

BBN REPORT-8384

## **An Integrated Architecture for Attack Attribution**

Contract No. N66001-00-8038

December 31, 2003

Prepared for:

NSA R23, Network Traceback & Attack Attribution  
National Security Agency  
9800 Savage Road Suite 6534  
Fort Meade, MD 20755

Prepared by:

BBN Technologies  
10 Moulton St.  
Cambridge, MA 02138

# An Integrated Architecture for Attack Attribution

W. Timothy Strayer, Christine E. Jones, Isidro Castineyra,  
Joel B. Levin, and Regina Rosales Hain

BBN Technologies  
10 Moulton Street, Cambridge, MA 02138

{strayer, cej, isidro, rrhain}@bbn.com  
levin@cybertrails.com

December 31, 2003

## Abstract

Anonymity is important to perpetrators of network-based attacks. One of the simplest ways to remain anonymous is to hide the source of an attack by chaining together multiple connections into an *extended connection*. This is typically done by logging into a remote host, then from there logging into a third and fourth and so on until, at the final host, an attack is launched. These intermediate hosts are called *stepping stones*. Tracing such an attack back to the original source is difficult. Some techniques exist to trace individual connections, but tracing an extended connection requires identifying related connection pairs at each stepping stone.

This paper examines the problems and approaches to connection tracing, focusing on tracing extended connections across stepping stones. We survey the literature and discuss the several techniques that have been offered so far for discovering related connection pairs, and offer a taxonomy of these techniques. We then discuss a set of experiments performed on four selected algorithms to compare them and gain better understanding of their relative strengths and weaknesses. An architecture for an integrated attack attribution system, including both stepping stone detection and IP traceback, is offered, followed by concluding remarks and observations. Our future work will include constructing the master function and installing stepping stone detection extensions into SPIE to provide a more complete traceback solution.

## 1 Introduction

The Internet is and likely will for quite some time remain vulnerable to malicious attacks. In spite of the development of many sophisticated defense mechanisms such as intrusion detection systems and firewalls, attacks continue to increase [13]. This is due in part because of a lack of accountability: The anonymous nature of the Internet and its protocols makes it difficult to accurately identify the source of a network attack when the perpetrator wishes to conceal it. In fact, an attacker can generate attacks that appear to have originated from anywhere or nowhere.

There are several reasons why computer networks, especially the Internet, are particularly prone for attacks. First, since networks facilitate remote operations, an attacker may be physically separated from the target. This separation provides some degree of protection.

Second, the design of the Internet emphasizes fault tolerance, efficiency, and usefulness over accountability. The legitimacy of IP source addresses is not universally enforced. Routing algorithms are purposefully stateless to facilitate rapid recovery or rerouting of traffic after failure. Login identifiers also hide identity; rather than being the true name of the individual, it is a handle without a strong binding to any real identifying properties at all. In fact, the lack of a strong binding of user to individual is a universal problem, and techniques like PKI (public key infrastructure) seek to make identity and authentication based on identity an integral part of the network. So far such efforts have failed to achieve traction.

---

Finally, the sociological aspects of the Internet support the establishment and maintenance of loosely coordinated subcultures, complete with group dynamics and peer pressures, some of which reward daring feats. From chat rooms to hacker and cracker communities, individuals say and do things they may never attempt off-line because they have created an alter-ego for their presence on-line, where real names and identities are not being revealed.

Anonymity is a liberating differentiator; inhibitions are relaxed when the fear of being identified is reduced or removed. This is true for most social situations, but more nefariously, anonymity emboldens individuals with ill intentions to act in destructive ways. Often, network-based attacks are perpetrated by individuals seeking to hide their identities.

This paper surveys the research into source identification and attack attribution. In particular, it examines the problem of tracing attacks across so-called *stepping stones* [29, 21], which link interactive connections into a chain from the originating source to the ultimate victim. Section 2 describes the attack attribution problem and its two components: IP traceback and stepping stone detection. Section 3 presents a taxonomy of stepping stone detection techniques, and provides a description of previous work that espouse those techniques. Section 4 describes our test environment and the results of our own experiments on a selected set of promising stepping stone detection algorithms. In Section 5, we offer an integrated architecture for attack attribution, including IP packet tracing and stepping stone detection. We make concluding remarks and observations in Section 6, including an outline of how we intend to take the results of this paper—especially the proposed integrated architecture—and fully realize it by extending SPIE with stepping stone detection functionality. The system, when implemented, will be the most complete traceback system to date.

While we couch the attribution problem in terms of attacks, what constitutes an attack may be subjective. Further, there is nothing inherent about an attack that would limit the need to trace packets only to those packets related to an attack. Therefore, while we discuss “attacks” and “attack packets,” all of the detection and tracing techniques and algorithms discussed in this paper are applicable to any packet or group of packets, regardless of why someone is interested in tracing their origin.

## 2 The Attack Attribution Problem

Attributing an attack to a particular source or set of sources requires understanding what can happen to the packets used to perpetuate the attack as they traverse the network. The IP routing infrastructure is stateless and based largely on destination addresses; the source address plays virtually no role in the forwarding of a packet to its destination other than providing a return address in the case of bidirectional communication. In this respect, IP packets are essentially fire-and-forget types of delivery mechanisms; once a packet is introduced into the network, there is no need for the packet to maintain any relationship with its source. The source IP address carries no semantic of trust, but it is the only clue built into the network infrastructure as to the proper source. Attackers take advantage of this property of the IP protocol by manipulating—either directly or indirectly—the source address of attack packets to obscure their true origin.

In fact, an attacker can simply include an incorrect source address within a packet with impunity. This is referred to as *spoofing*. No entity in an IP network is officially responsible for ensuring the source address of a packet is correct. Many routers do employ a technique called *ingress filtering* [11] to limit source addresses of IP datagrams from a stub network to addresses belonging to that network. Not all routers, however, have the resources necessary to examine the source address of each incoming packet, and ingress filtering provides no protection on transit networks. Furthermore, spoofed source addresses are legitimately used by network address translators (NATs) [9], Mobile IP [23], and various unidirectional link technologies such as hybrid satellite architectures.

In addition to directly manipulating the source address of packets, an attacker can rely on applications and services within the network to modify source addresses for seemingly legitimate reasons. The manner in which these services operate vary. Some are basic request/response services, such as ICMP or DNS, that automatically reply to received packets. Attackers simply send request packets with incorrect source addresses that name the victim(s) to which the reply is sent. Such attacks are called *smurf* or *reflector* [22] attacks. Other applications allow users to relay

---

communications through intermediate hosts, such as remote logins. Intermediate traversal of a host has the effect of disconnecting the attack stream into separate flows, each with apparently different sources. By chaining a series of connections across a set of such intermediate hosts, henceforth referred to as *stepping stones*, the original source address can be obscured.

The attack attribution problem, therefore, becomes a matter of determining the source of the attack packets and, when those attack packets are generated as a direct result of some application level activity, tracing back across those applications. This effectively divides the attribution problem into two parts: (1) finding the source of a flow of attack packets, called the IP Traceback Problem, and (2) discovering which sources are acting to launder the attack, called the Stepping Stone Problem. Consequently, we identify three types of attack sources:

**Originating Source** The *originating source* of an attack, also referred to as the *attack source*, is the point of origin from which the attacker injects traffic into the network. In the presence of an extended connection, the originating source is the host that initiates the first connection in the connection chain.

**Laundering Source** Intermediate hosts (or routers acting as hosts) along the traversed path of an attack are considered to be *laundering sources* in that they are exploited to conceal the originating source. The laundering host is also referred to as a stepping stone.

**Immediate Source** The actual host to issue a packet is considered the *immediate source*, also referred to as the *packet source*. The host may be either the originating source or an intermediate laundering source.

## 2.1 Packet Source Identification: The IP Traceback Problem

The ability to identify the immediate source of packets is a necessary first step in identifying the originating source of an attack. Yet identifying a packet's source is complicated by both legitimate actions taken upon the packet by the routers as well as the always-present possibility of malicious actors along the packet's path.

In particular, a traceback system must not be confounded by an attacker that subverts a router with the intent to confuse the tracing system. If one assumes that any router along the path may be co-opted to assist in concealing a packet's source, it becomes obvious that one must attempt to discern not only the packet's source, but its entire path through the network. If a path can be traced through any number of non-subverted routers, then it must either terminate at the source of the packet, or pass through a subverted router. Subverted routers may be considered to be co-conspirators and treated appropriately.

The result of a packet traceback is an *attack path*. This attack path consists of each router traversed by the packet on its journey from source to the victim. Multiple paths may be identified. This can happen for two primary reasons. First, a compromised router can fabricate trace information such that multiple (but incorrect) sources are named. Second, multiple indistinguishable packets may be injected into the network from different, but legitimate, sources, possibly to confound the traceback system. Therefore, a traceback system should construct an *attack graph* composed of one or more attack paths, as shown in Figure 1.

It is important to note that IP packets may be modified during the forwarding process. In addition to the standard decrementing of the time to live (TTL) field and checksum recomputation, IP packets may be further changed by intermediate routers. Packet *transformation* [15] may be the result of valid processing, router error, or malicious intent. Packet transformations resulting from errors or malicious behavior need only be traced to the point of transformation, since the transforming router either needs to be fixed or can be considered a co-conspirator. However, traceback systems should trace packets through valid transformations back to the source of the original packet. This is especially important since attackers may further conceal their identities by engineering attack packets to be legitimately transformed by an established network protocol.

In the presence of subverted routers, an attack graph may incorrectly identify a host as being the source of malicious packets; that is, an attack graph may contain *false positives*. This is an unavoidable consequence of admitting the

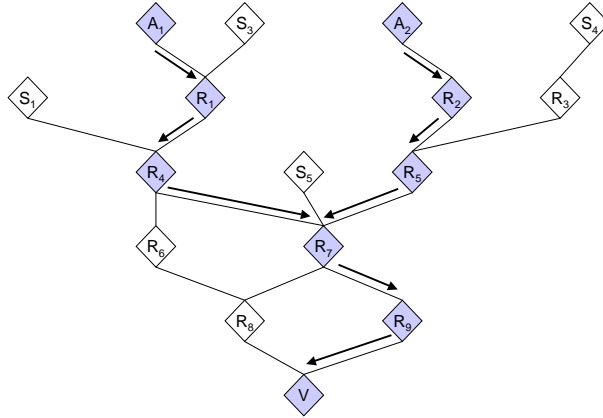


Figure 1: An attack graph containing attack paths for two identical packets injected by  $A_1$  and  $A_2$  and received by the victim,  $V$ . The arrows indicate links traversed by the packet; nodes on an attack path are shaded:  $\{A_1, R_1, R_4, R_7, R_9, V\}$  and  $\{A_2, R_2, R_5, R_7, R_9, V\}$ .

possibility of subverted routers and tradeoffs between accuracy and resources. While attempting to minimize false positives, an ideal traceback system produces no *false negatives*; that is, it must never exonerate an attacker by not including the attacker in the attack graph.

Further, when a traceback system is deployed, it must not reduce the privacy of IP communications. In particular, entities not involved in the generation, forwarding, or receipt of a packet should not be able to gain access to its contents through the IP traceback system. An ideal IP traceback system must not expand the eavesdropping capabilities of a malicious party.

### 2.1.1 Current Traceback Systems

Recent work in IP traceback has resulted in a number of proposed traceback techniques. Belenky [1] offers an in-depth survey of packet traceback systems, including a framework for evaluating and comparing their viability. The approaches taken by these schemes can be broadly categorized as follows:

**Route inference** Route inference was pioneered by Burch and Cheswick [5] who considered the restricted problem of large packet flows and proposed a novel technique that systematically floods candidate network links. By watching for variations in the received packet flow due to the restricted link bandwidth, they are able to infer the flow’s route. This requires considerable knowledge of network topology and the ability to generate large packet floods on arbitrary network links.

**Specialized routing** Some ISPs have developed their own ad hoc routing mechanisms for automatically conducting input debugging across their networks. Schnackenberg *et al.* [28] propose a special Intruder Detection and Isolation Protocol (IDIP) to facilitate interaction between routers involved in a traceback effort; however, IDIP does not specify how participating entities should track packet traffic. Even with automated tools, each router in the ISP must support input debugging or logging mechanisms that are not common in today’s high-speed routers.

To avoid this requirement, Stone [33] suggests constructing an overlay network connecting all the edge routers of an ISP. By using a deliberately simple topology of specialized routers, suspicious flows can be dynamically rerouted across the special tracking network for analysis. This approach, however, has two major shortcom-

---

ings: (1) The attack must be sufficiently long-lived to allow the ISP to effect the rerouting before the relevant flow terminates, and (2) the routing change is perceptible by the attacker.

**End-host packet auditing** End-host auditing schemes distribute trace state at the hosts rather than in the network. Routers notify the packet destination of their presence on the route. Proposed techniques include those by Savage *et al.* [27] and Bellovin [2], which explore in-band and out-of-band signaling, respectively. Savage *et al.* employ a packet marking scheme (enhanced by Song and Perrig [31]) that encodes the information in rarely-used fields within the IP header. Bellovin’s scheme (and later extensions by Wu *et al.* [19]) simply sends the audit information in an ICMP message.

Because of the high overhead involved, neither Savage nor Bellovin attempt to provide audit information for every packet. Instead, each employs probabilistic methods that allow sufficiently large packet flows to be traced. By providing partial information on a subset of packets in a flow, auditing routers enable an end host to reconstruct the entire path traversed by the packet flow after receiving a sufficient number of packets belonging to the flow.

**Network packet logging** An obvious (and naive) approach to traceback is simply to log packets at various points throughout the network and then use appropriate extraction techniques to discover a packet’s path through the network. Sager [25] suggests such a monitoring approach. Logging requires no computation on the router’s fast path; however, the effectiveness of the logs is limited by the amount of space available to store them. Given today’s link speeds, packet logs quickly grow to intractable sizes, even over relatively short time frames. Storage requirements can be reduced by sampling techniques similar to those of end-host schemes, but down-sampling reduces the probability of detecting small flows and does not alleviate the security issues raised by storing complete packets in the router.

Alternatively, routers can be tasked to perform more sophisticated auditing, extracting a smaller amount of information as packets are forwarded. The Source Path Isolation Engine (SPIE) [30] uses a hash-based technique to support the traceback of individual packets. In SPIE, traffic auditing is accomplished by computing and compactly storing packet digests rather than storing packets themselves. This both reduces the storage requirements by several orders of magnitude over current log-based techniques and preserves traffic confidentiality. Additionally, SPIE enables tracing packets through packet transformations, and is supported by an architectural framework for deploying SPIE within a network infrastructure.

### 2.1.2 SPIE: The Source Path Isolation Engine

SPIE is a traceback technique that provides traceback of *individual* packets, explicit traceback support of packet transforms, generation of attack graphs that identify all possible packet sources, no false negatives and minimal false positives, and preservation of current levels of network privacy. In addition, SPIE is efficient and scalable.

Despite its clear benefits, SPIE has two shortcomings. First, as initially proposed and currently implemented, the router through which a packet is first introduced into the network does not identify the specific host that generated the packet. Known and documented link-layer extensions to SPIE solve this problem [12, 26]. By recording and storing packet digests per router interface rather than for the router as an aggregate, and utilizing link-layer MAC addresses of hosts, it is possible to extend SPIE with the capability of specifically identifying originating hosts.

Second, SPIE must be fully deployed in a network to realize complete traceback functionality. SPIE performs what is essentially a reverse path walk through the network by visiting each router and asking that router if the given packet has been seen at that router. Consequently, each router in the network must be instrumented with a SPIE data collection agent. An alternate approach allows for SPIE to be implemented in passive tap boxes that reside next to each router. Reducing the coverage—that is, deploying SPIE more sparsely—may result in inaccurate traces, especially if packet transformations occur within routers that are not monitored by SPIE.

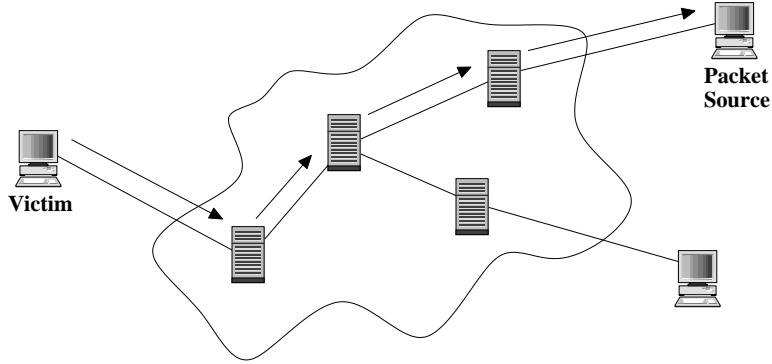


Figure 2: Traceback from the victim to the immediate packet source.

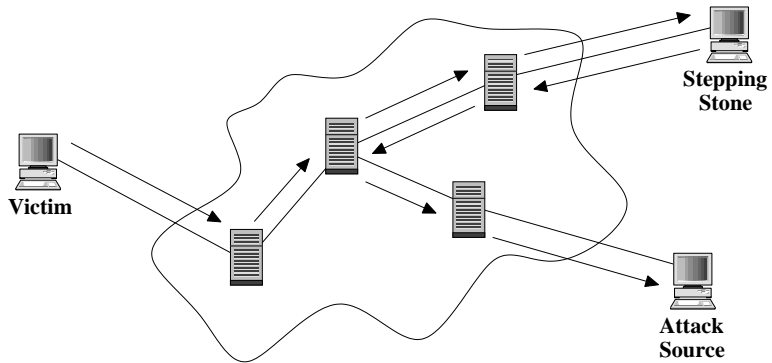


Figure 3: Traceback across an extended connection from the victim, through stepping stones (laundering hosts), to the source of the attack.

The details of SPIE and how it fits into an integrated attack attribution system are addressed in greater detail in Section 5.2.

## 2.2 Packet Causality Detection: The Stepping Stone Problem

The IP traceback systems surveyed above are oriented to packet tracing. They aim to produce an attack graph showing the path of one or more packets through the network, as shown in Figure 2. The resulting attack graph for a packet trace generally consists of routers that a packet traversed; the only hosts involved are the source of the packet and the victim. A traceback that reaches a host has identified a potential source of a packet. However, this does not mean that the source of the attack has necessarily been determined. The host may actually be a laundering host mid-stream of the actual attack path.

Tracing an attack path through laundering hosts requires the discovery of an association between two connections with endpoints at that host such that these connections act as consecutive links in a chain of connections. The chain of connections between an originating (or attack) source and the victim form what is called an *extended connection*, as shown in Figure 3.

There is a body of emerging work on detecting such laundering hosts, or stepping stones. The emphasis of such work has been on identifying two connections as related links in a larger chain of connections.

### 2.2.1 Classifications

We briefly define three classes of intermediate hosts used by an attacker to launder attack data and to obfuscate the attack path (a more expansive description is given by Lee and Shields [17]). An extended connection employed for

---

launching an attack may involve any type and any number of such intermediate hosts.

### **Stepping Stones**

The simplest type of intermediate host is a *stepping stone* host. Such a host is compromised to the extent that the attacker can log into the host and establish an outgoing interactive connection (such as *telnet*, *rlogin*, or *ssh*) to the next host in the attack path. The attacker's communications are not fundamentally altered, though enough may change to evade some methods of detection. For instance, if one or both connections are established using *ssh*, encryption will prevent use of any traceback method based on packet content. Processing delays in the host may superficially alter timing of packets. An attacker will usually chain multiple stepping stones together to further screen the attack's origin.

An attacker need not deeply compromise a host to create a stepping stone. All that is needed is access to an ordinary user's privileges to log in and to make outgoing connections. Such access can be gained, for instance, by dictionary attacks on password files or by snooping for passwords in Internet cafes or computer labs. Root access (or equivalent) is rarely required, though a more permanent back door could be more easily established with such privileges.

One difficulty with the detection of stepping stones is the large number of legitimate uses of extended connections. Commonly, access to hosts by legitimate users from outside a network is via a trusted gateway host through which the user connects to the target host. And frequently, by habit if for no other reason, a user will casually connect to some other host as needed by beginning a new session from his current target host [38] (at least one of the present authors regularly uses stepping stone connections of both these types).

### **Zombies**

Consider an intermediate host at which incoming communication is transformed such that the resulting outgoing communication appears to be wholly unconnected, and delays measured in hours or days are introduced prior to the establishment of the outgoing communication. Such behavior defines a *zombie* host. The attacker's incoming communication may be script and code to install a Trojan, and the attack output might come days or months later (say in response to a *cron* table entry). Another zombie may accept simple trigger commands from the attacker to execute previously planted code that issues entirely different output, for instance as part of a distributed denial-of-service (DDoS) attack.

The lack of apparent relationship between the incoming attack stream and the outgoing attack data, as well as the extremely long interval that may ensue between the end of one and the start of the other, makes the traceback problem in the network a very difficult one. Consider the example of a DDoS attack being triggered by a timer. Once the attack is under way, it is generally easy to determine the immediate source of the last link or two in the attack path. But once the zombie running the Trojan script is identified, it is difficult to associate the attack output with the incoming communication that generated the attack. Since the incoming connection (over which the Trojan was downloaded and installed) could have existed weeks or more in the past, the network retains no information about it. An investigator must examine host logs, if access can be obtained, and even then a cleverly written Trojan may have altered those logs when installed. Even if the DDoS attack is initiated by a packet containing a trigger command, there may very likely be insufficient data to associate that trigger packet with the outgoing attack connection.

### **Reflectors**

Finally, it is possible to launder an attack through an innocent, intact host operating normally, by using it as a *reflector*. For instance, an attacker logged into a host (possibly as part of an attack chain) may generate packets with the IP address of the ultimate attack victim forged as the source of the packets. The response packets are directed to the victim and constitute the actual attack. By bouncing such spoofed packets off a large number of normally operating reflectors, massive quantities of bogus responses are directed at the victim. This attack must be detected in the network, as there is no need for the attacker ever to have communicated with the reflector to set up the attack, hence no evidence of the attack or any tampering can be found on the host.

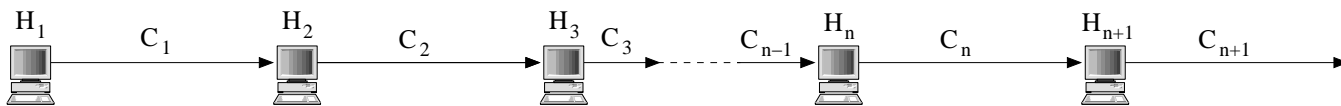


Figure 4: A chain of connections showing directionality. The hosts to the left are the initiators of the connections, so they are considered “upstream,” while the hosts to the right are the targets of the connections and therefore are considered “downstream.”

Paxson [22] considers a number of defenses against reflector attacks. By employing packet classification and filtering techniques to identify attack packets and rate limit their responses, these defenses mitigate the effects of the attack, and allow a significant portion of the victim host’s services to continue. Further, some IP traceback techniques such as SPIE provide for tracing across certain ICMP reflectors.

### 2.2.2 Methods of Detection

Without loss of generality, consider an attacker as being “upstream” and its victim as being “downstream,” and the direction of a connection as being toward the victim, i.e., “going downstream.” This is a convention without regard to whether a detection method examines unidirectional or bidirectional connections.

Assume that an attack path includes connections chained together between a number of hosts; that  $C_1$  represents the connection from host  $H_1$  to host  $H_2$ ,  $C_2$  the connection from  $H_2$  to  $H_3$ , and so on, as shown in Figure 4. IP traceback techniques provide a means of identifying the host  $H_n$  that is the immediate source of attack packets arriving at host  $H_{n+1}$  on connection  $C_n$ . We would like to extend this trace further by identifying upstream connection  $C_{n-1}$  on the attack path and continuing back up the chain of connections to discover the attacker.

In general, this is accomplished by examining all the incoming connections at  $H_n$  to associate one of them with  $C_n$ . This is done by examining some set of characteristics of the outgoing stream of packets comprising  $C_n$  and searching for an incoming stream of packets with matching characteristics. The correlation between each of these incoming streams and  $C_n$  can be computed in various ways, as shown in Section 3, and the best match that achieves more than some minimum correlation is the most likely candidate for connection  $C_{n-1}$ .

The above is an example of *direct correlation*, where both connections have an end-point in common. There is also the concept of *indirect correlation*. Assume that  $C_i$ ,  $C_{i+1}$ , and  $C_{i+2}$  are consecutive connections in the attack path. Further, suppose there is not sufficient access to the network(s) or routers traversed by  $C_{i+1}$  to install or access tools capable of reporting on connections within. It may still be possible to correlate the packet stream of  $C_i$  as it leaves host  $H_i$  for  $H_{i+1}$  with some other packet stream on a connection exiting host  $H_{i+2}$  for  $H_{i+3}$ ; if so, we can consider this second connection as a likely candidate for  $C_{i+2}$ . For such an indirect correlation, the criteria for deciding that the correlation is significant are likely to be different from those for a direct correlation.

There are a number of connection characteristics which may be compared in an attempt to determine whether two or more connections are correlated. If  $C_i$  and  $C_j$  are connections in an attack path, any characteristic which remains unchanged at the intermediate host or hosts between them—or, in other words, is invariant—may be used to determine if two such connections are associated. The only truly invariant relationship between correlated connections is causality. A packet or event on  $C_i$  *must* occur earlier in time than the corresponding packet or event on a downstream connection  $C_j$ , or no correlation can exist. However, accurately establishing causality depends on precision of clocks and relative timing error between networks.

Other characteristics of connections that may be analyzed for correlation are separated into two main categories: content and timing. Two connections in a chain may contain nearly identical sequences of data bytes, assuming there is no encryption. Even so, there may be some variations (consider that one connection may use *telnet* while another uses *rlogin*) due to differing sequences of protocol options, or bytes being repackaged differently into packets. Nevertheless the main part of the stream of data bytes will be nearly identical. Use of encryption in a connection

---

(using *ssh*, for instance) removes the possibility of correlation by content; however, depending on the type of cipher, packet size or byte count over time may still be a variable that can be correlated between connections.

Timing characteristics are not affected by encryption or any other transformation of the content. Variables available for analysis include the transmission times of packets or bytes, start time and lengths of packet bursts, and start and end times of idle periods. We will see below that the ending time of idle intervals is a particularly useful connection characteristic [38].

Yet, the content and timing characteristics are not completely invariant between connections in a chain; connection content and timing may be transformed at intermediate hosts, even with the simplest of stepping stones. Some transformations are “naturally occurring”—processing or propagation delay variations, for instance. However, application-level processes at intermediate hosts may be deliberately modified or created to transform connections with the goal of complicating the correlation process. Transformation of connections causes the algorithms used to detect correlated connections to produce results that are somewhat imprecise. The correlation between two connections is thus an indication of the *probability* that the connections are associated in an attack path. Consequently, reduction of false positives must be a consideration of any technique developed to detect correlated connections.

### 2.2.3 *Legitimate Sources of Imprecision*

Some causes of variation that effect the correlation between two associated connections are natural consequences of normal network operation. Legitimate causes of variation between correlated connections include:

**Propagation Delay** Propagation delay affects the various timing-based characteristics of connections. There are variations in packet transmission times whenever alternate or multiple routes are used within or between networks. Some jitter can be introduced by variations in local queuing at routers traversed by a connection. Also transmission times of packets of varying lengths are affected if the links operate at different data rates.

**Packetization Variation** Packetization variation can occur when, for instance, a stream of characters arriving on an incoming *telnet* connection is buffered on an intermediate host and then forwarded on the next connection in the chain. The basic unit of *telnet* data is the character or byte, so there is no reason for the packaging of the characters on the outgoing connection to be identical to that of the incoming characters. If the incoming and outgoing protocols are different, e.g., *telnet* vs. *rlogin*, the packetization of the outgoing character stream is more likely to be altered. Hence any analysis of the content must be able to examine the character stream without regard to packet boundaries.

**Error and Flow Control** Any error that invokes retransmission at any point in a connection chain will affect the timing characteristics of the connections downstream of the retransmission. (Effect of packet retransmission on content of packets may be detected by examination of the packet headers if a protocol such as TCP is used.) Also, at any point in the chain, a flow control mechanism may alter the timing of subsequent connections in the chain; this applies to hardware and link-layer flow control, as well as congestion mitigation techniques such as those used in network-layer and higher protocols (e.g., fair queuing or random early detection).

### 2.2.4 *Imprecision by Evasion Techniques*

Attackers may attempt to evade detection by actively modifying connections such that they appear unrelated. As pointed out by Donoho *et al.* [8], the information that the attacker wishes to transmit could be embedded steganographically, but this type of evasion would be hard to implement and execute. For the rest of this paper we concentrate on the following evasion techniques that can be applied on interactive sessions which use the *telnet/ssh* paradigm. These techniques can be used singly or in combination.

**Encryption** Encrypting as part of a stepping stone foils detection techniques based purely on content analysis. For

---

example, one could use an *ssh* session as the downstream leg of a stepping stone pair for which the first leg is a *telnet* session.

**Delay Manipulation** The attacker can also add delay in more or less random amounts to traffic crossing a stepping stone with the intention of thwarting detection techniques based on timing analysis. Note that for interactive sessions there typically exists a maximum delay that the attacker is willing to experience. (In their paper on the “multiscale” stepping stone detection technique [8], Donoho *et al.* present a simple transcoder (*inter-keystroke shuffling* (LIS)) that illustrates this idea.)

**Chaff Insertion** Attackers can add extra data into their connection in the hope of avoiding content-based and/or timing-based analysis techniques. As a simple example, an attacker could add a number of characters followed by the same number of DEL (delete) characters.

### 3 Taxonomy of Stepping Stone Detection and Traceback Techniques

This section reviews current stepping stone detection and traceback techniques proposed in the literature. The techniques can be organized along the following axis:

**Host-based vs. network-based** Some techniques require being present on the actual host that is the stepping stone. These techniques suffer from deployment, scalability, and trust problems, but are best positioned for detecting the chaining of connections. Network-based techniques examine traffic streams, attempting to pull from the flow of packets enough data to surmise the existence of a stepping stone host somewhere on the network.

**Proactive vs. reactive detection** Proactive techniques continuously and without being prompted attempt to identify the presence of stepping stones in the traffic they are monitoring. Reactive techniques are driven by the occurrence of certain specific events.

**Real-time vs. off-line analysis** It is possible for some techniques to issue detection results within a small amount of time of the existence of the stepping stone. These techniques calculate the probability of correlation faster than the occurrence of the events they are correlating. Other techniques require *post mortem* analysis either because they are computationally expensive or the data collection is too distributed.

**Passive vs. active monitoring** Passive monitoring techniques examine the events, traffic or otherwise, without causing additional events or manipulating those that naturally occur. Active monitoring allows the introduction of artifacts such as markers in the packets or in the data within the packets in order to detect the presence of stepping stones.

Section 3.1 examines host-based techniques; Section 3.2 techniques based on correlation of connection content, and Section 3.3 timing correlation techniques, including techniques adapted from message-flow detection schemes for wireless networks. We chose algorithms from four techniques as candidates for further study, one from content-based detection, and three from timing correlation, each exhibiting interesting properties.

#### 3.1 Host-Based Auditing

The techniques described in this section are classified as host-based because the laundering hosts participate in the detection and tracing of connection chains. Hosts are tasked with the auditing of incoming and outgoing packet flows to maintain correlation information. The type of data collected and the methods of collection and reporting vary with the auditing system.

Yet, all host-based approaches to stepping stone detection suffer from a fundamental flaw—trust is inappropriately extended to the hosts that form the connection chain. The correlation of flows within a connection chain relies on the information maintained and reported by *all* hosts within that chain. These hosts are quite probably compromised, so

---

no guarantee can be asserted with regard to the correctness of the correlation data, undermining the entire system. In addition, the required participation of all hosts significantly reduces system scalability: hosts need to be specially and individually configured. Auditing increases the load at hosts which often are resource poor. Often, an assumption is made that attackers are not spoofing source addresses which may not necessarily be the case.

The following discussion provides a brief description of the host-based auditing systems proposed to date.

### 3.1.1 Distributed Intrusion Detection System (DIDS)

The first documented work to address the stepping stone problem is regarded to be that of Snapp *et al.* [29]. They developed the Distributed Intrusion Detection System (DIDS) which monitors and analyzes events within a network. Both host and network monitors are distributed throughout a network to audit activity and report any events of interest to the centralized DIDS *director*. The data is then aggregated and analyzed for security purposes, much like any generic IDS, and appropriate action is taken if necessary.

When applicable, audited events are attributed to specific users. Users are generally distinguished by user identifiers (UIDs). However, there does not exist a one-to-one mapping between users and UIDs; a user often maintains access to multiple UIDs (legitimate or not) on various hosts throughout a network environment. DIDS addresses this by assigning a network-user identification (NID) to users when they first log into the monitored network. Any time that a user changes UIDs by logging into a different account (either on the same host or a remote host) the new UID is associated with the NID. This allows the DIDS to track user behavior as the user steps across various hosts in the network.

### 3.1.2 Caller Identification System (CIS)

Jung *et al.* [16] took the approach of verifying the origin and path of a remote connection prior to its establishment with the use of the Caller Identification System (CIS). All hosts within a CIS-enabled network filter login attempts through a server that performs “authentication” functions. A CIS server solicits information about the chaining history of a remote connection from previous hosts in the chain, verifies that information from all previous hosts coincide, and records the information for use by subsequent downstream hosts.

Essentially, the hosts  $\langle h_1, h_2, \dots, h_n \rangle$  forming a connection chain maintain an expanding view of the chain. When a remote connection is initiated at  $h_n$  from host  $h_{n-1}$ , the CIS server on  $h_n$  queries  $h_{n-1}$  about its view of the connection. Host  $h_{n-1}$ , assuming it knows about the connection, returns a list of previous hosts through which the connection traversed. Next,  $h_n$  verifies the information by asking each identified host about the connection. The integrity of the remote connection is established (and the connection granted) if the information from all hosts concur, and host  $h_n$  caches the current view of the connection chain. Therefore, the origin of all network connections is known and traceback is not required.

### 3.1.3 Session Token Protocol (STOP)

Similar to the CIS, the Session Token Protocol (STOP) developed by Carrier and Shields [7] recursively queries previous hosts in a connection chain, but only for forensic purposes, not for *a priori* verification of connection integrity. STOP is an extension of the Identification Protocol [14], a method of determining the client-side identity of a TCP connection. The protocol was extended to return secure tokens rather than actual user names, collect additional process data, and to further query, when requested, the previous host from which the identified user connected.

A STOP server listens on a known port of a host, and can be queried about connections initiated from that host. A token consisting of a hash of the user and connection information is returned; this prevents STOP from being used for invasive means. If an identified user is remote at a queried host as well, the STOP server can be directed to contact the STOP server of the host from which the user is remotely connected. Therefore, each host  $h_n$  of a connection chain records the STOP token of  $h_{n-1}$  within the chain. A token can be presented to system administrators to

---

request the actual connection information, which may include a token received from another host. Assuming that all involved hosts are running uncompromised STOP servers, this chain of tokens can be forensically traced off-line to the host of origin.

### 3.1.4 Process Origin Information

Buchholz and Shields [4] present another forensic tool that correlates incoming and outgoing network connections at hosts. Their method is to associate origin information with each process in the system table. The origin of a process is defined to be either *local* or *remote*. A local process is initiated by a user physically located at the host or by the system itself, whereas a remote process is initiated by users connected through a remote host. Origin information is only recorded in the case of remote processes.

To gain system access from a remote location, a user makes use of services offered by the remote host. Many systems offer well-known remote login services such as *telnet* or *ssh* that provide a user with an interactive session. After a successful remote login procedure, the origin information is recorded and associated with that process. All additional processes started during the remote session inherit the origin information. The origin information, at a minimum, consists of the IP five-tuple  $\langle \textit{source address}, \textit{destination address}, \textit{source port}, \textit{destination port}, \textit{protocol} \rangle$ . This information can be extracted from incoming packets and, therefore, no cooperation is required of other hosts. No specific mechanisms were identified to utilize the origin information for traceback.

## 3.2 Content Correlation

Content-based detection schemes assume that the contents of the connections are available—i.e., the connections are not encrypted—and are preserved across connections. Of the two techniques presented here, one—Thumbprints—is passive, the other—Watermarking—is active.

### 3.2.1 Thumbprints

Staniford-Chen and Heberlein [32] introduce the idea of *thumbprints* as short summaries of the contents of connections, and use these thumbprints as the basis for correlating two connections at a stepping stone, thereby tracing a connection chain across stepping stones back to its point of origin. A thumbprint is a very small piece of information calculated over a particular interval of a connection. In this way, the thumbprint is similar to a checksum. Thumbprints from several intervals of two connections are compared to determine if the connections have the same content and, consequently, belong to the same connection chain. All related connections are then pieced together to trace the extended connection across the stepping stones.

To allow the algorithm to compute similarities of content in the individual connections in the chain, this approach relies on the assumption that the content of the extended connection is invariant at all points in the chain. In fact, if the invariance assumption holds, it is not necessary to implement the thumbprinting system at every node in the network; rather, several well-placed nodes can collect and compare thumbprints to construct the trace without perfect information.

The thumbprinting algorithm must be able to accurately distinguish connection pairs from all other unrelated connections. A series of thumbprints are constructed over a connection by breaking the connection up into intervals and calculating the thumbprint over each interval. This implies that timing is an issue. Even if all hosts were clocking the intervals at exactly the same clock time, the clocks in each host may be skewed, shifting the input characters forward or backward in the interval. Similarly, propagation and queuing delays can shift the input characters so that intervals nearer the source will contain more newer characters and intervals nearer the ultimate destination will contain more older characters.

Since the extended connection is comprised of several independent connections, each with its own error recovery and packetization characteristics, the raw input character stream of one connection may look different from another because of packet reordering, lost data, retransmitted packets, and packet sizes. As a result, the approach described

---

operates after the packet-related variations in the connections have been removed—that is, at the transport layer—but the authors do suggest that perhaps the error tolerance of the thumbprinting algorithm would be sufficient to deal with these errors.

The thumbprint algorithm used by Staniford-Chen and Heberlein is referred to as a *local thumbprint* because it only depends locally on the character stream. The idea is to find a function that takes a sequence of characters and maps it onto a small vector of real numbers. To obtain this vector the algorithm first produces a histogram of the sequence. Each one of the real numbers is the result of the cross product between the histogram and a given “weighting” vector. This diminishes the impact of missed or out of order characters. In fact, this scheme does not take order into consideration at all, although the authors offer a couple that do, but their preliminary experiments suggest that order-conserving functions make little overall difference.

This approach is vulnerable to several countermeasures, most notably the ease of disguising the content of the extended connection, particularly by encryption, but also by simple techniques such as those described by Donoho *et al.* [8]. Nonetheless, we have chosen this algorithm as the most promising example of a content-based correlation method, and will present evaluation results on it and three other timing-based techniques in Section 4.

### 3.2.2 Watermarking

Wang *et al.* [36] suggest a network-based approach for tracing across stepping stones by injecting into the reverse stream a unique “watermark” that can be noticed by monitors in the network. Their technique, called *Sleepy Watermark Tracing* (SWT) does not introduce overhead into the system until the trace is started (this aspect they call “sleepy”) yet is “active” in the sense that the system actively traces the path rather than relying on passive data collection. The detection is separated from the trace, requiring no node other than the intrusion target to have intrusion detection capability.

SWT exploits the observation that interactive intrusions with chained connections are bidirectional and symmetric at the granularity of connections. This assumption stems from the authors’ assessment of the threat, that attacks aim to gain unauthorized access rather than commit a denial of service. Given this, the bidirectional nature of the connections is essentially what enables the insertion of watermarks into the response traffic stream. To work, however, SWT makes the assumption that content remains invariant across each connection in the connection chain (this implies that encryption is not being used).

Wang *et al.* also point out that passive network-based approaches suffer from the computational complexity of matching  $m$  concurrent incoming connections with  $n$  concurrent outgoing ones, yielding  $O(m \times n)$  comparisons. This active watermark approach does not require any matching, either prior to or after the detection of the intrusion; the watermark is injected into the return stream and followed as it crosses the various stepping stones. Thus, there is no need to record all incoming and outgoing connections, nor to correlate these connections *a priori*. Further, a trace is executed only when needed.

An intrusion detection system (IDS) is the initiator of the SWT tracing, triggering the trace once it detects an intrusion. The watermark is inserted into the stream by a watermark-enabled application—that is, a modified version of *telnetd* or *rlogind*. Incoming and outgoing connections are then correlated by noticing that the watermark from an incoming connection is being sent on an outgoing connection.

A watermark is a small piece of information—easily embedded, easily retrieved, but hard to notice—used to uniquely identify a connection. The watermark must be able to remain invariant as it traverses multiple connections (hence the assumption prohibiting encryption). Since each application has its own traffic characteristics, a watermark that meets these criteria for one application may be different than a watermark used in another application.

This observation may cause a problem similar to encryption. If two connections in the chain use different applications, say *telnet* on one link and *rlogin* on another, then the watermark cannot be assumed to be invariant across each of these applications. Although not addressed in the paper, perhaps the authors make an assumption that watermarks

---

for similar applications like those based on text use watermarks that are similar enough to be correlated. Text-based watermarks such as added extra spaces and backspaces in certain patterns would certainly survive across any application that preserved the text. Instead of stating that the watermarks are application specific, the authors may mean that the watermarks are specific to a class of similar applications.

Watermarks are correlated along the path to form the trace across the chain of connections. This correlation happens at so-called “guardian” gateways (that is, at routers near the hosts that are the actual stepping stones) because they are assumed trustworthy, where hosts are not. The accuracy of the correlation is based only on the uniqueness of the watermark.

Wang and Reeves [34] extend the Sleepy Watermark Tracing work by considering the inter-packet delays as the means of encoding the watermark signal into the packet stream. Manipulation of the inter-packet delays replaces the insertion of a content-based watermark as suggested above. This is essentially creating a covert channel for communication between the endpoints of the extended connection; the ultimate destination—the victim—manipulates the sending of packets on the return path so that the pattern of their inter-packet delays forms a message that can be detected at other parts of the network. As long as there are enough packets available in the return path, a message of arbitrary complexity (and resistance to perturbation) can be constructed.

Consider a packet stream. The inter-packet delay is a result of the packet generation at the originating application and each application at a stepping stone, the protocol processing at the stepping stone hosts, and the processing of the packet at each router along the path. The authors have run a set of experiments to show that the inter-packet delay does not suffer much variance even over extremely long extended connections with many stepping stones. This finding suggests that the manipulation of the inter-packet delay can be preserved across the extended connection as well. Mapping certain delay values to symbols encodes those symbols into the packet stream.

The larger the difference in delays used to encode the symbols, the more robust the encoding is against delay perturbation from the network. However, there is a limit to how large a delay value can be while maintaining the interactive nature of the packet stream. Furthermore, the authors seek to make this scheme robust against arbitrary induced perturbations. To address both of these issues, the authors suggest further encoding the symbols with redundancy codes for error correction, specifically, a Hamming Code. Consequently, given a maximum perturbation, one can calculate the Hamming distance required to tolerate that perturbation. The experiments given show that their method performs well in the presence of random packet perturbations.

Like the Sleepy Watermark Tracing scheme, this scheme requires that there are watermark detectors throughout the network, and that they are able to detect the start of a watermark. Unlike the SWT, the watermark scheme is dependent on a ready source of packets. Running out of packets, and hence not fully communicating the watermark, destroys the ability to correlate the connections in the connection chain. Further, the parameters of the watermarking scheme must be held in secret, for if they are known to an attacker, the attacker can exceed the tolerable perturbation of inter-packet delays and destroy the scheme.

### 3.3 Timing Correlation

Content-based detection techniques fail under several circumstances—for example when the content is encrypted. Timing analysis can be used in these situations. The approaches based on timing analysis are grounded on the ideas that inter-packet delay characteristics—in particular of interactive connections—are distinctive enough to identify connections, and that inter-packet delays of both encrypted and unencrypted connections are preserved across many router hops and stepping stones.

The schemes described in this section compare pairs of packet streams by looking at their timing characteristics. They differ on the specific feature extracted and how they are compared. These choices determine how robust they are to countermeasures and how well they scale.

---

### 3.3.1 ON/OFF Periods

Zhang and Paxon [38] describe a method to detect stepping stones based on representing candidate data streams as a series of ON/OFF periods. A monitor would typically be set up at the network access point of a site. This scheme can also be used to find correlations between connections at different points in the network, such as the access points of a multi-homed site. This requires that the monitors be time-synchronized.

Connections are represented as follows. For a given data stream, an interval is considered to be an OFF period if no data has flown for more than  $T_{\text{idle}}$  seconds, where  $T_{\text{idle}}$  is a parameter which the authors set to 0.5 seconds. A packet is said to have data only if it carries new (not retransmitted nor keepalive) data in its TCP payload. When a packet with non-empty payload appears, the flow ends its OFF period. Two OFF periods belonging to different streams are said to be correlated if their ending times differ by  $\leq \delta$ .

Let  $C_1$  and  $C_2$  be two connections,  $OFF_1$  and  $OFF_2$  be the number of OFF periods in each, and  $OFF_{1,2}$  be the number of these that are correlated.  $C_1$  and  $C_2$  are a stepping connection pair if

$$\frac{OFF_{1,2}}{\min(OFF_1, OFF_2)} \geq \gamma$$

where  $\gamma$  is a control parameter.

As a refinement, the authors use the number of *consecutive* coincidences as a factor in declaring two connections linked. Specifically, in addition to the test above they require:

$$OFF_{1,2}^* \geq \min_{\text{csc}} \text{ and } \frac{OFF_{1,2}^*}{\min(OFF_1, OFF_2)} \geq \gamma'$$

where  $\min_{\text{csc}}$  and  $\gamma'$  are new control parameters, and  $OFF_{1,2}^*$  is the number of consecutive correlated OFF periods. The results reported in the paper show this technique to be effective, even though it has not been tested in the presence of evasive measures.

This method has been implemented in Bro, a real-time intrusion detection system [21].<sup>1</sup> Deployed techniques are not very common, but it makes evaluating the technique easier, both within the Bro system and using the Bro system as ground truth. Nonetheless, there is no reported performance evaluation of this scheme when subjected to evasion attempts based on adding random delay. To that end, we have decided to construct our own implementation of this ON/OFF technique for further testing and possible inclusion in the integrated attack attribution system described in Section 5.

### 3.3.2 Deviation

Yoda and Etoh [37] describe a method to detect stepping stones based on the “deviation” between two connections. The deviation is defined as the difference between the average propagation delay and the minimum propagation delay between the two connections. According to this paper, “experiments show that the deviation for streams of packets in the same chain of connections is much smaller than that for a pair of unrelated streams.” The paper focuses on *telnet* and *rlogin* streams.

This scheme logs, for each packet of interest, a timestamp, the packet’s destination and source IP address, and its TCP header.

---

<sup>1</sup>Zhang and Paxon included an additional content-based technique for detecting stepping stone connection pairs, based on noticing certain well-known common text strings (e.g., the common “Last login:” comment from Unix-based operating systems) in packets. For the purposes of the current study we are only looking at the timing-based algorithm.

---

Assume that  $C_2$  is causally dependent from  $C_1$ —i.e.,  $C_1$  is upstream from  $C_2$ . The deviation between connections  $C_1$  and  $C_2$  is calculated as follows. For each connection, the algorithm constructs a graph with the timestamp value in the  $x$  axis and the TCP connection’s sequence number in the  $y$  axis, ignoring retransmitted packets. The graphs are conceptually superposed and the graph of  $C_2$  is repositioned along both  $x$  and  $y$  axis until the average horizontal distance between the two graphs is minimized. One must take care to preserve the causal relationship between the connections—i.e., the repositioned graph for  $C_2$  should always be below that of  $C_1$ . The paper then presents a method to calculate the deviation between two connections.

This paper does not, however, report on real-time implementations. Similarly, the scheme’s performance against evasion has not been evaluated.

### 3.3.3 Inter-Packet Delay (IPD)

In [35], Wang *et al.* present a two-phased stepping stone identification scheme. Their method detects correlations using inter-packet timing characteristics of both encrypted and unencrypted connections. The packet streams are filtered before processing by eliminating the following sources of error: duplicated packets, retransmitted packets, and ACK-only packets.

The first phase of the method finds “correlation points” between two packet streams. The second phase determines from the set of correlation points to what degree the streams are related.

Correlation points are found by the following algorithm. For a unidirectional flow of  $n > 1$  packets, let  $t_i$  be the timestamp of the  $i^{th}$  packet observed at some point in the network—timestamps do not need to be synchronized. The  $i^{th}$  inter-packet delay (IPD) is defined as

$$d_i = t_{i+1} - t_i$$

The inter-packet delay vector, then, is  $\langle d_1, \dots, d_n \rangle$ . A window of this vector is defined as

$$W_{j,s}(\langle d_1, \dots, d_n \rangle) = \langle d_j, \dots, d_{j+s-1} \rangle$$

where  $1 \leq j \leq n$  represents the starting point of the window, and  $1 \leq s \leq n - j + 1$  is the size of the window.

The authors consider several possible definitions of a function called  $\text{sim}()$ , the “similarity” between two vectors. These are functions of the two vectors with range  $[0, 1]$ . The more similar the vectors, the closer the similarity measure is to 1.

Given two packet flows  $X$  and  $Y$  whose IPD vectors are  $\langle x_1, \dots, x_m \rangle$  and  $\langle y_1, \dots, y_n \rangle$  respectively, the paper focuses on  $W_{j,s}(X)$  and  $W_{j+k,s}(Y)$ , where  $s$  is the size of both windows,  $j$  the start point for the window on connection  $X$ , and  $k$  the offset between the windows. For a given window size  $s$ , the tuple  $(j, j + k)$ —i.e., the values of the start of the windows—is defined as a correlation point if the maximum taken over the offset value  $k$  of the similarity measure of  $\text{sim}()$  is greater than a given threshold  $\delta$ . That is,

$$\max_k \text{sim}(W_{j,s}(X), W_{j+k,s}(Y)) \geq \delta$$

the maximum taken over appropriate values of  $k$ .

In the case when  $Y$  is a delayed version of  $X$ , all the correlation points would have the same correlation offset  $k$ . Therefore, all the correlation points  $(x, y)$  will be perfectly covered by a linear function  $x = y + k$ . If  $Y$  is a downstream version of  $X$  in a stepping stone chain, the set of correlation points would also be expected to be approximately covered by a linear function.

The paper presents several candidate similarity measures. A particularly successful one is the Min/Max Sum ratio. This is the ratio between the summation of the minimum elements and the summation of the maximum elements.

$$\text{sim}(X, Y, j, k, s) = \frac{\sum_{i=j}^{j+s+1} \min(x_i, y_{i+k})}{\sum_{i=j}^{j+s+1} \max(x_i, y_{i+k})}$$

The range of this function is  $[0, 1]$ . Only when  $x_i = y_{i+k}$  for  $i = j, \dots, j + k - 1$ , will this measure have the value 1.

The second phase of the process uses the Correlation Value Function (CVF) to decide if two streams are correlated. CVF is calculated as follows. After obtaining a set of correlation points—i.e.,  $(j_1, j_1 + k_1), (j_2, j_2 + k_2), \dots, (j_n, j_n + k_n)$ —they are represented as two  $n$ -dimensional vectors  $C_x = \langle j_1, \dots, j_n \rangle$  and  $C_y = \langle j_1 + k_1, \dots, j_n + k_n \rangle$ , then

$$CVF(C_x, C_y) = \begin{cases} 0 & n = 0 \\ \rho(C_x, C_y) & n > 1 \\ 1 & n = 1 \end{cases}$$

where  $\rho()$  is the standard correlation function:

$$\rho(C_x, C_y) = \frac{\sum_{i=1}^n (j_i - E(C_x)) \times (j_i + k_i - E(C_y))}{\sqrt{[\sum_{i=1}^n (j_i - E(C_x))^2] \times [\sum_{i=1}^n (j_i + k_i - E(C_y))^2]}}$$

$CVF()$  has a range of  $[-1, 1]$ . The closer to 1 the value of  $CVF()$ , the more related the connections are assumed to be, and the closer to -1, the more the connections are completely counter to each other.

The scheme, therefore, is defined by choosing the window width  $s$ , the similarity function  $\text{sim}()$ , and its related threshold  $\delta$ .

The paper does not report on real-time implementations, nor is the scheme evaluated against evasion.

### 3.3.4 Multiscale Detection

Donoho *et al.* [8] present an approach that addresses directly the issue of countermeasures. The paper consider countermeasures that consist of local jittering of packet arrival times and also the addition of superfluous packets. The authors assume that the intruder has a “maximum delay tolerance.” The paper concludes that there are theoretical limits on the ability of attackers to disguise their traffic for sufficiently long connections. The conclusions are asymptotic, and require further analysis to determine if intruders can avoid detection by keeping connections short. The paper considers proof-of-concept issues with no discussion of systems-level or scaling issues.

There are two restrictions on the connections. For connections  $C_1$  and  $C_2$ , let  $N_1(t)$  be the *cumulative character counting function* on the untransformed stream  $C_1$ :

$$N_1(t) = \# \text{ of symbols in } C_1 \text{ on } [0, t)$$

and similarly for  $N_2(t)$ . The restrictions are:

1. Causality:  $N_2(t) \leq N_1(t)$
2. Maximum tolerable delay:  $N_2(t) \geq N_1(t + \Delta)$  where  $\Delta$  is the maximum tolerable delay

Though the paper uses wavelet terms, its results can be expressed in terms of windowed averages. Define

$$\alpha_{j,k}^i = \frac{N_i(k \cdot 2^j + 2^j) - N_i(k \cdot 2^j)}{2^j}$$

That is,  $\alpha_{j,k}^i$  is the symbol arrival rate measured in the interval  $t \in [k \cdot 2^j, k \cdot 2^j + 2^j)$ . For higher values of  $j$  the averages are taken over larger and farther apart intervals. The value  $2^j$  is called the *scale* of the average.

Using the ideas of the paper and changing terminology we can define,

$$\delta_{j,k}^i = \frac{[N_i(k \cdot 2^j + 2^j) - N_i(k \cdot 2^j + 2^{j-1})] - [N_i(k \cdot 2^j + 2^{j-1}) - N_i(k \cdot 2^j)]}{\sqrt{2^j}}$$

That is,  $\delta_{j,k}^i$  takes the difference between the number of symbols arriving in the first half of the interval  $t \in [k \cdot 2^j, k \cdot 2^j + 2^j)$  and the number of packets in the second half of the interval, divided by the square root of the length of this interval.  $\delta_{j,k}^i$  gives an indication of how the symbol arrival rate changes inside the interval. For positive values, the rate is decreasing, and vice versa.

The paper contains the following results. If  $N_1(t)$  is a Poisson stream at rate  $\lambda$ , then

$$\begin{aligned} \alpha_{j,k}^1 &\approx \lambda \pm O_P(1/\sqrt{2^j}), \text{ where } O_P() \text{ denotes the asymptotic order in probability} \\ |\alpha_{j,k}^1 - \alpha_{j,k}^2| &\leq O_P(j/2^j) \\ |\alpha_{j,k}^1 - \alpha_{j,k}^2| &\ll |\alpha_{j,k}^1| \text{ at long time scales} \end{aligned}$$

and

$$\begin{aligned} \delta_{j,k}^1 &\approx O_P(1/\sqrt{2^j}), \quad |\delta_{j,k}^1 - \delta_{j,k}^2| \leq O_P(j/2^j) \\ |\delta_{j,k}^1 - \delta_{j,k}^2| &\ll |\delta_{j,k}^1| \text{ at long time scales} \end{aligned}$$

At a given scale  $2^j$ , the similarity between two symbol streams 1 and 2 can be looked at by correlating the  $\delta_{j,k}$  vectors of the two streams at that scale. That is,

$$Corr(j) = \frac{\sum_k \delta_{j,k}^1 \cdot \delta_{j,k}^2}{\sqrt{\sum_k (\delta_{j,k}^1)^2 \cdot \sum_k (\delta_{j,k}^2)^2}}$$

A different way to look at this is the following. Assume that symbol arrival data has been recorded in the interval  $[0, 2^{j_{max}})$ . After choosing a scale  $2^j$ , where  $j \leq j_{max}$ , we calculate for the two symbol streams that we are comparing the value  $\delta_{j,k}$  in the following way. For each subinterval of size  $2^j$  take the number of symbols arriving in the first half of the subinterval and subtract the number of symbols arriving in the second half; then divide this result by the square root of the subinterval length. The two vectors  $\langle \delta_{j,1}^1, \dots, \delta_{j,2^{j-1}}^1 \rangle$  and  $\langle \delta_{j,1}^2, \dots, \delta_{j,2^{j-1}}^2 \rangle$  are then compared by using the correlation function.

If the two symbols streams are independent, we would expect the correlation value to be low. On the other hand, for two related symbol streams—like those in a stepping stone—there should be a high correlation value as long as the chosen scale is sufficiently large. We should choose  $2^j \gg \Delta$ , where  $\Delta$  is the maximum delay introduced by the stepping stone—the length of the measuring interval will minimize the effect of the offset introduced by the added delay. The paper includes simulations backing this point.

The paper also suggests that chaff introduction will not hide the correlation between streams.

This scheme has yet to be implemented in a scalable system, and it requires relatively long connections, but it is the first method to address robustness to added delay jitter and introduction of chaff. Consequently, we have chosen this algorithm for further study.

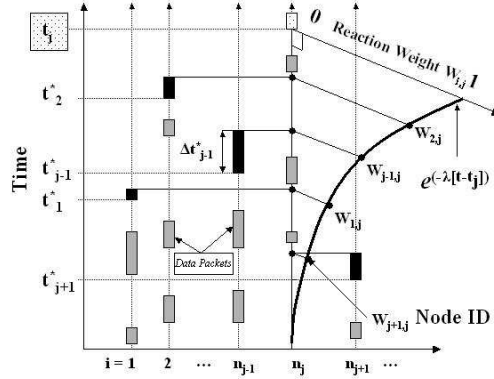


Figure 5: Notional representation of the state-space weight assignment process. Event  $t_j$  has just occurred at node  $n_j$  (uppermost box on time axis). Each vertical trace depicts packet transmissions from a node (gray with the most recent in black). Every other node is given weight  $W_{i,j}$  in vector  $W_j$  based upon its value on the exponential tail falling off from  $t_j$ .

### 3.3.5 State-Space Correlation from Wireless Topology Discovery

A problem similar to the discovering of stepping stones is discovering wireless transmitter/receiver pairs based only on the transmission events of nodes in a wireless network. In both cases there is a component of pairwise matching based on some evidence of relationship. In the wireless network, we consider a probe that can collect transmissions but cannot read the content of the transmissions. Several nodes are being monitored, and the probe can distinguish each node's transmissions. A node's transmissions may consist of several flows, each flow with an independent destination. Nodes in the network act as routers, receiving transmissions and forwarding them along toward the destination. The task is to match transmitters with receivers to form a path through the network.

The intuitive solution is to separate the transmissions by node, then attempt to match one node's transmissions against other nodes' transmissions to look for a reactive relationship—that is, we assume that a routing node will forward the packet as soon as it can, near in time after it received the packet. Therefore, a node that consistently seems to be transmitting just after another node has a high probability of communicating with the second node.

During work at BBN on the DARPA-funded Wolfpack program,<sup>2</sup> we developed several new techniques to match transmitting nodes with apparent receivers [20]. One uses a signal processing technique called *coherence* to perform cross spectral analysis of each node's transmissions. Another, described here in more detail, uses a Lagrangian state-space technique. While both methods are described in terms of discovering wireless network topology (without access to traffic contents), the techniques are applicable to discovering stepping stones.

The state-space technique can compute the degree at which two different traces are related using standard windowed time-frequency techniques such as cross spectral densities or its normalized counterpart, cross-coherence, to capture variations in the signal relationships as they evolve over time.

This technique examines the structure of all recorded traces from a purely causal perspective. The underlying approach is based on fundamental assumptions about the recorded traffic flows from the perspective of a given event (transmission), rather than from the perspective of the network node.

The state-space algorithm is based upon two assumptions. First, the likelihood of a transmission being a response to

<sup>2</sup>See <http://www.darpa.mil/ato/programs/wolfpack.htm>

a *prior* transmission generally decreases as the elapsed time between them increases. This is a direct consequence of causality with the added stipulation that a network tries to operate efficiently. Loosely speaking, we expect related packets to be located (temporally) closer than unrelated ones. Second, the inter-arrival times between a *fixed* event and any other event are approximately Poisson distributed.<sup>3</sup> This assumption may seem somewhat counter-intuitive in light of recent research in the long range dependency (related to self-similarity or fractal structure) of network traffic as observed in [6]. However, this reference provides a metric for this dependency  $H$ , the Hurst Parameter. We have empirically measured  $H$  for all our simulations, and have found that long range-dependency behavior can be neglected for time scales on the order of 4.5 ms or less. The average inter-event times for our networks occur on the order of 1 ms, which falls comfortably within the time scale where traces can be characterized by classical second-order statistics such as the Poisson distribution.

The algorithm uses a state-space representation of the nodes to generate a *conversation probability matrix* (CPM). The state vector of node  $j$  is updated at each detected transmission by node  $j$  by computing a weighted sum using the previous state vector and a new weight vector  $W_j$  consisting of weighted contributions from each node  $i$ . Each resulting  $i, j$  is an entry in the CPM and corresponds to the probability that the transmission generated by node  $j$  is due to one generated previously by node  $i$ . Figure 5 shows a notional representation of the weighting assignment. The constant  $\lambda$  is the Poisson parameter estimated from the trace statistics. One further aspect of the algorithm results from the causality constraint: the weight for node  $i$  is set to zero if an instigating transmission from node  $i$  ended after the start of the current transmission from node  $j$ .

The algorithm, as modified for stepping stone detection from the original wireless node detection, proceeds as follows. We assume the number of connections  $n$  is bounded, which we shall collectively refer to as a cluster. To represent the *packet centric* state of the cluster as a function of time, we define the mapping  $T : R \rightarrow R^{n \times 2}$ , where

$$T(t) = (t_i^*, \Delta t_i^*)_{i=1, \dots, n}, \quad (1)$$

and  $T(0) = \underline{0}$ , where  $t_i^*$  is the time of the most recent packet ( $\leq t$ ) for node  $i$  and  $\Delta t_i^*$  is the corresponding duration (packet length). From an implementation standpoint,  $T$  is easily updated on a per packet basis by simply replacing the appropriate row with the trace data.

We then apply the causality assertion in the following manner. Let us say that a packet  $p_k$  has just occurred at time  $t_k$  from node  $n_k$ . We then assign a weight,  $W_{p_k}^i \in [0, 1]$ , which quantifies the likelihood that this packet is a reaction to a prior packet from connection  $i$ , as follows. Let

$$\tau_k^i = t_k - [t_i^* + \Delta t_i^*] \equiv t_k - (T(t_k))_i \cdot [1, 1]^{transpose} \quad (2)$$

which is the time interval between the end of the transmission of the last packet from connection  $i$  until  $t_k$ . Then we determine a weight for each node from:

$$W_{p_k}^i \equiv W(\tau_k^i) = \begin{cases} e^{-\lambda \tau_k^i} & \text{if } \tau_k^i > 0 \\ 0 & \text{otherwise.} \end{cases} \quad (3)$$

The first case corresponds to assigning an exponentially decaying ( $\lambda > 0$ ) weight according to how much time has transpired since connection  $i$  last transmitted ( $\tau_k^i$ ). The second case ensures causality—that is, we are not allowing a packet to react to another before it arrives and thus a weight of zero is assigned. Figure 5 provides a notional representation for the Lagrangian based weight assignment process as given in Eq. 3. The vertical lines in the background depict sample time traces for a subset of packets (horizontal line) for a given trace collection point. Each of the shaded rectangles on the time traces represents a packet transmission on the connection labeling the

<sup>3</sup>We are currently working on extensions to this algorithm that use empirically derived statistical models.



Figure 6: Three hosts used to test stepping stone detection algorithms. The initiator establishes an interactive remote login connection to the stepping stone, and from the stepping stone another interactive remote connection is established to the target.

timeline. Black shaded rectangles represent the last packet from the corresponding connection, while grey rectangles are earlier packets. The rectangle beginning at time  $t_k$  is the *Lagrangian* packet  $p_k$ , which dictates the local selection of black rectangles that are quantitatively captured by the State Equation (Eq. 1). The next step is to calculate the associated weights (i.e., the likelihood weights as discussed above) by substituting each connection’s time lag,  $\tau_k^i$ , measured from  $t_k$  to the top of each black rectangle (Eq. 3). This stage is depicted as the darker horizontal lines intersecting connection  $n_k$ ’s timeline and ascent to the weight as governed by the exponential tail (which appears to rise out of the plane). We now have a very fast method for assigning weight to potential “instigating” connections measuring the likelihood of causality.

The only (and final) component of this technique that remains to be determined is the positive constant  $\lambda$  of Eq. 3. To address this, we apply the second assertion from above, that the probability that packet  $p_k$  is due to connection  $i$  assuming a Poisson random variable time delay  $\mathbf{t}$ , or, symbolically:

$$P(e_k \text{ due to node } i) = P(\mathbf{t} > \tau_k^i) = \mathbf{1} - \mathbf{P}(\mathbf{t} \leq \tau_k^i), \quad (4)$$

which is simply  $e^{-\lambda\tau_k^i}$ , where  $\lambda$  is the Poisson parameter. Thus, by estimating  $\lambda$ , we can assign probabilities concurrently with the calculations of Eq. 3 at no additional computational expense.

This technique, while promising, has yet to be applied to detecting stepping stones. Consequently, we have chosen this algorithm as well as the three others identified above for evaluation.

## 4 Experiments

Based on the preceding literature survey of stepping stone detection techniques, we chose four promising algorithms for a more in-depth study. Each of the algorithms were independently implemented to verify and understand their performance characteristics. The four algorithms are:

- Thumbprints (Section 3.2.1)
- ON/OFF Periods (Section 3.3.1)
- Multiscale Detection (Section 3.3.4)
- State-Space Correlation (Section 3.3.5)

### 4.1 Connection Traffic Data

As a first step, we chose a controlled environment in which to generate and collect traffic from known stepping stone pairs, as shown in Figure 6. This allowed us to focus on the effectiveness of each algorithm in detecting correlations. In future experimentation, we plan to subject the algorithms to traffic collected from a real-world environment.

Figure 6 shows three hosts used for our experiments, one of which was the connection originator, another the target, and between them the stepping stone. We initiated one or more connection sessions from the originator host to the

---

stepping stone, and from the stepping stone we further initiated connections to the target. The connection sessions were captured using the *autoexpect* tool from the *Expect* [18] program collection. This allowed us to replay the sessions as needed using the *Expect* program. The connection traffic was collected at the stepping stone using *tcpdump*.

#### 4.1.1 Traffic Content

We generated a set of connection sessions that consisted of permutations of several interactive remote login applications, transmission modes, and connection behaviors.

The three basic mixes of applications included *telnet-telnet*, *telnet-ssh*, and *ssh-ssh*. The reason for using both the *ssh* and *telnet* applications is that three of the evaluated algorithms are timing based and, in theory, not subject to confusion by encryption. The *ssh* application uses encryption whereas *telnet* does not. Additionally, the *ssh* and *telnet* applications may exhibit different timing characteristics resulting in dissimilar performance results for those timing-based algorithms.

For each of these application mixes, we constructed a trace that emphasized directory walks (as one may do when exploring a new system) and one that emphasized transcription (typing from a written page into a file). We also ran a set of traces in character mode (one packet per character typed) and line mode (a line is buffered and sent only when the carriage return is struck).

#### 4.1.2 Keystroke Distribution

The *Expect* program enables control over the inter-spacing of key strokes using a *Weibull* distribution. Therefore, output from a replayed *autoexpect* script is able to resemble a user typing at a keyboard. In addition, by adjusting parameters to the Weibull distribution function, the variability and speed of the output can be modified. *Expect*'s Weibull distribution is controlled by the following five parameters:

- Average interarrival time of characters
- Average interarrival time of words
- Measure of variability
- Minimum interarrival time
- Maximum interarrival time

The first parameter varies the interarrival time from one character to the next, and the second parameter enables simulation of the subtle pauses that occasionally occur at word transitions. The shape of the distribution is controlled by the third parameter, which essentially determines the variability of the interarrival times produced. The last two parameters bound the interarrival times within an interval.

As future work, the *Expect* program, and its built in support of the Weibull distribution, may be used to introduce timing variations at the stepping stone as a means of testing the detection algorithms' robustness against active evasion.

#### 4.1.3 Duration

The duration of the generated connections ranged from three to five minutes, sufficient time for the algorithms to detect stepping stones. However, the algorithms were implemented such that correlation results were able to be displayed at any time during a connection session.

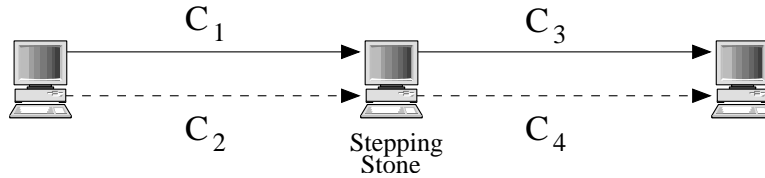


Figure 7: Connection samples used to illustrate the performance of the stepping stone detection algorithms.

#### 4.1.4 Filters

Using the *tcpdump* tool with which we originally collected the connection traffic, we were able to filter the traffic to include only the connections that we generated. This had the effect of removing all non-direct stepping stones from the sampled traffic, since studying direct stepping stones was our goal for this initial round of experimentation. Further work will consider the detection of indirect stepping stones.

We also filtered sampled traffic to consider the upstream and downstream portions of connection sessions both separately and together. We wanted to determine to what degree the upstream and downstream portions of correlated connections (which are obviously related as well) impacted the performance of an algorithm.

Additional filters were applied as needed to control the connection data processed by the algorithms. This enabled us to better understand certain aspects of the various algorithms. For example, we created a connection set that had four connections entering a stepping stone and four corresponding outgoing connections. We then created a second connection set by filtering out three of the outgoing connections. This allowed us to compare the detection results with that of all four outgoing connections present.

## 4.2 Performance Results

Three representative sets of traffic samples are used to illustrate the performance of the four evaluated stepping stone detection algorithms. Each sample contains the four connections  $C_1$ ,  $C_2$ ,  $C_3$ , and  $C_4$  as depicted in Figure 7. One set of traffic uses the *ssh* protocol, another the *telnet* protocol, and the third a mix of *ssh* and *telnet*. The connection samples contain only the upstream portion of the connections; the downstream acknowledgment channels were filtered. Therefore, there are four potential combinations of direct stepping stone connections— $C_1 \rightarrow C_3$ ,  $C_1 \rightarrow C_4$ ,  $C_2 \rightarrow C_3$ ,  $C_2 \rightarrow C_4$ —of which  $C_1 \rightarrow C_3$  and  $C_2 \rightarrow C_4$  are the actual stepping stone connection pairs.

We evaluated each of the four detection algorithms based on two criteria. First, and most important, the algorithms were evaluated according to how accurate and consistent they were in distinguishing stepping stone connection pairs from uncorrelated connection pairs. Then, assuming each algorithm was sufficiently able to distinguish stepping stone pairs, the amount of connection data required to correlate a connection pair, both in number of seconds and number of packets, was considered. (The number of seconds serves only to give an idea of how fast, under normal interactive conditions, the algorithms detect the stepping stone; it is the number of packets that actually is the more meaningful metric, since this tells us how much input is required for detection.)

Two points are important to note. First, the performance metrics graphed for each algorithm are specific to each individual algorithm and cannot (yet) be directly compared. This is because each algorithm produces its own measure of correlation, and has its own method for determining when a correlation can be declared. The second point is that, of the four algorithms tested, only the ON/OFF algorithm had an actual implementation constructed by the authors. All other algorithms were only described in their papers. As far as we know, this is the first time these algorithms have been implemented beyond simple proof-of-concept scripts. This also means that the papers gave little help with respect to properly choosing the various parameters for the algorithms.

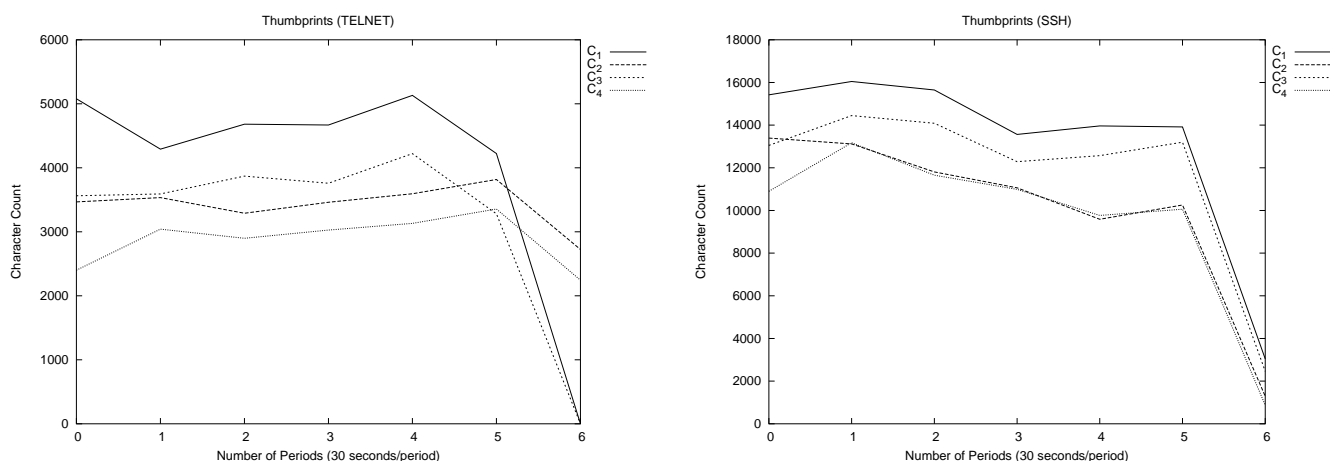


Figure 8: Performance results of the Thumbprints algorithm.

#### 4.2.1 Thumbprints

As a first step in evaluating the Thumbprint algorithm [32], we simply counted the total number of characters arriving on connections for specified time intervals. We did not consider the distribution of individual characters, and further work needs to expand the algorithm to include the weighting of individual characters (and sequences of characters) based on observed traffic patterns, as suggested by but not completely described in [32]. Note that because we are observing number of characters rather than actual character content, this algorithm can be applied to *ssh* connections as well as *telnet* connections. This is counter to the idea that content-based algorithms are not useful for encrypted connections.

Preliminary results indicate that this simplified version of the Thumbprints algorithm is capable of distinguishing stepping stone connections for both the *telnet* and *ssh* applications as shown in the performance graphs (left and right, respectively) of Figure 8. The total character counts of two correlated connections do not necessarily match during the lifetime of both connections; however, the change in characters trends in the same direction from time period to time period. For example, the character counts of  $C_1$  and  $C_3$  as graphed in Figure 8 maintain a consistent delta of characters across the time periods such that the graph of the character counts for the two connections maintain the same slope. Non-correlated connections may have similar character counts but have dissimilar changes in character counts between time periods.

Although very simple, this algorithm looks promising. From the graphical representation of our results we can distinguish stepping stone connections. A method of using the delta between connections' character counts to mathematically isolate correlated connections must be determined. Also, we need to extend the algorithm to include the notion of a "weighting" vector as described in [32]. However, by including the character content this algorithm will no longer work on *ssh* connections because the encryption used by the *ssh* application changes character content from one connection to another.

#### 4.2.2 ON/OFF Periods

The algorithm developed in [38], which is based on a connection's ON/OFF periods, was implemented by the co-author Vern Paxson within the Bro intrusion detection system [21]. We isolated the stepping stone detection algorithm from the publicly available Bro software package<sup>4</sup> and ported it to our test environment.

<sup>4</sup><http://www.icir.org/vern/bro-info.html>

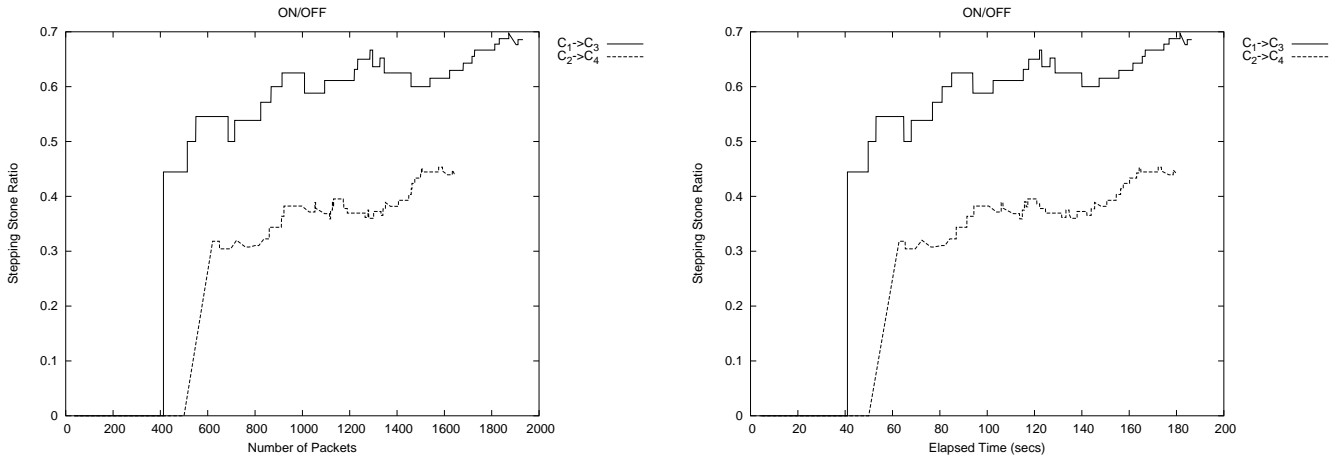


Figure 9: Performance results of the ON/OFF Period algorithm.

The detection algorithm requires that a pair of connections pass multiple “tests” before being declared a stepping stone pair. Two connections must maintain causality, meaning that OFF periods must occur in the same order. In other words,  $C_1$  of Figure 7 must always enter an idle period before  $C_3$ . Also, two connections must have a minimum number of consecutive coincidences (i.e., the OFF periods are within a specified delta). If these two criteria are met, then a ratio that indicates the likelihood that the two connections are a stepping stone pair is calculated. Various parameters govern the thresholds in determining stepping stone connections; we reused the parameter settings of the original implementation because the authors studied the parameters to determine the most appropriate values.

Figure 9 graphs the stepping stone ratio for connection pairs throughout the connection lifetimes, both in number of packets and elapsed time in seconds. This ratio is calculated only for connection pairs that meet the first two criteria—causality and consecutive coincidences. Therefore, although  $C_1 \rightarrow C_4$  (and  $C_2 \rightarrow C_3$ ) are potential stepping stone connection pairs, no ratio is calculated because they fail to meet the minimum criteria. The original authors’ determined that a ratio of 0.3 strongly indicates that two connections are a stepping stone pair. As can be seen by Figure 9, the connection pairs  $C_1 \rightarrow C_3$  and  $C_2 \rightarrow C_4$  strengthen this conclusion.

This algorithm is the most mature of the four algorithms we studied. However, additional study is required to fully understand the parameters to the algorithm, especially when evasion techniques that modify the timing of idle periods between connections are introduced.

#### 4.2.3 Multiscale Detection

We implemented the Multiscale algorithm based on descriptions provided in [8]. The algorithm compares connections based on the number of packets transmitted over each connection during configured time periods, specifically the difference of packets received. The implementation splits an interval, as defined in the algorithm description of Section 3.3.4, into two time periods corresponding to the first and second halves of the interval. The graphs displayed in Figure 10 refer to these periods.

This algorithm is sensitive to the length of the time periods. Results that clearly identify stepping stone connections may be obtained for a given time period. But, for another choice of time period length, correlation values for actual stepping stone pairs are below uncorrelated pairs. And for yet another choice of time period length, distinguishing stepping stone connections from other connections becomes impossible.

Figure 10 shows the results of the algorithm given two different choices of time period length. The graph on the

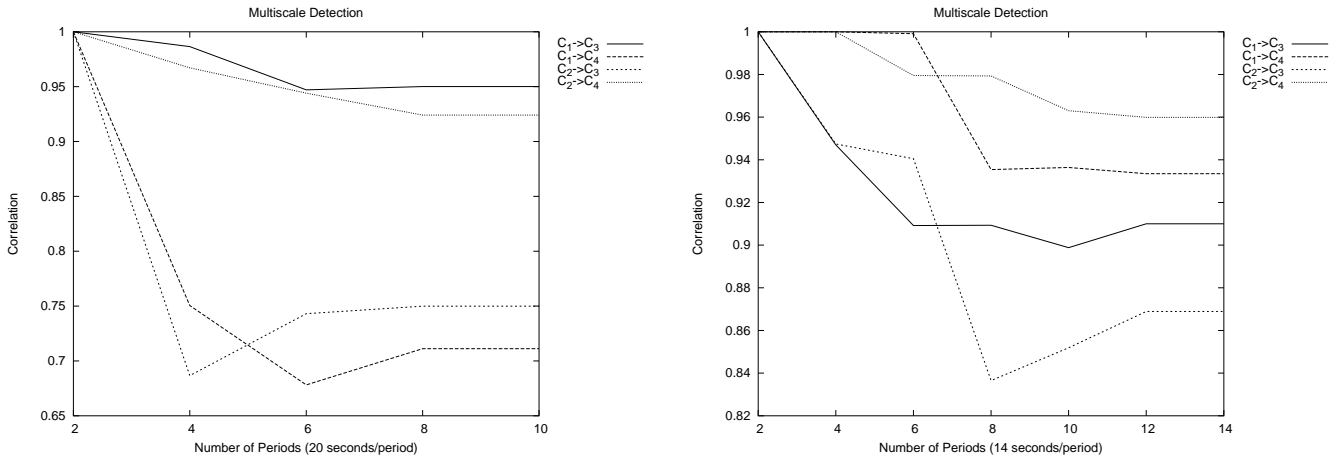


Figure 10: Performance results of the Multiscale Detection algorithm.

left shows the results of the algorithm given 20 second time periods; the choice of 20 seconds clearly shows that the connection pairs  $C_1 \rightarrow C_3$  and  $C_2 \rightarrow C_4$ , actual stepping stone pairs, to be more highly correlated. The graph on the right shows the results of the algorithm given 14 second time periods; the choice of 14 seconds, on the other hand, gives results that do not show stepping stone connections to be more highly correlated.

Clearly, more work needs to be done to determine this algorithm’s utility. The algorithm’s sensitivity to the choice of time period lengths needs to be understood and methods of alleviating the effect of the time period choice may need to be devised. Further, the appropriate correlation threshold to identify stepping stone connection pairs needs to be established.

#### 4.2.4 State-Space Correlation

An implementation of the State-Space Correlation algorithm was originally developed at BBN on the DARPA-funded Wolfpack program. This implementation was ported to work within the stepping stone test environment. During the experimentation process, a couple minor adjustments were then made to the algorithm itself such that the algorithm better fit the new problem constraints. Some of the assumptions made within the wireless environment, such as inability to read transmission content, are not applicable within the context of the stepping stone problem.

Figure 11 provides performance results of the State-Space Correlation algorithm with respect to a sample of *ssh* connection traffic as shown in Figure 7. The performance results are represented in terms of a correlation metric between 0 and 1, with values closer to 1 indicating a greater likelihood that two connections are a stepping stone pair. The correlation metric is graphed against both number of packets and elapsed time in seconds. The algorithm correctly distinguishes both stepping stone pairs— $C_1 \rightarrow C_3$  and  $C_2 \rightarrow C_4$ —within approximately 100 packets or 20 seconds of interactive traffic.

Similar results were collected for all the connection samples to which the algorithm was applied. For the limited experimentation conducted to date, the State-Space Correlation algorithm accurately and consistently distinguishes stepping stone pairs from non-correlated connection pairs. However, more extensive study must be conducted, especially with the goal of better understanding the parameters governing the algorithm and the correlation metric output.

The State-Space Correlation algorithm assumes that the interarrival times between a *fixed* event (i.e., a transmitted packet) and any other event (i.e., a subsequently transmitted packet) are approximately Poisson distributed, where  $\lambda$  is the Poisson parameter. The current algorithm calculates the  $\lambda$  constant *a priori* based on the pre-sampled traffic.

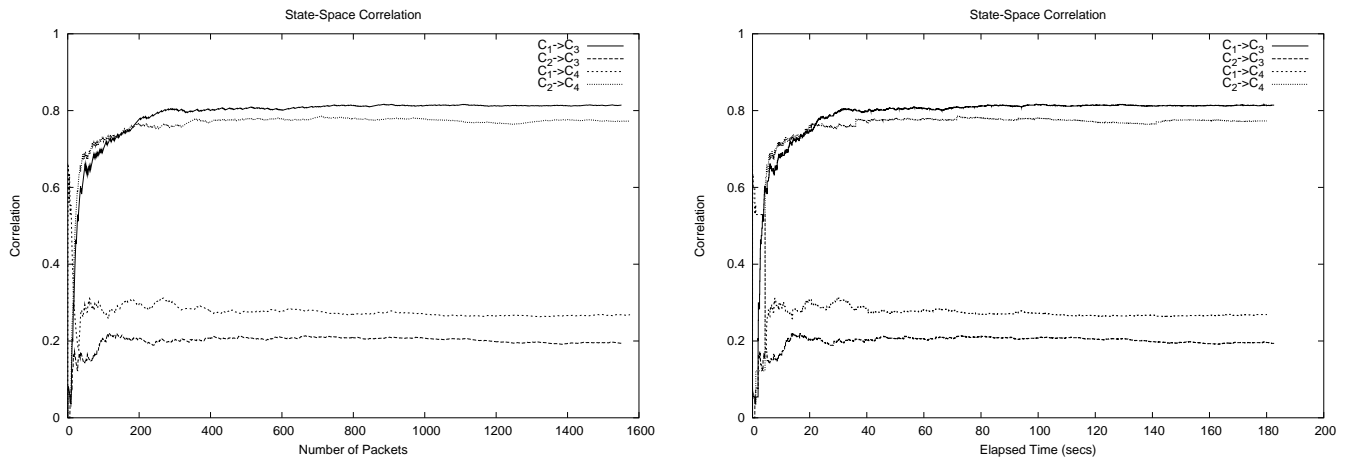


Figure 11: Performance results of the State-Space Correlation algorithm.

Future work needs to determine the method of setting this parameter, whether it is a constant value determined from a large sample of traffic from an appropriate network environment, calculated dynamically as the algorithm processes packet events, or possibly an entirely different statistical model that is empirically derived.

Other parameters that need to be more closely examined are the event duration and minimum weight threshold. Currently, the implementation sets an event’s duration to a constant value, although each packet event differs in duration. The algorithm must be modified to more accurately account for a packet event’s duration based on the packet size and network capacity. A minimum weight threshold is applied within the algorithm and should be set based on a stepping stone’s queuing delay.

Finally the output of the State-Space algorithm must be better understood so that an appropriate correlation threshold may be applied that accurately detects stepping stone pairs. The correlation metric may be effected as the number of sampled connections increases. Our experiments used sample traces with a relatively small number of connections. We need to expand our experiments to include large connection samples. The result may be that we need to adjust our correlation threshold based on the number of active connections being monitored.

### 4.3 Discussion

The performance results presented are preliminary in several dimensions. Each of the algorithms require more rigorous experimentation to determine appropriate parameters and thresholds for detecting stepping stones. At least one of the algorithms—the Thumbprints—needs to be extended to include other correlation considerations described by the authors. The algorithms also need to be tested with an expanded set of connection traffic, including traces from actual routers.

We also need to consider active evasion on the part of the attacker and legitimate sources of timing variation introduced in the network. Attackers can attempt to evade detection by modifying the connection content or timing at various points in the network, as described in Section 2.2.4. Connection variations are also inherent in a network due to such things as bottleneck routers, satellite links, quality of service (QoS), etc.

After more in-depth study and broader understanding of each algorithms strengths and weaknesses, we need to form a master function that combines the individual results of the algorithms to produce an overall stepping stone profile for each connection.

---

## 5 Integrated Source Attribution

The identification of the originating host of an attack requires being able to trace the attack back across not only the path of the attack packet or packets—giving the immediate source—but also back across any stepping stones that link several connections into the extended connection used to effect the attack. There are several approaches to packet tracing, as discussed in Section 2.1, and several approaches to detecting stepping stones, as discussed in Section 3, but to date no approach has been suggested and analyzed that combines the two approaches into an integrated attack attribution system. This section outlines such a system, using SPIE as the packet tracing infrastructure and extending it to include traces across stepping stones.

Recall from Section 2.2 that an *extended connection* is constructed by a series of individual connections linked in a pairwise fashion and related by carrying the same content (plaintext or encrypted) serially across each connection from the originating source to the ultimate destination. A *connection pair* is any two adjacent connections in the extended connection—that is, of the four endpoints of the two connections, two endpoints are the same host. The stepping stone detection techniques surveyed in Section 3 attempt to determine the correlation between all pairs of connections such that the pairs related by being part of the same extended connection will have correlation scores that are higher than those that are not related.

These techniques make several simplifying assumptions about the network that may not be valid in an operational environment. The first assumption is that the source address is valid because TCP connections require bidirectional communication. It is true that an attacker cannot effectively use a connection, especially an interactive one, with a spoofed source address, but this does not mean that the network is preserving the source address. The source address is actively modified by the network when the source is part of a privately addressed network, as one connected to the Internet via a network address translator (NAT) [9]. In this case, the source address is an address from the non-routable private address space as defined by RFC 1918 [24]. The attacker establishes a connection from his private network through a NAT to a compromised system in the Internet, possibly also within a privately addressed or corporate network where the IP addresses are not routable or not known.

The second assumption is that connection pairs are easily stitched together to form the fully reconstructed extended connection. Consider three connections between four hosts,  $H_1 \leftrightarrow H_2$ ,  $H_2 \leftrightarrow H_3$ , and  $H_3 \leftrightarrow H_4$ . The pair  $(H_1 \leftrightarrow H_2, H_2 \leftrightarrow H_3)$  can be detected at some point where all traffic into and out of  $H_2$  is seen. Likewise with the pair  $(H_2 \leftrightarrow H_3, H_3 \leftrightarrow H_4)$ . However, matching the two pairs into an extended connection assumes that, given one pair, the stitching algorithm knows where to find the other. The obvious clue is  $H_3$ 's address, but this suggests that there must be some global mapping of each host's address onto the data collection point handling the detection of stepping stones for that host. Outside of maintaining such a mapping, the easiest and most effective way to find the data collection point is if those points are closely tied to the routing infrastructure since the mapping of hosts onto routers is fairly well understood.

Integrating stepping stone detection with IP traceback addresses these two problems. A good IP traceback system does not rely on the source address in the packet during traceback. Since a packet can be easily associated with a connection, tracing any one of the packets in a connection will find the origin of the connection, even if that origin is obscured in some way. Further, a good IP traceback system will assemble the entire path of a packet, specifically each router that a packet passed through. This implies that the traceback system is closely integrated with or somehow relies upon the routing infrastructure of the network.

The remainder of this section provides a general stepping stone detection architecture suitable for housing any one or several of the stepping stone algorithms described in Section 3. It then offers much more detail on the Source Path Isolation Engine as a candidate IP traceback system that can meet the needs of an integrated attack attribution system, especially by providing the infrastructure for supporting inclusion of a stepping stone detection architecture. Following that are specifics of how SPIE can be extended. This section concludes with a discussion of the impact of a hostile or sparse environment on SPIE, IP traceback, and stepping stone detection.

---

## 5.1 Stepping Stone Detection Architecture

The stepping stone detection techniques described in the literature and surveyed in Section 3 focus more on the effectiveness of the offered algorithm and much less on how that algorithm can be practically deployed in a network. Generally the algorithms are tested using traffic traces to measure their results against ground truth. They are rarely if ever placed in multiple locations in live networks to measure how well detected stones can be stitched together to trace back across the extended connection.

Closest to a deployable system, perhaps, is the Bro [21] work by Paxson. Bro is an intrusion detection system that passively monitors network traffic for certain types of events at various levels. One type of event Bro can look for is the presence of stepping stones based on the algorithm by Zhang and Paxson [38]. Bro uses policy scripts made up of event handlers that specify what to do whenever an interesting event occurs. These event handlers can draw from a rich set of reactions to events, from setting alerts to calling functions. From these building blocks it is possible to envisage a set of cooperating Bro installations that piece together the components of an extended connection, but this functionality is beyond the Bro distribution and would require considerable software construction.

Each of the stepping stone algorithms in the literature have certain advantages and disadvantages, making them more or less suitable for certain types of networks or network traffic. There are three axes that define the space within which the algorithms fit: how much data is required for the algorithm to produce a usable result, whether the algorithm is content based or timing based, and how resistant the algorithm is to evasive techniques.

We have chosen four algorithms for study, as reported in Section 4: Thumbprints, ON/OFF Periods, Multiscale Detection, and State-Space Correlation.

Our approach is to consider a stepping stone detection system that is comprised of several algorithms, with one *master function* that monitors each of the individual algorithms and aggregates their results into a composite score. In this way, we hope to exploit the advantages and mitigate the disadvantages of the selected algorithms.

The master function must aggregate the results of these algorithms into a single score to be used to determine the correlated pairs of connections passing through this detection point. No stepping stone detection algorithm can issue a result with complete certainty; there is some degree of likelihood associated with each answer. Further, each of these algorithms uses its own method for determining a likely connection pair. Some use thresholds, some use probabilities, and some make declarative statements. The master function must be able to assimilate these answers, give them the proper weight, and construct an accurate representation of the consensus.

There is one further complication. Since we have included one algorithm (Thumbprints) that is based on content correlation, this algorithm will certainly not detect stepping stones when the connections are using encryption, while others (hopefully) will. Also, the Multiscale algorithm is claimed to be resistant to evasive techniques, while others may not be so. The master function must consider the strengths of algorithms when aggregating their results so not to discount the presence of a stepping stone when consensus cannot be achieved.

While testing the correlation of the forward packet streams of the connections, our experiments showed that the reverse packet streams also have strong correlation. Since a connection's forward and reverse packets can easily be identified, we feel that correlation on *both* the forward path and the reverse path adds to the overall strength of the correlation result. Consequently, the master function should take this into consideration when identifying stepping stones.

Our stepping stone detection architecture also considers the placement of the detection systems. We assume that hosts are not multi-homed, and therefore use a single identifiable router for all incoming and outgoing traffic. This means that the router serving a host acting as stepping stone will see the traffic for both connections in the connection pair. This router is called a *stub router*. Given that the router is the only sure point of commonality for the traffic, we assert that this is the appropriate location for stepping stone detectors. Further, given an IP traceback system that can determine the ingress router for a given packet, this stub router will be the ingress router for packets associated

---

with the outgoing connection originating at the host behind the router. It will also be the last router to have seen the packets on the incoming connection and, therefore, be the natural starting place for a trace of packets associated with that incoming connection to determine that connection's ingress point (which possibly has evidence of another stepping stone, and, therefore, another connection to trace).

This stepping stone detection architecture suggests that it be coupled with an IP traceback system that can trace single packets, can determine the ingress router, and has a presence at each router in the network. SPIE, described next, is such a traceback system.

## 5.2 IP Traceback Architecture using SPIE

The Source Path Isolation Engine (SPIE) [30] is a log-based traceback system that uses efficient auditing techniques at network routers to support the traceback of individual IP packets. Traffic auditing is accomplished by computing and compactly storing packet digests rather than storing the packets themselves. Every packet traversing a SPIE-enhanced router is recorded in a *digest table*; digest tables are paged at a specified rate and are representative of the traffic forwarded by the router during a particular time interval. A cache of digest tables is maintained for recently forwarded traffic.

If a packet is determined to be offensive by some intrusion detection system (or judged interesting by some other metric), a trace request is dispatched to the SPIE system which in turn queries routers for packet digests of the relevant time periods. The results of this query are used in a simulated reverse-path flooding algorithm to build an attack graph that indicates the packet's source(s).

Historically, tracing individual packets has required prohibitive amounts of memory; one of SPIE's key innovations is to reduce the memory requirement (down to 0.5% of link bandwidth per unit time) through the use of Bloom filters [3]. By storing only packet digests, and not the packets themselves, SPIE also does not increase a network's vulnerability to eavesdropping. Therefore, SPIE's hash-based traffic auditing allows routers to efficiently determine if they forwarded a particular packet within a specified time interval while maintaining the privacy of unrelated traffic.

Rather than hash the entire packet, our research indicates that the first 8 bytes of payload, along with the immutable fields from the header, are sufficient to differentiate almost all non-identical packets.

### 5.2.1 Bloom Filters

Constructing a digest table containing packet digests corresponding to the traffic forwarded by a router for a given time interval is a challenging task. SPIE implements digest tables using space-efficient data structures known as Bloom filters [3]. A Bloom filter computes  $k$  distinct packet digests for each packet using independent uniform hash functions, and uses the  $n$ -bit results to index into a  $2^n$  single-bit array. The array is initialized to all zeros, and bits are set to one as packets are received. Figure 12 depicts a Bloom filter with  $k$  hash functions.

Membership tests can be conducted simply by computing the  $k$  digests on the packet in question and checking the indicated bit positions. If any one of them is zero, the packet was not forwarded by this router. If, however, all the bits are one, it is highly likely the packet was forwarded. It is possible that some set of other insertions caused all the bits to be set, creating a *false positive*, but the rate of such false positives can be controlled by only allowing an individual Bloom filter to store a limited number of digests [10]. A saturated filter is swapped out for a new, empty filter, and archived for later querying.

### 5.2.2 Architecture

The tasks of packet auditing, query processing, and attack graph generation are dispersed among separate components in the SPIE system. Figure 13 shows the three major architectural components of the SPIE system. Each SPIE-enhanced router has a Data Generation Agent (DGA) associated with it. The DGA produces packet digests of

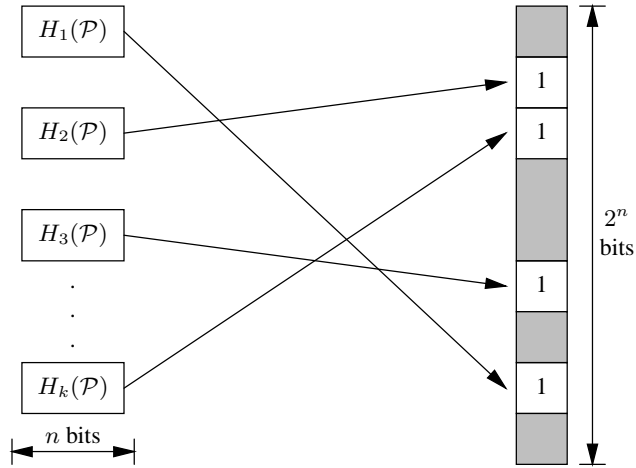


Figure 12: For each packet received, SPIE computes  $k$  independent  $n$ -bit digests, and sets the corresponding bits in the  $2^n$ -bit digest table.

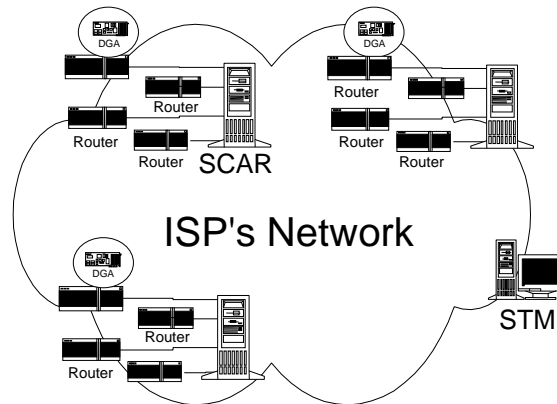


Figure 13: The SPIE network infrastructure, consisting of Data Generation Agents (DGAs), SPIE Collection and Reduction Agents (SCARs), and a SPIE Traceback Manager (STM).

---

each packet as it is forwarded through the router, and stores the digests in time-stamped digest tables. The tables are paged every so often, and represent the set of traffic forwarded by the router for a particular interval of time. Each table is annotated with the time interval and the set of hash functions used to compute the packet digests over that interval. The digest tables are stored locally at the DGA for some period of time, depending on the resource constraints of the router.

SCARs are responsible for a particular region of the network, serving as data concentration points for several routers and facilitating traceback of any packets that traverse the region. When a trace is requested, each SCAR produces an attack graph for its particular region. The attack graphs from each SCAR are grafted together to form a complete attack graph by the SPIE Traceback Manager (STM).

The STM controls the whole SPIE system. The STM is the interface to the intrusion detection system or other entity requesting a packet trace. When a request is presented to the STM, it verifies the authenticity of the request, dispatches the request to the appropriate SCARs, gathers the resulting attack graphs, and assembles them into a complete attack graph. Upon completion of the traceback process, the STM replies to the intrusion detection system with the final attack graph.

### 5.2.3 Traceback Processing

Before the traceback process can begin, an attack packet must be identified. Most likely, an intrusion detection system will determine that an exceptional event has occurred and provide the STM with a packet,  $\mathcal{P}$ , victim,  $V$ , and time of attack,  $T$ . SPIE places two constraints on the intrusion detection system: the victim must be expressed in terms of the last-hop router, not the end host itself, and the attack packet must be identified in a timely fashion. The first requirement provides the query process with a starting point; the latter stems from the fact that traceback must be initiated before the appropriate digest tables are overwritten by the DGAs. This time constraint is directly related to the amount of resources dedicated to the storage of traffic digests.

Upon receipt of a traceback request, the STM dispatches the query to the relevant SCARs for processing. Beginning at the SCAR responsible for the victim's region of the network, the STM sends a query message containing  $\mathcal{P}$ ,  $V$ , and  $T$  as provided by the intrusion detection system. The SCAR polls its DGAs and responds with a partial attack graph, the time  $T'$  the packet entered the region, and the entering packet itself  $\mathcal{P}'$  (it may have been transformed, possibly multiple times, within the region).

The attack graph either terminates within the region managed by the SCAR, in which case a source has been identified, or it contains nodes at the edge of the SCAR's network region. In the latter case the STM sends a new query for the transformed packet  $\mathcal{P}'$  to the SCAR for the abutting network region. This query uses the border router between the two network regions as its victim,  $V'$ , and  $T'$  as the time of attack. This process continues until all branches of the attack graph terminate, either at a source within the network, or at the edge of the SPIE system. The STM then constructs a composite attack graph which it returns to the intrusion detection system.

## 5.3 Extensions for Stepping Stone Detection

Consider any given packet that is part of a connection. It is easy to associate that packet with all other packets in the same connection by matching the source and destination IP addresses and ports; all packets that exactly share these values are, for some period of time, packets within the same connection. We caution that this is for some period of time because connection identification material can be reused after it has been held unused for an amount of time far exceeding the lifetime of any packet within the network. Nonetheless, any packet that is part of a connection can be definitively identified as part of that connection. Consequently, any given packet from a connection can be used by SPIE to trace to the source of that connection.

The trace begins with just such a packet. SPIE traces this packet to the router nearest to the immediate source. Since we assume that the hosts are not multi-homed, any connection that is part of a stepping stone connection pair must

---

also have come through this router. The stepping stone detection algorithms, therefore, are deployed at or near these (and all other) routers which serve as the first hop for any hosts. Assume that the stepping stone detection algorithms have produced a table of connection pairs such that, given an outgoing connection, the table indicates the correlated incoming connection. In order to continue the trace across the stepping stone, SPIE must query this connection pair table to determine the next connection to trace. Since the SPIE trace already has a representative packet within the outgoing connection, it is easy to look up the outgoing connection in the table. The table returns the corresponding incoming connection. But SPIE requires a specific example of a packet to conduct the trace; it cannot construct such a packet given only the connection identification material (IP addresses and ports). Therefore, we must add another field in the connection pair table—a representative packet. Now, when SPIE consults the connection pair table, the result will be the incoming connection and one packet from the connection. SPIE can then construct a new query based on the packet from the incoming connection, and continue the trace.

Actually, SPIE requires three pieces of information for a trace: the packet, the time the packet was seen, and the last router known to have seen the packet. The table, therefore, must return not only the packet but also the time that packet was seen. The last router known to have seen the packet is the current router since this must be the last router used by the incoming connection.

Since SPIE is sensitive about the age of the packets it traces—the older the packet, the less likely evidence of the packet’s path is still kept in the Bloom filters in the SPIE DGAs—the packet that gets installed into the connection pair table should be the last packet seen on the incoming connection, along with the time that packet was seen. This gives SPIE the best chance of completing the trace with the incoming connection’s representative packet since it is the newest packet on that connection.

SPIE then conducts a normal trace to once again find the ingress router for the connection. Again, the connection pair table is consulted, and additional packet traces may be launched (even tracing through NAT boxes, if necessary), until at some point the table does not report a correlated connection. At this point, the trace terminates with the last ingress router, which is as far as SPIE can go. However, since the traced packet is part of a connection and, therefore, must have a valid source address that can be used for returning packets such as acknowledgments, the host that is the actual originating source can be easily determined.

The core of SPIE remains largely unmodified. The major change required in order to extend SPIE to use the information from the connection pair tables is how SPIE reacts at the termination of the packet trace. Now SPIE must recognize that the trace has reached the last router, consult the connection pair table within (or near) that router, and construct a new query based on the results from the table. There are details, of course, to be considered, such as making sure that the security model applied to SPIE can also be extended to include the trace across stepping stones, how the connection pair table purges stale data, and what is the appropriate response if a trace fails without reaching an immediate source. However, these and other questions are part of future research; here we have just laid out the foundation for an integrated attack attribution system.

## **5.4 Source Attribution in Sparse and Hostile Environments**

The algorithms and techniques discussed in this paper have interesting and advantageous properties under ideal circumstances. This section will examine SPIE in particular and the stepping stone detection problem in general from the perspective of a sparsely deployed or hostile environment.

### *5.4.1 SPIE in Sparse and Hostile Environments*

The SPIE model is predicated on full network coverage—that is, in order to reconstruct a packet’s path through the network, a traceback system like SPIE must be able to “visit” each router and ask if that router had seen the packet. Reducing the coverage of routers in the network necessarily reduces the accuracy of the answers, but there are some circumstances and methods by which deployment can be thinned without having an unacceptable impact on the reliability of the trace results.

---

When a router is not covered by SPIE, then there is a gap in the query process. Since SPIE’s false positives are engineered to be low, simply skipping over this router and continuing the query at that router’s neighbors is sufficient, but it does increase the likelihood of false positives. Assume that the probability of not getting a false positive at router  $R_1$  and its neighbor  $R_2$  is  $p$  for each. If the packet is seen at  $R_1$ , then the probability of a false positive at  $R_2$  is  $1 - p$ . But the probability of a false positive at  $R_2$  without any knowledge of whether the packet was seen at  $R_1$  is  $1 - p^2$ , which is greater. The math works in a similar way for each additional router that is skipped.

More rigorously, the SPIE model can handle skipped routers by subsuming those routers’ neighbors. Given a router  $R$  with some number of neighbors  $N(R)$ , where none of those neighbors is instrumented with SPIE, then the new set of neighbors for  $R$  is the set of all neighbors of all of  $R$ ’s neighbors, or  $N(N(R))$ . Mathematically, this explodes the stress on the Bloom filter, requiring the filter to either be much larger or much more quickly replaced in order to maintain the same rate of false positives.

The above argument works well only if there are no transformations possible at the skipped router. Since a transform dramatically changes a packet, so much so that a new packet is essentially reconstructed for the trace when a transform has been detected, it is impossible to continue the trace across routers that are not participating in recording transformations. Consequently, such a network must either instrument all transforming routers with SPIE or disallow any routers from transforming the packets. This means blocking tunnels, NATs, source routing, and ICMP reflections.

While the above argument is true, one way to allow transforms at skipped routers is to try to guess all transforms possible at the skipped router and form queries for those contingencies. Since transforms generally require additional information to be saved at the DGA in order to invert the transform during a query, trying to guess all of the possible values may be very difficult.

Nonetheless, if one had to choose where to place SPIE taps, the perimeter of the network is clearly the best place. Here one can watch the traffic entering and leaving the network enclave, and SPIE can answer the important question of which entry point did this packet gain access to the network.

In a hostile environment, one can expect no SPIE deployment at all, little knowledge of the network topography, no knowledge of internal transforms, and active countermeasures to any probing. Visibility is limited to the friendly gateways bordering the hostile network. One possibility is to use SPIE as a *post mortem* analysis tool operating on packet traces and some additional notion of the network topology (perhaps obtained through covert tomographic methods). In this case, SPIE is essentially deployed in a simulated network representing the best understanding of the hostile target environment.

#### 5.4.2 Stepping Stone Detection in Sparse and Hostile Environments

The ability to gather trace information at locations where stepping stone hosts are likely to be is of most concern when deploying stepping stone detectors in a sparse or hostile environment. The locations of most concern are stub routers. If there is a choice about limited deployment, instrumenting stub routers is the first place to expend resources. However, this is precisely the most difficult place to gain access in a hostile environment.

An interesting property of correlating chains of interactive connections is that it is largely transitive. Given three connections,  $C_1$ ,  $C_2$ , and  $C_3$ , direct correlation (as explained in Section 2.2.2) examines the relationship of  $C_1$  to  $C_2$ , and  $C_2$  to  $C_3$ . Indirect correlation suggests that if  $C_1$  and  $C_3$  are both related to  $C_2$ , then  $C_1$  can be correlated to  $C_3$ . This makes sense intuitively because the same data is being transported across all three connections. This implies that connection data collected at distributed locations throughout the network can be centrally examined for indirect correlation. Certainly this increases the total number of connections to be examined.

A side effect of including stepping stone detection with packet tracing in a sparse environment is that the detection of a stepping stone may actually mitigate the lack of absolutely accurate results from the packet traceback. Consider a sparsely deployed SPIE system, where transforms may occur within the skipped routers. If a trace is “lost” because

---

of insufficient coverage, but we find that there is a stepping stone related to the connection of which the traced packet was a part, we can use the example packet from the correlated connection (the other connection in the connection pair that meet at the stepping stone) to resume the trace.

## 6 Conclusion

This paper has presented the foundation of and an architecture for an integrated attack attribution system. IP traceback provides the infrastructure and the definitive identification of immediate sources—the endpoints of connections, if the packet being traced is a part of a connection—and the stepping stone detection algorithms provide the means to continue the trace across the laundering hosts to find the originating source.

We have surveyed IP traceback schemes, with particular attention paid to the Source Path Isolation Engine as a solution that lends itself nicely to supporting stepping stone detection in an integrated system. We have also surveyed stepping stone detection techniques, and offered a taxonomy of these based where the algorithms are deployed and what data the algorithms use to determine connection pair correlation.

Four stepping stone detection algorithms appear to be most promising, three of which are based on timing analysis, and one on content correlation. Of the timing-based algorithms, the ON/OFF Periods algorithm seems efficient, focusing on short horizon correlation to deliver a result quickly. This algorithm does not attempt to be resistant to evasive techniques such as jitter and chaff, but is not perturbed by the use of encryption. The Multiscale algorithm was specifically designed to resist evasion, and does so with a longer correlation horizon, so its results are delivered later in the process. The third timing-based algorithm is a new algorithm developed for wireless node detection and modified to detect stepping stones. It uses a Lagrangian state-space to correlate cause-and-effect events. The fourth algorithm, Thumbprints, is based on correlation of content, so it will not perform well in the presence of encryption but has potential to be efficient in cleartext sessions.

We have conducted a set of experiments on the four chosen algorithms, simplifying the test environment to focus only on the detection of the stepping stone. Each of the algorithms clearly detects the stepping stone. We found that we had to run series of tests to find appropriate settings for tunable parameters since the descriptions of the algorithms did not generally give good guidelines for how they are set. It is also not clear that much study has been done on how to automate these algorithms—that is, how to adjust the parameters in such a way that there is high confidence that stepping stones of various types will be detected. We were surprised at how sensitive some of the algorithms were to very slight adjustments, which does not provide confidence in their robustness. Nonetheless, this study represents an independent evaluation of the algorithms from the literature, and in most cases, the first time these algorithms have been implemented with deployment in mind.

Of particular note, we believe, is the State-Space Correlation algorithm. This algorithm was developed initially for detecting wireless receivers based on their reaction to transmissions. While the original wireless problem seems similar to detecting stepping stones, our work represents the first time this algorithm has been applied to the stepping stone detection problem. The initial results are promising. It will be interesting to further study this algorithm to determine its efficiency in detection (especially compared with the other algorithms), and its effectiveness in withstanding evasion.

We have offered an architecture for building a composite stepping stone detection system using these four (or any combination of these or other) algorithms. The effectiveness of this approach hinges on a master function to aggregate the results of each detection algorithm and present a single score. Doing so requires the master function to understand the relative strengths and weaknesses of the algorithms, and to discount or promote one over others in different circumstances to play to the strengths. We have outlined the requirements for the master function, but have found that actually designing it for these four specific algorithms requires further development of the implementations of the algorithms and continued experimentation.

This stepping stone detection system is easily integrated with the SPIE traceback system. The SPIE system traces

---

through situations that can confuse or obscure the true source of a connection, such as network address translation (NAT) boxes, to the ingress router nearest the immediate source of the connection. From there, SPIE asks the router about connection pairs, and a table built by the stepping stone algorithms returns the incoming connection correlated with the connection being traced. SPIE then continues the trace using a representative packet from this incoming connection, and the process repeats at that connection's ingress router.

An integrated system such as described in this paper is capable of performing end-to-end traces that cannot be done individually by either of the two original individual components—SPIE and stepping stone detectors. SPIE by itself is unable to trace across a host that is creating a stepping stone. The stepping stone detectors cannot link pairs of stepping stones without the help of a network-wide component. SPIE provides that glue. The two approaches complement each other in an integrated system to provide much needed intruder traceback.

## References

- [1] BELENKY, A., AND ANSARI, N. On IP traceback. *IEEE Communications Magazine* 41, 7 (2003), 142–153.
- [2] BELLOVIN, S. M., LEECH, M., AND TAYLOR, T. ICMP traceback messages. Internet Draft, Oct. 2001. `draft-ietf-itrace-01.txt` (work in progress).
- [3] BLOOM, B. H. Space/time trade-offs in hash coding with allowable errors. *Communications of ACM* 13, 7 (July 1970), 422–426.
- [4] BUCHHOLZ, F. P., AND SHIELDS, C. Providing process origin information to aid in network traceback. In *Proc. USENIX Annual Technical Conference* (June 2002).
- [5] BURCH, H., AND CHESWICK, B. Tracing anonymous packets to their approximate source. In *Proc. USENIX LISA '00* (Dec. 2000).
- [6] CAPPE, O., MOULINES, E., PESQUET, J.-C., PETROPULU, A., AND YANG, X. Long-range dependence and heavy-tail modeling for teletraffic data. *IEEE Signal Processing Magazine* 19-3 (2002), 14–27.
- [7] CARRIER, B., AND SHIELDS, C. A recursive session token protocol for use in computer forensics and tcp traceback. In *Proc. IEEE Infocom '02* (June 2002).
- [8] DONOHO, D. L., FLESIA, A. G., SHANKAR, U., PAXSON, V., COIT, J., AND STANIFORD, S. Multiscale stepping-stone detection: Detecting pairs of jittered interactive streams by exploiting maximum tolerable delay. In *Proc. International Symposium on Recent Advances in Intrusion Detection* (Oct. 2002), pp. 17–35.
- [9] EGEVANG, K., AND FRANCIS, P. The ip network address translator. RFC 1631, May 1994.
- [10] FAN, L., CAO, P., ALMEIDA, J., AND BRODER, A. Z. Summary cache: a scalable wide-area web cache sharing protocol. *ACM/IEEE Trans. on Networking* 8, 3 (2000), 281–293.
- [11] FERGUSON, P., AND SENIE, D. Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing. RFC 2267, Jan. 1998.
- [12] HAZEYAMA, H., OE, M., AND KADOBAYASHI, Y. A layer-2 extension to hash-based IP traceback. *IEICE Trans. on Information & Systems*. to appear in November 2003 issue.
- [13] HOWARD, J. D. An analysis of security incidents on the internet, 1989 - 1995. PhD Thesis, Apr. 1997. <http://www.cert.org/research/JHThesis/Start.html>.
- [14] JOHNS, M. S. Identification protocol. RFC 1413, Feb. 1993.
- [15] JONES, C. E., TCHAKOUNTIO, F., SNOEREN, A. C., SCHWARTZ, B., CLEMENTS, R. C., CONDELL, M., PARTRIDGE, C., AND STRAYER, W. T. Traceback of ip packet transformations. Internal technical memo, BBN Technologies, 2002.
- [16] JUNG, H. T., KIM, H. L., SEO, Y. M., CHOE, G., MIN, S. L., AND KIM, C. S. Caller identification system in the internet environment. In *Proc. USENIX Security Symposium '93* (Oct. 1993).
- [17] LEE, S. C., AND SHIELDS, C. Tracing the source of network attack: A technical, legal and societal problem. In *Proc. IEEE Systems, Man, and Cybernetics Information Assurance Workshop* (2001).
- [18] LIBES, D. The Expect home page. Tech. rep., National Institute of Standards and Technology. <http://expect.nist.gov/>.
- [19] MANKIN, A., MASSEY, D., WU, C.-L., WU, S. F., AND ZHANG, L. On design and evaluation of "intention-driven" ICMP traceback. In *Proc. IEEE International Conference on Computer Communications and Networks* (Oct. 2001).
- [20] PARTRIDGE, C., COUSINS, D. B., JACKSON, A. W., KRISHNAN, R., SAXENA, T., AND STRAYER, W. T. Using signal processing to analyze wireless data traffic. In *Proc. ACM Workshop on Wireless Security (WiSe)* (Sept. 2002).
- [21] PAXSON, V. Bro: A system for detecting network intruders in real-time. In *Proc. USENIX Security Symposium, Jan. 1998* (Jan. 1998), pp. 1–11.

- 
- [22] PAXSON, V. An analysis of using reflectors for distributed denial-of-service attacks. *ACM Comp. Comm. Review* 31, 3 (2001).
- [23] PERKINS, C. E. IP mobility support for IPv4. RFC 3344, Aug. 2002.
- [24] REKHTER, Y., MOSKOWITZ, B., KARRENBERG, D., DE GROOT, G. J., AND LEAR, E. Address allocation for private internets. RFC 1918, Feb. 1996.
- [25] SAGER, G. Security fun with OCxmon and cfbwd. Internet 2 Working Group Meeting, Nov. 1998. <http://www.caida.org/projects/NGI/content/security/1198>.
- [26] SANCHEZ, L. A., MILLIKEN, W. C., SNOEREN, A. C., TCHAKOUNTIO, F., JONES, C. E., KENT, S. T., PARTRIDGE, C., AND STRAYER, W. T. Hardware support for a hash-based IP traceback. In *Proc. Second DARPA Information Survivability Conference and Exposition* (June 2001), vol. 2, pp. 146–152.
- [27] SAVAGE, S., WETHERALL, D., KARLIN, A., AND ANDERSON, T. Network support for IP traceback. *ACM/IEEE Trans. on Networking* 9, 3 (June 2001), 226–239.
- [28] SCHNACKENBERG, D., DJAHANDARI, K., AND STERNE, D. Infrastructure for intrusion detection and response. In *Proc. First DARPA Information Survivability Conference and Exposition* (Jan. 2000).
- [29] SNAPP, S. R., BRENTANO, J., DIAS, G. V., GOAN, T. L., HEBERLEIN, L. T., HO, C.-L., LEVITT, K. N., MUKHERJEE, B., SMAHA, S. E., GRANCE, T., TEAL, D. M., AND MANSUR, D. DIDS (distributed intrusion detection system) - motivation, architecture, and an early prototype. In *Proc. National Computer Security Conference* (Oct. 1991), pp. 167–176.
- [30] SNOEREN, A. C., PARTRIDGE, C., SANCHEZ, L. A., JONES, C. E., TCHAKOUNTIO, F., SCHWARTZ, B., KENT, S. T., AND STRAYER, W. T. Single-packet IP traceback. *ACM/IEEE Trans. on Networking* (Dec. 2002).
- [31] SONG, D. X., AND PERRIG, A. Advanced and authenticated marking schemes for IP traceback. In *Proc. IEEE Infocom '01* (Apr. 2001).
- [32] STANIFORD-CHEN, S., AND HEBERLEIN, L. T. Holding intruders accountable on the internet. In *Proc. IEEE Symposium on Security and Privacy '95* (May 1995), pp. 39–49.
- [33] STONE, R. CenterTrack: An IP overlay network for tracking DoS fbods. In *Proc. USENIX Security Symposium '00* (Aug. 2000).
- [34] WANG, X., AND REEVES, D. S. Robust correlation of encrypted attack traffic through stepping stones by manipulation of interpacket delays. In *Proc. ACM Symposium on Computer and Communications Security (CCS)* (Oct. 2003).
- [35] WANG, X., REEVES, D. S., AND WU, S. F. Inter-packet delay based correlation for tracing encrypted connections through stepping stones. In *Proc. European Symposium on Research in Computer Security* (Oct. 2002), pp. 244–263.
- [36] WANG, X., REEVES, D. S., WU, S. F., AND YUILL, J. Sleepy watermark tracing: An active network-based intrusion response framework. In *Proc. International Conference on Information Security* (June 2001), pp. 369–384.
- [37] YODA, K., AND ETOH, H. Finding a connection chain for tracing intruders. In *Proc. European Symposium on Research in Computer Security* (Oct. 2000), pp. 191–205.
- [38] ZHANG, Y., AND PAXSON, V. Detecting stepping stones. In *Proc. USENIX Security Symposium '00* (Aug. 2000), pp. 171–184.