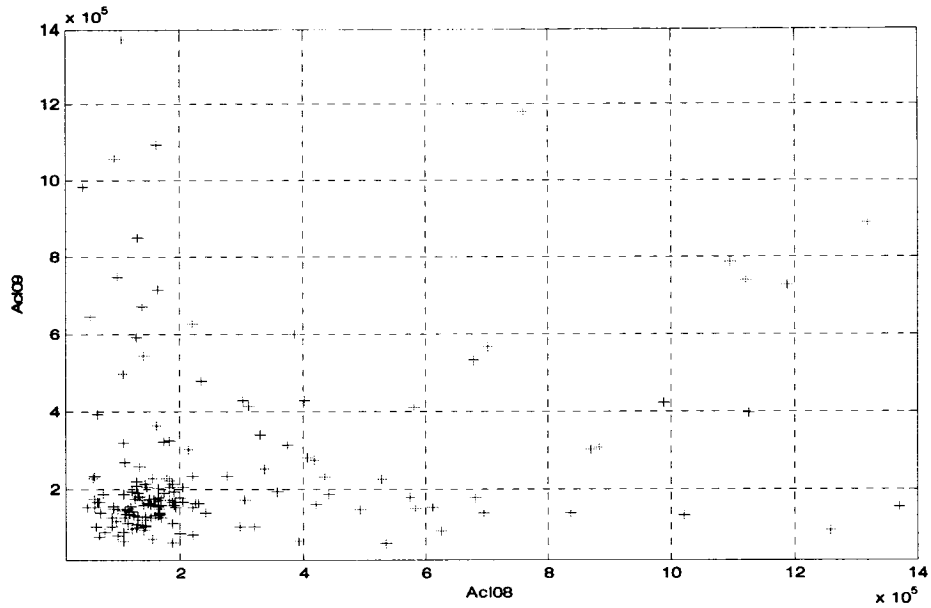


Figure 6.3 Unmatched T-thumbprints correlation before processing

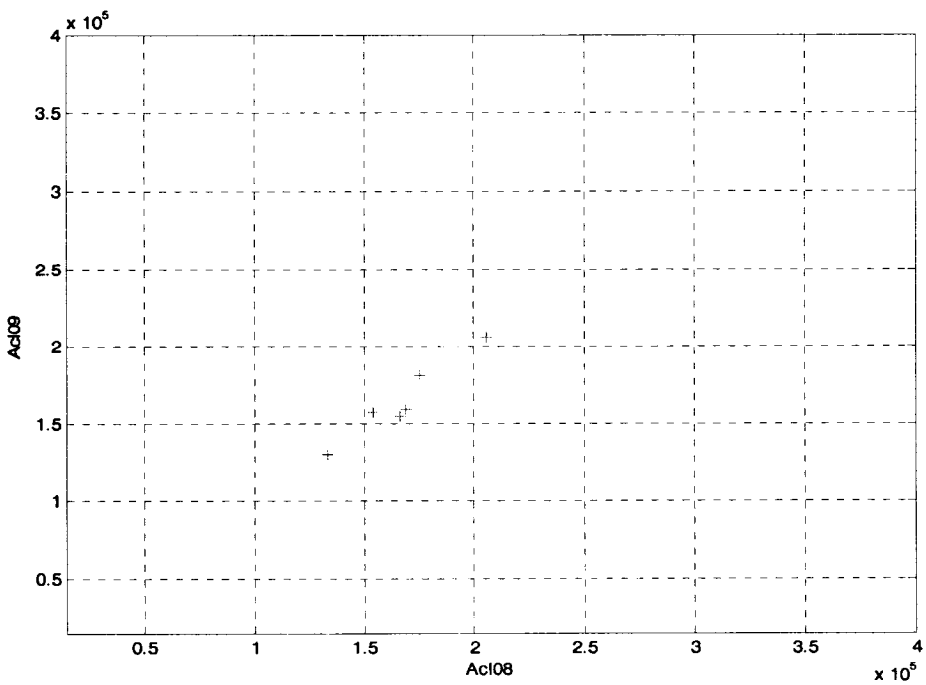


Figure 6.4 Unmatched T-thumbprints correlation after processing

We use adaptability to measure T-thumbprint performance. The adaptability is defined as the variation of *MR* between two T-thumbprints over the distance between two hosts.

Figure 6.5 shows the performance of T-thumbprint, where the Y axis stands for the *MR* of the two T-thumbprints, and the X axis stands for the distance between two hosts. Figure 6.5 shows that if the distance is increased, the *MR* will be decreased for the T-thumbprints in the same chain. The method we used to increase the distance between two hosts is to connect another two hosts more times because there is not any method to purely increase the distance between two fixed hosts without inserting any other hosts between them. As long as we introduce more hosts, it should affect the performance of the T-thumbprint negatively. In fact, the real performance of T-thumbprint is a little better than what is shown in Figure 6.5. That is, Figure 6.5 shows us only the lower bound performance of T-thumbprint.

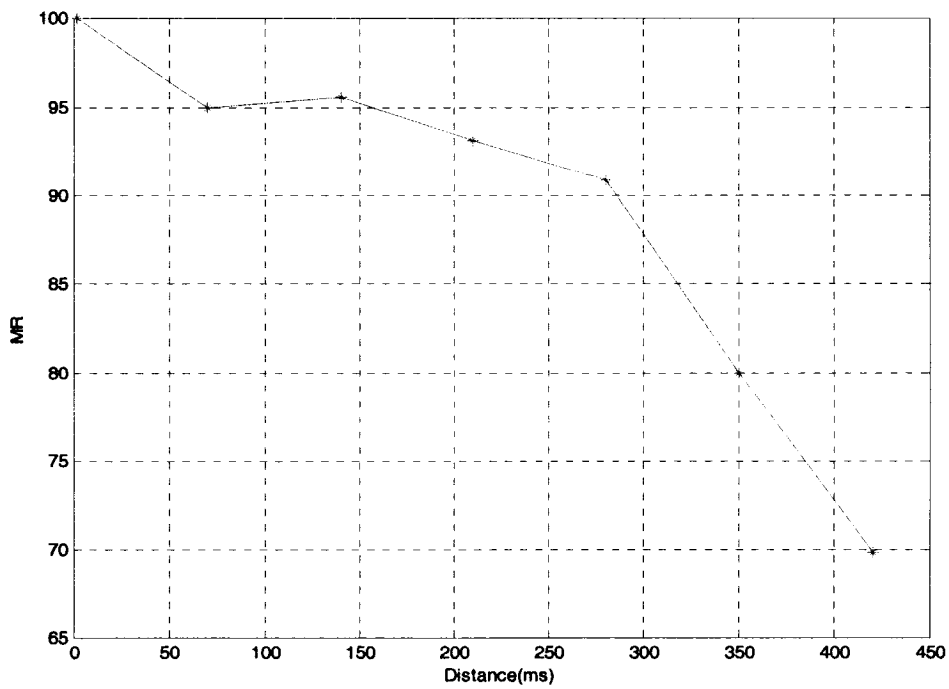


Figure 6.5 Performance of T-thumbprint

6.1.6 Discussion on Resisting Intruder Evasions

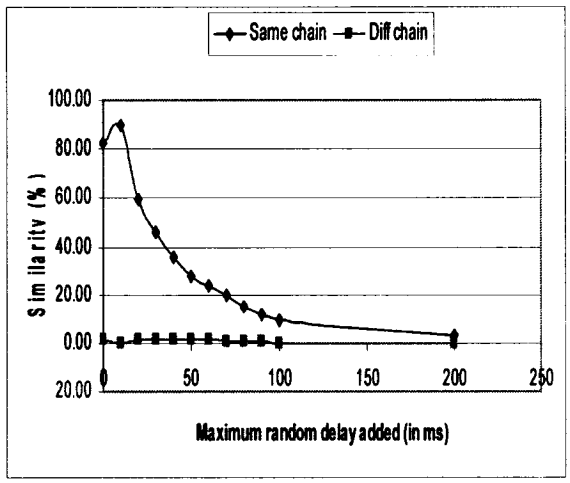
Tracing intruders back with T-thumbprint has been discussed. Most probably, intruders who are aware of the risks of being traced try to evade the traceback by modifying their connections. To evade the T-thumbprint, they may randomly delay the outgoing packets or randomly inject some characters into the connection so that the outgoing and incoming connections appear unrelated. The fact is T-thumbprint depends on the intruder's keystroke speed, which was claimed by V. Paxson to follow a Pareto distribution [Pax95]. If more characters are inserted into the stream, it is very difficult to maintain a Pareto distribution without carefully processing. We can detect the intruder's evasion by checking if a T-thumbprint breaks the Pareto distribution. This method does not always work because if the manipulation is processed carefully, the intruder can still make the stream while keeping the Pareto distribution. However, at least the T-thumbprint method makes intruders work harder. Furthermore, even if the T-thumbprint of the manipulated chain could follow a Pareto distribution, we are still able to handle them, but with some limitations.

Note that intruders can only delay the outgoing packets, not accelerate them. Another fact is an intruder cannot tolerate a much longer delay. This means there is an upper bound for an intruder's packet delay.

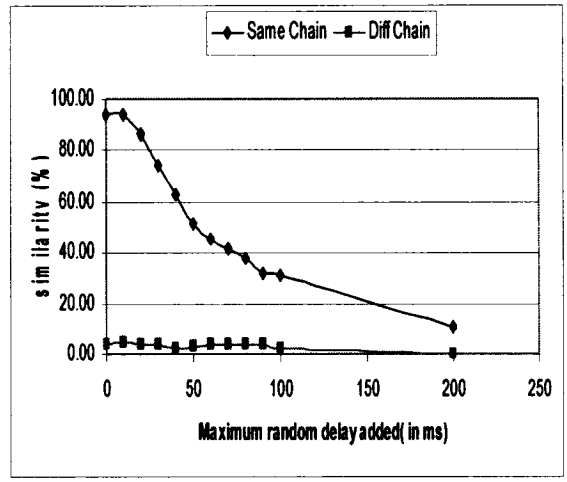
Suppose we have two time sequences $N_1(t)$, and $N_2(t)$ to represent two T-thumbprints, respectively, where $N_1(t)$ is the original sequence, and $N_2(t)$ is the manipulated sequence. It makes sense to make the following two assumptions: (1) The character emerges in the manipulated sequence $N_2(t)$ if and only if it has emerged in its original sequence $N_1(t)$;

(2) If one character emerges in its original sequence $N_1(t)$, it must emerge in its manipulated sequence $N_2(t)$ within a certain time interval. Donoho, *et al.*, [Don02] pointed out in theory that the two sequences are still related under the two assumptions above. That is, the scaling coefficients of the two sequences wavelet transform must be very close at long time scales. So even in the time domain, we could not correlate the two sequences, but we could still correlate them in the frequency domain. The problem is that we would need to monitor the chain for a longer time. The correlating algorithm would not work on the time-frequency domain because it is time-domain based; we are still working on a time-frequency domain T-thumbprint correlating algorithm.

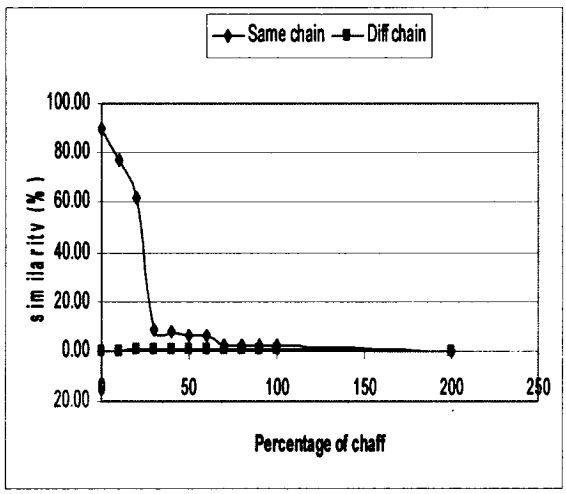
We are going to discuss how T-thumbprints resist intruders' manipulations, such as random delay and chaff perturbation. Figure 6.6 shows how the T-thumbprint performed when the chain was manipulated under different situations: random delay and chaff perturbation.



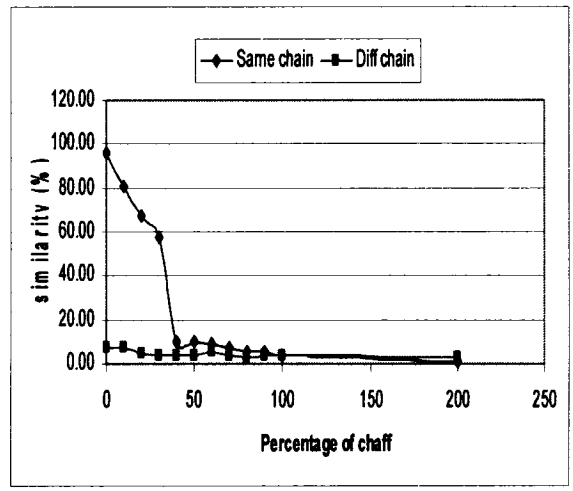
(a) Random delay added and correlated with ε=10%



(b) Random delay added and correlated with ε=20%



(c) Chaff added and correlated with ε=10%



(d) Chaff added and correlated with ε=25%

Figure 6.6 Performance of T-thumbprint on resisting attacker's chaff and random delay manipulation

Suppose that an intruder manipulated the connections by randomly delaying all the packets in the chain. The correlating algorithm could still use T-thumbprint to correlate the connections, Figure 6.6 (a) and (b) show this case; the difference between them is that different ε are used to correlate T-thumbprint. We made a program to simulate the

random delay manipulation. Suppose $N_1[1,...,m]$, $N_2[1,...,n]$, $N_3[1,...,k]$, $N_4[1,...,j]$ are four sequences; it is already known that N_1 and N_2 are correlated. For each element of N_2, N_3, and N_4, a random number is generated but limited by a maximum value Max-delay and added to elements of N_2, N_3, and N_4; then new sequences N_2', N_3', and N_4' are made. We correlate N_1 with N_2', N_3', and N_4', repectively by changing Max-delay from 10 miliseconds to 200 milliseconds. In Figure 6.6 (a) and (b), the Diffchain curve shows the correlation results of N_1 and N_2', and the Samechain curve represents the maximum *MR* between N_1 and N_2', N_3', N_4' respectively. The results show that the correlating algorithm could distinguish one connection from the others under a maximum random delay of up to 100 millisecond with ε=10%. This method will be stronger in resisting random delay manipulation with bigger ε, such as ε=10%.

Suppose that the attacker manipulates the connection by randomly injecting some characters into the chain, can the correlating algorithm still handle this situation? The experimental results in Figure 6.6(c) and (d) show that our approach can resist an attacker's injecting chaff attack to a certain extent, but adjusting ε cannot make the results better. Here we still assume that we have four sequences $N_1[1,...,m]$, $N_2[1,...,n]$, $N_3[1,...,k]$, $N_4[1,...,j]$, each of which represents one T-thumbprint, where N_1 and N_2 are already known to be correlated. N_2', N_3', and N_4' are generated using a simulating program with different injecting rates from 0% to 200%; Figure 6.6 (c) and (d) show the results. From the results shown we know that the correlating algorithm can be used to correlate T-thumbprints to the extent that chaff is limited to within probably 30%.

6.1.7 Finding the Longest Similar Subsequence of T-thumbprints

We have proposed a correlating algorithm to compare two T-thumbprints to see if they are related. But the problem is far from being solved because we assumed that the two T-thumbprints have the same start point, which simplified our correlation algorithm. We studied the sequence correlation formally, and propose a formal definition of the problem by introducing ε-similarity, partial sum and longest ε-similar subsequence (LSS) and cast it as a generalization of the well-known Longest Common Subsequence (LCS) problem. It is not a trivial task to define what is similar in two sequences of numbers which may differ in length. The definition and solution we derive here can be used in comparing many different thumbprints.

The LSS problem is much more complicated than the LCS problem due to the partial sums involved. We analyzed the property of partial sums, and proposed to focus on the minimum matched partial sum (MMPS). The Longest ε-Similar Subsequence (LSS) problem has an optimal structure similar to that of LCS problem [Wan06].

With the dynamic programming technique, we have an algorithm with complexity $O(m^2n^2)$ to find the optimal solution to the LSS problem. Based on the property of partial sums as defined in [Wan06], we come up with a more efficient optimal algorithm with time complexity of $O(mn(m+n))$. Practically, matches of very large sized minimum matched partial sum (MMPS), which sum up a large number of elements together, are likely false match-ups in thumbprint application. By limiting the size of MMPS to a constant number s, we reduced the complexity of a suboptimal solution to $O(smn)$.

6.2 Using RTT-thumbprint to Trace Stepping-stone Intrusion Back

We have developed T-thumbprint, which is defined as a sequence of intervals between timestamps of two continuous send packets in a connection, to trace intruders back with the advantages of efficiency, secrecy, and robustness. But it does not provide a full solution in resisting intruders' random delay and chaff manipulation.

We propose a new time-based thumbprint, Round-Trip Time- (RTT-) thumbprint, to characterize packets in a session, as well as two algorithms (exhaustive and heuristic) to correlate RTT-thumbprints. Instead of using timestamps of send packets or contents in a connection, RTT-thumbprint uses a sequence of timestamp pairs between each send packet and its corresponding echo packets to characterize a connection. The experimental results and analysis showed that the RTT-thumbprint can handle intruders' random delay and chaff manipulation better than T-thumbprint.

6.2.1 Preliminaries and Motivations

6.2.1.1 Preliminaries

We make the following assumptions: (1) the research object is limited to an interactive connection session made by *telnet*, *rlogin*, *rsh*, *ssh* or other similar tools; (2) the thumbprints are collected at approximately the same time intervals; (3) any users, when connected to a host, may need to pause to read, think, or respond to the previous operations, and the gaps between two consecutive operations caused by human interaction are measured in seconds. These gaps are considerably larger than a typical round-trip time of a network; (4) a user can only delay each packet sent or received; any

superfluous packets inserted into a connection will be removed before reaching the destination.

We first define some terminology. Given two sequences T: $\{(t_{11}, t_{12}), (t_{21}, t_{22}), \ldots, (t_{n1}, t_{n2})\}$, and U: $\{(u_{11}, u_{12}), (u_{21}, u_{22}), \ldots, (u_{m1}, u_{m2})\}$ with length n, and m respectively, we assume that the conditions $0<t_{i1}<t_{i2}$, $0<u_{j1}<u_{j2}$, where $i=1,\ldots, n$, and $j=1,\ldots, m$, are satisfied for sequences T and U. We define element-inclusion, and sequence-inclusion in the following:

Element-inclusion: For any timestamp pairs $t = (t_{i1}, t_{i2})$, and $u = (u_{j1}, u_{j2})$ if the condition $0< t_{i1}< u_{j1} < u_{j2}< t_{i2}$ is satisfied, then we say u is included in t, we denote it as $u\subset t$.

Sequence-inclusion: For two sequences U and T of length n, we say that sequence U is included in sequence T, denoted $U\subset T$, if $U[i]\subset T[i]$, where $i=1,\ldots, n$.

The sequence-inclusion definition only gives us the result that one sequence is completely included in another sequence. However, most times, when we correlate sequences which come from thumbprints, we need to determine whether a part of a sequence is included in another sequence. Under this situation, the sequence-inclusion problem becomes computing a longest inclusion subsequence.

Longest Inclusion Subsequence (LIS): Given the two sequences T and U above with lengths m and n respectively, if (1) there is a subsequence U' of U and a subsequence T' of T, such that U' is included in T' and (2) there are no other U' and T' with longer lengths that are included in T, we define U' as the longest (common) inclusion subsequence of U and T. The problem of computing a longest subsequence from one

sequence that is included in another sequence is the longest inclusion subsequence problem.

The length of the longest inclusion subsequence can then be used to measure the similarity of two sequences of timestamp pairs. The similarity ratio (*SR*) of two sequences is defined as $p/\min(m, n)$, where p is the length of a longest inclusion subsequence.

6.2.1.2 Motivations

Once a chain is established between an intruder and a victim, it has been shown [Yan05] that every packet sent from a intruder is decrypted and then encrypted in each host in between and forwarded to the victim at the end of a chain; a corresponding packet is echoed from the victim and propagated to the intruders' side in a similar way. If we monitor the outgoing connection of each host in between, we can observe the send packet from this host and one echo packet from the adjacent host downstream. This fact motivates us to consider these send and echo packet pairs as a unique characteristic for identifying a connection. Thus, we can use each sequence of packet RTT timestamps to characterize an encrypted connection.

We assume there are two compromised hosts, Host *i*, and Host *j*, that belong to one chain $<C_1, C_2, \ldots, C_5, C_6>$, as shown in Figure 6.7. Host *i* has two incoming connections: C_1, C_3, and two outgoing connections: C_2, C_4; Host *j* has two incoming connections: C_5, C_7, and two outgoing connections: C_6, C_8. If we monitor all the send and echo packets of each outgoing connection of Hosts *i* and *j* continuously, and match them, we should get a sequence like $\{(s_1, e_1), (s_2, e_2), \ldots, (s_n, e_n)\}$, where each s_i represents one send packet, and

e_i represents one echo packet. If we only take the first pair for each sequence and put the timestamps information into each pair, then we get the information for each outgoing connection, C_2: (t_{2s1}, t_{2e1}), C_4: (t_{4s1}, t_{4e1}), C_6: (t_{6s1}, t_{6e1}), C_8: (t_{8s1}, t_{8e1}). If connection C_2 and connection C_6 are in the same chain, the relations $0 < t_{2s1} < t_{6s1}$, and $t_{2e1} > t_{6e1} > 0$ must be true. If two connections are not in the same chain, such as C_4, and C_8, the relations above are not likely to be true. However, if we check more consecutive pairs, the probability that the relations are held for the two connections that are not in the same chain should be very low. In other words, we can get a gap between a probability that two connections are in the same chain and a probability that two connections are not in the same chain. If this gap is higher than a predefined threshold, we can safely consider that the two thumbprints belong to the same connection chain; otherwise, they do not. We use the timestamps of the consecutive matched pairs in one connection as our thumbprint; this can also be used to identify a connection uniquely and to trace back intruders.

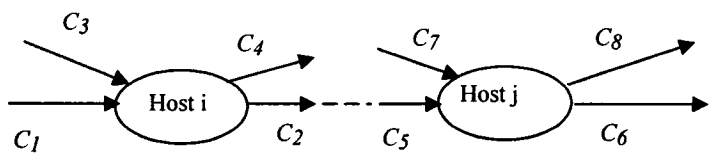


Figure 6.7 Illustration of the basic idea of RTT-thumbprint

6.2.2 RTT-thumbprint

Given a sequence of 'Send' and its matched 'Echo' packet pairs $\{(s_1, e_1), (s_2, e_2), \ldots, (s_n, e_n)\}$ from Host *1* to Host *2*, let $\{(t_{s1}, t_{e1}), (t_{s2}, t_{e2}), \ldots, (t_{sn}, t_{en})\}$ be the corresponding

timestamps of $\{(s_1, e_1), (s_2, e_2), \ldots, (s_n, e_n)\}$. We define a RTT temporal thumbprint (RTT-thumbprint) of a connection to be a sequence $\{(t_{s1}, t_{e1}), (t_{s2}, t_{e2}), \ldots, (t_{sn}, t_{en})\}$. Each element represents a timestamp pair of matched send and echo packets. In the following sections, for convenience, we usually use array $T[1, 2, \ldots, n]$ to represent a RTT-thumbprint.

The length of a RTT-thumbprint (n) is vital in tracing intruders; how to select the size depends on the network. For a local network, n can be relatively small because of less network fluctuation, such as 64. But for a wide area network, n should be larger because of serious network fluctuation. A thumbprint for an incoming connection is called an incoming RTT-thumbprint, denoted as iRTT-thumbprint, and a thumbprint for an outgoing connection is called an outgoing RTT-thumbprint, denoted as oRTT-thumbprint. For convenience, we usually use RTT-thumbprint to represent both incoming and outgoing thumbprints.

The first step of creating an RTT thumbprint is to match send packets with echo packets; the algorithm to do so can be found in [Yan05]. The issue about RTT-thumbprint is how to determine if two RTT-thumbprints match. For example, in some cases, we need to determine if the iT-thumbprint is included in the oT-thumbprint of the same host, or to determine if the oT-thumbprint of one host is included in the oT-thumbprint of another host so as to decide if the two hosts are in the same connection chain. We use SR to determine if two RTT-thumbprints are similar. Assuming two RTT-thumbprints $T = \{(t_{s1}, t_{e1}), (t_{s2}, t_{e2}), \ldots, (t_{sn}, t_{en})\}$ and $U = \{(u_{s1}, u_{e1}), (u_{s2}, u_{e2}), \ldots, (u_{sn}, u_{en})\}$, respectively, we determine the longest inclusion sequence of T and U. The problem of the element-inclusion relation incurs a high false positive in some cases. To avoid this

problem and to be aware that if Host i and Host j are in a same chain, the time gap that a packet propagates from Host i to Host j is supposed to approximately equal to the time gap that the corresponding echo packet propagates from Host j to Host i. To be practical, for any given ε between 0 and 1, we usually use the inequality (6.4) to determine element-inclusion upon two corresponding pairs: (t_{si}, t_{ei}), and (u_{sj}, u_{ej}):

$$\Delta_1 > 0, \Delta_2 > 0, \text{ and } \frac{\left||\Delta_1| - |\Delta_2|\right|}{|\Delta_1| + |\Delta_2|} < \varepsilon, \tag{6.4}$$

where $\Delta_1 = u_{sj} - t_{si}$, $\Delta_2 = t_{ei} - u_{ej}$ and supposing there is no clock skew between the two hosts. This is to avoid every small RTT (u_{sj}, u_{ej}) to fall into (t_{si}, t_{ei}). We will match only those that fall somewhere in the middle of the other interval.

6.2.3 Correlating Algorithm

We address issues related to collecting and correlating RTT-thumbprints. When we collect a RTT-thumbprint in a host, the most important and difficult issue is to find the echoed packet for each packet sent. We summarize the reason briefly here.

First, any lost packets during transmission are retransmitted either automatically by the sending client not having received an acknowledgement or on request of the receiving server. Retransmission of the same packet continues until either an acknowledgement is received or until the connection timeout expires. This affects matching a send with an echo packet because we are faced with the case that one echo packet could match two or more send packets. Second, cumulative acknowledgements may take place; this mechanism benefits reducing network traffic, but complicates the packet matching. Third, the size of the transmit window is not necessarily one, so more packets can be allowed to

send continuously without receiving any Ack packet. More Send-Ack-Echo overlapping each other makes packet matching difficult. Finally, the packets only communicating between two adjacent hosts, such as Ignore packet, Keep-alive and Key re-exchange packet, will also make packet matching complex, as well as complicating RTT-thumbprint correlating because these kinds of packets do not propagate to the victim side. In summary, there is no one-to-one mapping between send packets and echo packets, which makes packet-matching difficult.

Another reason that affects RTT-thumbprint correlation is clock skew. We collect each RTT-thumbprint based on the local host clock, which may be different from other host clocks. We cannot directly compare two RTT-thumbprints with each coming from a different host by using relation (6.4) because there may be clock skew; we are not sure how much this is as it is changing all the time. So if we want to correlate two RTT-thumbprints correctly, we need to determine or estimate the clock skew between two hosts first. We discuss the method to estimate clock skew in the following.

Given two RTT-thumbprints $T = \{(t_{s1}, t_{e1}), (t_{s2}, t_{e2}), \ldots, (t_{sn}, t_{en})\}$, and $U = \{(u_{s1}, u_{e1}), (u_{s2}, u_{e2}), \ldots, (u_{sn}, u_{en})\}$, supposing there is no clock skew problem, the exhaustive solution to correlating them is to check if U is included in T by comparing with each element of T using relation (6.4) until all the elements in U have been checked. The problem with this solution is that it takes a long time to get the correlating results. Considering a packet propagation ideal scenario, if T and U are in the same chain, the first element of U is supposed to be included in the first element of T, and so on for others. Even in a non-ideal situation, it is not necessary to compare each element in U with all the elements in T. We propose a heuristic algorithm to correlate RTT-thumbprints; with

this algorithm, instead of comparing each element of U with all the elements in T, we only check N elements from the current position of T (N is a predefined number; in our experiment we selected $N = 4$).

Clock skew is a very important factor for correlating RTT-thumbprints. We propose a method to estimate clock skew between two hosts approximately. Assuming that T and U are RTT-thumbprints collected at Host i and Host j, respectively, and $U[j]$ is included in $T[i]$, we exploit the fact that the timestamp of a send packet in Host i plus the time this packet propagates from Host i to Host j should approximately be equal to the timestamp of the send packet collected in Host j. However, due to the clock skew, there will be some difference. We can estimate the packet propagation time and use that to estimate the clock skew for this pair (i,j):

$$cs = (t_{si} + p_t) - u_{sj}, \tag{6.5}$$

where $p_t = \frac{\Delta_t - \Delta_u}{2}$, $\Delta_t = t_{ei} - t_{si}, \Delta_u = u_{ej} - u_{sj}$, and cs: clock skew.

6.2.3.1 Exhaustive algorithm

For each estimated clock skew, the exhaustive algorithm computes the longest inclusion sequence of two thumbprints. This algorithm is shown in detail as the following:

```
program Exhaustive algorithm
  function main(){
  float sr = 0;
  for (i=0; i<=NumT; i++)
     for (j=0; j<=NumU; j++){
        Δt = tei-tsi; Δu=uej-usj; pt=(Δt-Δu)/2;
        cs = usj-(tsi + pt);
        if (compute_SR(cs)>sr) sr = compute_SR(cs);
     }
  return (sr);
  }
  function Compute_SR(cs)
  CurrPT = CurrPU = 0;
```

```
  while (there are more elements in U and T){
     lbT=CurrPT; lbU=CurrPU; ubT=NumT; ubU=NumU;
     Match=flase; i=lbU;
     while (i<=ubU && !Match){
        j=lbT;
        while (j<=ubT && !Match){
           Δ1=(usj-cs)-tsi; Δ2=tei-(uej-cs);
           ral=abs(abs(Δ1)-abs(Δ2))/(abs(Δ1)+abs(Δ2));
           if (Δ1>0 && Δ2>0 && ral<ε){
              counter++; CurrPT=j+1; CurrPU=i+1;
              Match=true;
           };
           j++;
        }
        i++;
     }
     if (Match) {currPT=j; currPU=i;}
     else {currPT++; currPU++;};
    }
    return SR=counter/min(NumT , NumU);
  }
end.
```

In the algorithm above, U and T are two sequences corresponding to two RTT-thumbprints; 'counter' represents the largest number of elements in U that are included in T; *NumT*, and *NumU* are the lengths of T and U; *lbT* and *lbU* are the lower bounds of T and U; *ubT* and *ubU* are the upper bounds of T, and U.

The premise that we can use Equation (6.5) to estimate the clock skew is that we assume two corresponded elements $T[i]$ and $U[j]$ are generated by the same packet. Since we do not know this, we have to compute all such pairs and find the best solution. The exhaustive method to estimate clock skew is to traverse all the elements in T and U; this means we get nm values to approximate the clock skew. We try each value to compute SR, and the largest one of SRs will be the SR with LIS between T and U. Thus, the exhaustive algorithm has time complexity $O(m^2n^2)$.

6.2.3.2 Heuristic algorithm

The exhaustive algorithm can give us the best solution for correlating RTT-thumbprint, but with the penalty of inefficiency. In most cases it is not necessary to compute all the pairs in T and U to estimate the clock skew. If the two thumbprints are collected at roughly the same time (this can be guaranteed by assumption 2) , most probably $U[i]$ will correspond to one of the elements between $T[i]$ and $T[i+N]$, where N is a predefined number. The heuristic algorithm is similar to the exhaustive one, but the outer loop in the main algorithm is limited to N iterations, as well as the loop in the function *Compute_SR (cs)*. We will not show this algorithm here because the two are very similar. The experimental results showed that the heuristic algorithm has as good performance as the exhaustive one but is more efficient.

We estimated the number of the basic operations in the heuristic algorithm. There are mN estimated clock skews. For each one, we correlate T and U with $O(mN)$ basic operations. Therefore, we have $O(m^2N^2)$ basic operations altogether. The ratio between the heuristic computation and the exhaustive computation is $O(\frac{m^2N^2}{m^2n^2}) = O(\frac{N^2}{n^2})$. If we select N=4, and n=64, the computation time of the heuristic algorithm is only 0.39% of that of the exhaustive algorithm. One experimental result showed that when correlating two thumbprints each with length 64, the heuristic algorithm cost less than one second while the exhaustive one cost hundreds of seconds.

6.2.4 Experimental Results and Analysis

The test environment was set up in the Computer Science Department Lab, at the University of Houston. We made a program *TT* (Thumbprint Traceback) by using Libpcap to simulate tracing intrusions back. There are two hosts Acl08 and Acl09 in local campus domain 'cs.uh.edu' under our control, and there are other hosts Mex (in Mexico), Epic (in California), and Bayou (on the campus) which are not under our control, but we do have regular accounts to login to those machines. We did our experiment conservatively so as to make our results more reliable. We established four connection chains Host *i* (*i*=1 to 4)→ Acl09 → Mex → Acl08 → Epic → Bayou by using SSH along the same path and did the experiment at the same location and the same time, with the same contents as input. This would increase the probability that a connection from one chain was included in another connection from a different chain. This is the worst case scenario that RTT-thumbprint can handle.

Table 6.2 shows one of the RTT-thumbprints correlating results; the first line is for the heuristic algorithm and the second line for the exhaustive algorithm. The value shown on the table is *SR*, which means how many elements in *U* (collected in Acl08) are included in *T* (collected in Acl09). It is clear that two thumbprints of the same connection chain have very high similarity while those not in the same chain, have very low similarity. The results also show us that the heuristic algorithm can be as good as the exhaustive one in *SR*, but the heuristic is much more efficient and can be used in real-time connection traceback.

Table 6.2 RTT-thumbprint exhaustive and heuristic correlating results

Connections at Acl09	Connections at Acl08			
	C_0(%)	C_1(%)	C_2(%)	C_3(%)
C_0(%)	100.00 100.00	1.26 0.00	0.57 0.57	0.52 0.52
C_1(%)	–	92.63 94.66	0.57 0.57	0.00 0.00
C_2(%)	–	–	87.42 91.25	0.62 0.62
C_3(%)	–	–	–	89.00 93.71

6.2.5 Discussion of Resistance to Intruder Evasions

RTT-thumbprint is more robust than other methods used to trace back intruders in resisting intruder's evasion. Most probably, intruders who are aware of the risks of being traced try to evade the traceback by modifying their connections. To defeat the RTT-thumbprint, they may randomly delay the outgoing packets or randomly inject some packets into the connection and so that the relayed outgoing and incoming connections appear unrelated.

Random delay manipulation cannot affect RTT-thumbprint to trace intruders back. We make an assumption that each packet can only be delayed, not accelerated. Suppose all the packets in Host *j* are delayed by an intruder, u_{sj}, and u_{ej} are bigger. No matter how large u_{sj} is, it is always larger than t_{si}, and therefore relation (6.4) holds all the time for send packets. Echoed packet e_i in Host *i* is always later than the echo packet e_j in Host *j*, so t_{ei} is always bigger than u_{ej} and therefore relation (6.4) holds no matter how an intruder delays the packets in Host *j*. RTT-thumbprint can completely resist intruders' random delay.

Chaff manipulation cannot affect RTT-thumbprint to trace back intruders too much. We first assume that any chaff will be removed shortly before reaching the destination. We assume an intruder inserts some packets in Host j, and then removes them in Host $j+k$ (here k is any integer that guarantees Host $i+k$ is not the destination host). Because all the packets inserted in Host j do not reach the destination, we cannot capture the matched pair for the chaff either in Host i or Host j. Obviously, in this case chaff does not affect the performance of RTT-thumbprint to trace intruders back. The only effect is to make downstream and upstream propagation time unsymmetrical, and then affect RTT-thumbprint correlating. But we can handle this problem by adjusting ε. Therefore, RTT-thumbprint can resist intruders' chaff evasion to a certain degree. This point needs to be further studied.

6.3 Tracing Stepping-stone Intrusion Back

There are two ways to trace intruders back by using T-thumbprint or RTT-thumbprint. We use T-thumbprint as an example to explain our traceback framework. One is to use oT-thumbprint only to do traceback. The other is to combine oT-thumbprint and iT-thumbprint together to trace back. Figure 6.8 shows the scenario in which several hosts are connected by one connection chain $<C_1, C_2, C_3, C_4>$, and where T-thumbprint$_i$ represents the thumbprint of corresponding connection C_i. It is assumed that the attacker connects to Host 1 first, and eventually connects to the victim.

The first way to trace back is to use oT-thumbprint only. We have oT-thumbprint$_4$ at Host 3 for outgoing connection C_4. What we need to do first of all at Host 3 is to request all outgoing T-thumbprints of each upstream host that connect to Host 3 directly. The

second step is to correlate oT-thumbprint$_4$ with all other oT-thumbprints requested at Host 3 to decide which connection is in the same chain as C_4. We can do the same thing in Host 2 as that we did in Host 3 to traceback which connection among all the incoming connections of Host 2 is in the same chain with C_3. Recursively, we will eventually trace back to the intruder only with outgoing T-thumbprint. The problems with this way are inefficiency, overloading the network, and difficulty in synchronizing oT-thumbprints. It is inefficient because it needs to compare all the outgoing T-thumbprints of each host related to each incoming connection. It overloads the network because this method needs to transfer many oT-thumbprints over the network. It is difficult to synchronize oT-thumbprints because we cannot guarantee the oT-thumbprint transferred is in the same time interval with the local oT-thumbprint. To overcome the shortcomings of this method, there is another way to trace intrusions back.

The second way is to combine incoming T-thumbprints with outgoing T-thumbprints to trace back intruders. Unlike the previous method, it is not necessary to transfer oT-thumbprints over a network. In the first step one needs is to correlate oT-thumbprint$_4$ with all of the iT-thumbprints at Host 3 to decide which incoming connection is in the same chain as C_4. The second step is to request Host 2 which incoming connection of Host 2 is in the same chain as C_3. We do the same thing recursively and eventually get the intruder location. The main advantage of this method is we can guarantee the iT-thumbprint and oT-thumbprint are in the same time interval because we use the same process on the same host to collect the incoming and outgoing T-thumbprints at the same time intervals.

Another important issue is that we can trace intrusions back with T-thumbprint in real time. T-thumbprint is not large. Typically, it is 8x4 bytes long on a local area network, and 128x4 bytes long on the Internet. T-thumbprints could be generated and correlated within one second. So it is possible to trace intruders back in real time by using T-thumbprint.

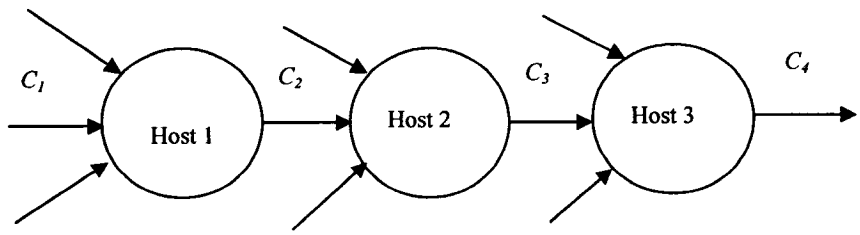


Figure 6.8 Illustration of using T-thumbprint to trace intrusions back

7. CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

Our work mainly consists of four focal parts. The first is to detect stepping-stones by examining the incoming and outgoing connections. The second is to detect stepping-stone intrusion by estimating the length of the connection chain. The third is to match TCP/IP packets; the matching approaches are used in many of our algorithms. The fourth is to trace intrusions back. We distinguish detecting stepping-stones from detecting stepping-stone intrusion. We are the first to detect stepping-stone intrusion by matching TCP/IP packets.

We proposed to detect stepping-stones by checking TCP request-response traffic. We found that the difference between the number of requests of an outgoing connection and the number of responses of an incoming connection is a random walk process. If the two connections are relayed, this random walk process is bounded no matter whether the two connections are manipulated or not. Our approach to detect stepping-stone was to monitor this difference to see if it always walks within a range. The false alarm rate and misdetection rate really depend on the number of packets captured. The higher the number of packets captured, the lower the false alarm rate and the misdetection rate. We derived formulas to compute the least number of packets to be monitored in order to obtain a given false positive rates or false negative rates. The analysis shows that our approach is better than Blum's; the latter has been considered to be the best approach in

detecting stepping-stones and resisting intruders' evasion, in terms of the number of packets needed with a given false positive rate even if the connections are manipulated. One disadvantage with our method is that it is difficult to determine a set of boundaries for all cases.

We developed two approaches to detect stepping-stone intrusion, the step-function and the network fluctuation-based approaches. Using step-function to detect stepping stone intrusion exploits the idea that if we monitor a connection chain and collect all the Sends and Echoes from the start to the end, all the RTTs form steps with each step corresponding to one compromised host. That is, if we count the steps, the length of a connection chain in terms of connections can be estimated. This length indicates the number of hosts compromised by a user. The longer the number of connections, the higher the probability the user is an intruder. The experimental results showed that this approach has a low false positive rate for detecting stepping-stone intrusion. However, this approach has the problem of step aggregation [Yan05-1] which means if two steps are close together, it may not be able to distinguish the levels. Another approach we proposed to detect stepping-stone intrusion uses network fluctuation to estimate the length of a connection chain. The basic idea is the longer a connection chain, the higher the fluctuation of RTTs. This approach mainly focuses on solving the problem of step aggregation in the step-function approach, as well as on solving the yardstick problem in Yung's method [Yun02]. The experimental results on the Internet showed that this method can be used to a certain degree to determine if there is a stepping-stone intrusion. Compared with the previous approaches, this method has the ability to handle the step

aggregation problem, and the biased yardstick problem to some degree; its results are more monotonic than Yung's, and it can be implemented easily and executed efficiently.

The key issue of the two approaches above for detecting stepping-stone detection is to compute RTTs by matching TCP/IP packets. The third part of our work is packet-matching algorithms. We proposed three methods to match TCP/IP packets: the TCP/IP protocol-based matching approach; the clustering-partitioning matching approach; and the standard deviation-based matching approach. The first approach can be applied when we need to match packets in a real-time situation. The second approach can be applied when we do not need to match packets in a strictly real-time situation. The third approach can be applied to a situation in which the length of a connection chain is relatively stable. Its result is mainly used by the network fluctuation-based approach to detect stepping-stone intrusion. Because of strict real-time conditions, the TCP/IP protocol-based approach cannot get both high matching-rate and high matching-accuracy. It has a tradeoff between the packet matching-rate and the packet matching-accuracy.

Unlike this method to match packets locally, the clustering-partitioning and standard deviation-based approaches match packets globally because of the relaxed time restriction. The two matching approaches can achieve a good balance between high matching-rate and high matching-accuracy. The disadvantage of the clustering-partitioning matching approach is that the selection of parameters affects the matching results. The advantage of this approach is that it can be applied to match packets in multiple levels of RTTs. The advantage of the standard deviation-based approach is its

matching result does not depend on any parameters, but it is only suitable for computing a single level of RTTs.

We developed two thumbprints which can be used to trace intrusions back. One is the T-thumbprint, another one is the RTT-thumbprint. The basic idea of the T-thumbprint is to use the gaps between consecutive send packets to identify a connection. We proposed an algorithm to correlate two such T-thumbprints. The simulation results showed that the similarity between two relayed connections is much higher than that between non-relayed connections. A disadvantage of the T-thumbprint is that it is weak in resisting intruders' evasion. The RTT-thumbprint can do a better job in this case. The basic idea of the RTT-thumbprint is to make use of the fact that the timestamps of a send-echo pair of a downstream host must be included in the corresponding pair of an upstream host. By computing the inclusion rate of two sequences containing send-echo pairs between different hosts, it is possible to detect if the two hosts are relayed. The advantage of this method is that it can resist intruders' time perturbation; it also resists intruders' chaff perturbation to a certain degree, and is better than other existing approaches.

7.2 Future Work

We have proposed some approaches to detect stepping-stones and intrusion. But the problem of detecting stepping-stone intrusion is far from being completely solved. The research presented in this dissertation needs more work in several areas.

1) In the network fluctuation-based stepping-stone intrusion detection approach, we use the standard deviation of RTTs to characterize network fluctuation. There is

a need to find a better way to represent the variation of RTTs, as well as network traffic. The RTTs of a TCP session can be viewed as a signal. If we use signal processing technique to analyze the RTT information, probably we could find a better way to characterize the RTTs. For example, we tried to apply Wavelet and Fourier transformation to analyze the RTT signal. Unfortunately we did not make much progress in this approach. But more studies are still needed in this area.

2) We have developed three approaches to match TCP/IP packets, but each approach has its own limitation. More work is still needed to improve the packet-matching algorithm to make it possible to match multilevel RTTs without depending on any predefined parameters.

3) In the RTT-thumbprint correlation efficient algorithm, we assume that the two sequences have the same starting point. A modified algorithm to find the longest similar subsequence of RTT-thumbprints is important and still necessary to develop. A real traceback system based on RTT-thumbprints also needs development. How to make the RTT-thumbprint more stable in resisting intruders' evasion is also significant and critical.

4) All the approaches we developed to detect stepping-stone intrusion can only resist intruders' evasion to a certain degree. To develop an algorithm which can better resist intruders' time and chaff perturbation is still significant in terms of detecting and preventing stepping-stone intrusion.

5) An unsolved problem in estimating the connection chain length is upstream detection. Currently we detect stepping-stone intrusion by estimating the downstream length which depends on where the sensor is. Had the sensor been very close to the victim side, it would have been impossible to detect stepping-stone intrusion without introducing false negatives. The way to solve this problem is to estimate the length of whole a connection chain from the intruder's site to the victim's site. A good estimation of the upstream connection length can significantly improve the performance of our intrusion detection method.

6) The purpose of matching TCP/IP packets is to compute the RTTs of a session. We are working on developing a way to compute the RTTs directly by applying queuing theory to a TCP session. From the point of a sensor (one of the stepping-stones), a downstream connection chain can be simulated by a queue. The input of the queue is the requests from the sensor while the output is the responses. The RTT for each request is the time that the request stays in the queue until there is a response. It would be difficult to estimate this time gap, but the average queuing time for some requests would be more easily estimated by applying queuing theory. More study is needed to focus on this idea.

BIBLIOGRAPHY

[Ber92] D. Bertsekas, R. Gallager, "*Data Networks*," Englewood Cliffs, NJ: Prentice Hall Inc., 1992.

[Bis02] M. Bishop, "*Computer Security-Art and Science*," Boston, MA: Pearson Education, Inc., 2002.

[Blu04] A. Blum, D. Song, S. Venkataraman, "Detection of Interactive Stepping-Stones: Algorithms and Confidence Bounds," *Proceedings of International Symposium on Recent Advance in Intrusion Detection*, pp. 20-35, Sophia Antipolis, France, September 2004.

[Bor90] D. Borman, "Telnet Linemode Option," RFC 1184, Network Information Center, SRI International, Menlo Park, CA, October 1990.

[Cer05] CERT, http://www.cert.org, July 2005.

[Che95] S. Staniford-Chen, L. T. Heberlein, "Holding Intruders Accountable on the Internet," *Proceedings of the 1995 IEEE Symposium on Security and Privacy*, pp. 39-49, Oakland, CA, May 1995.

[Cla03] M. P. Clark, "*Data Networks, IP and the Internet Protocols, Design and Operation*," West Sessex, England: John Wiley & Sons, 2003.

[Cox02] I. J. Cox, M. L. Miller, J. A. Bloom, "*Digital Watermarking*," San Francisco, CA: Morgan-Kaufmann Publishers, 2002.

[Cox65] D. Cox, H. Miller, "*The Theory of Stochastic Process*," New York, NY: John Wiley & Sons Inc., 1965.

[DNWS04] Data Nerds Web Site, "Winpcap and Windump," http://www.datanerds.net, July 2004.

[Don02] D. L. Donoho, A. G. Flesia, et al., "Multiscale Stepping-Stones Detection: Detecting Pairs of Jittered Interactive Streams by Exploiting Maximum Tolerable Delay," *Proceedings of International Symposium on Recent Advance in Intrusion Detection* , pp. 49-64, Zurich, Switzerland, 2002.

[Fel68] W. Feller, "*An Introduction to Probability Theory and Its Applications*," Volume I, New York, NY: John Wiley & Sons, Inc., 1968.

[For02] B. Forouzan, "*TCP/IP Protocol Suite*," (Second Edition) Columbus, OH: McGraw-Hill, 2002.

[Fri99] M. Friedman, A. Kandel, "*Introduction to Pattern Recognition: Statistical, Structural, Neural, and Fuzzy Logic Approaches*," River Edge, London: NJ World Scientific Publishing Co., 1999.

[Fro94] V. Frost, B. Melamed, "Traffic Modeling for Telecommunications Networks," *IEEE Communications Magazine*, 32(3): 70-80, 1994.

[Gol06] D. Gollmann, "*Computer Security*," West Sussex, England: John Wiley & Sons Ltd., 2006.

[Hea90] R. Heady, G.Luger, A. Maccabe, M. Servilla, "The Architecture of a Network Level Intrusion Detection System," *Technical Report,* Department of Computer Science, University of New Mexico, August 1990.

[Heb92] L. T. Heberlein, K. Levitt, B. Mukherjee. "Inter-network Security Monitor: An Intrusion-Detection System for Large-Scale Networks," *Proceedings of 15th National Computer Security Conference*, pp. 262-271, San Jose, CA, October 1992.

[Jai88] A. Jain, R. Dubes, "*Algorithms for Clustering Data*," Englewood Cliffs, NJ: Prentice Hall, Inc., 1988.

[Joh70] N. I. Johnson, S. Kotz, "*Continuous Univariate Distributions*-1," New York, NY: John Wiley & Sons, Inc., 1970.

[Jun93] H. Jung, et al., "Caller Identification System in the Internet Environment," *Proceedings of 4th USENIX Security Symposium*, pp. 69-78, Santa Clara, October 1993.

[Kao96] E. Kao, "*An Introduction to Stochastic Processes*," New York, NY: Duxbury Press, 1996.

[KFG93] C. Kao, D. A. Frincke, T. Goan Jr., et al., "Analysis of an Algorithm for Distributed Recognition and Accountability," *Proceedings of the 1st ACM Conference on Computer and Communication Security*, pp. 154-164, Fairfax, VA, 1993.

[Kiz05] J. M. Kizza, "*Computer Network Security*," New York, NY: Springer Science + Business Media, Inc., 2005.

[Kru05] C. Kruegel, F. Valeur, G. Vigna, "*Intrusion Detection and Correlation*," New York, NY: Springer Science + Business Media, Inc., 2005.

[LBNL04] Lawrence Berkeley National Laboratory (LBNL), "The Packet Capture Library," ftp://ftp.ee.lbl.gov/libpcap.tar.z, March 2004.

[Li98] Q. Li, D. L. Mills, "On the Long-range Dependence of Packet Round-trip Delays in Internet," *Proceedings of International Conference on Communications*, Vol. 1:pp. 1185-1192, Atlanta, GA June 1998.

[Mai03] E. Maiwald, "*Network Security-A Beginner's Guide*," Emeryville, CA: MacGraw-Hill/Osborne, 2003.

[Mal03] S. Malik, "*Network Security Principles and Practices*," Indianapolis, IN: Cisco Press, 2003.

[Mar01] D. J. Marchette, "*Computer Intrusion Detection and Network Monitoring*," New York, NY: Springer-Verlag New York, Inc., 2001.

[Mir96] B. Mirkin, "*Mathematical Classification and Clustering*," Dordrecht, The Netherlands: Kluwer Academic Publishers, 1996.

[Pax95] V. Paxson, S. Floyd, "Wide-area Traffic: The Failure of Poisson Modeling," *IEEE/ACM Transactions on Networking*, 1995 (3):226-244, 1995.

[Pen05] P. Peng, P. Ning, D. Reeves, X. Wang, "Active Timing-Based Correlation of Perturbed Traffic Flows with Chaff Packets," *Proceedings of 25th IEEE International Conference on Distributed Computing Systems Workshops*, pp. 107-113, Columbus, OH, June 2005.

[Pro01] P. E. Proctor, "*Practical Intrusion Detection Handbook*," Upper Saddle River, NJ: Prentice-Hall, Inc., 2001.

[RFC81] University of Southern California, "Transmission Control Protocol," RFC 793, 1981.

[Sav01] S. Savage, D. Wetherall, A. Karlin, T. Anderson. "Network Support for IP Traceback," *ACM/IEEE Transactions on Networking*, 9(3): 226-237, June 2001.

[Sno02] A. Snoeren, C. Partridge, L. Sanchez, C. Jones, F. Tchakountio, B. Schwartz, S. Kent,W. Strayer, "Single-Packet IP Traceback," *ACM/IEEE Transactions on Networking*, 19(6): 721-734, December 2002.

[Sol05] M. G. Solomon, M. Chapple, "*Information Security Illuminated*," Sudbury, MA: Jones and Bartlett Publishers, 2005.

[Sna91] S. Snapp et al., "DIDS (Distributed Intrusion Detection System)-Motivation, Architecture, and Early Prototype," *Proceedings of 14th National Computer Security Conference*, pp. 167-176, Columbus, OH, October 1991.

[TCS85] Department of Defense Standard, "Department of Defense Trusted Computer System Evaluation Criteria," *http://csrc.nist.gov/publications/history/dod85.pdf*, August 1985.

[USC81] University of Southern California, "Transmission Control Protocol", RFC 793, September 1981.

[Wan01] X. Wang, D. S. Reeves, S. F. Wu, J. Yuill, "Sleepy Watermark Tracing: An Active Network-based Intrusion Response Framework," *Proceedings of 16th International Conference on Information Security*, pp. 369-384, Paris, France, June 2001.

[Wan02] X. Wang, D. Reeves, and S. Wu, "Inter-Packet Delay-based Correlation for Tracing Encrypted Connections through Stepping Stones," *Proceedings of 7th European Symposium on Research in Computer Security, Lecture Notes in Computer Science*. Vol. 2502, pp. 244-263, Zurich, Switzerland, October 2002.

[Wan03] X. Wang, D. S. Reeves, "Robust Correlation of Encrypted Attack Traffic through Stepping-Stones by Manipulation of Inter packet Delays," *Proceedings of the 10th ACM Conference on Computer and Communications Security*, pp. 20-29, Washington DC, October 2003.

[Wan04] X. Wang, "The Loop Fallacy and Serialization in Tracing Intrusion Connections Through Stepping Stones," *Proceedings of the 2004 ACM Symposium on Applied Computing*, pp. 404-411, Nicosia, Cyprus, March 2004.

[Wan06][Yan06-1] M. Wan, S. Huang, J. Yang, "Finding the Longest Similar Subsequence of Thumbprints for Intrusion Detection," *Proceedings of 20th IEEE International Conference on Advanced Information Networking and Applications*, Vol. 1: 255-260, Vienna, Austria, April 2006.

[Yan04] J. Yang, S. Huang, "A Real-Time algorithm to Detect Long Connection Chains of Interactive Terminal Sessions," *Proceedings of 3rd ACM International Conference on Information Security*, pp. 198-203, Shanghai, China, November 2004.

[Yan05] J. Yang, S. Huang, "Matching TCP Packets and Its Application to the Detection of Long Connection Chains," *Proceedings of 19th IEEE International Conference on Advanced Information Networking and Applications*, Vol. 1: 1005-1010, Taipei, China, March 2005.

[Yan05-1] J. Yang, S. Huang, "Characterizing and Estimating Network Fluctuation for Detecting Interactive Stepping-Stone Intrusion," *Proceedings of International Conference on Communication, Network and Information Security*, pp. 70-75, Phoenix, AZ, November 2005.

[Yan05-2] J. Yang, S. Huang, "A Way by Estimating the Variation of TCP Packet Round-Trip Time to Detect Stepping-Stone Intrusion (Extended Abstract)," *the web site of the 21st Annual Computer Security Applications Conference*: http://www.acsac.org/2005/techblitz/yang.pdf, Tucson, AZ, December 2005.

[Yan05-3] J. Yang, S. Huang, "Correlating Temporal Thumbprint for Tracing Intruders," *Proceedings of 3rd International Conference on Computing, Communications and Control Technologies*, pp.236-241, Austin, TX, July 2005.

[Yan05-4] J. Yang, S. Huang, "Improved Thumbprint and Its Application for Intrusion Detection," *Lecture Notes in Computer Science (LNCS)* by Springer-Verlag, Vol. 3619: 433-442, 2005.

[Yan06] J. Yang, S. Huang, M. Wan, "A Clustering-Partitioning Algorithm to Find TCP Packet Round-Trip Time for Intrusion Detection," *Proceedings of 20th IEEE International Conference on Advanced Information Networking and Applications*, Vol. 1: 231-236, Vienna, Austria, April 2006.

[Ylo96] T. Ylonen, "SSH—Secure Login Connections Over the Internet," *Proceedings of 6th USENIX Security Symposium*, pp. 37-42, San Jose, CA, July 1996.

[Ylo04-1] T. Ylonen, "SSH Protocol Architecture," draft –IETF document, http://www.ietf.org/internet-drafts/draft-ietf-secsh-architecture-16.txt, June, 2004.

[Ylo04-2] T. Ylonen, "SSH Transport Layer Protocol," draft IETF document, http://www.ietf.org/internet-drafts/draft-ietf-secsh-transport-18.txt, June 2004.

[Yod00] K. Yoda, H. Etoh, "Finding Connection Chain for Tracing Intruders," *Proceedings of the 6th European Symposium on Research in Computer Security,* pp. 191-205, Toulouse, France, October 2000.

[Yun02] K. H. Yung, "Detecting Long Connecting Chains of Interactive Terminal Sessions," *Proceedings of International Symposium on Recent Advance in Intrusion Detection*, pp.1-16, Zurich, Switzerland, September 2002.

[Zha00] Y. Zhang, V. Paxson. "Detecting Stepping Stones," *Proceedings of the 9th USENIX Security Symposium*, pp 67-81, Denver, CO, August 2000.