Modeling Network Traffic via Identifying Encrypted Packets to Detect Stepping-stone Intrusion under the Framework of Heterogonous Packet Encryption

Jianhua Yang*, Noah Neundorfer, and Lixin Wang

Columbus State University, Columbus, Georgia, USA
{yang_jianhua, noah_neundorfer, wang_lixin}@columbusstate.edu

Received: July 20, 2022; Accepted: September 24, 2022; Published: November 30, 2022

Abstract

Exploiting stepping-stones to launch attacks has been widely used by most professional attackers. There are two reasons for doing this: first, it is hard to detect intrusion via stepping-stones; second, even though such intrusions can be detected, it is almost impossible to capture the intruders. There have been many algorithms developed to detect stepping-stone intrusion. In this paper, we propose a novel approach to detect stepping-stone intrusion by modelling and identifying encrypted network traffic of a host. Commonly, attackers use Secure Shell (SSH) to hide their identity. SSH securely connects two hosts together and encrypts their interactions. One connection of SSH leads to another on a different host, and again until the attacker becomes untraceable from a victim host. The previous work detecting a stepping-stone by viewing the lengths of encrypted packets assumed that the encryption algorithm used for both encryptions would remain the same. In this research, we explore an algorithm to determine if a host is used as a stepping-stone by focusing on the length sequences of incoming and outgoing packets under heterogeneous encryption algorithms. The performance of the algorithm proposed was assessed by experiments over the Internet. The results show that the average match rate for relayed connections was in the range of 94% to 96%, but for un-relayed connections, the average match rate never rose above 15%, and was often much lower.

Keywords: Stepping-stone, Intrusion detection, Modelling network traffic, Heterogeneous packet encryption

1 Introduction

Due to the emerging of computer networks, and especially the Internet that was open to the public use in the early of 1990s, remotely accessing a computing host became much easier than before. This change brings us benefits and convenience in terms of exploiting computing system, but also put most of the servers under the risk of attacking. Since the end of 1990s, more and more attackers, esp. professional hackers, launched their attacks via compromised computing hosts. One obvious benefit of doing so is to avoid detection and capturing. In this type of attack, attackers make a TCP connection chain to connect the comprised computing hosts. Attackers launch their attacks to victims behind such a connection chain. The longer a connection is, the better protection provided. The computer hosts compromised by attackers to make a TCP connection chain are called stepping-stones [1]. The attacks launched via stepping-stones are called stepping-stone intrusion (SSI).

Journal of Internet Services and Information Security (JISIS), volume: 12, number: 4 (November), pp. 56-73
DOI:10.58346/JISIS.2022.I4.004

*Corresponding author: TSYS School of Computer Science, Columbus State University, 4225 University Avenue, Columbus, GA, USA 31907, Tel: 001-706-507-8180, Web: http://csc.columbusstate.edu/yang

Since 1995, especially after 2000, many algorithms have been proposed and developed to detect stepping-stone intrusions. The methods proposed to detect stepping-stone intrusion can be divided into two categories: host-based, and network-based. The basic idea of host-based methods is to decide if a host is used as a stepping-stone by monitoring and comparing its incoming and outgoing network traffic. In other words, if the incoming network traffic of a host matches with any outgoing traffic, the computing host is highly suspicious to be compromised as a stepping-stone. Of course, this may introduce false-positive errors because some legal network applications may make use of one or two stepping-stones. The basic idea of network-based stepping-stone intrusion detection (SSID) is to estimate the length of the TCP connection chain between attackers and victims. If the number of compromised hosts, which is also the length of a TCP connection chain, is more than three, it is highly suspicious that the connection chain is used for SSI. The reason that number "Three" is used by most researchers is we found that most legal network applications rarely use three or more hosts to access its targets. One common sense is that the longer a connection chain is, the slower the computer network communication. So legal applications rarely access a remote server via a connection chain involving three or more than three hosts due to inefficient network communication.

Network-based SSID methods include the following approaches. The first approach is to estimate the length of a downstream connection chain via processing network packets proposed by K. Yung, [2] in 2002. The second one is the step-function proposed by J. Yang, etc. [3] in 2004. The third one is the clustering-partitioning approach proposed by J. Yang [4] in 2007. The fourth approach is to mine network traffic with the k-Means to estimate the length of a TCP connection chain [5] proposed by L. Wang, etc., in 2021. The most recent one is the algorithm to estimate the length of a downstream connection using packet crossover [6] proposed by L. Wang, etc.

K. Yung [2] proposed an approach to detect stepping-stone intrusion by estimating the length of a downstream connection chain. His idea is to use the length of the connection chain from the sensor to its adjacent connected host to estimate the length of the connection from a sensor to the target host. We assume Host i is the sensor where a detection program resides. The objective of the algorithm is to estimate the length of the connection from Host i to the target Host n. If a Send packet is sent from Host i to Host n via Host i+1, the packet must be acknowledged first by Host i+1 which is immediately following Host i in the connection established by intruders. We compute the RTT (round-trip time) between the Send packet and its corresponding ACK from the adjacent host along the connection chain. We denote this one as ACK-RTT. The Send packet arrives at Host n which is the end of the connection chain, then be echoed. When the Echo packet is received by the sensor, it is trivial to compute the RTT between the Send and the Echo packet which is denoted as ECHO-RTT. The ratio between ACK-RTT and ECHORTT can approximately estimate the length of the downstream connection chain from the sensor. The more the ratio is close to zero, the longer the downstream connection chain is.

K. Yung's method cannot estimate the length of a connection accurately. The measure bar ACK-RTT may introduce false negative or positive errors. If ACK-RTT is large enough to dominate the length of the whole connection chain, the ratio between ACK-RTT and ECHO-RTT would be large enough to make us conclude that there is no stepping-stone at all even though there are more than three connections in the connection chain. On the contrary, if ACK-RTT is very small, even though there is only one connection in the chain, the ratio between ACK-RTT and ECHO-RTT might be small which leads to a stepping-stone intrusion detected.

The best way to detect stepping-stone intrusion is to estimate the length of the downstream connection chain accurately. Using step-functions is one way to estimate the length of a connection chain in a LAN accurately. In a LAN, packets captured are easy to be matched since there is no packets crossover. Each request can match its immediate following response. If we can monitor a connection chain from the beginning which has only one connection to the time which has n connections, we match all the Send packets with all the Echo packets and compute all the corresponding RTTs. We found that all the RTTs

can form different steps if we put them in a two dimensional coordinate system.

Step-functions can work well in a LAN. It cannot get good performance in the Internet context. Most intruders use the Internet to launch their attacks through stepping-stones. The clustering-partitioning algorithm [4] can detect stepping-stone in the Internet through accurately estimating the length of a long connection chain.

The idea behind this algorithm is that the RTTs of different Send packets, from connection chains with the same length must be bounded and clustered around one center. If we monitor a connection chain from the beginning to the end and collect all the Send and Echo packets, we know it cannot directly match each Send with its immediate following Echo due to requests and responses crossover. But it is clear that each Send must match with one Echo after the Send packet. Even though we do not know which Echo is the one to match which Send, we can compute all the time differences between the Send and all the Echoes after the Send. One of the differences must be the RTT we seek. If we do the same computation for each Send, we get a time difference set. If we mine the set, the RTTs representing different length of the connection chain could be partitioned in different clusters.

In 2021, Dr. L. Wang, etc. proposed an efficient approach to estimate the length of connection chain by mining network traffic using the k-means clustering. This method overcomes then existing approaches for connection-chain-based SSID either are not effective or require a large number of TCP packets to be captured and processed. The algorithm proposed in [5] can accurately determine the length of a connection chain without requiring a large number of packets collected. This method does not work effectively if there are many outliers in the packets' RTTs.

In [6], Dr. L. Wang, etc., proposed to make use of packet crossover to estimate the length of the downstream connection of a connection chain. Packet crossover is a defined as a phenomenon in which a new Send (request) packet meets an Echo (response) packet of a previous Send packet along the connection chain between a client host and a server host. The proposed detection algorithm does not require a large number of TCP packets captured and processed, and thus it is efficient. It was verified in the paper that the length of a downstream connection chain strictly increases with the packet crossover ratio.

Host-based SSID methods include many different approaches. The algorithm proposed in this paper belongs to Host-based SSID. We will summarize briefly the host-based SSID approaches developed since 1995. The initial idea of the algorithm proposed in this paper was first published in the proceedings of 36th International conference version on advanced information networking and applications. In this paper, we present the improvements we did from the conference version.

In 1995, Staniford Chen [7] proposed to use "content-thumbprint" of the incoming and outgoing traffic of a host to determine if the host is used as stepping-stone. A content-thumbprint is a 24-byte signature of the payload of a packet regardless the payload size. Different payloads may share the same signature if the hash function is not well designed. This may incur collision issue and bring false-positive errors. Fortunately, Mr. Chen carefully designed the hash function to avoid the collision issue. His method does not work for encrypted sessions, such as a TCP connection chain made by using OpenSSH. Dr. Y.Zhang and V. Paxson proposed a "time-thumbprint" [1] approach for SSID in 2000. A time-thumbprint is generated based on packets' timestamps which are determined by the clock of a computing host, and it is independent of packets' payloads. This method applies to encrypted sessions for SSID. However, a time-thumbprint may be affected by intruders' manipulation, such as time-jittering and/or chaff-perturbation. K. Yoda et al. [8] proposed a session deviation-based detection approach by setting up monitors for packets at many nodes on the Internet to store attackers' activities. It suffers from the similar issues as the time-thumbprint method. In 2004, Dr. A. Blum et al. proposed a Detect-Attacks-Chaff SSID (DAC SSID) [9] by counting the number of packets of a connection. Dr. Blum employed the idea from Computational Learning Theory and conducted analysis using the concept of random walks. Blum's method for SSID used the idea that two sessions are relayed if and only if the difference between the packet numbers in the two sessions is bounded above. Blum's method does not work effectively in

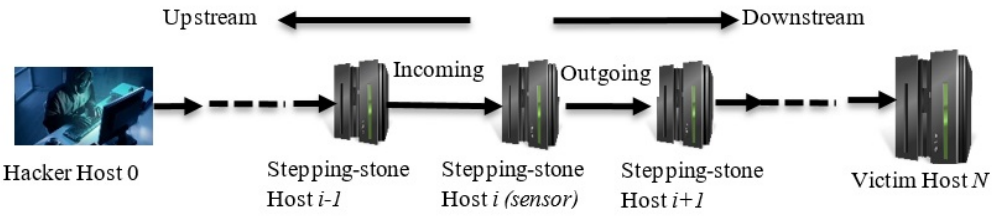

Figure 1: An example connection chain

resisting to intruders' evasion manipulation with chaff-perturbation as the upper bound for the number of monitored packets is huge.

Dr. J. Yang, et al. proposed to employ random walk for SSID and resist attackers' session manipulation [10]. Two improved approaches based on random walk were proposed in 2015, and 2016 respectively. One is RTT-based random walk algorithm to detect stepping-stone intrusion [11], another one is packet cross-matching with RTT-based random walk for SSID [12]. All the above three algorithms can resist intruders' chaff-perturbation to a certain degree. Dr. L. Wang, et al. proposed a framework to test resistency of detection algorithms for stepping-stone intrusion on time-jittering session manipulation [13].

In 2021, Dr. J. Yang, et al. proposed an approach of modelling network traffic and exploiting encrypted packets to detect stepping-stone intrusion [14]. In order to exploit the encrypted packets of the incoming and outgoing connection of a host, Dr. J. Yang, et al. assumed that the two connections use the same encryption algorithm, which may not always be true. Dr. J. Yang improved the algorithm published in [14] to make it work regardless if the encryption algorithms from incoming and outgoing connections respectively are the same or not. The initial result was published in the proceedings of 36th International Conference on Advanced Information Networking and Applications, Sydney, Australia, April, 2022 [15]. In this paper, we redesign the algorithms to "Find Potential Sequence Match", and "Select Best Matches" to improve the SSI detection rate. We introduce two important skills to improve the SSID performance: Crossover and Shorten. "Crossover" is used to determine if a potential range overlaps with an already selected Range. "Shorten" method can reduce the size of a range until it no longer has crossover, potentially reducing the range to empty. In addition to the above updates, more detail testing results to verify the proposed approach are presented in this paper.

The rest of the paper is laid out as the following. Preliminaries will be given in Section 2. In Section 3, we present the way to identify an encrypted packet. We will discuss the approach to model network traffic in Section 4. The detection algorithm details are introduced in Section 5. Experimental results and its analysis are presented in Section 6. We conclude the whole paper and point out the future research work in Section 7.

Professional Hackers' propensity to use stepping-stone intrusion is a well-documented facet of cybersecurity. In our attempts to detect this method of hiding an attacker's IP, it is useful to have an example model of a connection chain to more accurately refer to the parts of it. The figure below is one such model. For this example, where Host 0 is the attacker and Host N is the victim (where malicious activity is likely occurring), Host i is what we consider our "sensor" node, that is, the node where we attempt stepping-stone detection.

1.1 SSH Connection Chain

The SSH protocol allows secure, encrypted connections to a host running an SSH server. Attackers use sequential SSH protocols to create a "chain" of hosts that transmits commands from an attacker to a victim. Each connection in a chain is separately encrypted, and the unencrypted data cannot be accessed at any stage. One benefit for an attacker of a stepping-stone connection is that the attacker's IP is incredibly difficult to trace back from the victim.

Each "stone" in the connection only knows it has an incoming and outgoing SSH connection, but does not know if the connections match, or the IP addresses of stones not adjacent to itself. An attacker who wishes to use stepping-stone intrusion usually will use 3 or more individual "stones", often in geographically diverse locations, to create a very difficult trace path leading back to them.

1.2 Send and Echo Definition

Important topics to discuss for studying SSH connections are TCP headers, IP headers, and SSH packet formation and encryption. Both the TCP and IP header, since they are below the application layer on the OSI model, remain unencrypted, and can be used to match packets.

A TCP header contains the essential information for a reliable connection, like ports, sequence and acknowledgement numbers and other flags. The IP header contains the source and destination IP, as well as other data. Since we know where the TCP header ends, it is trivial to calculate the length of the enclosed data, that is, the length of an SSH packet.

While all SSH packets are essentially created the same, we need to define crucial differences between packets that will be used in this research. The first and most important difference is between Send and Echo packets. When plaintext is enclosed into SSH packets, we then will distinguish them based on where they depart from. A **Send packet is a packet that travels from Host 0 (client side) towards the SSH server of Host N, that is in the direction of the victim. A packet traveling from the SSH server of Host N towards Host 0, or from the victim towards the attacker, is an Echo packet**. An echo packet usually contains a repeat, or confirmation, of the message sent by the corresponding Send packet. So, a Send packet leaves Host 0, travels to Host N, and then a corresponding Echo packet, with the exact same, or different plaintext, is sent back to Host 0 from Host N.

The other type of Echo packet is a "Data-Echo packet"(or just Data packet). This is a packet that returns from the victim host where a command is executed containing information that is a response for the command execution (ex. The plaintext: "Document Downloads Pictures etc.. . . " is contained in the Data-Echo packets returned when the victim host executes an "ls" command). For the purpose of this paper, the term "Echo packet" will always refer to a packet repeating the command from a send packet. The term Data packet or Data-Echo packet will be used to refer to packets containing the response for an executed command.

1.3 Packet Encryption

In SSH, packet encryption is performed using one of several ciphers. Before performing this encryption, the application creates a basic SSH packet. This packet contains several fields. The first is a 4-byte field that contains the length of the packet in bytes. The second is an one-byte field for padding length, containing the length of the padding in bytes, then two fields, payload and padding, of variable length. It may also contain a MAC field.

The Payload field is where the data, or the plaintext, of the message is stored, with each letter of plaintext being stored as a standard one-byte char. The padding field is slightly more complex. A certain amount of random padding is added in order to lengthen the packet to a certain total length. **The amount of padding added should increase the total length of the first four fields combined (excluding a**

Figure 2: The structure of an SSH packet

Text length chacha20-poly1305@openssh.comText length aes128-gcm@openssh.com Text length aes256-gcm@openssh.com 1-2 Characters .Length(36) 1-2 Characters .Length(36) 1-2 Characters .Length(36) 3-10 Characters .Length(44) 3-10 Characters .Length(52) 3-10 Characters .Length(52) 11-18 Characters .Length(52) 11-18 Characters .Length(52) 11-18 Characters .Length(52) 19-26 Characters .Length(60) 19-26 Characters .Length(68) 19-26 Characters .Length(68) 27-34 Characters .Length(68) 27-34 Characters .Length(68) 27-34 Characters .Length(68) 35-42 Characters .Length(76) 35-42 Characters .Length(84) 35-42 Characters .Length(84) 43-50 Characters .Length(84) 43-50 Characters .Length(84) 43-50 Characters .Length(84) 51-58 Characters .Length(92) 51-58 Characters .Length(100) 51-58 Characters .Length(100) 59-66 Characters .Length(100) 59-66 Characters .Length(100) 59-66 Characters .Length(100) 67-74 Characters .Length(108) 67-74 Characters .Length(116) 67-74 Characters .Length(116)

Figure 3: An example of packet lengths across encryption algorithms

MAC) to a multiple of either 8 or the cipher block size, whichever is larger. So if the cipher block size is 16, the total length must be a multiple of 16 (16, 32, 48, etc.). If the cipher block size was 4 or had no block size (see stream cipher's), the total length must be a multiple of 8 (8, 16, 24, 32, 40, etc.). Once the packet is generated in plaintext, it is encrypted using the session key agreed upon by the client and the server. The first four fields must be encrypted, and as it often is, the MAC varies.

Once this packet is fully generated, it is then passed down the OSI model, adding a TCP header, IP header, etc. Then it is sent to the next host along the connection chain. When an encrypted packet is received at a sensor or an intermediate host, the packet is not fully unencrypted down to its plaintext, but a new SSH packet is made, and the payload is moved to the new packet's payload field. New padding is generated, and the packet is encrypted again **using the session key from the next connection**. This means that each individual connection (i-1 to i, and i to i+1) is separately encrypted.

For example, say we had an attacker, Host A, set up a connection using chacha20 to our sensor, Host B, and then another connection using aes256-gcm to the victim, Host C. When a packet is sent, it would be encrypted with chacha20, sent to host B, where it is then unencrypted, re-encrypted in aes256-gcm, and then sent to Host C.

This process essentially means that we have no guarantee that a packet entering and leaving a host will have the same length, even if the unencrypted plaintext is the exact same.

1.4 The Length of an Encrypted Packet with Different Encryption Algorithms

What we do know when looking at SSH packets, is that when multiple packets with the same amount of plaintext (the same number of chars) are encrypted, **they will have the same encrypted packet length**. If we know these lengths, we can estimate the number of characters present in any given packet. This occurs because of the combination of two factors: the length of any given character is always one-byte, and the length of a packet will always be padded to a multiple of a certain, knowable value. This means that upon testing, we get results like the ones below.

As we can observe in **??**, the number of plaintext characters used in any packet encrypted by a given cipher has a predictable, consistent packet length. What's more, when we compare the differing packet lengths of the 6 ciphers that are available for use in the baseline installation of OpenSSH (the most commonly used SSH software), we find that **the total difference in the length of an encrypted packet with the same amount of plaintext will never exceed 8 bytes**. This again is due to the combination of two factors: the padding will always increase the packet length to a multiple of a certain value, and

the maximum that this value can be with these baseline ciphers is 16. This second fact is based on all included ciphers having a block size of 16 bytes or less. While this holds true for all useable ciphers, then we know for a fact that the difference in length not exceeding 8 bytes will stay true. While seemingly inconsequential, this deduction is incredibly important in our ability to match packets and will be used often.

1.5 Critical Definitions of Packets

Finally, we will briefly review some critical definitions for the following analysis. The first of these is a match packet. A match packet refers to a packet that is observed entering a sensor, and then leaving the sensor (After having been decrypted, reformed, and re-encrypted as described section 2.3). Our ability to find match packets is critical in determining if a sensor is being used as a stepping-stone. Knowing that a packet matches another means it must be part of a connection chain, which usually means it is part of a stepping-stone intrusion.

A Minimum Length packet is one that has the minimum number of characters (1 character/one-byte) present as the payload of the packet. These packets are the shortest length possible for their respective cipher. This terminology is important because, **when typing normally in a terminal into a connection chain, each individual character will be sent as a separate minimum length packet**. This behavior is important to packet sequencing and will be discussed further in Sections 4 and 5.

Finally, it is important to note the distinctions of an exit packet. An exit packet is a packet with the SSH command "exit". This packet is executed slightly different to other packets. If a chain from host 0 to host N is created, when the exit command is issued (the enter key is pressed with "exit" currently in the terminal), it will travel down the connection chain as a normal packet until reaching host N-1. **The exit command will then be executed on host N-1, not host N. This means the exit packet will not be sent to the last host and will not necessarily have a matching echo packet**. This is important to note, as not understanding this can cause confusion when some of the last packets in a sequence can match up differently from a standard send-echo relationship.

2 Identifying Encrypted Packet

In order to successfully modelling a sensor's traffic in a way that can be used to determine if it is being used as a stepping-stone, we must focus on finding Match packets (described in section 2.5). To find these Match packets, we use certain identifying features of packets. We focus on four key characteristics: whether a packet is a send or echo packet, packet length, the number of identical length sequential packets, and the timestamp of the packet. All four of these characteristics are simple to find, and the process of finding them will be expanded upon later.

The Send or Echo state of a packet is the simplest field to use. An incoming Send packet can only ever be matched with an outgoing Send packet, vice versa. Packet Length is slightly more complex to use, as different algorithms present different length packets for identical plaintext, but as stated above: two packets with identical plaintext, when encrypted with different algorithms, will always have a difference in length of 8 bytes or less[1]. Therefore, any packet with a greater difference is not a matched packet.

The crux of this method is the number of Identical Length Sequential Packets: the number of consecutive packets with the same length. So, 4 minimum length consecutive packets would be recorded as such. By separating these sequences whenever a command is executed, we can create highly accurate sequences of packets that identify two connections.

[1]This assumption always holds true given that none of the allowed ciphers have a block size exceeding 16 bytes. All standard ciphers are block size 16 or less. This difference value must be increased if allowed ciphers have a larger block size.

```
265.78.195240225 168.27.2.107 168.27.2.106 SSH 110 Client: Encrypted Packet (len=44)
266.78.195669822 168.27.2.106 168.27.2.103 SSH 118 Client: Encrypted Packet (len=52)
267.78.196655384 168.27.2.103 168.27.2.106 SSH 118 Server: Encrypted Packet (len=52)
269.78.197168392 168.27.2.106 168.27.2.107 SSH 110 Server: Encrypted Packet (len=44)
```

Figure 4: An example of packet nestling

The timestamp is important because we can match packets up across a sensor because they perform a process of "nestling". Nestling is the process where Host B receives a Send packet, sends its own send packet to Host C, received Host C's Echo packet, and then sends its own Echo packet back to Host A. The important factor we can observe in this nesting behavior is that: an incoming Send packet can only be matched to an outgoing Send packet with a later timestamp, and an incoming Echo packet can only be matched to an outgoing Echo packet with a later timestamp. Based on this behavior, we can accurately identify encrypted packets and their associate match packets

3 Modeling Network Traffic

3.1 ILSPS Creation

We model network traffic by creating a list of Identical Length Sequential Packets Series (ILSPS's). An example sequence created with this model will be a list as such:

$$\{\cdots$$
$$\{SEND/ECHO, numPackets, packetLength, startTS, endTS\},$$
$$\{SEND/ECHO, numPackets, packetLength, startTS, endTS\},$$
$$\cdots\}$$

A single member of the above list is referred to as an ILSPS. An ILSPS is composed of the following fields:

- SEND/ECHO simply lists whether the ILSPS is a sequence of Send packets or Echo packets. Current work can only use Send packet sequencing, due to complex and inconsistent behavior exhibited by echo packets[2], but the implementation leaves room for future improvement.
- The 'numPackets' field lists the number of sequential, identical, packets of the element. Four identical minimum length packets can be noted by a "4" in this field.
- The field 'packetLength' lists the length of the packets.
- The 'startTS' field notes the timestamp of the first packet that makes up this element, while the 'endTS' field notes the last packet's timestamp. If the 'numPackets' field is 1, then these two fields will have the same value.

When creating the elements, but not when recording them, they also have an additional field with one of 3 values: UNCAPPED, PARTCAPPED, or CAPPED. This value is used to determine whether

[2]Echo packets sometimes will "separate" or "reform" at nodes. This is believed to be based on the time in between a packet being received and sent out, but as of yet occurs into inconsistent a manner for effective use of Echo packet matching.

the next packet looked at is eligible to be included in this element. When we determine that a command has been executed, we change the values to CAPPED, so that the next sequence of packets begins a new element. In this way, we divide the elements up by whenever a command is invoked. **This creates a sequence of ILSPS which is unique to the set of commands and the connection that the attacker uses**.

To create the sequence, we look at each packet individually. If the packet is a Send packet, it is added to the sequence. If the packet is an Echo packet, it affects the current sequence's CAPPED value. For Send packets, if the packet matches the length of the current UNCAPPED or PARTCAPPED element, then the 'numPackets' field of the element is increased, and the 'endTS' field is set equal to this packet's timestamp. In any other case, we create a new element using this packet. The field 'numPackets' is initialized to one, 'packetLength' is the length of the packet, and both 'startTS' and 'endTS' are set equal to this packet's timestamp. Either way, we then move to the next packet.

There are two criteria for Echo packets causing element capping. The first is receiving an Echo packet with a different length than the previous Send packet. When this occurs, it means data has been sent that is not a strict echo message, usually a response to a user's command (ex. "Documents Downloads Pictures etc…" as a response to an "ls" command). Receiving this Echo packet immediately switches the current element to CAPPED, and begins a new sequence. The second criteria is if multiple correct length packets are received. When an Echo packet with a correct length is received, we designate the current element as PARTCAPPED. If another Echo packet with a correct length is received, before another send is received, we then change it to CAPPED. The reason for this is that sometimes the data echoed as a response to a command has the same packet length as the previous Send packet. This causes an issue where even though response data is received, it is only seen as a strict Echo packet, so the element would not be CAPPED.

A current issue with this method lies with exit packets (A packet with an "exit" message). An exit packet will be executed on the Sensor node, instead of the destination node. So, if an attacker at Host A sends an exit packet to a sensor at Host B, the packet is sent as plaintext normally, but the command is then performed at Host B to close the connection, not at Host C. This means that when there is an exit packet, the sequences of an incoming connection and an outgoing connection will look slightly different at the end of the sequences. This issue is difficult to manage, as it is impossible to know what an exit packet is just by viewing the incoming connection. Thankfully, these account for a small fraction of the total packets sent, so the Exit packet lowers the accuracy with a negligible amount for sequences with more than 20 packets. Therefore, we treat the echo packet error rate as negligible.

3.2 Timestamp Basling

Since, for the purposes of testing, we must collect the data at individual times, we do run into the problem of the timestamps being incomparable. For instance, every timestamp from Test 3 must be after every timestamp from Test 2. For reasons that will become clear in Section 5, this prevents us from performing accurate comparison to test the strength of the algorithm. To the end of having comparable data, we perform a process of "Timestamp baseline" where we take the first timestamp in a sequence of ILSPS, set it to 0, and then subtract the initial timestamp from every other in the sequence. This means that all sequences will effectively start from 0 ns, and we will be able to compare them as we will see in the following section.

Algorithm 1: Find Potential Matches

```
primaryIndex, OUTIndex  = 0
INRange,OUTRange  =  [0,0]
MatchesFound = []
INMax = len(incomingArray)
OUTMax = len(outgoingArray)

while(primaryIndex < INMax):
    for each INStartIndex in range(primaryIndex, INMax):
        for INEndIndex in reversed(range(INStartIndex+MINIMUMMATCHLENGTH, INMax)):
            INRange = [INStartIndex, INEndIndex]

            If INRange is NOT fully contained in previous Match:

                rangeLength = INRange[1] - INRange[0]

                for OUTIndex in range(0, OUTMax):
                OUTRange = [OUTIndex, OUTIndex+rangeLength]
                If Outrange is outside of the OUTARRAY:
                                    Break;
                If INRange and OUTRange match our conditions for matching:
                   MatchesFound.append(INRrange, Outrange, their respective data)
    primaryIndex += 1
```

Figure 5: Select match Algorithm

4 Detecting Stepping-stone Intrusion

Once each connection has been turned into a list of ILSPS's, we then attempt to match this list to other lists. For example, we would compare "incomingConnection1" with "outgoingConnection3" to see if their lists indicate they may be a relayed connection.

4.1 Find Potential Sequence Matches

Once we have all sequences prepared, we then compare each incoming sequence to each outgoing sequence, line by line. For each set of two element, the 'numPackets' fields must match exactly, the 'packetLength' fields must be within 8 of each other, and both incoming timestamps (startTS and endTS) must be before their respective outgoing timestamps of the other element.

Once each connection has been turned into a list of ILSPS's, we then attempt to match this list to other lists. For example, we would compare "incomingConnection1" with "outgoingConnection3" to see if their lists indicate they may be a relayed connection.

If all the above fields are within their respective parameters, then the elements are considered to be matching, the total number of matching packets for this specific sequence comparison is increased by an amount equal to the 'numPackets' field. Regardless of the matching status, the total number of compared packets is increased by the 'numPackets' field as well.

4.1.1 Wholly Within

In the Algorithm 1 pseudocode "Find Potential Matches", we have the line "If INRange is NOT fully contained in previous Match" This line refers to the "WhollyWithin" function. If the range currently being checked is fully within ANY previous selected Range, we will not choose that line, ex. Range(3,7) is fully within Range(1,10). There are two reasons for this function: A large range is a better indicator of

Algorithm 2: Select Correct Matches

```
MatchesFound = Data from Find Potential Matches
tempMatches = []
SelectedMatches = []
while (len(MatchesFound) > 0):
        tempMatches = MatchesFound.copy()
        largestMatch = [[0,0]]
        for each Match in tempMatches:
                If Match is empty ([]):
                        MatchesFound.remove(Match)
                 else if Match crosses over with selectedMatches:
                        MatchesFound.remove(Match)
                        newMatch = removeCrossoverItems(Match)
                                MatchesFound.append(newMatch)
                 else if Match is larger than largestMatch:
                        largestMatch = Match

        if(largestMatch != [[0,0]]):
                SelectedMatches.append(largestMatch)
                 for num in range(largestMatch[0][0],largestMatch[0][1]):
                        usedInIndex.append(num)
                for num in range(largestMatch[2][0],largestMatch[2][1]):
                        usedOutIndex.append(num)
         MatchesFound.remove(largestMatch)
```

Figure 6: Select match Algorithm2

a relayed connection than a shorter range, and it greatly reduces the computational speed of the algorithm in future steps.

4.2 Select Best Matches

Once we have selected a sequence of Potential Matches, we then must determine which of these Matches will be used for the result calculation. To do this, the Algorithm 2 “Select Correct Matches” picks Matches from the set of Potential Matches.

4.2.1 Crossover and Shorten

In selecting the correct Matches, we have two important functions: Crossover and Shorten. The first, Crossover, is used to determine if a potential range overlaps with an already selected Range. For example, the potential Range(1,9) would crossover with the already selected Range(8,16). This can also occur in the case where a range, say Range(8,12), is used twice, but is matched with two distinct ranges on the outgoing side, ex. Range(9,13) and Range(37,41). Obviously, the same packets cannot be used as part of two separate sequences matches, so the Crossover function detects this.

To remedy this problem, we use the Shorten method. This method will reduce the size of a range until it no longer has crossover, potentially reducing the range to empty. The Shorten operation will apply on both parts of the Match, and will reduce the associated range in the same manner. Ex. if incoming Range(7,12) is reduced to Range(8,10), then outgoing Range(8,13) is also reduced in the same manner, to Range(9,11). If at the end of this operation, the Range is smaller than the MinumMatchLength chosen for the program, the range is automatically discarded.

Together, these two functions ensure that the same ILSPS is not used twice, for incoming or outgoing connections, which would compromise the efficiency of the program.

4.3 Match Weighting

Initially, the program treated every ILSP as worth the same, they would be measured, and ultimately the number of correct matches would be divided by the number of total matches, to get an accuracy percentage. Over testing, we found that the more sequential packets in an ILSP, the better an indicator it was, and therefore the more weight it should be given. As such, we instead calculate the total number of packets within all the ILSP's of a connection, and divide the matched number of packets by the total number of packets, which gives us a weighted result that is a much better indicator of relayed connections.

4.4 Result Determination

Once each comparison is processed, the data is output, showing the matching rates of individual connections, as well as average matching rates for types of connections. While we know which connections are actually relayed in the data, the program does not, and indicates there is a relayed connection when the matching percentage is above 75%, and an unrelated otherwise.

5 Experimental Results and Analysis

5.1 Experimental Setup

In order to test our algorithm, we devised a basic setup consisting of 8 hosts (7 connections) using the SSH protocol, and recorded data at every incoming and outgoing connection. This setup used a local host (CCT30), and then 7 external hosts using AWS servers AWS1, AWS2, …, AWS7 from Amazon cloud. The connection chain is CCT30 $\rightarrow$ AWS1 $\rightarrow$ AWS2 $\rightarrow$ AWS3 $\rightarrow$ AWS4 $\rightarrow$ AWS5 $\rightarrow$ AWS6 $\rightarrow$ AWS7. All the traffic generate at CCT30, and finally come to AWS7 via stepping-stones AWS1 to AWS6. We collect network traffic at the incoming and outgoing connections of each host of AWS1 to AWS6. These hosts were geographically distributed to create a more realistic testing environment. Table 1 shows all the AWS servers' geographic location, and other information.

Server Name	**OS**	**Public IP Address**	**Private IP Address**	**Geographic Location**
AWS1	Ubuntu	34.239.127.118	172.31.86.245	Virginia, USA
AWS2	Ubuntu	52.59.96.142	172.31.29.198	Frankfurt, Germany
AWS3	Ubuntu	18.133.230.26	172.31.39.118	London, UK
AWS4	Ubuntu	52.60.79.102	173.31.12.163	Vancouver, Canada
AWS5	Ubuntu	52.194.232.92	172.31.0.70	Tokyo, Japan
AWS6	Ubuntu	34.248.116.10	172.31.32.141	Dublin, Ireland
AWS7	Ubuntu	52.67.153.198	172.31.13.98	San Paulo, Brazil

Table 1: AWS servers' IP and geographic location

5.2 Experimental Data Collection

For data collection, we used a separate local host (CCT31) located in the Lab of Computer Science, Columbus State University, to collect the data at every point, we used the tcpdump command as follows for incoming connections,

sudo tcpdump -nn -tt "(dst port 22 && dst xxx.xxx.xxx.xxx) || (src xxx.xxx.xxx.xxx && src port 22)" >AWS1_IN_AttackX_TestX.txt

And the below tcpdump command for outgoing connections

sudo tcpdump -nn -tt "(dst port 22 && src xxx.xxx.xxx.xxx) || (dst xxx.xxx.xxx.xx && src port 22)" >AWS1 OUT AttackX TestX.txt

Figure 7: tcp dump command

In both cases, "xxx.xxx.xxx.xxx" referred to the hosts own ip address, and X referred to the appreciate attack/test number. We ran these commands on all the hosts in the chain (only outgoing and incoming for the first and last links, respectively) to compile the data.

5.3 Experimental Data Processing

After the data was fully collected, we processed the data in preparation for analysis. We first used the method of ILSPS creation at detailed in Section 4, along with the timestamp baselining process. Once we had the ILSPS' we run the data through the packet sequencing method we detailed in Section 5 to the received results.

5.4 Experimental Results and Analysis

A set of three attacking scenarios were designed [14] in order to test the effectiveness of this algorithm. The three attackers use three different scripts to access a victim in three different tasks. The first attacker logs into the victim as a root user, accesses and downloads the shadow file from the ./etc directory. The second attacker opens a file already in the victim machine and appends data onto it. The third attacker deletes and creates a modified version of a file already on the victim computer. The command scripts used by the three attackers are listed below, respectively:

First attacking scenario:
pwd
whoami
sudo su
ls
cd /etc
ls -a
scp -p shadow attacker usernameattacker IP:/home/seed/Documents exit

Second attacking scenario:
whoami
pwd
cd /home/seed/Documents
ls
nano text file.txt ls
cat hello.txt
exit

Third attacking scenario:

whoami
pwd
cd /home/seed/Documents
ls
nano hello.txt
ls
cat hello.txt
exit

Each attack spanned a chain of 8 hosts as we mentioned before, 6 of which performed packet matching detection to verify the performance of stepping-stone detection algorithm proposed, and each attacking scenario was performed in 10 separate tests. For these attacks, an incoming connection of a particular test was compared with the outgoing attacks of its respective tests. The correct connection (ex. Attack1-Test4 with Attack1-Test4) match rate was determined as well as the incorrect connection (ex. Attack1-Test4 with Attack3-Test4) match rate. The following Table 2 to Table 7 show the testing results from ASW1-AWS6 respectively. In the tables, we use short name AK representing Attacking Scenario, and T is the short name of "Test". Each table shows the match rate between the incoming connection of attacking scenario x (x=1, 2, 3), such as AK1 (attacking scenario 1, x=1), and the outgoingconnection of all the three attacking scenarios, such as AK1, AK2, and AK3. Apparently, AK1 matches with AK1, but not matching with AK2, or AK3. Similar results hold for AK2, AK3 matching with AK1, AK2, and AK3. In each of the following table, the column in red color shows the matching rate for matched attacking scenario, and the columns in black color show the matching rate for unmatched attacking scenarios. We use Table 2 as an example to help readers understand the results. In Table 2, the first three columns shows the match rate between the incoming connection of attacking scenario 1 and the outgoing connection of attacking scenario 1, scenario 2, and scenario 3 respectively. Similarly, the second and the third four columns present the match rate for attacking scenario 2, and scenario 3 respectively. The column in red color of the first four columns shows the match rate between the incoming connection and the outgoing connection of attacking scenario 1 (a relayed connection) is above 90%, but match rate for un-relayed connection is below 12%. Similar results can be obtained from other tables. In Table 8, it shows the average match rate for all 10 tests of all the AWS servers.

AK1	**AK1**	**AK2**	**AK3**	**AK2**	**AK1**	**AK2**	**AK3**	**AK3**	**AK1**	**AK2**	**AK3**
T1	99.06	0	11.21	T1	0	98.36	9.01	T1	0	10.45	91.50
T2	99.12	0	0	T2	0	97.26	21.91	T2	0	0	96.47
T3	99.12	0	9.64	T3	0	81.11	7.77	T3	5.44	4.08	91.83
T4	96.24	0	3.75	T4	0	96.92	9.23	T4	0	5.16	99.35
T5	94.87	0	0	T5	0	96.92	0	T5	0	5.29	99.33
T6	95.41	0	0	T6	0	96.22	13.2	T6	0	2.43	97.56
T7	99.13	10.43	0	T7	0	90.00	8.57	T7	6	4	99.33
T8	99.16	0	0	T8	0	92.20	6.48	T8	0	0	96.81
T9	92.91	0	0	T9	0	90.14	8.45	T9	8.67	0	97.10
T10	91.11	0	10	T10	0	97.33	20	T10	10.22	3.4	95.45

Table 2: Testing match rate results at AWS1

This simple Tests scenario demonstrates the strength of the algorithm. The average match rate for relayed connections was in the range of 94% to 96%. The lowest matching rate found for any relayed connection was 82.35%, and every other relayed connection had a matching rate of 85% or higher. For un-relayed connections, the average match rate never rose above 15%, and was often much lower. The highest found matching rate for an unrelated connection was 31.11%. **The difference between the lowest match rate of a correct connection and the highest match count of an incorrect connection**

AK1	AK1	AK2	AK3	AK2	AK1	AK2	AK3	AK3	AK1	AK2	AK3
T1	99.06	8.41	0	T1	0	99.18	6.5	T1	0	14.47	99.34
T2	99.12	0	0	T2	0	98.64	10.81	T2	0	9.09	93.7
T3	90.35	0	0	T3	0	98.88	8.88	T3	0	6.33	98.59
T4	90.22	0	0	T4	0	90.90	9.09	T4	11.61	12.25	99.35
T5	99.13	0	5.17	T5	0	98.48	9.09	T5	0	6	99.33
T6	90.82	0	0	T6	0	98.14	0	T6	0	1.95	97.56
T7	99.13	0	0	T7	0	91.54	8.54	T7	8	4	99.33
T8	99.16	0	0	T8	0	93.58	8.97	T8	0	4.45	96.81
T9	99.20	0	0	T9	0	92.95	8.45	T9	5.2	2.89	97.10
T10	91.11	0	13.33	T10	0	97.40	9.09	T10	6.85	6.85	97.14

Table 3: Testing match rate results at AWS2

AK1	AK1	AK2	AK3	AK2	AK1	AK2	AK3	AK3	AK1	AK2	AK3
T1	99.06	0	0	T1	0	98.64	21.62	T1	0	14.47	99.34
T2	99.12	0	0	T2	0	98.88	15.55	T2	0	9.09	93.70
T3	95.57	0	0	T3	9.09	89.39	9.09	T3	0	6.33	98.59
T4	96.21	0	6.06	T4	0	98.48	0	T4	11.61	12.25	99.35
T5	99.13	0	6.03	T5	0	90.14	8.45	T5	0	6	99.33
T6	95.45	0	0	T6	0	98.14	0	T6	0	1.95	97.56
T7	99.13	0	0	T7	0	90.14	8.54	T7	8	4	99.33
T8	99.16	0	0	T8	0	93.58	17.94	T8	0	4.45	96.81
T9	99.20	0	0	T9	0	92.95	8.45	T9	5.2	2.89	97.10
T10	97.2	6.14	4.46	T10	0	97.36	8.45	T10	6.85	6.85	97.14

Table 4: Testing match rate results at AWS3

AK1	AK1	AK2	AK3	AK2	AK1	AK2	AK3	AK3	AK1	AK2	AK3
T1	99.06	0	5.6	T1	0	95.93	3.25	T1	0	19.07	96.71
T2	95.61	0	0	T2	10.81	98.64	18.91	T2	0	9.15	96.47
T3	95.57	0	0	T3	0	98.88	18.88	T3	0	2.79	97.90
T4	99.13	0	4.54	T4	0	98.48	9.09	T4	0	3.22	99.35
T5	99.13	0	6.89	T5	0	91.04	14.92	T5	0	4.66	99.33
T6	95.45	0	0	T6	0	98.14	0	T6	5.85	2.43	97.56
T7	99.13	0	0	T7	0	90.00	0	T7	5.33	0	98.66
T8	99.16	0	6.66	T8	0	93.58	19.23	T8	0	3.82	4.9
T9	99.20	0	0	T9	0	97.18	8.45	T9	5.2	2.89	97.10
T10	97.2	6.14	17.3	T10	0	98.66	18.66	T10	9.71	2.85	99.42

Table 5: Testing match rate results at AWS4

AK1	AK1	AK2	AK3	AK2	AK1	AK2	AK3	AK3	AK1	AK2	AK3
T1	95.32	0	10.28	T1	0	95.93	4.06	T1	0	24.34	96.71
T2	95.61	0	0	T2	0	93.24	10.81	T2	0	2.81	96.47
T3	95.57	9.73	0	T3	0	94.44	12.22	T3	0	4.92	96.47
T4	88.63	0	5.3	T4	9.09	92.42	18.18	T4	3.87	5.16	96.77
T5	95.68	0	6.03	T5	0	90.90	15.15	T5	0	7.33	99.33
T6	95.45	0	5.45	T6	0	90.74	11.11	T6	3.9	2.43	97.56
T7	95.65	0	0	T7	0	91.54	12.67	T7	7.94	0	91.39
T8	95.83	0	10	T8	5.12	93.58	19.23	T8	2.54	3.82	86.62
T9	98.41	0	0	T9	7.14	91.42	17.14	T9	2.89	2.89	97.10
T10	99.44	0	7.82	T10	12	92.00	16	T10	11.42	2.85	97.14

Table 6: Testing match rate results at AWS5

AK1	**AK1**	**AK2**	**AK3**	**AK2**	**AK1**	**AK2**	**AK3**	**AK3**	**AK1**	**AK2**	**AK3**
T1	99.06	0	5.6	T1	5.69	99.18	7.31	T1	6.57	15.78	99.34
T2	99.12	0	0	T2	12.16	98.64	24.32	T2	0	4.92	99.29
T3	76.99	7.07	0	T3	0	95.55	31.11	T3	7.74	14.78	99.29
T4	99.24	0	6.06	T4	0	78.78	21.21	T4	6.45	16.77	98.06
T5	99.13	0	0	T5	0	89.23	12.3	T5	0	18.66	94
T6	93.63	0	20	T6	0	98.14	11.11	T6	3.41	10.73	97.56
T7	99.13	0	0	T7	6.94	90.27	8.33	T7	8	13.33	99.33
T8	94.16	0	5	T8	0	88.46	26.92	T8	3.79	9.49	92.4
T9	84.92	0	0	T9	7.04	97.18	30.98	T9	12.13	0	99.42
T10	99.44	10.61	12.84	T10	0	98.68	7.89	T10	16	6.85	99.42

Table 7: Testing match rate results at AWS6

	Attack1	Attack2	Attack3
Attack1	96.40%	0.98%	3.42%
Attack2	1.42%	94.56%	11.97%
Attack3	3.62%	6.35%	96.92%

Table 8: CrossAttack, Weighted Results, With TS baselining.

was over 40%.

6 Conclusion

The contents of encrypted packets are not accessible, but the header fields of an encrypted packet are not encrypted. We found that the length of an encrypted packets following a certain rule regardless its encryption algorithm. In this paper, we modelling network traffic as a sequence of Send and Echo packets including the number of packets captured in a certain period of time, packet length, starting timestamp, and ending timestamp. We obtain two sequences from the incoming and outgoing connection of a host respectively. In this paper, we propose an algorithm to find potential sequence matches and select the best match to determining the similarity of the two sequences. The higher the similarity of two sequences, the higher suspicious a host is used as a stepping-stone. Multiple experimental results over the Internet Amazon cloud servers show that the average matching rate for relayed connections was between 94% and 96%. For un-relayed connections, the average matching rate never rose above 15%. This method also has a significant advantage over some other monitoring methods by being totally position-independent: it does not matter where the sensor is located in a connection chain when performing stepping-stone intrusion detection.

Author Contribution

Dr. J. Yang supervised the research project, and designed the algorithm. Mr. N. Neundorfer implemented the algorithm, set up the experiments, collected data, and processed the data. Dr. L. Wang worked on supervising data collection and processing.

Data Availability

The network traffic packets collected for this research project are available on the following shared Google Drive:
https://drive.google.com/drive/folders/1lgeKJ_tLD2IKlx9iZHPPCM-7ab-WIYFN.

Conflicts of Interest

We, as the authors, declare no conflict of interest regarding the publication of this paper.

Acknowledgments

This research project of Drs. Jianhua Yang and Lixin Wang is supported by the National Security Agency NCAE-C Research Grant (H98230-20-1-0293) with Columbus State University, Georgia, USA.

References

[1] Y. Zhang and V. Paxson. Detecting stepping stones. In *Proc. of the 9th USENIX Security Symposium (USENIX'00), Denver, Colorado, USA*, pages 71–84. USENIX Association, August 2000.

[2] K.H. Yung. Detecting long connecting chains of interactive terminal sessions. In *Proc. of the 5thInternational Symposium on Recent Advance in Intrusion Detection (RAID'02), Zurich, Switzerland*, volume 2516 of *Lecture Notes in Computer Science*, pages 1–16. Springer, Berlin Heidelberg, October 2002.

[3] J. Yang and S.H.S. Huang. A real-time algorithm to detect long connection chains of interactive terminal sessions. In *Proc. of 3rd ACM International Conference on Information Security (INFOSECU'04), Shanghai, China*, pages 198–203. ACM, November 2004.

[4] J. Yang and S.-H. S. Huang. Mining tcp/ip packets to detect stepping-stone intrusion. *Journal of Computers and Security*, 26:479–484, December 2007.

[5] L. Wang, J. Yang, X. Xu, and P.-J. Wan. Mining network traffic with the k-means clustering algorithm for stepping-stone intrusion detection. *Wireless Communications and Mobile Computing*, 2021:1–9, March 2021.

[6] L. Wang, J. Yang, and A. Lee. An effective approach for stepping-stone intrusion detection using packet crossover. In *Proc. of the 23rd World Conference on Information Security Applications (WISA2022), MAISON GLAD, Jeju Island, Korea*, pages 198–203. WISA, August 2022.

[7] S. Staniford-Chen and L.T. Heberlein. Holding intruders accountable on the internet. In *Proc. of IEEE Symposium on Security and Privacy, Oakland, California, USA*, pages 39–49. CISC, May 1995.

[8] K. Yoda and H. Etoh. Finding connection chain for tracing intruders. In *Proc. of 6th European Symposium on Research in Computer Security (ESORICS'00), Toulouse, France*, volume 1985 of *Lecture Notes in Computer Science*, pages 31–42. Springer, Berlin Heidelberg, October 2000.

[9] A. Blum, D. Song, and S. Venkataraman. Detection of interactive stepping stones: Algorithms and confidence bounds. In *Proc. of 7th International Symposium on Recent Advance in Intrusion Detection (RAID'04), Sophia Antipolis, France*, volume 324 of *Lecture Notes in Computer Science*, pages 20–35. Springer, Berlin Heidelberg, September 2004.

[10] J. Yang, B. Lee, and S.S–H. Huang. Monitoring network traffic to detect stepping-stone intrusion. In *Proc. of 22nd IEEE International Conference on Advanced Information Networking and Applications (AINA 2008), Okinawa , Japan*, pages 56–61. IEEE, March 2004.

[11] Jianhua Yang and Yongzhong Zhang. Rtt-based random walk approach to detect stepping-stone intrusion. In *Proc. of IEEE 29th International Conference on Advanced Information Networking and Applications (AINA'15), Gwangju, South Korea*, pages 558–563. IEEE, March 2015.

[12] J. Yang. Resistance to chaff attack through tcp/ip packet cross-matching and rtt-based random walk. In *Proc. of the 30th IEEE International Conference on Advanced Information Networking and Applications (AINA'16), Crans-Montana, Switzerland*, pages 784–789. IEEE, March 2016.

[13] L. Wang, J. Yang, X. Xu, and P.-J. Wan. Mining network traffic with the k-means clustering algorithm for stepping-stone intrusion detectiona framework to test resistance of detection algorithms for stepping-stone intrusion on time-jittering manipulation. *Wireless Communications and Mobile Computing*, 2021:1–9, August 2021.

[14] J. Yang, L. Wang, and S. Shakya. Modelling network traffic and exploiting encrypted packets to detect stepping-stone intrusions. *Journal of Internet Service and Information Security*, 12:2–25, February 2022.

[15] N. Neundorfer, J. Yang, and L. Wang. Modelling network traffic via identifying encrypted packets to detect stepping-stone intrusion under the framework of heterogonous packet encryption. In *Proc. of 36th International Conference on Advanced Information Networking and Applications (AINA'22), Sydney, Australia*, volume 450 of *Lecture Notes in Computer Science*, pages 516–527. Springer, Berlin Heidelberg, April 2022.

Author Biography

Jianhua Yang is currently working at TSYS School of Computer Science, Columbus State University (CSU), Columbus, GA USA as a full Professor. He received his Ph.D. degree on Computer Science from University of Houston, USA at 2006, M.S. and B.S. degree on Electronic Engineering from Shandong University, China at 1987, and 1990 respectively. Dr. Yang has published more than 60 peer-reviewed journal papers and conference proceedings on cybersecurity. He has been awarded by NSA, NSF, and DoD with amount of more than half million dollars. His current research interest is computer network and information security.

Noah Neundorfer worked as a Research assistant for the TSYS School of Computer Science, Columbus State University, while working on this project funded by the NSA. He also participated in a research experience for undergraduates at Montana State University. He currently is studying for a bachelor's degree in computer science with a cybersecurity focus at CSU. Noah has assisted in the creation, publication, and presentation of several cybersecurity works at conferences and in publications, mainly in the topics of Network Intrusion detection and Malware Identification and analyzation

Lixin Wang is currently an Associate Professor of computer science at Columbus State University, Columbus, GA USA. He holds a Ph.D. degree in Computer Science from Illinois Institute of Technology, Chicago IL. His research interests include Network Security, Intrusion Detection, Wireless Networks, Algorithm Design and Analysis. He has been conducting top quality research and published 47+ peer-reviewed high-quality research papers, most of which are published on leading Computer Science journals or top-tier Computer Science conferences. Since 2011, Dr. Wang has been awarded ten federal grants (from NSF, NSA, DoE, USAR, or DoEd of USA) as the PI or Co-PI with the total awarded amount more than $2.6 million USD.