Sleepy Watermark Tracing: An Active Network-Based Intrusion Response Framework*

Xinyuan Wang[†], Douglas S. Reeves [†‡], S. Felix Wu[††], Jim Yuill[†]
[†]*Department of Computer Science*
[‡]*Department of Electrical and Computer Engineering*
North Carolina State University
USA

[††]*Department of Computer Science*
University of California at Davis
USA

Key words: Intrusion Response, Intrusion Tracing, Active Security, and Network Security

Abstract: Network-based intrusion has become a serious threat to today's highly networked information systems, yet the overwhelming majority of current network security mechanisms are "passive" in response to network-based attacks. In particular, tracing and detection of the source of network-based intrusion has been left largely untouched in existing intrusion detection mechanisms. The fact that intruders can log in through a series of hosts before attacking the final target makes it extremely difficult to trace back the real source of network-based intrusions.

In this paper, we apply active networking principles to address the problem of tracing network-based intrusion with such chained connections, and propose a novel intrusion response framework: Sleepy Watermark Tracing (SWT). SWT is "sleepy" in that it does not introduce overhead when no intrusion is detected. Yet it is "active" in that when an intrusion is detected, the target will inject a watermark into the backward connection of the intrusion, and wake up and collaborate with intermediate routers along the intrusion path. By integrating a sleepy intrusion response scheme, a watermark correlation technique and an active tracing protocol, SWT provides a highly efficient and accurate source tracing on interactive intrusions through chained telnet or rlogin. Our

* This work has been supported by the Defense Advanced Projects Agency, administered by AFOSR under contract F30602-99-1-0540

prototype shows that SWT can trace back to the farthest trustworthy security gateway to the origin of intrusion, within one keystroke by the intruder. With its unique active tracing, SWT can even trace when intrusion connections are idle.

1. INTRODUCTION

Network-based attacks have become a major concern to today's highly networked mission critical information system. Existing network security mechanisms such as IDS, Firewall and IPSEC have not completely addressed the problem of network-based attacks. They are "passive" in front of network-based attacks and tend to be host-based. There is no automatic network-wide response even when attacks are detected.

One major problem in building an effective response to network-based attacks is the lack of source identification. Without effective source tracing, the attacked victim is blind at defending network-based attacks, and no effective intrusion countermeasures such as blocking and containing can be implemented. Network-based attacks can not be effectively repelled or eliminated until its source is known.

A complete solution to the problem of tracing network-based attacks is complicated by different anonymity gaining techniques used by different network-based attacks. For example, distributed denial-of-service (DDoS) attacks are usually generated from multiple previously-compromised slave machines, under control of a remote master machine. The unidirectional flooding traffic from slave machines usually comes with a "spoofed" source IP address, which makes it difficult to trace even the slave machines. For bi-directional, interactive intrusions, one of the most widely used techniques to conceal their true origin is to connect through "stepping stones" [11]: intruders connect through a series of intermediate hosts before attacking the final target. All these techniques are easy to implement and use, making source tracing of network-based attacks among the hardest network security problems

In this paper, we focus on the real-time tracing of interactive intrusions that utilizes connection chains to disguise their source. A real-time solution to this problem not only enables us to stop or deter network-based intrusion near its source, but also helps to deter DDoS by better protecting hosts from being compromised into slave machines. While there are several approaches have been proposed to address the tracing problem of intrusion connection chains, they are all passive and tend to be isolated, and their lack of real-time network-wide coordination severely limits their practical use in real-time tracing in the current Internet.

On the other hand, active network [2, 9] is an emerging framework that seeks to increase the programmability of computer networks and network components. It enables user and application to dynamically control how packets are handled. This customized packet processing opens new ways of securing networks that was not available in traditional passive networks.

In this paper, we apply active networks principle to address the problem of tracing network-based intrusion with chained connections, and present a novel intrusion response framework: *Sleepy Watermark Tracing (SWT)*. SWT is "sleepy" in that it does not introduce any overhead when there is no intrusion detected. Yet it is "active" in that when there is intrusion detected, it will trigger and coordinate network-wide tracing at real-time. SWT exploits the observations: 1) interactive intrusions with chained connections are bi-directional and symmetric at the granularity of connections; 2) application level contents are invariant across connection chains. By "injecting" carefully designed watermarks into the backwards-response traffic of the intrusion connection chain, SWT is able to trace through the intrusion connection chains at real-time - within a single keystroke by the intruder. Through its unique active tracing, SWT can trace through the connection chain even when the intruder is silent. All these represent substantial improvements over existing capabilities for tracing interactive intrusions with a chained connection.

In the next section, we discuss the general tracing problem and give a brief overview of existing tracing approaches. In section 3, we describe the general Sleepy Watermark Tracing method. In section 4, we present the SWT architecture. In section 5, we describe our prototype implementation of SWT and experimental results. In section 6, we conclude with possible future work.

2. TRACING PROBLEM AND APPROACHES

Given a series of computer hosts H_1, H_2, … H_n (n>2), when a person (or a program) sequentially connects from H_i into H_{i+1} (i=1,2,..n-1), we refer to the sequence of connections on <H_1, H_2, … H_n> as a *connection chain*, or *chained connection*. The *tracing problem* of a connection chain is, given H_n of a connection chain, to identify H_{n-1}, … H_1.

Tracing the source of intrusion through a connection chain over the Internet is a difficult problem. It requires network-wide collaboration among hosts in the network, and yet some of the hosts may be compromised and not trustworthy, As a network security mechanism, intrusion source tracing should be based on trust of appropriate network resources, and be robust

against compromised hosts in the network. To trace back the chained connections through multiple hosts, effective correlation is needed at intermediate nodes. Because network-based intrusion in today's high-speed network can be very short, correlation at intermediate nodes needs to be fast and accurate. Additionally, to scale the tracing system to the Internet, the tracing system should have minimum overhead while providing a fast response to detected network-based intrusion.

In general, tracing approaches for a connection chain can be divided into two categories: host-based and network-based, each of which can further be classified into either active or passive. Table 1 provides a classification of existing tracing approaches, as well as our proposed tracing mechanism.

Table 1. Classification of Existing Tracing Approaches and SWT

	Passive	**Active**
Host-based	DIDS	Caller ID
	CIS	
Network-based	Thumbprinting	IDIP
	Timing-based	SWT
	Deviation-based	

Distributed Intrusion Detection System (DIDS) [7] developed at UC Davis is a host-based tracing mechanism that attempts to keep track of all the users in the network and account for all activities to network-wide intrusion detection systems. Each monitored host in theDIDS domain collects audit trails and sends audit abstracts to a centralized DIDS director for analysis. While DIDS is capable of keeping track of all users moving around the network through normal login within the DIDS domain, it seems not feasible in large-scale network such as the Internet, because of its centralized monitoring of network activities.

The Caller Identification System (CIS) [5] is another host-based tracing mechanism. It eliminates centralized control by utilizing a truly distributed model. Each host along the login chain keeps a record about its view of the login chain so far. When the user from the *n-1*th host attempts to login into the *n*th host, the *n*th host asks the *n-1*th host about its view of the login chain of that user, which should be 1,2 ... *n*-1 ideally. The *n*th host then queries host *n*-1 to 1 about their views of the login chain and so on. Only when the login chain information from all queried hosts matches, will the login be granted at the *n*th host. While CIS attempts to maintain the integrity of login chain by reviewing information from hosts along the login chain, it introduces excessive overheard to the normal login process.

Caller ID, described by Stuart Staniford-Chen [3] , is yet another interesting host-based approach that is said to be used by the Air Force. Caller ID is controversial in that it actually utilizes the same break-in

technique used by intruders to break into the hosts along the connection chain reversibly. If the intruder from H_0 connects through H_1, $H2 \ldots H_{n-1}$ to the final target H_n, the network security personnel at H_n first breaks into $H_{n-1;}$ from there they can find out the intruder comes from H_{n-2}, then they break into H_{n-2} and so on. Eventually they can find the origin of the intruder. One compelling advantage of Caller ID is that it is scalable to the Internet. It is also efficient in the sense that it introduces less overhead compared to DIDS and CIS. But its manual approach makes it difficult to trace short intrusion in today's high-speed network. Besides it legal complications, Caller ID also has the drawback that one must perform manual tracing on the host where the intruder is active, which is easily-noticed by the intruder.

The fundamental problem with the host-based tracing approach is its trust model. Host-based tracing places its trust upon the monitored hosts themselves. In specific, it depends on the correlation of connections at every host in the connection chain. If one host is compromised and is providing misleading correlation information, the whole tracing system is fooled. Because host-based tracing requires participation and trust of every host involved in the network-based intrusion, it is very difficult to be applied in the context of the public Internet.

Network-based tracing is the other category of tracing approaches. Neither does it require the participation of monitored hosts, nor does it place its trust on the monitored hosts. It is based on the property of network connections: the application level content of chained connections is invariant across the connection chain. In particular, the thumbprint [3] is a pioneering correlation technique that utilizes a small quantity of information to summarize connections. Ideally it can uniquely distinguish a connection from unrelated connections and correlate those related connections in the same connection chain. While thumbprinting can be useful even when only part of the Internet implements it, it depends on clock synchronization to match thumbprints of corresponding intervals of connections. It also is vulnerable to retransmission variation. This severely limits its usefulness in real-time tracing.

The timing-based scheme [11] by Zhang and Paxson is a novel network-based correlation scheme for detecting stepping stones across the connection chain. The correlation is based on the distinctive timing characteristics of interactive traffic, rather than connection contents. It pioneered new ways of correlating encrypted connections. It requires no clock synchronization and it is robust against retransmission variation. However, because its timing characteristics are defined over the entire duration of each connection to be correlated, it is difficult to be used in real-time correlation.

The deviation-based approach [10] by Yoda and Etoh is another network-based correlation scheme. It defines the minimum average delay gap between the packet streams of two TCP connections as *deviation*. The deviation considers both timing characteristics and the TCP sequence number, and it does not depend on the TCP payload. Similar to the timing-based approach, the deviation-based approach does not require clock synchronization and is robust against retransmission variations. However it is difficult to be used in real-time correlation as the deviation is defined over all the packets of a connection. Another drawback of deviation-based approach is that it correlates only TCP connections.

One fundamental problem with passive network-based approaches is its computational complexity. Because it passively monitors and compares network traffic, it needs to record all the concurrent incoming and outgoing connections even when there is no intrusion to trace. To correlate at any host in the connection chain, it needs to match every concurrent incoming connection with every concurrent outgoing connection at that host. That is, for a host with m concurrent incoming connections and n concurrent outgoing connections, the passive network-based correlation approach would take $O(m \cdot n)$ comparisons, in addition to the $O(m+n)$ scanning and recording of concurrent connections.

On the other hand, the active network-based approach dynamically controls how connections are correlated through customized packet processing. It does not need to record all the concurrent incoming and outgoing connections at any host in the connection chain. It does not need to match each concurrent incoming connection with each concurrent outgoing connection. For a host with m concurrent incoming connections and n concurrent outgoing connections, the active network-based approach is able to correlate within *time dependent only on the number of connections being actively traced*, in addition to the $O(m+n)$ scanning of concurrent connections.

IDIP (Intrusion Identification and Isolation Protocol) [6] is a proposal by Boeing's Dynamic Cooperating Boundary Controllers Program that uses an active approach to trace the incoming path and source of intrusion. In the proposal, boundary controllers collaboratively locate and block the intruder by exchanging intrusion detection information, namely, attack descriptions. While it does not require any boundary controller to record any connections for correlation, its intrusion tracing is closely coupled with intrusion detection. The effectiveness of IDIP depends on the effectiveness of intrusion identification through the attack description at each boundary controller. Therefore IDIP requires each boundary controller to have the same intrusion detection capability as the IDS at the intrusion target host. It

is questionable whether the intermediate boundary controller is able to identify an intrusion based on a hard-coded attack description.

3. SLEEPY WATERMARK TRACING OVERVIEW

SWT is an active network-based tracing framework. It is "sleepy" in that it does not introduce overhead when no intrusion is detected. Yet it is "active" in that when an intrusion is detected, the target will inject a watermark into the backward connection of the intrusion and “wakes up” intermediate routers along the intrusion path.

By watermarking selected packets and processing them accordingly, SWT provides many potential advantages over existing intrusion tracing approaches. 1) SWT separate intrusion tracing from intrusion detection and it does not require any node other than the intrusion target to have the intrusion detection capability. 2) Unlike thumbprinting, timing-based and deviation-based approaches, SWT does not need to record all the concurrent incoming and outgoing connections at any node, and it does not require matching each of the incoming connections with each of the outgoing connections for correlation at any node. 3) SWT requires no clock synchronization and is robust against retransmission variation. 4) SWT traces only when needed. 5) So far the most compelling advantage of SWT is its correlation accuracy and efficiency. By using watermarks, SWT can trace the intrusion connection chain to its origin within a single keystroke of the intruder. With its unique active tracing, SWT can trace the intrusion connection chain back to its origin even when the intruder is is inactive. 6) We have found that SWT can be implemented efficiently. It does not introduce any noticeable overhead to routers, and it only requires a few network server applications at the intrusion target host to be modified to inject watermarks.

In the remainder of this section, we describe the SWT model concepts and assumptions.

3.1 Basic SWT Concepts

In order to keep track of network-based intrusions to hosts, it is desirable to monitor hosts through the nearest router or gateways. This is termed a *Guardian Gateway*. We define the *Incoming Guardian Gateway* of host *H* as the nearest router that forwards incoming traffic to H and the *Outgoing Guardian Gateway* of host *H* as the nearest router that forwards outgoing traffic from *H*. It is possible that one host has more than one incoming or

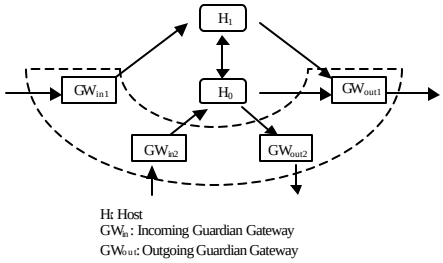

Figure 1: Guardian Gateway Set

outgoing guardian gateway. We define the union of incoming and outgoing guardian gateways of a host as its *Guardian Gateway Set* (e.g., {GW_{in1}, GW_{in2}, GW_{out1}, GW_{out2}} in Figure 1). For any guardian gateway set G, we define those hosts as *Guarded Host* of G whose guardian gateway set is a subset of G. For a host H, while the traffic between H and its directly-connected neighbour hosts does not pass through any gateways, the traffic between H and any non-directly-connected hosts must pass through its guardian gateway set.

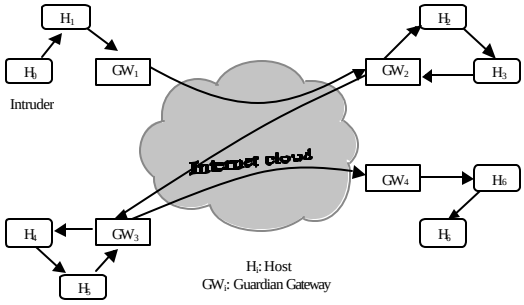

Figure 2: Tracing Model

We further define a *leap* as one connection step between hosts within a connection chain (e.g., $<H_i , H_{i+1}>$ in Figure 2). One leap may consist of multiple hops (or links in the physical network) and the two guardian gateways of the two end hosts. A leap can be specified by a 5-tuple consisting of

<protocol number, source ip address, source port number, destination ip address, destination port number>

Now the tracing problem of chained intrusion is defined as discovering and sequencing the guardian gateways of those hosts in the intrusion path, or (equivalently) as finding the leaps along the intrusion path.

3.2 Basic SWT Assumptions

We have identified the following assumptions that motivate and constrain our design:

Intrusions are interactive and bidirectional,

Routers are trust worthy and hosts are not trust worthy,

Each host has a single SWT guardian gateway and

There is no link-to-link encryption.

The first two assumptions represent our assessments of the nature of the intrusions. Here we refer to intrusions as those attacks aiming to gain unauthorized access, rather than denial of service attacks. A study of CERT security incidents [4] indicates that almost all security incidents, especially unauthorized access incidents, happened at computer hosts rather than routers or gateways. Therefore we believe our assumption to trust routers will cover most intrusion cases. In case there are indeed compromised routers involved in intrusion, the compromised router will be effectively indistinguishable from an attacker. The compromised router needs to be addressed first, before the tracing of the intrusion can go any further. In this case SWT can still trace to the farthest trustworthy guardian gateway.

The assumption of each host having a single SWT guardian gateway is only for simplifying the presentation of the SWT architecture. In case some host has multiple SWT guardian gateways, the guardian gateway set will be used in SWT tracing.

The final assumption represents the inherent limitation of any tracing based on network content. We believe that correlation of encrypted connections in real-time is still an open problem.

4. SLEEPY WATERMARK TRACING ARCHITECTURE

In general, the Sleepy Watermark Tracing Architecture consists of two complementing parties, namely, the *SWT guarded host* and the *SWT guardian gateway*. The SWT guarded host is the host that supports and thus is protected by SWT. The SWT guardian gateway supports SWT. In our trust model, each SWT guarded host has a unique SWT guardian gateway, and it maintains a pointer to its SWT guardian gateway. Each SWT guardian gateway may guard one or more SWT guarded hosts and it maintains the list of its SWT guarded hosts.

IDS and watermark-enabled applications at a SWT guarded host are SWT supporting components. In particular, IDS refers to an application level

interface to any Intrusion Detection System ; this is the ultimate initiator of SWT tracing. It interacts with SWT subsystem within SWT guarded host and triggers active watermark tracing once it detects an intrusion. Watermark enabled applications are those network service applications (such as telnetd, rlogind) that have been modified to inject arbitrary watermarks upon request.

The core of Sleepy Watermark Tracing consists of three interacting components: Sleepy Intrusion Response (SIR), Watermark Correlation (WMC) and Active Tracing (AT). In particular, Sleepy Intrusion Response accepts tracing requests from IDS, coordinates active tracing and keeps track of tracing information of intrusions. Watermark Correlation correlates incoming and outgoing connections through watermarks. Active Tracing coordinates different parties in the network to collaboratively trace the incoming path and source of intrusions.

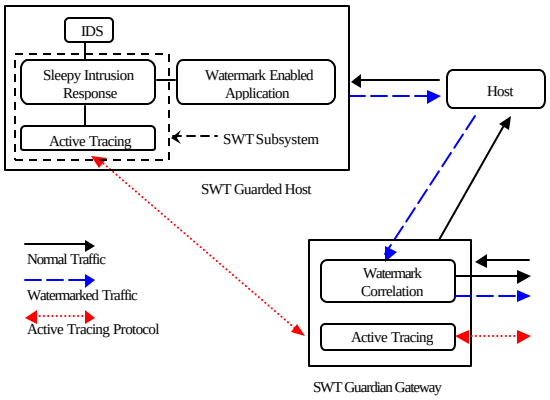

Figure 3: SWT Architecture

These three components work tightly together across SWT hosts and SWT guarded gateways. In specific, SIR and AT form the SWT subsystem within a SWT guarded host. Upon request from IDS, SIR coordinates a WM- enabled application and the AT module to initiate active tracing from the SWT guarded host to SWT guardian gateways. At the SWT gateway, the AT module receives tracing requests and provides watermarks to the WMC module. This module in turn provides AT module information about the next-leap SWT guardian gateway by correlating incoming and outgoing connections. Once the SWT guardian gateway finds next leap information about an intrusion connection chain, AT will send trace information to the original host that initiated the whole tracing and notify the next leap SWT guardian gateway to start watermark tracing.

4.1 Sleepy Intrusion Response

SIR controls and coordinates overall SWT intrusion tracing. It is in a SWT guarded host and it interacts with IDS and WM-enabled applications in the same host. To achieve high efficiency, SIR introduces "sleepiness" into SWT. By default, the SWT system is inactive and in sleep mode. When IDS detects an intrusion, it triggers SWT tracing by notifying SIR with appropriate connection information. Upon request from IDS, SIR first registers the intrusion connection as active for a configurable period of time, if it is not active already. Then SIR triggers active tracing on its guardian gateway by sending out trace notification; Finally SIR notifies the WM-enabled application that terminates the intrusion connection to start injecting the requested watermark. SIR also keeps track of tracing information of intrusions returned by the SWT guardian gateway, and upon request from IDS, SIR will provide tracing information on any specific active intrusion. If within a timeout period there is no trace information returned from the SWT guardian gateways, or further trace notification from IDS on an active intrusion connection, that intrusion response component will become inactive ("fall asleep").

4.2 Watermark and Watermark-Enabled Applications

Conceptually, a *watermark* is a small piece of information that can be used to uniquely identify a connection. Ideally, a watermark should be easy to embed and retrieve and yet be invisible to normal users of network applications. In order to be used for correlation, a watermark must be able to traverse multiple connections and remain invariant (we assume that there is no encryption involved in the connections). Therefore, watermark belongs to the application layer and is application-specific.

One challenge in generating watermark is how to make watermarks invisible to end-users. For text based network applications such as telnet and rlogin, this is in many ways similar to hiding data in text [1], which is much more difficult than hiding data in pictures or sounds. The open space method is one of the major methods of data hiding in text files through manipulating white space. In particular, inserting spaces at the end of each line of text file will not be noticed by readers. But for network applications such as telnet and rlogin, simply inserting spaces will change the cursor position, and it is likely to be noticed by end users. Fortunately, the text being transferred to network applications is not necessarily the same as that being displayed. For example, the string

"**See me***abc\b\b\b \b*"

transferred to telnet or rlogin will be displayed as the string

"**See me**"

We define a *virtual null string* of a network application as a string that appears null to end users of the network application. For instance, "*abc\b\b\b \b*" is a virtual null string of telnet and rlogin. Therefore by using virtual null strings, we can make watermarks invisible to such network applications.

In order to achieve high confidence of correlation, it is desirable to have the probability of collision of randomly generated watermarks as low as possible. For $n > 1$ sites, assume each site independently generates a equi-probable random integer number between 1 and m, where $m >> n$; let $P(m, n)$ be the probability such that those n random numbers are different from each other. Then we have:

$$P(m,n) = \prod_{i=1}^{n-1} \frac{m-i}{m} = \prod_{i=1}^{n-1} \left(1 - \frac{i}{m}\right)$$

When $m > n^2$, we have:

$$\prod_{i=1}^{n-1} \left(1 - \frac{i}{m}\right) > 1 - \frac{n^2}{2m}$$

Therefore, given $n = 2^{32}$, having $m \geq 2^{73}$ will make $P(m, n) > 0.999$. That means having 73 random bits in watermarks is sufficient to cover the whole IPv4 address space such that the probability of collision of generated watermarks is less than 0.1%.

Because the watermark is application specific, it needs to be injected into backward traffic through the application itself. *Watermark-enabled* applications are those network server applications (such as telnetd, rlogind) that have been modified to be able to "inject" requested watermark into their response traffic upon request. A watermark-enabled application processes two messages from SIR : WM-Start and WM-End, where WM-Start notifies watermark-enabled application to start injecting the enclosed watermark for specified times, and WM-End notifies the watermark-enabled application to stop injecting the watermark.

4.3 Watermark Correlation

In order to trace back along the intrusion connection chain, a mechanism is needed to find and match adjacent connections that belong to the same connection chain. We refer to this adjacent connection matching mechanism as *correlation*. According to the SWT tracing model, the hosts along the intrusion connection chain are not trustworthy, therefore, SWT is designed

to correlate at SWT guardian gateways. Because the forward and backward traffic of intrusion connection chain is symmetric at the granularity of leaps, watermarks along the backward traffic could be used for correlation at SWT guardian gateways.

By referencing its SWT guarded hosts, the through traffic of a SWT guardian gateway can be divided into two classes: *guarded* and *bypassing* (Figure 4). We define *guarded traffic* of a SWT guardian gateway as the traffic that either terminates at or originates from one of the SWT guardian gateway's guarded hosts, and *bypassing traffic* as all other traffic. It is obvious that the SWT guardian gateway needs to scan only the guarded traffic for possible correlation. We further define an *incoming leap* of a SWT guardian gateway as the connection that terminates at one of the gateway's guarded hosts, and an *outgoing leap* of a SWT guardian gateway as the connection that originates from one of the gateway's guarded hosts (Figure 4). Thus correlation at SWT guardian gateway can be modeled as matching an outgoing leap with an incoming leap.

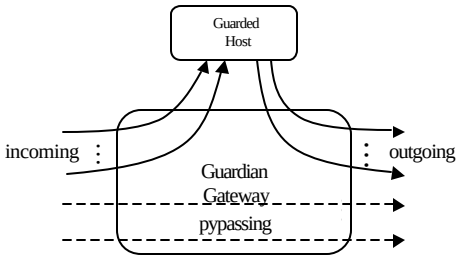

Figure 4: Guardian Gateway Correlation

One challenge of correlation at the SWT guardian gateway is that there may be multiple incoming and outgoing leaps through a single SWT guardian gateway. For a SWT guardian gateway with m incoming and n outgoing leaps, there are $m \cdot n$ combinations of possible matches after those $m+n$ leaps have been scanned. In specific, after m incoming leaps have been scanned, each of the n outgoing leaps scanned has m possible matches for correlation. Therefore exhaustive matching through multiple SWT guardian gateways would be complex and computationally expensive. To solve the connection matching combination explosion problem and achieve real-time response, SWT introduces the watermark as its basis for correlation.

With an identifying watermark injected to backward traffic of the intrusion connection chain, correlation at an intermediate SWT guardian gateways is simplified to scanning incoming and outgoing leaps and matching those with the same watermark. Specifically, when a SWT

guardian gateway scans incoming leaps, it registers any leap that has a registered watermark. When it scans outgoing leaps, it matches watermarked outgoing leaps with the incoming leap with same watermark.

The following observations can be made about watermark correlation:

- The accuracy of correlation is purely based on the uniqueness of the watermark, which is ultimately determined by SIR at the intrusion target host. This makes it possible to get very high confidence of correlation from tracing even a single watermarked packet.

- While the watermark is application specific, watermark correlation is generic. It has linear computation complexity across chained connections and requires no clock synchronization. Therefore it gives real-time response.

5. PROTOTYPE EXPERIMENTS

As a proof of concept, we have implemented a SWT prototype on FreeBSD 4.0. The prototype includes a SWT guarded host, SWT guardian gateways and a watermark-enabled application all running on the FreeBSD platforms.

We have performed two functional experiments on tracing a telnet connection chain: A $\Rightarrow$ B $\Rightarrow$ C $\Rightarrow$ D, where A is the source of intrusion and D is the final intrusion target. The first is to trace the intrusion source while the intruder is active. Our SWT prototype demonstrates the capability of real-time tracing of a single watermarked packet: SIR at host D gets all the trace information back to intrusion source A within one key stroke from intruder at A. The second experiment is to trace the intrusion source while the intruder is inactive or silent. By actively sending back a watermark from watermark-enabled telnetd, our SWT prototype also gets all the trace information lead to the intrusion source A. As we have expected, for each watermarked packet, SWT triggers one GWTraceOn message travel from D $\Rightarrow$ C $\Rightarrow$ B $\Rightarrow$ A, and two GWTraceInfo messages from C and B respectively.

To quantify the overheads incurred due to SWT itself, we have measured latency of SWT gateways with four different configurations:

- FreeBSD kernel IP forwarding without SWT;
- SWT configured to bypass traffic;
- divert socket IP forwarding without SWT;
- SWT configured to scan traffic.

The latency measurements were performed on a three node testbed configured in a straight line topology. The gateway at intermediate node was

a 233Hz Pentium PC with 32 MB RAM, 512KB cache, and two Netgear FA310TX 10/100 fast Ethernet adapters, running FreeBSD 4.0.

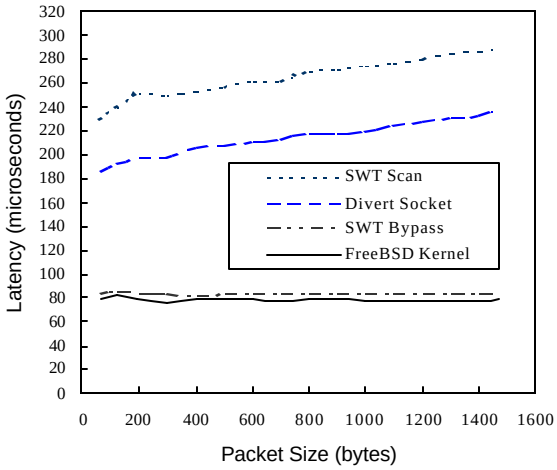

Figure 5: Latency of SWT Gateway

Figure 5 shows that the latency of FreeBSD kernel IP forwarding is about 78 µs, independent of packet sizes. It takes about 83 µs for the SWT gateway to bypass and forward IP packets of various sizes. The 5 µs latency difference comes from IPFW rule matching in the FreeBSD kernel. The latency of divert socket IP forwarding ranges from 186 µs to 239 µs depending on the size of IP packets. The 103 µs to 156 µs overhead for divert socket forwarding over kernel forwarding includes: (1) overhead for two context switches for data reading and writing; (2) overhead for data copy in and out of user space; (3) overhead for dispatching system calls. Compared with divert socket IP forwarding, SWT scanning takes about 50 µs more time to forward IP packets of various sizes. This indicates that the SWT gateway latency overhead due to SWT itself is about 50 µs.

6. CONCLUSIONS

In this paper, we have argued that network-wide, active intrusion response is needed in order to trace today's increasingly sophisticated network-based intrusions, which most likely utilize chained connections to hide their origin. We have presented SWT as an active network-based intrusion response framework and have shown that watermark can be used to construct highly accurate and efficient correlation for tracing chained

intrusion connections. Our prototype shows that SWT is able to trace back to the trustworthy SWT guardian gateway that is closest to the source of intrusion chain, within single keystroke of the intruder. By actively injecting watermark back to the intrusion connection, it is able to trace even when the intruder is silent.

By integration of Sleepy Intrusion Response, Watermark Correlation and Active Tracing, SWT provides highly effective, real-time and network-wide tracing of intrusions with chained connections. It is efficient, robust and scalable and it only requires some of the edge routers to participate tracing. Our experiment shows that SWT's own impact on a gateway's processing delay is only about 50 µs.

These results lead us to conclude that active network technology can indeed provide better and yet practical solutions to some of the most difficult network security problems. It is our hope that SWT could be a building block for more active network security mechanisms such as dynamic perimeter defense, and dynamic intrusion blocking and containment.

REFERENCES

1 W. Bender, D. Gruhl, N. Morimoto and A. Lu. Technique for Data Hiding. *IBM Systems Journal*, Vol. 35, Nos. 3&4, 1996.

2 K. L. Calvert, S. Bhattacharjee and E. Zegura. Directions in Active Networks. *IEEE Communication Magazine*, 1998

3 S. Staniford-Chen, L. T. Heberlein. Holding Intruders Accountable on the Internet. In Proceedings of *IEEE Symposium on Security and Privacy*, 1995.

4 J. D. Howard. An Analysis of Security Incidents on The Internet 1989 - 1995, PhD Thesis, http://www.cert.org/research/JHThesis/Start.html, April 1997.

5 H. Jung, et al. Caller Identification System in the Internet Environment. In *Proceedings of 4th USENIX Security Symposium*, 1993.

6 D. Schnackenberg. Dynamic Cooperating Boundary Controllers. http://www.darpa.mil/ito/Summaries97/_E295_0.html, Boeing Defense and Space Group, March 1998.

7 S. Snapp, et all. DIDS (Distributed Intrusion Detection System) – Motivation, Architecture and Early Prototype. In *Proceedings of 14th National Computer Security Conference*, 1991.

8 X. Y. Wang. Survivability through Active Intrusion Response. In *Proceedings of 3rd IEEE Information Survivability Workshop (ISW-2000)*, October 2000.

9 D. Wetherall, J. Guttag and D. Tennenhouse. ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. In *Proceedings of IEEE OPENARCH '1998*, April 1998.

10 K. Yoda and H. Etoh. Finding a Connection Chain for Tracing Intruders. In *F. Guppens, Y. Deswarte, D. Gollmann and M. Waidner, editors, 6th European Symposium on Research in Computer Security – ESORICS 2000 LNCS-1895,* Toulouse, France, October 2000.

11 Y. Zhang and V. Paxson. Detecting Stepping Stones. In Proceedings of 9th USENIX Security Symposium, 2000.