STEPPING-STONE NETWORK ATTACK KIT (SNEAK) FOR EVADING TIMING-BASED DETECTION METHODS UNDER THE CLOAK OF CONSTANT RATE MULTIMEDIA STREAMS

by

JAIDEEP PADHYE

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

August 2008

To my mother Shubhada and father Dhananjay without whom

I wouldn't be where I am today.

ACKNOWLEDGEMENTS

First and foremost, I want to thank all members of *Information Security Lab* for providing an excellent and inspiring work environment. I would like to thank my supervising professor Dr. Matthew Wright, who gave me a chance to work with him. His uncompromising quest for excellence and supreme work ethics inspired me to reach my goals. I am grateful to Dr. Donggang Liu and Mr. Michael O'Dell for their interest in my research and for taking time to serve on my thesis committee.

Of all my colleagues, I want to express my special thanks to: Aniket Pingley, my room-mate during the past two years. Our whiteboard duals made complex problems really simple and I attribute a lot of my success to him. Kiran Mehta was always ready with a helping hand whenever I was stuck during the implementation phase of my project. I had lot of interesting and evening-filling discussions, not only about my research but also about other important things of life with him. I would like to thank all my colleagues for their support.

Finally, I owe special gratitude to my family for their continuous and unconditional support: to my mother Shubhada for being what she is, to my father Dhananjay for the interest he showed in my studies and the motivation he gave me during those tiring times, and to my brother Manoday for his enduring patience, understanding, and support.

June 5, 2008

ABSTRACT

STEPPING-STONE NETWORK ATTACK KIT (SNEAK) FOR EVADING TIMING-BASED DETECTION METHODS UNDER THE CLOAK OF CONSTANT RATE MULTIMEDIA STREAMS

Jaideep D. Padhye, M.S.

The University of Texas at Arlington, 2008

Supervising Professor: Matthew Wright

With the advent of the Internet, network-based security threats have been constantly on the rise. The source of an attack could be traced by studying the system logs and the source IP address of the attack can be used to identify and prosecute the attacker. To avoid getting traced and to mislead the forensic investigators, attackers usually compromise weaker nodes on less secure networks and use them as *stepping stones* to attack the victim. This technique makes it difficult for the investigators to trace the real source of attack.

Hence, it is important to research the stepping stone detection techniques so that the attackers can be apprehended. An interesting approach towards detecting stepping stones is to correlate incoming and outgoing streams at the stepping stone. A popular way of achieving this is to watermark packet streams as it is effective against a wide range of evasion techniques. Previous investigators have described a promising technique by which an attacker could effectively evade any timing-based detection technique, includ-

ing watermarking. Their basic idea was to remove timing information from the packet streams by disguising the attack traffic as constant rate multimedia stream.

In this thesis, we investigate the effectiveness and plausibility of this approach. We present the design and implementation details of *Stepping stone NEtwork Attack Kit (SNEAK)*, a system that implements the previously described evasion techniques. SNEAK includes implementations of two algorithms for managing traffic at the stepping stone. The first algorithm is the sender-side dropping algorithm, in which the stepping stone makes decisions about dropping packets as needed when packets are sent. The second algorithm is the receiver-side dropping algorithm, in which the stepping stone makes decisions about dropping packets as needed, when packets are received. To counter the packet drop and the packet loss, we maintain redundancy in the packet streams. Both algorithms are suitable for practical use, depending on the needs of the attacker. We defined metrics for robustness, usability and effectiveness, and we studied the trade-offs between them. We implemented a prototype of the *SNEAK* system and tested it on the PlanetLab network. Our prototype provides reliable transmission and reasonable performance for shell commands over at least two stepping stones and the traffic has the characteristics of a constant rate multimedia stream. We tested the effectiveness of *SNEAK* against a centroid-interval-based watermarking technique that is currently the best available timing-based detection technique. The experimental results indicate that timing information embedded in the incoming stream is completely eliminated in the outgoing stream. The results also demonstrate that *SNEAK* is suitable for practical use without affecting the overall usability of the system and *SNEAK* is effective against all timing based detection techniques. The experimental results demonstrate the need to consider the true potential of the attacker and develop detection methods that use more than low-level timing information to defeat such attacks.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Attack using stepping stones

Hackers attack systems on the Internet to steal sensitive information or to disrupt services and they need to cover their tracks to avoid being discovered and identified. One of the ways to cover their tracks is to launch attacks indirectly by relaying the attack through a chain of intermediate (previously compromised) systems called *stepping stones* (also known as *hop points*). The networks that carry these stepping stones are called *host networks*. The attacker constructs a chain of interactive connections using protocols like Telnet or SSH. The commands that the attacker types on his local terminal are relayed through the stepping stones until they reach the victim. The quality of communication over the Internet has improved multifolds, and targets can be attacked on the other side of the world without facing cumbersome delays. A basic attack using stepping stones is illustrated in Figure 1.1.

1.2 Stepping stone detection

The attacker uses the stepping stones to relay the traffic to the victim. The obvious way to detect stepping stones would be to compare the incoming stream with the outgoing stream at each of the suspected hosts. An intuitive approach towards stream comparison would be to compare the contents of the incoming and outgoing packets in a network to find packets with the same content. However, the use of encrypted communication protocols like *SSH*, have made this approach ineffective. We need to use other characteristics of the traffic such as inter-packet timing to detect stepping stones. Ac-

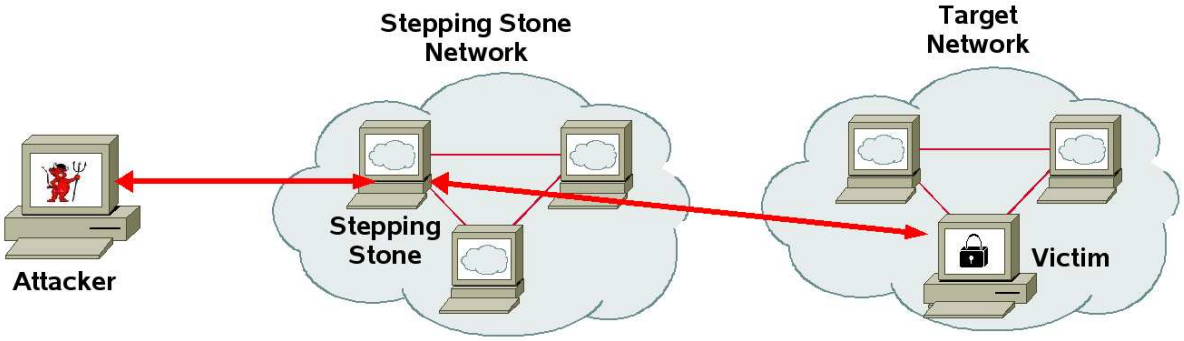

Figure 1.1 Basic attack using stepping stone.

tively perturbing the incoming stream by embedding a pattern and trying to detect that pattern in the outgoing streams is a promising approach. This approach can be made robust to random changes in packet timing introduced by the attacker [3].

In our work, we investigate the ways in which the attacker can evade detection. A simple technique would be to remove all the timing information from the packet streams and convert to a constant rate stream. Our primary assumption is that the constant-rate traffic will not be flagged as unusual. Since multimedia streams are becoming more and more common on the Internet, particularly with the growing use of Internet telephony (VoIP) and video content [4], we feel that this is a reasonable assumption. These trends may not hold in certain corporate and high-security environments, but these environments can easily be avoided while choosing stepping stones. As stepping stones are intermediate hosts and can be located in more open environments like homes and college campuses, even while the hacker aims for more secure targets.

In our proposed technique, the attacker uses stepping stones and disguises his traffic as an encrypted constant rate multimedia stream. Encrypted packets pass through each of the stepping stones in the attack path before making it to their destination. At each stepping stone between the attacker and the target, a *receiving process* receives and decrypts these packets and places the payloads in a small buffer. A different process

running on the stepping stone, the *sending process*, takes these payloads from the buffer, re-encrypts them and sends them at a constant rate to the next node, also disguised as an encrypted constant rate multimedia stream. Whenever the buffer is empty, a dummy packet is sent instead. The key principle is that the sending process is independent of the receiving process. The timing characteristics of the packets generated by the sending process have no dependence on the timing of packets from the incoming stream, so all timing information is removed.

1.3 Contribution

An earlier study of evasion of stepping stone detection simulated the attack and verified its effectiveness [2]. Although the simulations indicate the possibility of the attack being successful, the practical feasibility of the attack had not been studied. Our contribution was to implement the algorithm in the form of the SNEAK system and test it on PlanetLab network. We also discovered the shortcomings in the original algorithm and proposed changes to make it more effective. We then proposed an alternative algorithm that performs better. In this context, we discuss concepts of usability, robustness and effectiveness and define metrics to measure them. Finally we discuss the trade-off for configuring and selecting the algorithms to use.

1.4 Thesis Organization

In Chapter 2, we discuss the background concepts and the related work for our thesis. In Chapter 3, we discuss the design of SNEAK systems and discuss various components of the system in detail. In Chapter 4, we describe the prototype application and its implementation. Chapter 5 discusses the experimental setup and the results. Finally Chapter 6 concludes with ideas for future work.

CHAPTER 2

BACKGROUND

This chapter discusses *stepping stones*, including a real life scenario for their use. Then, we discuss the prior work in stepping stone detection and traceback. Further, we discuss two watermarking techniques and the basic evasion approach that forms the basis of this thesis.

2.1 Attack using stepping stones

A *connection chain* is a sequence of logins where a person logs into one computer, from there logs into another, and so on [5]. A *stepping stone* is any intermediate host on a connection chain [6]. An *attack path* is a connection chain that links all the stepping stones that are used to launch the attack. A stepping stone is usually a less secure unit, which is compromised by the attacker to use it to relay the attack traffic to its destination. A connection that carries the attack traffic to the destination is called an *upstream connection* and the one that brings back the response is called a *downstream connection*. Generally the attackers use terminal emulation programs like SSH and Telnet to create connection chain to the victim. Stepping stones thus formed are called *interactive stepping stones*. Although the attacker might use stepping stones to form a one-way communication, for this research project, we only investigate interactive stepping stones.

Stepping stones can be used for launching denial of service attacks or hacking into systems to steal secure data. Let us a consider a hypothetical case study where an attacker seeks to penetrate a tightly secured server and retrive some top secret data from a carefully monitored government network. We assume that the attacker has the

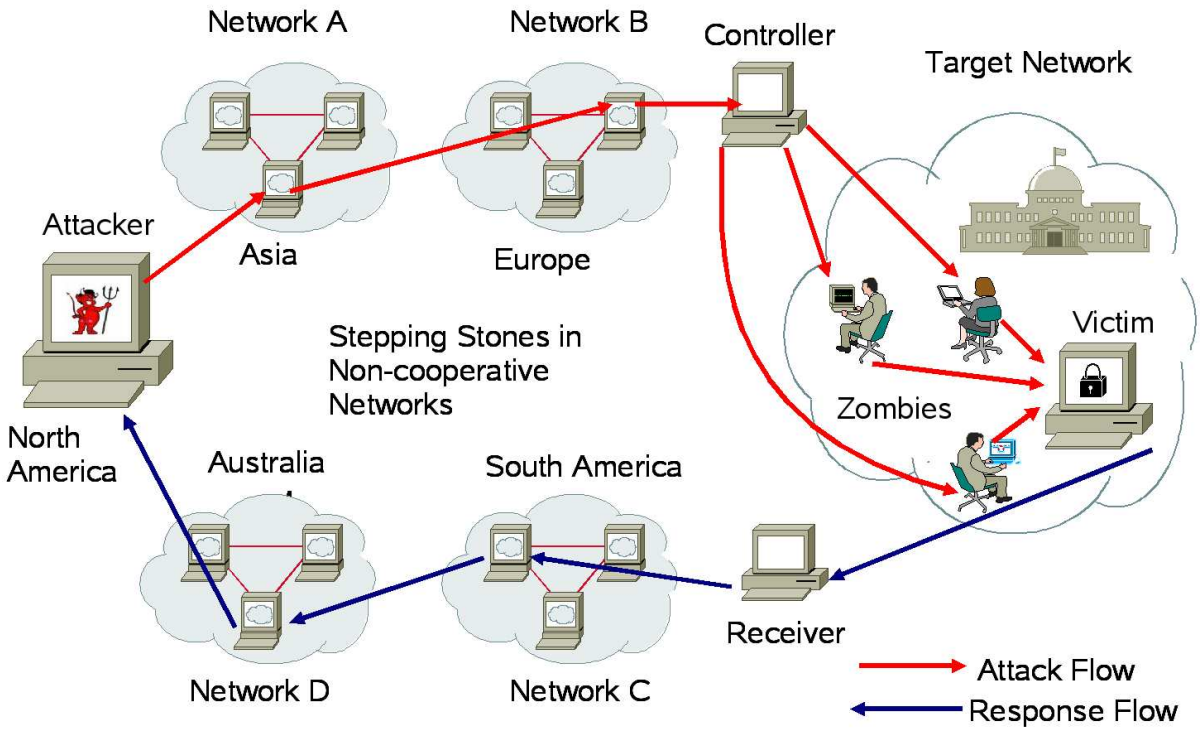

Figure 2.1 Real life attack using stepping stones. Source: [1]

technical expertise to carry out this task, but then needs to cover up his tracks to prevent forensic investigators from tracing him using the system logs. To ensure that the source of the attack is anonymous, he first selects nodes with weak security across geographically diverse locations as candidates for being stepping stones and compromises them. Then he selects and compromises two more weak nodes in the proximity of the target network, that are to act as the controller and receiver. Next he attacks the target network and compromises a few relatively less secure nodes to use them as *zombies*[1] inside the target network, that can later be used to launch an attack on the victim. This attack scenario, described in *Mitre workshop report* [1], has been illustrated in Figure 2.1. This approach guarantees anonymity to the attacker, as even if the forensic investigators manage to

[1]A zombie is computer that the hacker compromises and commands using a remote controller.

trace the attack path till the controller, they might not get access to the system logs on the stepping stones. To stay anonymous, the attacker can also use commercial services like anonymizer.com, but the information can be subpoenaed [7]. Thus, attack using stepping stones is the most favorable attack mechanism that guarantees anonymity for the attacker.

2.2 Traceback

Let us again consider the scenario depicted in Figure 2.1. To apprehend attackers and to deter future attacks, forensic investigators need a comprehensive and potent approach to track down the attackers to their source and apprehend them. One way of achieving the goal is to build a comprehensive legal mechanism and a lot of progress has been made in this regard [8]. However, to collect evidence and facilitate traceback of attackers, they need the cooperation of intermediate networks that carry the stepping stones, which can be very difficult in some cases. Due to international policies, the cooperation of the intermediate networks cannot be guaranteed. We need to find a way to trace the attacker that is simple yet effective. Hence, IP traceback and stepping stone detection have gained high priority in network security domain [1].

Few approaches towards IP traceback have been proposed that are broadly classified into two categories: Host-based and Network-based approach. The *host-based* approach [9] [10] requires some kind of monitoring software to be installed on each participating host. This approach has a disadvantage as the attacker can manipulate the results of the monitoring software if he has control over the host machine. The *network-based* approach [5] [6] [11] requires tracing software to be installed in network routers and switches. This ensures that the whole of the network comes under the purview of the scan and the hosts do not need to participate individually.

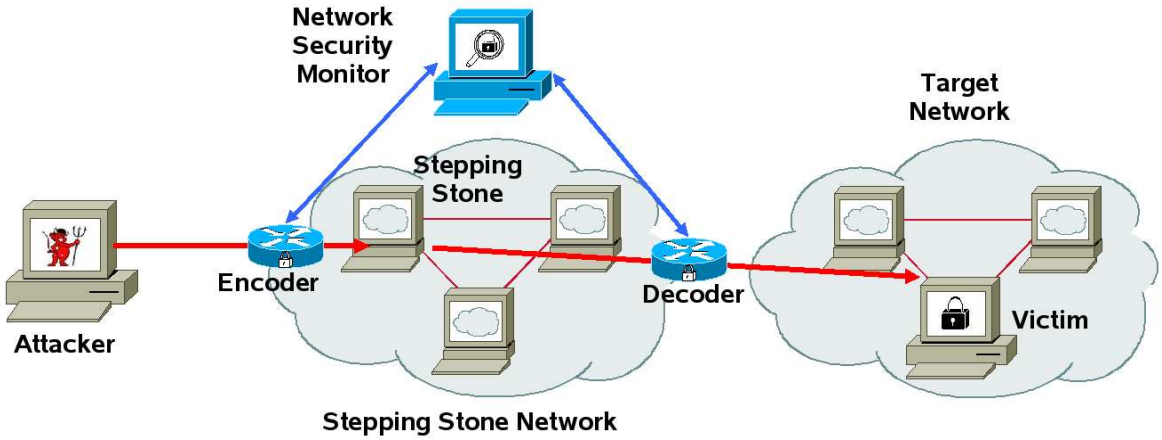

Figure 2.2 Basic approach towards stepping stone detection

2.3 Stepping Stone Detection

It is important for the investigators to detect whether a node was used as a stepping stone. The intuitive approach would be to try to correlate incoming and outgoing connections. This approach is implemented by intercepting the incoming and outgoing packet streams at a node. Network level devices fingerprint the incoming and outgoing packet streams and the information is sent to a network monitor that tries to match them. The packet stream could be fingerprinted based on its contents, inter-packet timings or any other parameter which is suitable for the purpose. The characteristics of the incoming and outgoing streams are compared to determine whether it is a stepping stone. The basic approach towards stepping stone detection is illustrated in Figure 2.2. Passive monitoring and active perturbation are the two main approaches for detection.

Passive monitoring involves correlating traffic streams based on characteristics of traffic. The problem of interactive stepping stones was formulated and proposed by Staniford and Heberlein [5]. They proposed a content-based algorithm that created thumbprints of packets streams and compared them. This approach was rendered in-

effective due to the growing use of encryption. Zhang and Paxson [6] were the first researchers who were able to correlate encrypted traffic streams. Their algorithm was based on the ON/OFF periods of the interactive traffic. This approach formed the basis of current research. A deviation-based approach was proposed by Yoda and Etoh [11] that used packet timings and sequence numbers to correlate packets streams. An approach purely based on inter-packet timings was proposed by Wang, Reeves and Wu [12]. Donoho et al. [13] proposed that an attacker can selectively or randomly delay packets at the stepping stone to perturb the timing characteristics of a connection. This seriously degrades the effectiveness of timing-based correlation.

Active perturbation of the packet streams turned out to be an effective approach against the perturbations introduced by the attacker. In this approach, an incoming packet stream is modified and those perturbations are detected in the outgoing streams. The easiest way would have been to modify the packet contents or the packet header or inserting additional packets in the stream. However, that would be easy to detect for the attacker and hence it was imperative that the perturbations be transparent. Wang and Reeves [3] proposed the first active watermark based correlation scheme that was robust against random perturbations by the attacker. The concept was later on adapted by Wang, Chen, and Jajodia [14] to track VoIP calls on the Internet. This scheme had many drawbacks however as it failed against packet drops by the attacker, it could not deal with cover traffic and it also failed in cases where the packet flows split, mixed or merged. It also needed sender/receiver synchronization. To address these defects, Wang et al. [15] improved their concept by introducing an interval-centroid-based watermarking technique. This technique could handle cover traffic introduced by the attacker and it could deal with flow splitting, mixing and merging. It also did not require the encoder and decoder to be synchronized. The technique is currently the best available in watermark-based stepping stone detection. We discuss this technique in detail in the Section 2.4.

2.4 Watermark-based correlation

The effectiveness of watermark-based stepping stone detection techniques make them an interesting topic for further research. Watermarking primarily consists of an encoding stage in that a network monitor embeds a watermark in the incoming packet stream and a decoding stage where you look for the watermark in the outgoing stream. A watermark is a sequence of binary bits thus, encoding the watermark involves modification of the packet timings in a way that creates a pattern that represents the watermark's bit sequence in the incoming stream. We discuss two ways in which the watermark can be encoded and decoded, that are of particular interest to us.

2.4.1 Probabilistic watermarking to trace VoIP calls

In this technique proposed by Wang et al. [14], a fixed number of packet pairs are chosen independently and probabilistically for encoding a particular bit. The chosen packets are delayed by a certain amount. Since the attacker doesn't know which packets are delayed, he cannot directly adjust the packet timings to degrade the watermark. Following is a brief description of the watermark embedding process:

- Consider a packet flow $P_1, ..., Pn$ with time stamps $t_1, ..., t_n$ respectively where ($t_i < t_j$ for $1 \leq i < j \leq n$).
- Sequentially look at first $n - d$ ($0 < d \ll n$) packets and determine independently, with probability $p = \frac{2r}{n-d}$, whether the current packet will be chosen.
- Independently and randomly select $2r$ distinct packets denoted as $P_{z1}, ..., P_{z2r} (1 \leq zk \leq n - d$ for $1 \leq k \leq 2r)$.
- For each of these packets say P_{zk}, another packet is chosen at a distance d to create $2r$ packet pairs: $< P_{zk}, P_{zk+d} > (d \geq 1, k = 1, ..., 2r)$.
- *Inter-Packet Delay (IPD)* is calculated for each of these packet pairs $< P_{zk+d}, P_{zk} >$ as: $ipd_{zk} = t_{zk+d} - t_{zk}, (k = 1, .., 2r)$.

- Thus, the 2r IPDs obtained are divided into two groups of equal size $ipd_{1,k,d}$ and $ipd_{2,k,d}$ $(k = 1, ..., r)$.
- Let $Y_{k,d} = \frac{(ipd_{1,k,d} - ipd_{2,k,d})}{2}$ $(k = 1, .., r)$ be the difference in between individual IPDs of the groups.
- Then, the average of group of normalized differences is represented as : $\overline{Y_{r,d}} = \frac{1}{r}\sum_{k=1}^{r} Y_{k,d}$.
- As $\overline{Y_{r,d}}$ is symmetrically centered around 0, increasing or decreasing it by an amount $a > 0$ will shift the distribution to the left or right. This property is used for embedding bits '0' or '1' by decreasing or increasing the value $\overline{Y_{r,d}}$.
- We decrease or increase $\overline{Y_{r,d}}$ by delaying the packets in $ipd_{1,k,d}$ or $ipd_{2,k,d}$ respectively.

For decoding the watermark, we calculate $\overline{Y_{r,d}}$ to decode the bits.

The advantage of this scheme is its robustness against random perturbations of packet timing by the attacker. It also guarantees even time adjustment and it is suitable to be applied to VoIP traffic that flows at near constant rate. The watermark that is embedded is difficult to detect for the attacker.

The disadvantage of this scheme is that it requires encoder/decoder synchronization to function properly and it is not robust against packet drops by the attacker. This scheme cannot handle inter-flow or intra-flow transformations.

2.4.2 Interval-centroid-based watermarking

This technique [15] has also been proposed by Wang et al. The time duration for embedding the watermark is divided into equal intervals and using the timings at which the packets appear in the interval, the centroid of the interval is calculated. The difference between time of arrival of a packet and the start time of the interval in which appears, is called as the *time difference*. The mean of time difference for all the packets in an interval is defined as the *interval-centroid*. The difference between values of interval-centroids is

used to encode or decode bits thus making it an interval-centroid-based technique. This technique is an improvised version of the previous watermarking algorithm described in the Section 2.4.1. It can be used to detect a stepping-stone attack even if the attacker uses cover traffic or packet flow splitting and merging. The common procedure for encoding and decoding bits is as follows:

- Create $2n$ intervals for the time duration T_d where $n = R * L$ where R is the redundancy number and L is the number of bits in the watermark.
- With equal probability, randomly create group A and B with n intervals each.
- Assign R intervals each from Group A and Group B to each bit $i \in L$.

To encode a bit '0' or '1', carry out following steps:

- For each bit $i \in L$, for each interval $j < 2R$ assigned to it, for each packet $k \in$ interval j, calculate $\Delta t_{i,j,k}$ and $\Delta t\prime_{i,j,k}$ where $\Delta t_{i,j,k}$ is time difference from start of interval and $\Delta t\prime_{i,j,k} = a + \frac{(T-a)\Delta t_{i,j,k}}{T}$ is the adjusted delay.
- To encode bit '1', delay the packets in Group A intervals by $\Delta t\prime_{i,j,k}$.
- To encode bit '0', delay the packets in Group B intervals by $\Delta t\prime_{i,j,k}$.

Please refer Appendix A, Algorithm 3 for further details.

To decode the bit '0' or '1', carry out following steps:

- Store the arrival timings of all the packets in the 2n intervals.
- Calculate the centroids of packets in each interval.
- For each bit $i \in L$, calculate the aggregated centroids $Cent_A$ and $Cent_B$ of Group A and Group B intervals.
- If $Cent_A - Cent_B$ is positive then we decode that bit as '1'.
- If $Cent_A - Cent_B$ is negative then we decode that bit as '0'.

Please refer Appendix A, Algorithm 4 for further details.

2.5 Evading detection

For evading detection, the attacker select nodes that are seldom used and are not monitored frequently, for using them as stepping stones. However, in our technique, the attacker selects stepping stones that experience a lot of incoming and outgoing traffic streams resulting due to various protocols like SSH, HTTP, constant rate multimedia etc. The high volume of traffic on the node helps the attacker to continue using the node as stepping stone without being detected. The attacker cloaks his attack traffic as any one of the legitimate traffic streams; let us consider constant rate multimedia traffic in this example. Thus, attacker's traffic stream remains indistinguishable from the traffic streams.

Venkateshaiah [2] proposed an algorithm that helps the attacker evade stepping stone detection by using buffering and chaff along with selective dropping of packets. The key idea was to remove any correlation between the input and the output streams. This can be achieved by making the output stream send packets at preselected times that are independent of the input stream. Data packets from the input stream are placed into a buffer. When the output stream sends a packet, it first checks the buffer for data. If data is present in buffer, it is sent, otherwise it sends a chaff packet. We could use any pre-determined pattern for the output stream, such as random or bursty, as long as it is independent from the input stream. We choose to use a constant-rate stream for both performance and the fact that it can be made to look like any number of multimedia streams. Performance is important for interactive sessions, and a constant rate stream with many packets will provide the output stream a chance to send buffered data soon after it arrives. However, if a constant rate packet stream is encountered and is not a multimedia packet stream, then it might be flagged as suspicious. So, we convert this traffic into a multimedia packet stream by encoding the attack data as the payload of a

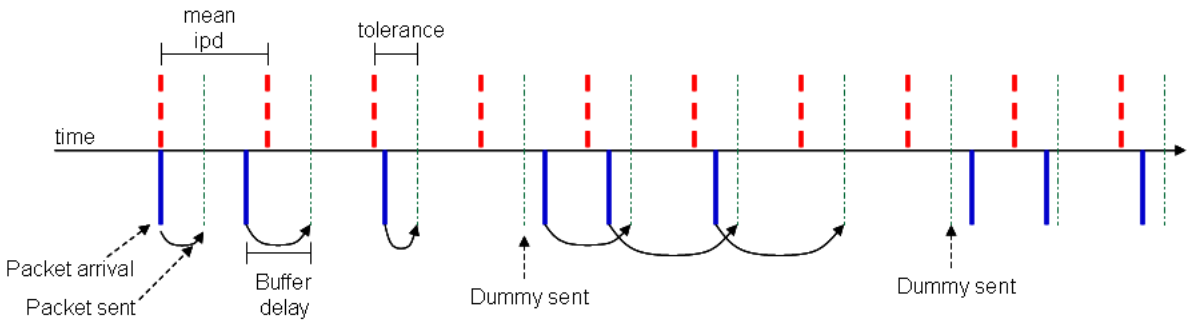

Figure 2.3 Traffic time-line. Source: [2]

multimedia packet and to avoid content-based detection, we encrypt the payload of the packets. Following is the summary of the algorithm:

- Profile the connection and obtain the standard deviation of inter-packet delays.
- When the first packet arrives, delay the packet by the maximum buffering delay.
  - From the arrival of first packet, wait for a tolerance of 3σ, and trigger a send event at intervals of duration $\overline{ipd}$, where σ is the standard deviation of the inter packet delays of the incoming packet stream.
  - From the second packet onwards, buffer the packets that arrive before an event is triggered.
- When a send event is triggered:
  - If there is no packet in the buffer, send a dummy packet.
  - If there are more than two packets in the buffer, drop the first packet and send the second packet.
  - If there are less than or equal to two packets in the buffer, send the first packet in the buffer.

The algorithm is illustrated in Figure 2.3. The authors have simulated their algorithm and verified that it is practically feasible to implement such an algorithm. The disadvantage of this algorithm is that it does not perform well under adverse network conditions.

CHAPTER 3

SNEAK SYSTEM DESCRIPTION

In this chapter we describe the components of SNEAK and define parameters for measuring the performance of the system. The system primarily consists of two algorithms that help in removing the timing information from the packet streams. We also discuss performance and consider the tradeoffs in the use of these algorithms.

3.1 Overview

As described in Section 2.4, the detection mechanism tries to correlate traffic flows for *stepping stone detection* or *IP traceback*. For our research, we are specifically interested in the timing-based detection approach and we develop a mechanism to evade the detection. The technique of active perturbation proposed by Wang et al. [15] is currently considered to be the best available detection mechanism. We try to defeat this detection mechanism using the approach proposed by Venkateshaiah [2] that uses buffering and chaff and selective dropping of packets to evade detection. This approach is described in Section 2.5. We use the same assumptions made by Venkateshaiah [2] that are as follows:

- The attacker has complete control over the stepping stone and he can install or modify any software on the system.
- The attacker has access to the victim and can install and operate a small server on the victim.
- The detection mechanism does not have prior knowledge of the presence of a stepping stone in the network.

- The attacker does not know which packets are watermarked by the detection mechanism.
- The attacker does not know about any of the watermarking parameters, i.e. delay, offset, etc.

3.2 SNEAK components

Consider an attacker who is using a stepping stone to attack his victim. For simplicity, let us assume that the attacker is using just one stepping stone. The detection mechanism is trying to expose the stepping stone by watermarking the incoming stream to the stepping stone and looking for the watermark in the outgoing streams. The attacker tries to evade detection by severely degrading the embedded watermark so that it is difficult to detect with accuracy. The attacker uses three components of SNEAK to carry out this attack: the Client, the Server, and the Agent. The attacker uses the Client to issue commands, while the Server runs on the victim, receives the commands, executing them on the shell, and sending back the replies. The Agent program resides on the stepping stone and removes all the timing information embedded by the detection mechanism. We concentrate mainly on the algorithms used in the Agent program as they form the crux of the evasion mechanism. The basic use of SNEAK is illustrated in Figure 3.1.

3.3 Robustness, Usability and Effectiveness

In this section, we formally define the terms *robustness*, *usability* and *effectiveness*, as used in the evaluation of SNEAK. Let us consider a typical SNEAK system where an attacker uses the Client, Agent, and Server to carry out an attack using stepping stones as illustrated in Figure 3.1. If the attacker sends a command and receives a reply, we

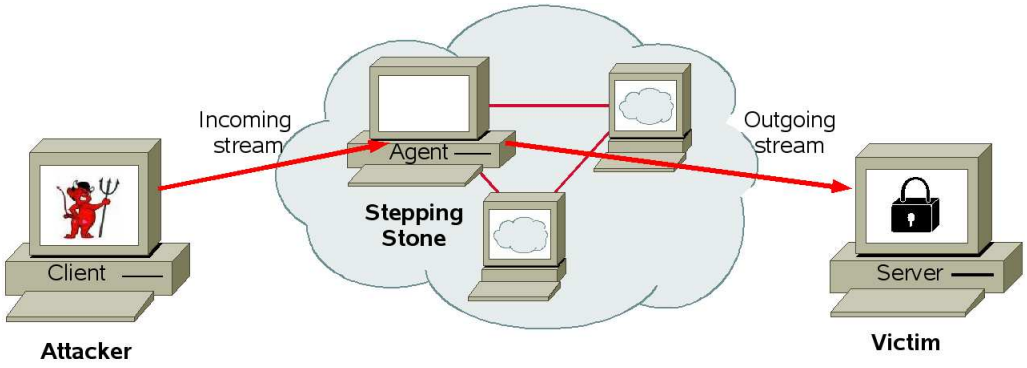

Figure 3.1 Basic attack using SNEAK

call it a *successful command* and the time elapsed between the issuing of the command and the receipt of the first reply packet is called as the *response time*. In the context of this attack, the percentage of successful commands is used as the metric for *robustness* and the response time is used as the metric for *usability*. For an ideal system, hundred percent success rate is essential and expected response time should be no worse than the case when SSH is used for attack. SSH protocol was designed to work on TCP protocol that provides a very good quality connection that gives lowest possible response time while giving high robustness.

Consider a scenario in which a network monitor embeds a binary watermark in the packet stream from the Client to the Agent and tries to detect the watermark in the stream from the Agent to the Server. Then, the Hamming distance[1] between the binary string embedded and detected by the monitor is called the *bit difference*. The average bit difference is used as the metric for *effectiveness*. If the average bit difference is too low, then the detection mechanism can have required amount of confidence about presence of a watermark. On the other hand if the average bit difference is too high, then it can be argued that the watermark is present, but it has been flipped. For the detection mechanism to have the least confidence, it is imperative that the bit difference

[1]For binary strings a and b, the Hamming distance is equal to number of ones in $a \oplus b$.

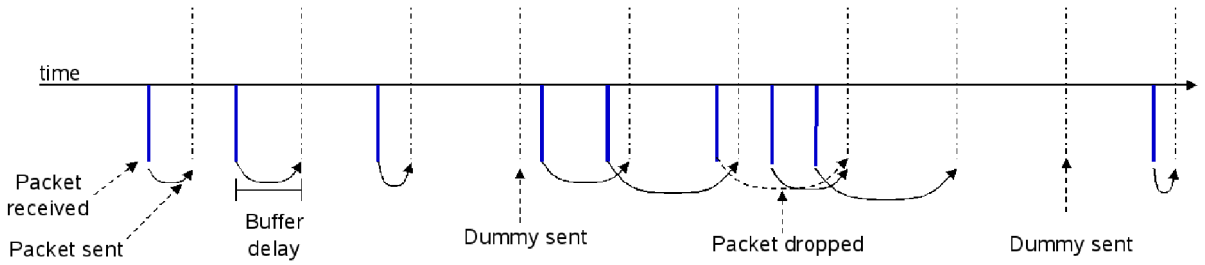

Figure 3.2 SNEAK sender-side dropping algorithm timeline

should be close to 50% of the total number of bits in the watermark. Additionally, it has been observed in our experiments that for an unwatermarked stream, the average bit difference observed is around the half way mark, which implies that the confidence of the detection mechanism is least for an unwatermarked packet stream. For example, consider a network monitor that is using sixteen bit watermarks for stepping stone detection. If the monitor finds the resulting average bit difference in the outgoing stream to be closer to eight bits, then the stream is classified as unwatermarked. If the outgoing stream is a result of an evasion algorithm used by an attacker, then we call it an *effective* algorithm.

3.4 Algorithm 1: Sender-side dropping algorithm

In this section, we describe the first algorithm proposed by Venkateshaiah [2]. Since the attacker does not know which packets are watermarked, he removes all the timing information by generating a constant rate output stream, independent of the packet timings in the input stream. For that, the attacker buffers packets that arrive early and drops packets when they arrive late. The attacker has the knowledge of the $rate$ at which he is sending packets from the Client. He triggers a send event at every $\overline{ipd} = 1/rate$ microseconds. The attacker divides the time into fixed-length *slots*, beginning with the arrival of the first packet. The length of each slot is $\overline{ipd} = 1/rate$, which is the mean inter-packet delay (IPD) at the source of the traffic. Since the attacker is unlikely to

generate a data packet in every slot, the Client adds cover traffic as needed to ensure a constant rate packet stream. It buffers and selectively drops packets using the following scheme. Each packet is expected to arrive in its respective slot, but packets may arrive earlier or later than expected due to the Internet and watermarking delays encountered by packets. If an incoming packet arrives early, it is delayed till the next available slot. If an incoming packet is delayed, i.e. it does not arrive in its slot, then a dummy packet is sent instead. When the delayed packet arrives, it is buffered and sent at the next available slot. Some packets may arrive very late immediately followed by the packets that arrive in time. If the sender finds more than 2 packets in the buffer, it drops the one that arrived first and sends the next one. We call this *Strategy 1*.

This algorithm was successful in generating a constant rate traffic stream at the output that was independent of input stream. It performed well under normal network conditions but gave increased response times under adverse network conditions. We found that the packets would queue up in the buffers and increased the buffering delays and thereby increased response time. We further discuss performance issues in detail in Section 3.6

We made some changes to the algorithm and added *Strategy 2* that requires the sender to drop packets more aggressively to prevent packet build up in the buffers. In adverse network conditions, we often find more than two packets in the buffer. In *Strategy 2*, we drop all but the last two packets in the buffer. Aggressively dropping packets also affects the overall robustness of the system. To deal with packet loss in general, we create redundancy in the number packets containing data and we call it *redundancy number 'r'*. Whenever we send a data packet, we send r copies of it. Sending these redundant copies of a packet does not cost us additional resources as they substitute the cover traffic (dummies) required to be sent in the absence of a data packet in the buffer. The cost of filtering out the redundant copies is minimal at the Server.

In the original algorithm by Venkateshaiah [2], the Agent first profiled the incoming connection and using $\overline{ipd} = 1/rate$ as the mean, calculated the standard deviation of IPDs called σ. It was assumed that jitter follows normal distribution. Prior research has suggested that jitter should be exponentially distributed [16] or gamma distributed [17]. The use of a normal distribution helps in designing the system for good performance, but this approach can be modified for various network conditions.With this knowledge of the distribution of the IPDs around the mean, the Agent could adjust the start time of the outgoing stream, in tolerance. In theory, tolerance was supposed to help in adjusting the start of send intervals to account for the normally distributed inter-packet delays. However, we found that the tolerance hardly had any effect on the robustness or usability of the algorithm. We have seen that even if we do not use tolerance and a packet comes later than expected, it gets buffered and is sent in the next slot. Figure 3.2 illustrates the timeline for the modified algorithm. For a full specification, see to Algorithm 1 of Appendix A.

3.5 Algorithm 2: Receiver-side dropping algorithm

Under adverse network conditions, usability and robustness of sender-side dropping algorithm was affected. The *Strategy 1* gave better robustness, but gave higher response times, while *Strategy 2* improved the response time but the robustness degraded. It was identified that the packet build up in the buffers was the root cause of problems. These issues are further discussed in detail in the Section 3.6. One way to solve the issue of packet build up would be to reduce the amount of data being buffered. We need to buffer only one copy of data packet and drop all the redundant copies. This can be done at the Agent while receiving the incoming packets and while sending the packet to the next hop, the redundancy can be reintroduced. This approach has two advantages: first we do not pay additional cost for buffering redundant packets, and second we maintain complete

redundancy on all hops that ensures that we have enough redundant packets to counter the drops.

In sender-side algorithm, we buffer all the packets including the late packets. In the case of a bunch of late packets, we buffer and and later drop them in case there is packet build up in the buffer. As a result we end up wasting some resources in the process. We would save some resources if we directly drop all those packets. We need a way to determine whether a packet is late, and the measure of lateness of a packet should depend on the prevailing network conditions. Hence, we need to profile the connection to determine the standard deviation of inter-packet delays. We use the connection profiling technique proposed by Venkateshaiah [2]. In this technique, we record the inter-packet delays (IPDs) of the incoming packet stream and we already have the knowledge of the *rate* at which we send the packets from the Client. We assume that the inter-packet delays are normally distributed when the packets reach the Agent. Based on that, we calculate the $\overline{ipd} = 1/rate$ and use it as mean to calculate the standard deviation σ as follows:

$$\sigma = \sqrt{\frac{1}{n-1} \sum_{i=1}^{(n-1)} (ipd_i - \overline{ipd})^2}$$

Here n is the total number of packets in consideration and ipd_i is the inter- packet delay between packet P_i and the previously received packet.

In this algorithm, we try to eliminate all the late packets by using σ to determine it's lateness. We know that the IPD distribution is a normal distribution, and hence we know that almost all the IPDs will lie within $3 \times \sigma$ of the mean. Thus, if the IPD is greater than $\overline{ipd} + 3 \times \sigma$, then we instantly drop the packet, otherwise we buffer it. Using this algorithm, we do not expect more than two packets in the buffer at any given time. When a send event is triggered, we send a data packet if it is available, or we send a dummy packet instead. The process can be summarized as follows:

- Profile the connection to determine the standard deviation σ of the IPDs.

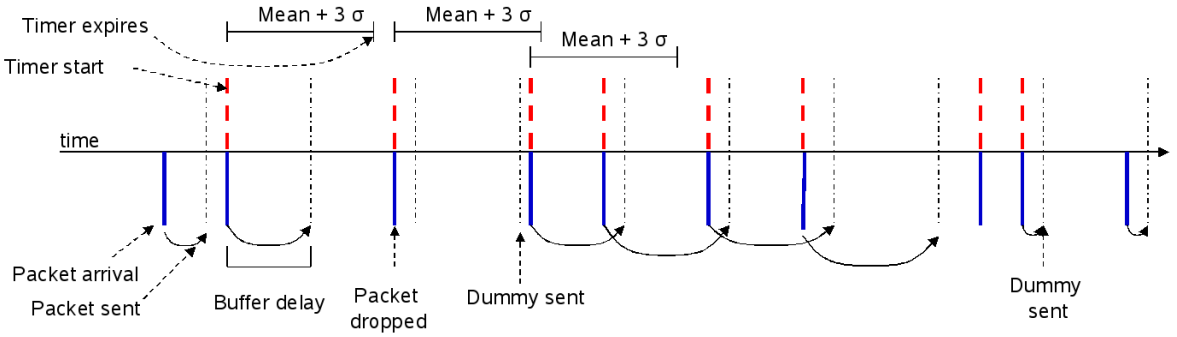

Figure 3.3 SNEAK receiver side dropping algorithm timeline

- The receiver thread accepts only those packets that have come within $\overline{ipd} + 3\sigma$ of the arrival time of the last received packet.
- After accepting a packet, put it in the buffer; discard any duplicates (see Section 4.2).
- The sender thread wakes up according to the constant rate schedule and checks for packets in the buffer.
  - If there is a packet in the buffer, remove it from the buffer, copy it, and send it; schedule the redundancy numbers worth of duplicates in the coming slots.
  - If there is no packet in the buffer, send a chaff packet.

Figure 3.3 illustrates the timeline for the algorithm. For further details, please refer Appendix A, Algorithm 2

3.6 Performance issues and trade off

Both the algorithms are equally effective against watermarks though they exhibit different robustness and usability characteristics. The use of a particular algorithm or a strategy should be dictated by the requirements of the user. In case of Algorithm 1, the fixes we introduced eliminated the cascading delays in case of the original algorithm under

adverse network conditions and redundancy in packet traffic balanced the aggressive dropping strategy. Although it increased the robustness of the algorithm and avoided cascaded delays in the buffer, the usability of the algorithm suffered up to a certain extent. To counter the high drop rate, we used redundant packets that get queued in the buffers till they get dropped at the server. Buffering all these packets required some additional resources, and those ultimately get wasted as the redundant packets are dropped at the server. However, it has been experienced that the success rate was really high. Thus, Algorithm 1 with both strategies performs well in terms of robustness.

In case of Algorithm 2, under normal network conditions, not more than two packets are enqueued in the buffer at one time. However, this algorithm causes packet build up in case of varying network conditions. Suppose if the network conditions are adverse when the connection is profiled and then improve over the course of the connection and become normal. If under the adverse conditions, the standard deviation σ is greater than or equal to the mean $\overline{ipd}$, then during the normal conditions, the factor by which it is greater decides the number of packets entering the buffer at a time. For ex: if σ becomes twice the mean $\overline{ipd}$, then two packets are put in the buffer. This condition can be avoided by putting an upper bound on the value of the σ. Another problem is that, this scheme will drop a packet that comes late even if there are no other packets waiting in the buffer. This cuts out late packets, decreasing average response time but also affecting the success rate. However, with an increase in redundancy number, this Algorithm 2, gives the required usability and robustness.

CHAPTER 4

SNEAK PROTOTYPE

In this chapter, we describe the design of an experimental prototype application that employs our algorithm to evade detection.

4.1 Prototype Design

Our goal is to create a pseudo-shell application that allows the attacker to run commands on the victim's machine. This is a proof of concept application and it lacks the superior shell functionality provided by Telnet and SSH. The software has been programmed in C on Linux kernel 2.6.22. We send packets using the Real-time Transport Protocol (RTP) [18], thus creating a packet stream that resembles VoIP or some other multimedia. RTP provides end-to-end delivery services for data with real-time characteristics, such as interactive audio and video data. Applications typically run RTP on top of UDP to make use of its multiplexing and checksum services. Many of the popular VoIP protocols such as H.323 and SIP use RTP as their transport level protocol [19].

We also need to encrypt the payload to prevent the contents of our packet streams from being inspected by monitoring software. Hence, we use the Secure Real-time Transport Protocol (SRTP) [20], a profile of RTP that provides confidentiality and message authentication. We used libsrtp-1.4.2, which is a freely available implementation of SRTP. We also used a fixed packet size of 64 bytes to match existing VoIP packets (packet sizes vary depending on the codec). Our application consists of three components: the *Client*, the *Server*, and the *Agent*. The Client resides on the attacker's machine, while the Server resides on the victim's machine. Note that, for attacking hosts that are yet to be com-

promised, the victim could be used as a stepping stone. The incoming SRTP stream and any outgoing packets caused by shell commands should be difficult to correlate, but the Server can add random delays to make it even more challenging. Finally, the Agent resides on the stepping stone and it relays the packets between the Client and the Server. For a multiple number of hops, we need to install the Agent on each of the nodes involved.

4.1.1 SNEAK Client

The Client program issues the attack commands that are to be run on the victim's machine. It uses separate threads for sending and receiving (the *sender* and the *receiver*). The sender continuously sends packets at a constant rate over a UDP socket, while the receiver simultaneously receives packets on the same socket and prints them out on the screen. The sender thread monitors the attacker host's standard input and prepares any typed commands to be sent to the Agent in the next time slot. If any commands or replies require more data than is available in our fixed packet size, we simply break the data into multiple packets and queue them all for sending one at a time in the constant rate stream. Whenever there is no command available, it keeps sending chaff packets. To create chaff packets, we simply fill the payload with "NUL" characters. Thus, the chaff packets are filtered out by the recipient by testing for the presence of all "NUL" characters in the payload. The use of AES counter mode prevents the observer from determining which packets are chaff and which are real.

4.1.2 SNEAK Server

The Server program responds to the received command by executing the command on the victim's machine and sending back the response to the Client. It has a two-thread design similar to the Client. The sender forks a child process that creates a shell and ties its standard input, output, and error descriptors with input and output pipes. When the

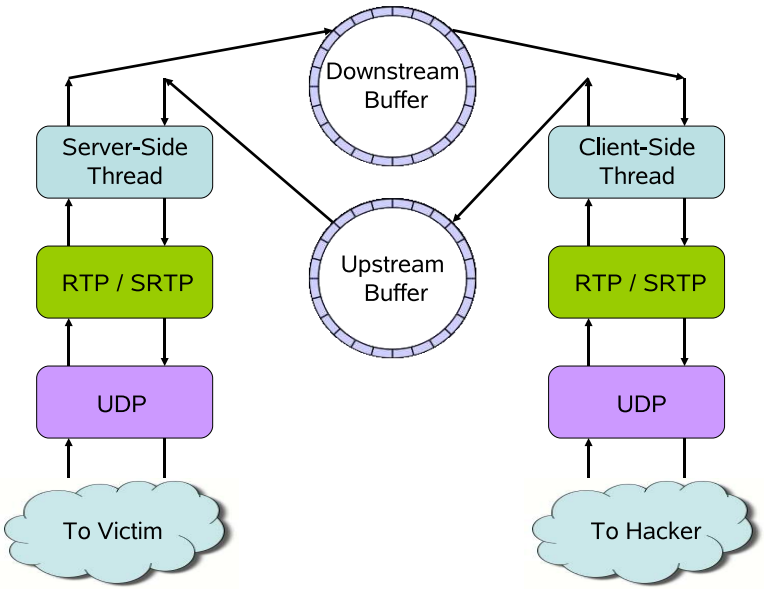

Figure 4.1 SNEAK Agent architecture

receiver receives a command on its UDP socket, it writes to one end of the input pipe. On execution of the command, the shell writes the output on the output file descriptor. The sender listens on the other end of the output pipe and sends the data back on the socket. Similar to the Client, the sender sends at a constant rate, sending data only at the designated times and sending chaff packets otherwise.

4.1.3 SNEAK Agent

The Agent program receives data from the previous host in the connection chain and sends it to the next host. It acts as a relay, while employing our algorithm, to selectively forward or drop packets. It primarily consists of four threads that do the work of sending and receiving in each direction and we call them client-side or server-side threads. The prefix client-side or server-side indicates that the data is coming from the direction of the Client or Server respectively but it may come from an Agent on another host. Another important component is the FIFO queue for each direction, that we call

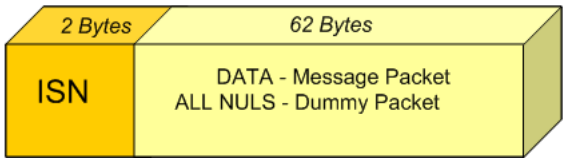

Figure 4.2 Structure of 64 Byte payload.

the upstream buffer (client-side to server-side) and the downstream buffer (server-side to client-side). Whenever any data appears on the client-side socket, the client-side receiver puts the packet in the upstream buffer. The server-side sender picks up a packet from the buffer in the next available slot and sends the packet towards the Server. Similarly, the server-side receiver collects responses from the server-side socket and puts them in the downstream buffer, from that the client-side sender gets them and sends them towards the Client. The architecture of Agent is illustrated in Figure 4.1.

4.2 Robustness mechanism

RTP and SRTP do not provide any quality of service guarantees. They don't guarantee timely delivery or prevent out-of-order delivery. Recreating a TCP-style retransmission mechanism would result in a complex protocol. Hence, for the sake of simplicity, we have implemented a simple mechanism that provides an acceptable level of robustness. For each packet containing a real message, we send multiple copies of the same packet over the network in succession while maintaining the constant packet rate. This increases the probability of the message reaching is destination. We call the number of copies of a packet sent over the network the *redundancy number*. For any network conditions, we can achieve the required level of robustness by increasing the redundancy number. Sending additional copies of the same packet doesn't increase the work for the application, as the application will continually send packets to maintain a constant rate stream.

Additionally, we must remove duplicate packets, so as not to execute commands repeatedly. We track the packets by storing a unique *internal sequence number (ISN)* in each packet sent (including chaff packets). For a 64-byte payload, the first two bytes are used for storing the ISN as demonstrated in Figure 4.2. We mark the packet number as received in a bit vector, according to its ISN, and we drop any duplicate packets that arrive later.

4.3 Miscellaneous implementation issues

We need to ensure that the Agent, which is supposed to be installed on the stepping stones does not consume too many resources in order to remain undetected. The markup tables needed to keep track of the sequence number of the packets received would consume lot of space if array of integers is used. So we used bit vectors to reduce the footprint of the process in memory. We also converted the bit manipulation functions to macros so that the speed of execution is not affected. For the part of profiling the connection,we use the GNU Scientific Library (GSL-1.11). It provides us with useful functions for calculating the standard deviation (σ) of the inter-packet delays of the arriving packets. We used circular buffers for the upstream and downstream buffers in the Agent. The buffers were accessed simultaneously by two threads, the sender and the receiver. To avoid the race condition, we employed mutual exclusion by using binary semaphores.

CHAPTER 5

EXPERIMENTS

In this chapter, we describe the experimental setup of the SNEAK system. We then describe the results on the PlanetLab experimental network.

5.1 SNEAK Experimental setup

The experimental setup involved the use of SNEAK for evading watermark-based detection. We decided to conduct our experiments over two hops but this system should work with as many hops as needed with linear increase in delay. The real life scenario of an attack using stepping stones would involve nodes at a considerable geographical distance, and hence we would expect significant latency and jitter. To test our software against such an environment, we chose nodes that were spread across the United States and we used two hops to attack the victim. For our experiments, our Client was placed on `isec.uta.edu` and we used two Agents that were placed at `ricepl-1.cs.rice.edu` and `pl2.csl.utoronto.ca` respectively. The Server was placed at `ricepl-3.cs.rice.edu`. We selected these nodes for their relatively high performance among PlanetLab nodes we tested. As PlanetLab uses virtualization, many nodes introduce high delays for interactive sessions even when no buffering is used. Figure 5.1 illustrates the experimental setup.

5.2 Watermarking mechanism

We use the algorithms described in the Section 2.4.2 to watermark the incoming packet streams and detect the watermark in the outgoing packet streams. Ideally the

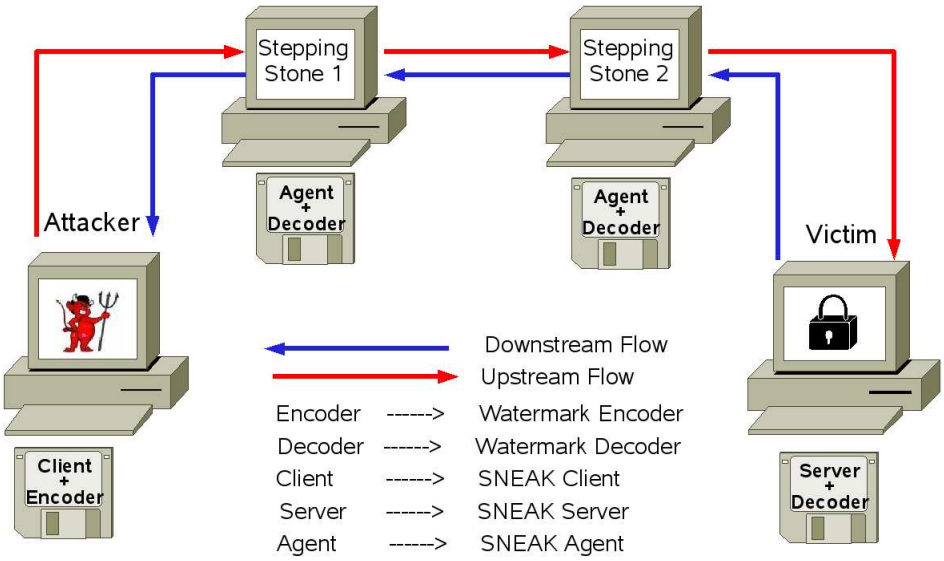

Figure 5.1 Experimental setup of SNEAK

watermark encoder would intercept the incoming traffic and embed the watermark and detect it by intercepting the outgoing stream. This would be typically done by a device like a router in the real world mechanism. However, we just needed a simple proof of concept mechanism to prove that SNEAK is effective against timing-based watermarks. So we coupled the watermarking encoder with the Client and embedded a decoder in the Agent and the Server. The Client encodes the watermark bits and the agent and server report the detection results by calculating the hamming distance between received bit stream and the original binary watermark string. Figure 5.1 illustrates the watermarking setup.

5.3 Results

We now describe the results we obtained by testing our prototype on the PlanetLab network. Our main focus was on testing the robustness, usability and effectiveness of SNEAK. We tested and compared the sender side drop algorithm i.e. Algorithm 1 and the receiver side drop algorithm i.e. Algorithm 2 with all the strategies. The packet rate

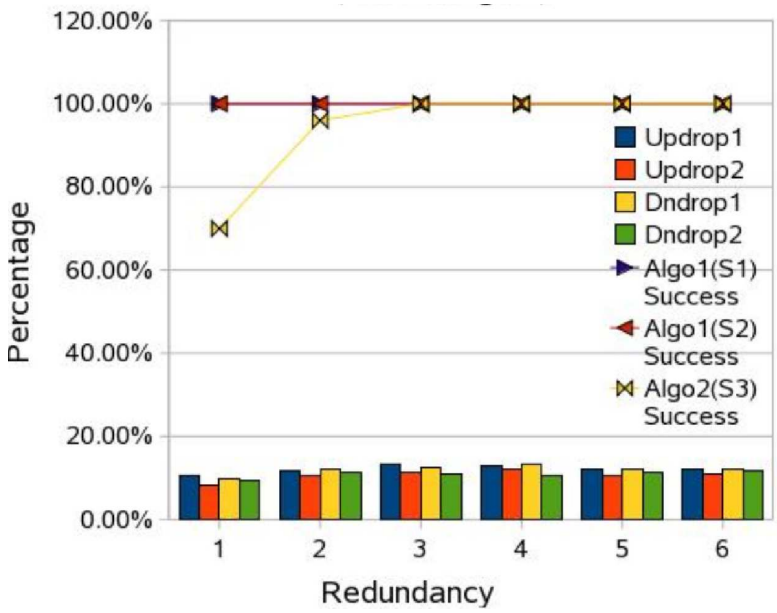

Figure 5.2 Success rates for SNEAK under normal conditions

was 40 packets/sec and the capacity of the upstream and downstream buffers was fixed at 10 packets per buffer. The watermarking mechanism used 16 bit watermarks with redundancy number as 6. For our measurements, we report results from five sessions, each with 10 commands, for a total of 50 commands. We did this for redundancy numbers between one and six, where redundancy number one means that only one copy of packet is sent i.e. no redundancy.

5.3.1 Robustness

To test whether the scheme described in Section 4.2 works to provide sufficient robustness for shell type applications, we tested the effect of redundancy number on the success rate of the queries. We defined success rate as the percentage of successful queries. To evaluate this, we issued a number of commands that would result in a single reply packet being sent, such as `uname` and `date`. One could estimate the success rates of commands resulting in multiple reply packets by the success rates we have obtained.

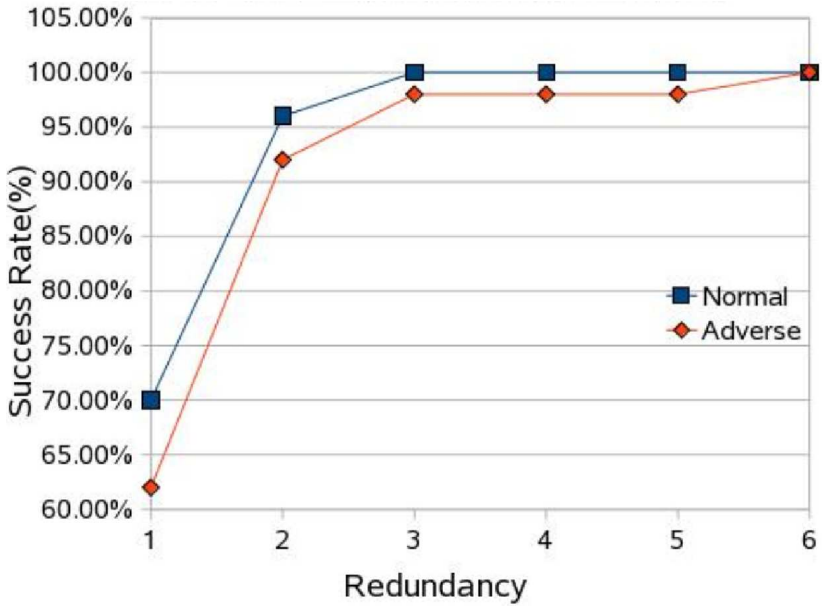

Figure 5.3 Robustness of Algorithm 2 under different network conditions

We show the results of this test in Figure 5.2. For Algorithm 1(Strategy 1 & 2), we see a 100% success rate from redundancy number one till six, i.e. even with no redundancy high robustness was achieved. While for Algorithm 2 (Strategy 3), 100% success rate was achieved at a redundancy number of three and above. Thus, to compensate the drops, we needed to increase the redundancy number to achieve desired robustness. Thus, for normal conditions, even at reasonably low redundancy levels, we have very high success rates. However, these numbers do get affected in case of adverse network conditions. For example, in case of adverse network conditions the Algorithm 2 (Strategy3) achieves desired robustness at redundancy number six as demonstrated in the Figure 5.3. Note that, in most cases, an attacker can rerun a command to get the result. Alternatively, the attacker could have the application adjust the redundancy number as needed so a few critical commands are sure to be correctly received.

Figure 5.2 also illustrates the drop rates experienced during the experiments. *Updrop1, Updrop2, Dndrop1 and Dndrop2* indicate the upstream drop1 (Agent1), upstream

drop2 (Agent2), downstream drop1 (Agent2) and downstream drop2 (Agent1) respectively. We see that no significant drops were experienced for Algorithm 1(Strategy 1 & 2) for redundancy number one to six. While consistant drop rates in the range of 10% to 15% are reported for the Algorithm 2(Strategy 3). The drop rates are relatively consistent across our experiments. Let us assume that all drops are independent events. If we consider the drop rate for the experiments with redundancy number of six, the chance that a given packet makes it to the Server and a response gets back is only 60%. However, with six duplicates, the chance that the query gets through and gets one valid response is $1 - (1 - 0.6)^6 = 99.4\%$. Allowing for some variation, we see that we get approximately what we would expect from our redundancy approach.

We tested the system for usability by measuring the maximum amount of data buffered for a session of 10 single packet commands and taking an average for the total number of sessions. Thus, each of the algorithms was profiled for the maximum amount of data buffered on average. As the buffering delays are associated with any packets that get buffered, the amount of data buffered can affect the response time of the system and thus will affect the overall usability of the system. Figure 5.4, Figure 5.5 and Figure 5.6 illustrate the data buffered by Algorithm 1(Strategy 1), Algorithm 1(Strategy 2) and Algorithm 2(Strategy 3) respectively. *Upbuf1, Upbuf2, Dnbuf1 and Dnbuf2* indicate the data buffered at upstream buffer1(Agent1), upstream buffer2(Agent2), downstream buffer1(Agent2) and downstream buffer2(Agent1) respectively. Each of the readings indicates the sum of all the data buffered for the session. It can be seen that, for Algorithm 2(Strategy 3) the data buffered always remains constant at 256 Bytes while the data buffered for Algorithm 1(Strategy 1 & 2) increases with the increase in redundancy number. These observations are consistent with our theory that the Algorithm 2 is the best in terms of usability as compared to Algorithm 1(Strategy1 & 2). It can also be

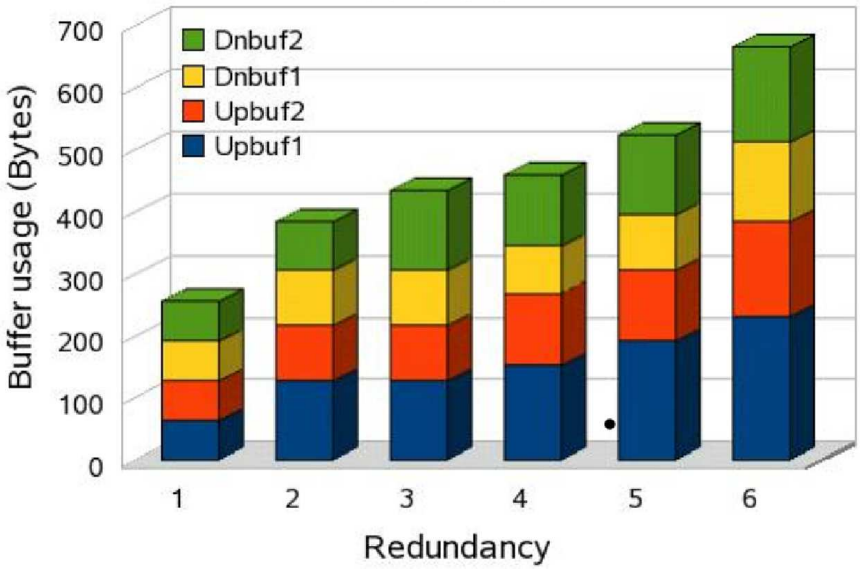

Figure 5.4 Total buffer usage for Algorithm 1(Strategy1)

noted that the maximum amount of data buffered is still less that 1 Kilobyte, which is really less and it helps in maintaining stealthiness for the Agent program.

5.3.2 Usability

We tested the usability of the system by first using it to execute a series of commands manually. We found the response times to be reasonable for interactive use. We also measured the response time of the system from the time the command is sent to the time the first response packet is received. We observed during the experiments that the response time of the system is largely dependent on the prevailing network conditions and the PlanetLab node responsiveness. Despite the variable network conditions, our system consistently provides response times of less than one second, which is sufficient for an attacker to get reasonable use from the system. The lowest response time of around 125 ms was recorded for the Algorithm 2(Strategy 3) at redundancy number six while the highest was around 200 ms which was noted for Algorithm 1(Strategy 2) for redundancy number 4. It can also be noted that the response time will depend upon

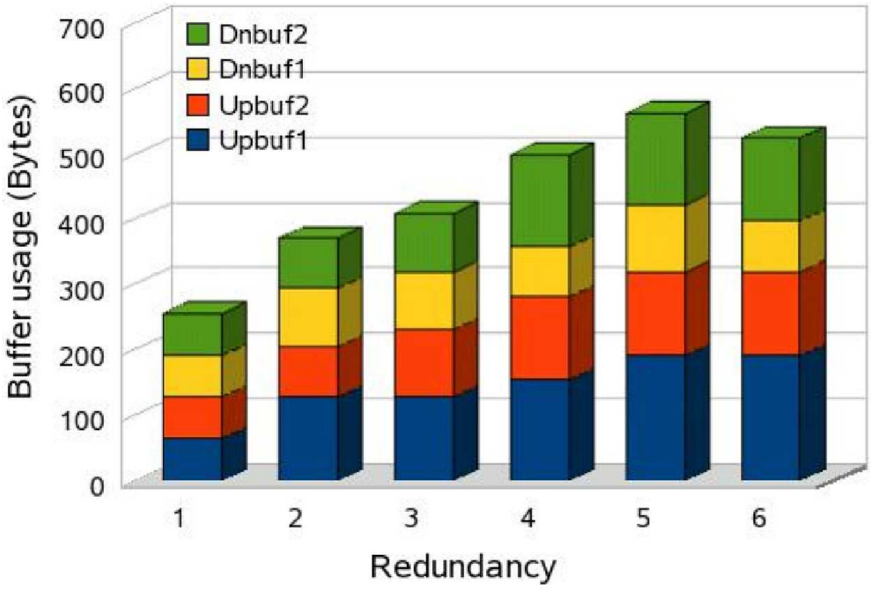

Figure 5.5 Total buffer usage for Algorithm 1(Strategy2)

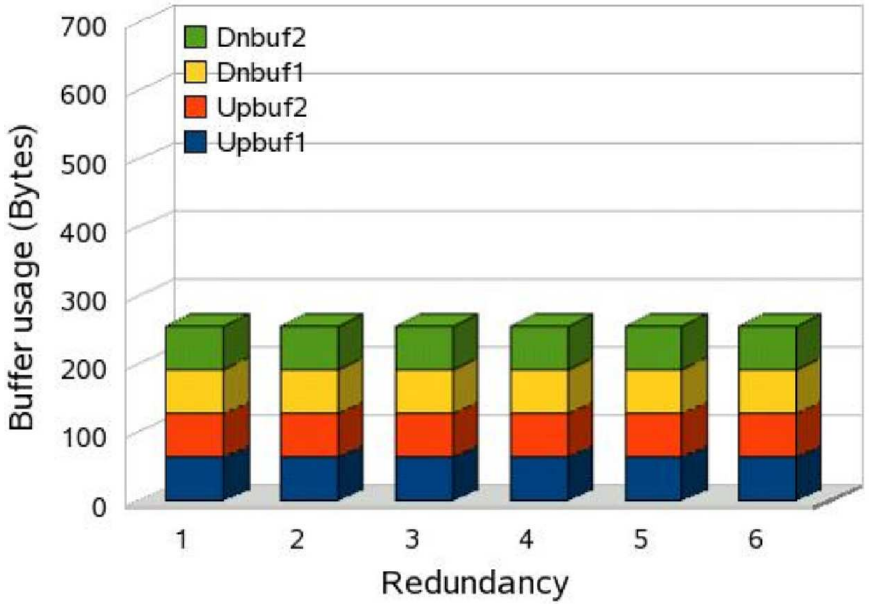

Figure 5.6 Total buffer usage for Algorithm 2(Strategy3)

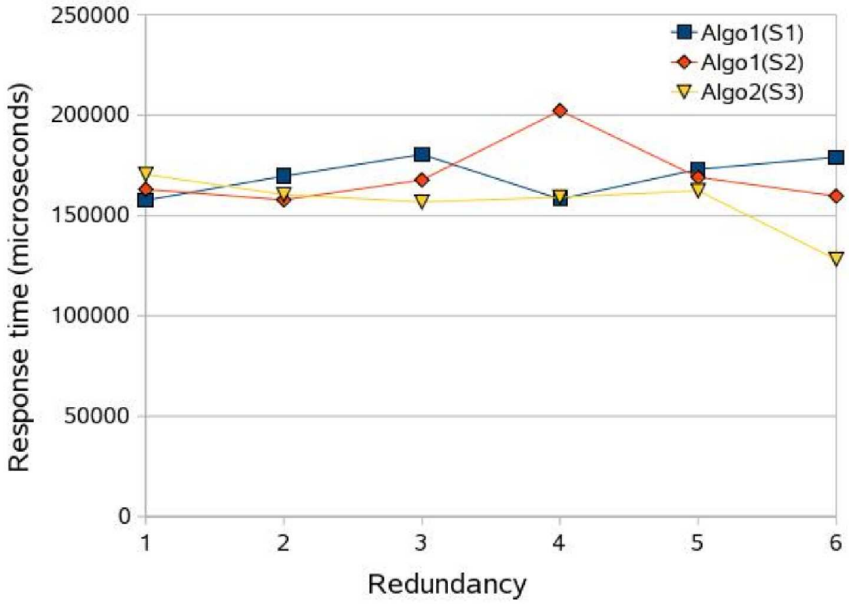

Figure 5.7 Comparison of response times

the amount of data buffered in the upstream and downstream buffers for that particular session. Buffering more data increases the response time, which affects the usability. We also tested the effects of running commands with long results. In particular we ran the command "ls /*", that resulted in high number of packet responses. We found that with redundancy number six, high robustness was achieved with both the algorithms. The Algorithm 2(Strategy 3) performed better than Algorithm 1(Strategy 1 & 2) in terms of usability.

5.3.3 Effectiveness

We now present the results of our experiments to test the effectiveness of SNEAK, in terms distribution of IPDs and the difference in bits. We also verify that our technique causes the watermarking approach of Wang et al. [15] [14] to fail.

It was observed that all 16 bits of the watermark are correctly detected for all the trials for Algorithm 1(Strategy 1 & 2) and Algorithm 2(Strategy 3) at the Agent1 i.e.

the bit difference was zero. This indicates that the watermark was correctly embedded by the Client. The Figure 5.8 indicates the difference in bits of watermarks detected at the Agent2 for Algorithm 1 (Strategy 1 & 2) and Algorithm 2(Strategy 3). The bit difference is distributed around the 8 bit mark with the values spread in the range of 4 to 12. The Figure 5.9 demonstrates the bit difference at the server and the bit difference is distributed around the 8 bit mark with the values ranging from 4 to 13. We measured the distribution of bit difference for an unwatermarked packet stream and it was seen that the distribution is normal around the 8 bit mark. Thus, the results indicate that the incoming packet stream at Agent2 and Server is unwatermarked and the timing information embedded by the Client has no bearing on it.

The Figure 5.10 demonstrates the distribution of inter packet delays(IPDs) for the watermarked stream from the Client to the Agent.It can be clearly seen that the stream is watermarked indicated by the spikes generating a pattern in the plot. The Figure 5.11 demonstrates the distribution of IPDs of the outgoing packet stream from the Agent. The use of SNEAK results in a packet stream that is devoid of any timing information and it flows from the Agent to the Server. Thus from these results, it is evident that SNEAK is effective against watermarking mechanism described in Section 2.4.2.

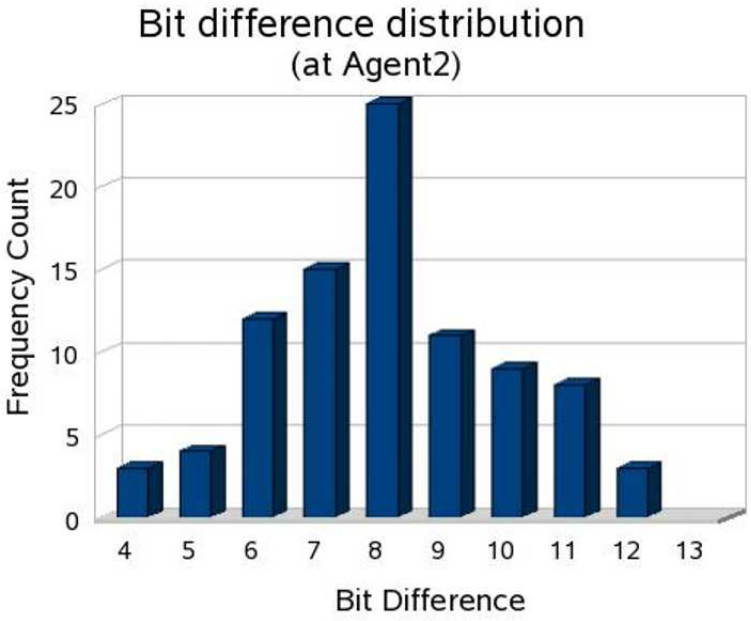

Figure 5.8 Distribution of bit difference at Agent2

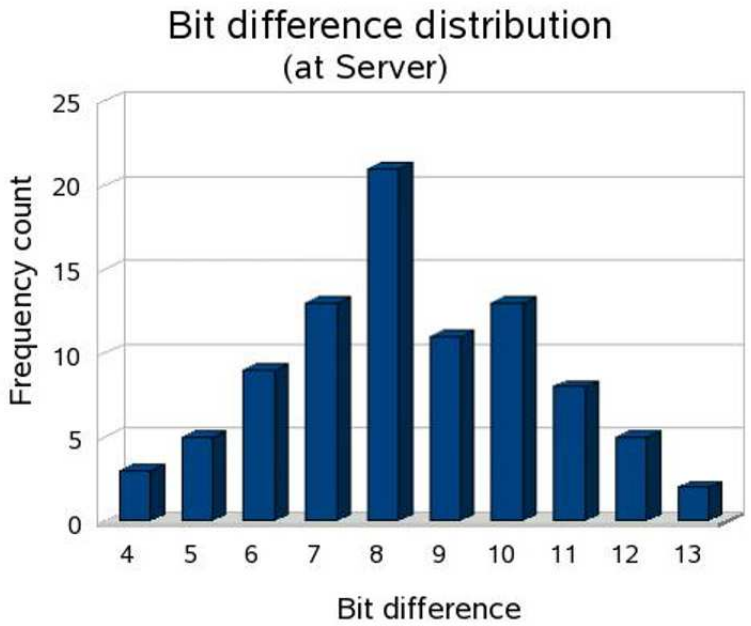

Figure 5.9 Distribution of bit difference at Server

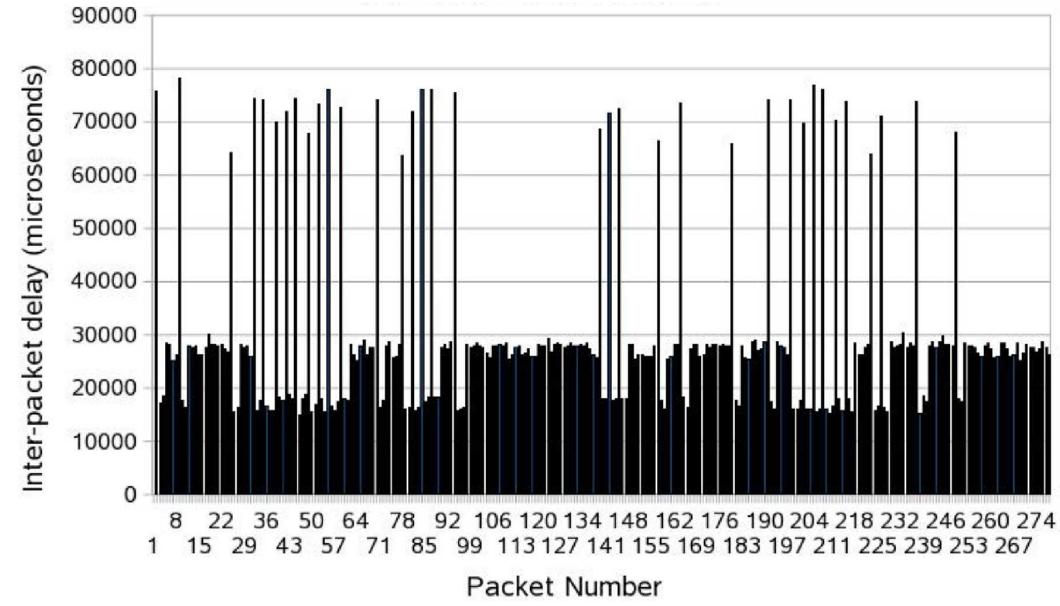

Figure 5.10 Distribution of IPDs for watermarked flows

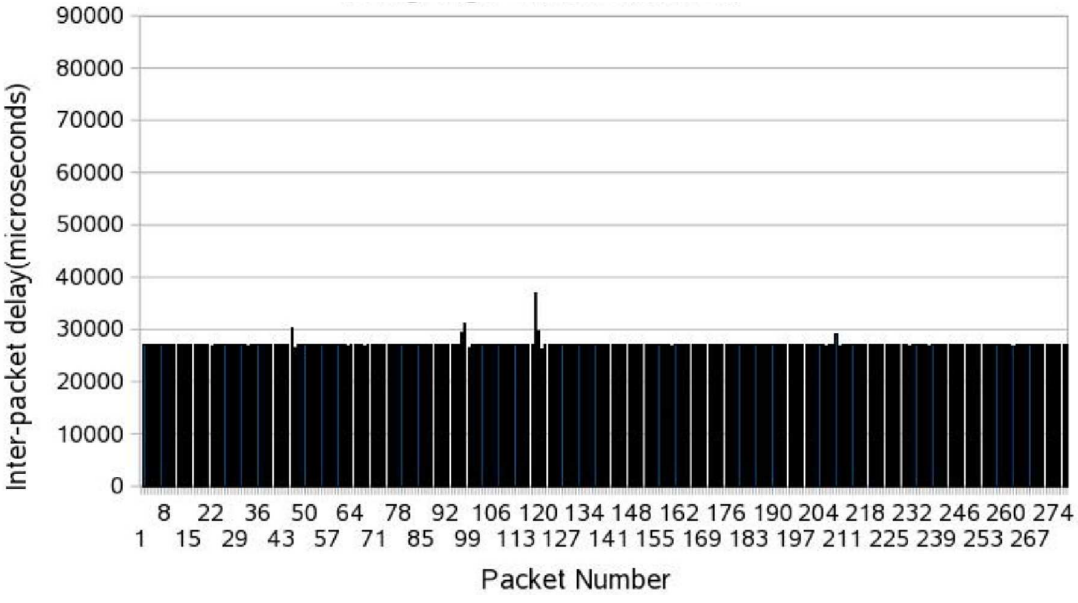

Figure 5.11 Distribution of IPDs after buffering and chaff

CHAPTER 6

CONCLUSION

In this thesis, we have improved the existing method for evading stepping stone detection schemes and proposed a new algorithm for the same. We have proved using experimental data that our system should be effective against all the timing-based detection schemes currently available. Our results also demonstrate that the watermark detection is degraded severely and yet the system remains usable. We have defined the concepts of usability, robustness and effectiveness and specified the metrics for measuring them. We also developed a prototype of SNEAK and verified that it was usable over two stepping stones on the PlanetLab network.

6.1 Future Work

Although our work largely discusses evasion techniques, our our algorithms could be applied by network administrators to secure their networks. As discussed by Gianvecchio and Wang [21], the watermarking technique proposed by Wang, Chen, and Jajodia [14] can be considered to be a type of covert channel. Thus, our technique also breaks timing based covert channels present in the network. Additionally, our technique might be useful for strengthening anonymity systems against watermark-based attacks. In both cases, however, there is a substantial cost that the system would pay for the increased security in terms of bandwidth and latency. The attacker in our system does not pay for the bandwidth and has rather low bandwidth and usability needs, making it more practical in the scenarios we have outlined. Hence, this aspect of the system needs to be investigated further.

APPENDIX A

ALGORITHMS

Algorithm 1: SNEAK algorithm with sender side drops

```
Input: Incoming watermarked packet stream
Output: Constant rate packet stream

/* Strategy1 -> Drop current, send next.
   Strategy2 -> Drop all, till only 2 remain */

/* Receiver thread */

begin
    Start receiving packets from previous hop with sequence number i;
    foreach packet_i where i ← 0 to n do
        if packet_i contains all NULs then
            Discard packet_i as dummy;
        else
            Enqueue packet_i in buffer;
        end
    end
end

/* Sender thread */

begin
    Start sending packets to next hop;
    Trigger send event after every ipd μs;
    foreach send event triggered do
        if buffer ≠ empty then
            if More than two packets in buffer then
                if Strategy1 then
                    Drop current and send next packet ;
                else/* Strategy2 */
                    Drop all but last two packets in buffer;
                    . Send next packet;
                end
            else
                Send current packet;
            end
        else
            Send dummy packet;
        end
    end
end
```

Algorithm 2: SNEAK algorithm with receiver side drops

Input: Incoming watermarked packet stream
Output: Constant rate packet stream

```
/* Strategy3 -> Don't buffer packets that come late */

/* Receiver thread */
```

begin
  Profile connection to get σ *using* $\overline{ipd}$ *as mean*;
  Start receiving packets from previous hop with sequence number i;
  foreach $packet_i$ *where* $i \leftarrow 0$ **to** n **do**
    $timestamp_{curr} \leftarrow$received packet timestamp;
    $IPD_i \leftarrow timestamp_{curr}$ - $timestamp_{prev}$;
    if $IPD_i \leq 3\sigma + \overline{ipd}$ **then**
      if $packet_i$ *contains all NULs* **then**
        Discard $packet_i$ as dummy;
      else
        if $packet_i$ *is marked* **then** `/* Check for duplicates */`
          Discard $packet_i$ as duplicate;
        else
          Enqueue the $packet_i$ in buffer;
          Marked $packet_i$ as received;
        end
      end
    else
      Drop $packet_i$;
    end
    $timestamp_{prev} \leftarrow timestamp_{curr}$;
  end
end

```
/* Sender thread */
```

begin
  Start sending packets to next hop;
  Trigger send event after every $\overline{ipd}$ μs;
  foreach *send event triggered* **do**
    if *buffer* $\neq$ *empty* **then**
      Send current packet;
    else
      Send dummy packet;
    end
  end
end

Algorithm 3: Realtime watermark encoding algorithm

Input: Tuple < o, T, RNG, s, R, L>
Output: Watermarked packet stream
Generate the watermark 'W', $2n$ time intervals where $n = R \times L$;
Distribute them randomly with a probability 0.5 in two groups A and B ;
foreach $i \in L$ *where* $1 \leftarrow 1$ **to** L **do**
  Randomly assign $r = \frac{n}{l}$ group A and B intervals with probability $\frac{1}{l}$;
  if $i \equiv 0$ **then**
    Mark group B intervals for delaying;
  else
    Mark group A intervals for delaying;
  end
end
Use delay vector 'D' for encoding the watermark ;
$max \leftarrow 2 \times R \times L$;
foreach *packet* $\in$ *interval* I_i *where* $i \leftarrow 1$ **to** max **do**
  if *first packet overall* **then**
    Note time stamp of the first interval;
  end
  Get time stamp and calculate interval number of current packet;
  Calculate time stamp and interval number of expected packet;
  if *Expected interval* $\neq$ *Current interval* **then**
    Mark expected packet as first packet in interval;
  else
    Increase expected packet number by one;
  end
  if I_i *marked for delay in D* **then**
    if *first packet in interval* **then** `/* Use above calculated values */`
      Calculate $\Delta t_{i,j,k}$ for the expected packet;
      Calculate $\Delta\prime t_{i,j,k}$ for the expected packet;
      Effective delay$\leftarrow \Delta\prime t_{i,j,k} - \Delta t_{i,j,k}$ + packet rate;
    else `/* Use information from previous packet */`
      Calculate $\Delta t_{i,j,k}$ for the expected packet using $\Delta t_{i,j,k(prev)}$;
      Calculate $\Delta\prime t_{i,j,k}$ for the expected packet;
      Effective delay$\leftarrow \Delta\prime t_{i,j,k} - \Delta\prime t_{i,j,k(prev)}$;
    end
    $\Delta\prime t_{i,j,k(prev)} \leftarrow \Delta\prime t_{i,j,k}$;
    $\Delta t_{i,j,k(prev)} \leftarrow \Delta t_{i,j,k}$;
  else
    Effective delay$\leftarrow$packet rate;
  end
end

Algorithm 4: Realtime watermark decoding algorithm

Input: Tuple < o, T, RNG, s, R, L>,Watermarked packet stream
Output: Decoded watermark
Generate watermark 'W', $2n$ time intervals where $n = R \times L$;
Distribute them randomly with a probability 0.5 in two groups A and B ;
foreach $i \in L$ *where* $1 \leftarrow 1$ **to** L **do**
  Randomly assign $r = \frac{n}{l}$ group A and B intervals with a probability $\frac{1}{l}$;
  Store intervals assigned to i and in a bit group vector 'BG';
end
$max \leftarrow 2 \times R \times L$;
foreach *packet* $\in$ *interval* I_i *where* $i \leftarrow 1$ **to** max **do**
  if *first received packet overall* **then**
    Note time stamp of the first interval;
  end
  Get time stamp and calculate interval number of current packet;
  if *Current interval* $\equiv$ max **then**
    foreach *bit* $i \in L$ *where* $i \leftarrow 0$ **to** *L-1* **do**
      Using BG, find intervals for bit i in group A and B;
      Calculate total packet delay and packet number for each group;
      Calculate Group A and Group B centroids;
      if *Group A centroid > Group B centroid* **then**
        Received bit is 1;
      else
        Received bit is 0;
      end
    end
    Reverse the binary watermark string and convert to integer;
    Calculate the hamming distance between W and received integer;
  else
    if *Current interval* $\equiv$ *Previous interval* **then** `/* Received packet lies in the same interval */`
      Add $\Delta t_{i,j,k}$ to the total packet delay;
      Increase the total packet number in the interval by 1;
    else `/* Received packet lies in new interval */`
      Store the total packet delay and number of packets for previous interval;
      Start calculating total packet delay for the new interval;
      Start calculating total number of packets for the new interval;
    end
  end
end

REFERENCES

[1] Department of the Air Force and Air Force Materiel Command, "Network attack traceback," April 2005.

[2] M. Venkateshaiah, "Evading existing stepping stone detection methods using buffering," Master's thesis, The University of Texas at Arlington, 2006.

[3] X. Wang and D. Reeves, "Robust correlation of encrypted attack traffic through stepping stones by manipulation of interpacket delays," *Proceedings of the 10th ACM conference on Computer and communication security*, pp. 20–29, 2003.

[4] I. Ellacoya Networks, "Ellacoya data shows web traffic overtakes peer-to-peer (P2P) as largest percentage of bandwidth on the network," Press Release, June 2007. [Online]. Available: http://ellacoya.com/news

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

[6] Y. Zhang and V. Paxson, "Detecting stepping stones," *Proceedings of the 9th USENIX Security Symposium*, pp. 171–184, 2000. [Online]. Available: citeseer.ist.psu.edu/article/zhang00detecting.html

[7] Anonymizer Inc, "Anonymizer Privacy Policy," February 2008.

[8] S. W. Brenner, "U.s. cybercrime law: Defining offenses," *Information Systems Frontiers*, vol. 6, no. 2, pp. 115–132, 2004.

[9] H. Jung, H. Kim, Y. Seo, G. Choe, S. Min, C. Kim, and K. Koh, "Caller Identification System in the Internet Environment," *Proceedings of 4th USENIX Security Symposium*, vol. 246, 1993.

[10] S. Snapp, J. Brentano, G. Dias, T. Goan, L. Heberlein, C. Ho, K. Levitt, B. Mukherjee, S. Smaha, T. Grance, *et al.*, "DIDS (Distributed Intrusion Detection System)-Motivation, Architecture, and an Early Prototype," *Proceedings of the 14th National Computer Security Conference*, pp. 167–176, 1991.

[11] Yoda, K. and Etoh, H., "Finding a Connection Chain for Tracing Intruders," *F. Guppens, Y. Deswarte, D. Gollmann and M. Waidner, editors, 6 thEuropean Symposium on Research in Computer Security–ESORICS 2000 LNCS-1895*, 2000.

[12] X. Wang, D. Reeves, and S. Wu, "Inter-packet delay-based correlation for tracing encrypted connections through stepping stones," 2002. [Online]. Available: citeseer.ist.psu.edu/wang02interpacket.html

[13] D. Donoho, A. Flesia, U. Shankar, V. Paxson, J. Coit, and S. Staniford, "Multiscale stepping-stone detection: detecting pairs of jittered interactive streams by exploiting maximum tolerable delay," 2002. [Online]. Available: citeseer.ist.psu.edu/donoho02multiscale.html

[14] X. Wang, S. Chen, and S. Jajodia, "Tracking anonymous peer-to-peer voip calls on the internet," in *CCS '05: Proceedings of the 12th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2005, pp. 81–91.

[15] Xinyuan Wang and Shiping Chen and Sushil Jajodia, "Network flow watermarking attack on low-latency anonymous communication systems," in *SP '07: Proceedings of the 2007 IEEE Symposium on Security and Privacy*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 116–130.

[16] A.Corlett, D. Pullin, S.Sargood, "Statistics of one-way Internet packet delays," March 2002. [Online]. Available: http://ftp.ist.utl.pt/pub/drafts/draft-corlett-statistics-of-packet-delays-00.txt

[17] Jean-Chrysotome Bolot, "End-to-end packet delay and loss behavior in the Internet," *SIGCOMM Comput. Commun. Rev.*, vol. 23, no. 4, pp. 289–298, 1993.

[18] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "Request for comments: 3550 RTP: A transport protocol for real-time applications," July 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3550.txt

[19] Packetizer Inc, "H.323 versus SIP: A Comparison," Website Article.

[20] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman, "Request for comments: 3711 the secure real-time transport protocol (SRTP)," March 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3711.txt

[21] S. Gianvecchio and H. Wang, "Detecting covert timing channels: an entropy-based approach," in *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2007, pp. 307–316.

BIOGRAPHICAL STATEMENT

Jaideep Padhye was born in India in 1983. He received his Bachelor of Engineering degree in Computer Technology from Nagpur University, India in 2005, and his Masters of Science degree in Computer Science and Engineering with a thesis from The University of Texas at Arlington(UTA) in 2008. He has been part of the Information Security(iSEC) Lab at UTA. His research interests include Network Security, Anonymity and Privacy on the Internet.