Detecting Anomalies in Active Insider Stepping Stone Attacks

Giovanni Di Crescenzo, Abhrajit Ghosh
Telcordia Technologies
Piscataway, NJ, USA
{giovanni,aghosh}@research.telcordia.com

Abhinay Kampasi*
Microsoft
Redmond, WA, USA
abhinay.kampasi@microsoft.com

Rajesh Talpade†
Niksun
Princeton, NJ, USA
rtalpade@niksun.com

Yin Zhang
University of Texas at Austin
Austin, TX, USA
yzhang@cs.utexas.edu

Abstract

Network attackers frequently use a chain of compromised intermediate nodes to attack a target machine and maintain anonymity. This chain of nodes between the attacker and the target is called a stepping stone chain. Various classes of algorithms have been proposed to detect stepping stones, timing correlation based algorithms being a recent one that is attracting significant research interest. However, the existing timing based algorithms are susceptible to failure if the attacker actively tries to evade detection using jitter or chaff. We have developed three anomaly detection algorithms to detect the presence of jitter and chaff in interactive connections, based on response time, edit distance and causality. Experiments performed on Deter using real-world traces and live traffic demonstrate that the algorithms perform well with very low false positives and false negatives and have a high success percentage of about 99%. These algorithms based on response times from the server and causality of traffic in both directions of an interactive connection have made the existing stepping stone detection framework more robust and resistant to evasion.

1 Introduction

Computer users increasingly rely on network communication technology to carry out a variety of business or personal transactions, possibly involving highly private or sensitive information. As accessibility of people's data and infrastructure from the Internet and other networks becomes much easier, the frequency, impact, and variety of cyber-attacks are expected to analogously grow with time. A key element for effectively countering cyber-threats is the ability to recognize the host that originated the attack. However, network attackers frequently use a chain of compromised intermediate nodes, called stepping stones, to attack a target machine because with this approach they might avoid being recognized and thus maintain anonymity. This is a big concern for intrusion prevention, detection and tolerance in autonomous systems; here, even if an intrusion is detected, only the last host from which the attack was launched is identified whereas the actual attack came from a different host. Instead, one would like to traceback a cyber-attack from the target to the origin host, identifying any intermediate stepping-stones that may have been used. The problem of detecting stepping stones was first addressed in the ground-breaking paper [1]. The earliest methods proposed in the literature followed different types of content-based approaches. However, content-based techniques are expensive because they involve payload analysis and also getting access to packet payload is not always possible due to privacy concerns. Another serious limitation in that much of the interactive traffic today is encrypted and hence content

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, volume: 2, number: 1, pp. 103-120

*Work done while at University of Texas at Austin

†Work done while at Telcordia Technologies

comparison is not possible. In a significant departure from content-based techniques, some timing-based methods were produced in [2]. Here, the methods focused on passive traffic monitoring but also raised the issue of active traffic perturbations. In this paper we complement the work of [2], by studying timing-based methods for the stepping stone problem and, more generally, the attack trace back problem, in a scenario that admits active traffic perturbations.

Our contribution. We observe that active traffic perturbations create traffic anomalies and therefore combine novel anomaly detection methods with novel timing-based stepping stone detection methods into an attacker trace back methodology that is expected to be much harder to evade. We design three anomaly detection algorithms to detect the presence of jitter and chaff in connections by correlating traffic in both directions of a connection. The "response-time based" algorithm uses the time interval between sending of a packet from the source and receipt of the response packet in the form of an echo from the destination to detect jitter. The "edit-distance based" and "causality based" algorithms rely on the fact that there is a strong causality relation between two directions of an interactive connection and are primarily used to detect chaff. If the attacker tries to obfuscate the traffic then this causality relation is broken resulting in the network traffic appearing anomalous.

Intrusions are a problem both in an Intranet and in an Internet setting. Although our algorithms make a number of assumptions that are more appropriate for Intranets and thus our overall solutions achieve a much higher level of success and practicality when facing active stepping stone attacks from network insiders. However, they achieve similar properties in any Internet setting as long as one can monitor hosts, connections and communication exchanged across them.

We performed experiments on Deter test bed. Real-world Internet traces from Telcordia Technologies and University of Texas Computer Sciences department were used along with some traces with jitter and chaff generated on Deter. The algorithms were also evaluated on live traffic. The implementation of these algorithms in Bro [3, 4] intrusion detection system shows that they are able to detect jitter and chaff with a high success rate of about 99% and very low false positives/negatives.

Related work. There has been considerable research on stepping stone detection. The initial line of research focused on content-based detection techniques, including comparing content over different streams looking for a high degree of correlation [1] and actively injecting content watermark into interactive traffic [5]. Timing-based stepping-stone detection was first proposed in [2] and, since then, timing-based stepping stone detection has become an active research area. Techniques proposed in this area include computing deviations between a known intruder stream and other concurrent streams on the Internet [6], using inter-packet delays to correlate connections in real-time [7], modifying timing-based algorithms for wireless node detection into stepping-stone detection algorithms [8]. A comparison study of several existing algorithms can be found in [8].

Research on timing-based stepping stone detection has then focused on making the algorithm more resistant to evasions like timing perturbation and chaffs. In [9] the authors assume that the intruder has a maximum delay tolerance, use wavelet analysis and conclude that for sufficiently long connections there are theoretical limits on the ability of attackers to disguise their traffic (through local jittering of packet arrival times and the insertion of chaffs). Their analysis is only asymptotic and a proof-of-concept, and makes the restrictive assumption that the chaff insertion process is independent of the packet arrival process. In [10] the authors proposed a watermark-based scheme, which detects correlation between streams of packets by actively injecting watermark into inter-packet delays. However, they assume that the attacker's timing perturbation of packets is independent and identically distributed, which may not hold in practice. The stepping stone detection algorithm proposed by [11] is based on monitoring the number of packets between connections. This paper uses computational learning theory and random walk analysis to provide provable upper bounds on the number of packets one needs to observe to confidently detect a stepping stone and to gives bounds on the amount of chaff that the attacker would need to inject in

order to evade detection. Despite recent progress in the stepping-stone detection area, it is still relatively easy for an attacker to evade existing stepping-stone detection algorithms. Specifically, [11] shows that an attacker can simulate two independent interactive flows (and thus evading detection by any existing algorithm) by inserting a small amount of chaff.

Significant research exists that addresses the problem of traceback of Distributed Denial of Service (traffic-flood-based) attacks. An excellent overview and comparison of relevant DDoS traceback techniques is provided in [12]. Most of the techniques require modification to IP protocols and routers/switches before they can be used. Since they are focused on DDoS, the traceback capability is limited to the actual end-hosts that send the traffic flood, i.e. they cannot handle multiple levels of stepping-stones.

Anomaly detection is a frequently occurring and widely studied problem in data analysis in various fields including networking [13]. One of the main challenges in anomaly detection is the case where anomalies result from adversarial behavior where attempts are made to mask patterns to make them appear normal to avoid detection [14]. There has been some amount of work in the past that relates to network intrusion detection via anomaly detection: [15] focuses on the detection of flooding DoS attacks; [16] identifies precursors of DoS attacks; [17] performs stateful intrusion detection of scanning followed by buffer overflow attacks using attack signatures; [18] uses a signature-based approach to detect routing attacks and stepping stone attacks and [19] discusses the detection of DoS attacks by applying SVM-based learning techniques. In [20] the authors apply machine learning based anomaly detection techniques to identify malicious nodes in a network.

There have been some anomaly detection techniques [21, 22, 23] proposed to detect stepping stones based on the time difference between send packet and the corresponding echo packet. This time difference is very small for normal connections but increases proportionally to the number of intermediate hosts in the chain. The response-time based anomaly detection algorithm proposed by us relies on the same concept but is used to detect the presence of jitter rather than stepping stones.

Organization of the paper. The rest of the paper is organized as follows. In Section 2 we discuss the attack model considered in the paper and exemplify how an attacker can evade stepping stone detection algorithms, by using as an example the method from [2]. In Section 3 we present our three anomaly detection algorithms and in Section 4 we outline the new attacker trace back methodology. In Section 5 we describe our evaluation experiments by outlining the experimental setup, the custom server implementation, and the evaluation of the three anomaly detection algorithms. Finally, we present our conclusions in Section 6. Our results have also previously appeared as [24].

2 Model and Algorithm Evasion

Our attack model, shown in Figure 1, considers an origin host (where the attacker is located), a final host (the attack target) and a stepping stone chain between attacker and target. In this model, the *stepping stone detection* problem consists of detecting whether a given node belongs to the chain between attacker and target, and the *attacker traceback* problem consists of detecting all stepping stones and the origin host associated with an attack to a target host. Iteratively solving the stepping stone detection problem for each stepping stone from the target host to the attacker host can be used as a solution for the attacker traceback problem. Attackers typically utilize interactive sessions (e.g., Telnet, SSH) between the origin host and the stepping stones, and between pairs of stepping stones, for initiating the attack. Monitoring the communication exchanged across these sessions is a typical initial step towards solving both problems.

As we will concentrate on timing-based methods, we will use the following basic definitions. A session can be characterized as a sequence of ON and OFF periods, as follows. When there is no data traffic on a session for more than T_{idle} seconds, the session is considered to be in an OFF period. We consider a packet as containing data only if it carries data in its TCP payload. When a packet with non-

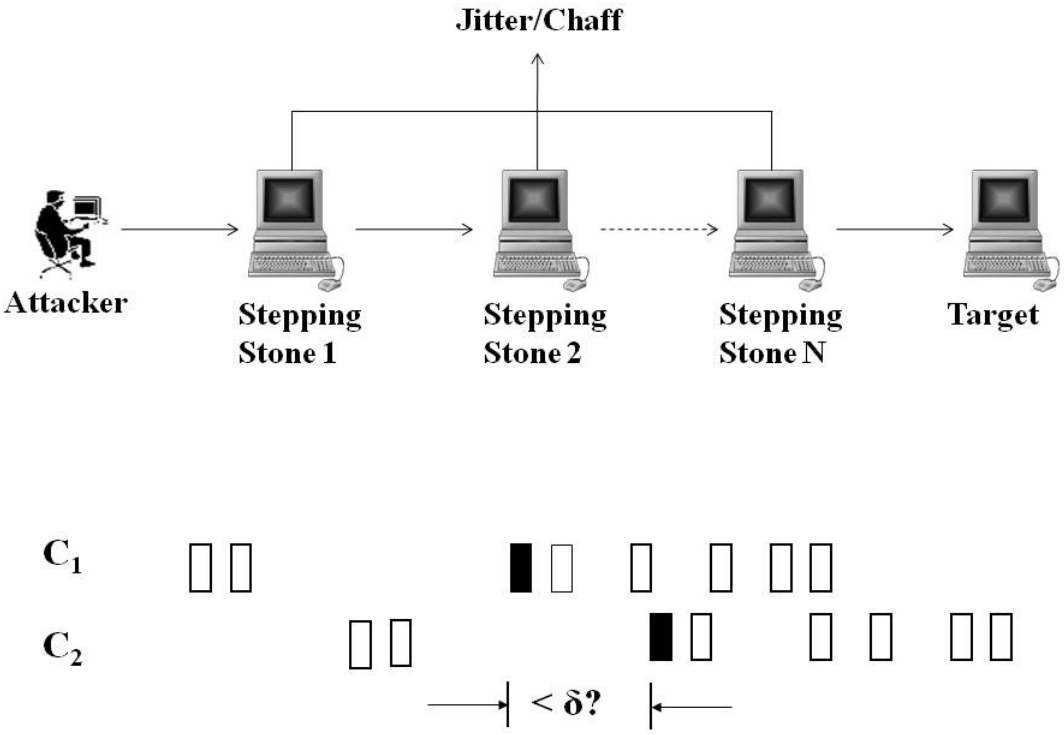

Figure 1: (Top): Stepping stone chain between attacker and target. (Bottom): Correlation of packets for connections C_1 and C_2 based on timing.

empty payload then appears, the flow ends its OFF period and begins an ON period, which lasts until the session again goes data-idle for T_{idle} seconds. The motivation for considering traffic as structured into ON and OFF periods comes from the strikingly distinct distribution of the spacing between user keystrokes. Studies of Internet traffic have found that keystroke inter-arrivals are very well described by a Pareto distribution with fixed parameters [25]. The parameters are such that the distribution exhibits infinite variance, which in practical terms means a very wide range of values. In particular, large values are not uncommon: about 25% of keystroke packets come 500 msec or more apart, and 15% come 1 sec or more apart (1.6% come 10 sec or more apart). Thus, interactive traffic will often have significant OFF times. We can then exploit the tendency of machine-driven, non-interactive traffic to send packets back-to-back, with a very short interval between them, to discriminate non-interactive traffic from interactive.

In the rest of this section we show that the attacker can evade an existing stepping stone detection algorithm if he injects sufficient amount of timing jitter or chaff packets at any of the intermediate stepping stones. We focus on the stepping stone detection algorithm from [2] but note that similar observations are expected to hold for other methods that do not specifically address these active traffic perturbations. We have leveraged Netcat [26] to implement a custom server that injects jitter and/or chaff in the connection chain. We ran the custom server in jitter mode, chaff mode and jitter+chaff mode and successfully evaded the algorithm in each of these modes.

2.1 A Stepping Stone Detection Algorithm

The stepping stone algorithm from [2] is based on the fact that if two nodes are part of a stepping stone chain, then the flow of traffic on these machines will be highly correlated. Each connection is split into a stream of ON-OFF periods. An OFF period starts if no data traffic has been observed on a connection for more than T_{idle} (set to 500 msec). Any packet seen after a connection is in an OFF period marks the end of the OFF period and the start of an ON period. If the difference between end times of OFF periods (or start times of ON periods) across two connections is less than δ (set to 80 msec), then these OFF periods are said to be correlated as shown in Figure 1.

Let OFF_1 and OFF_2 denote the total number of OFF periods within two connections C1 and C2 respectively and $OFF_{1,2}$ denote the number of correlated OFF periods. C_1 and C_2 form part of a stepping stone chain if

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

where γ is set to 0.3.

The experiments performed by the authors were able to detect most stepping stones with a low percentage of false positives/negatives. As defined, this algorithm did not address active traffic perturbations, such as jitter and chaff. We observed that if the attacker actively tries to evade the timing based algorithm by randomly injecting jitter, chaff or a combination of the two in the connection then the algorithm is rendered ineffective.

2.2 Evasion Using Jitter

If the attacker injects timing jitter or delay of more than δ msec in one of the connections, then he will be able to evade detection. This is because OFF periods are considered correlated only if their end times differ by less than δ. However, if the attacker uses a custom server to explicitly inject jitter greater than δ in one of the connections then the OFF periods between the two connections will never be correlated and the attacker will be able to evade detection. In this case, the attacker is exploiting the dependence of the algorithm on the parameter δ.

2.3 Evasion Using Chaff

If the attacker injects chaff packets randomly in one of the connections then the ratio of correlated OFF periods to the total OFF periods will reduce. Injecting sufficient chaff will cause this ratio to fall below γ and the attacker will be able to evade detection. In this case, the attacker is exploiting the dependence of the algorithm on the parameter γ.

3 Anomaly Detection Techniques

We have developed three algorithms to detect jitter and chaff based anomalies in interactive traffic. The response-time based algorithm detects jitter while the edit-distance based and causality based algorithms detect chaff. While, an attacker who is oblivious to the presence of a traceback solution will be detected using the timing-based stepping-stone detection algorithm, an attacker who attempts to evade detection by obfuscating traffic flows by introducing jitter/chaff will end up having his inter stepping stone traffic appear anomalous. Hence, the stepping stone detection algorithms together with the anomaly detection techniques form a robust attacker traceback methodology that is difficult to evade. All the anomaly detection algorithms are online and can detect jitter and chaff in live interactive traffic (as well as traces).

3.1 Response-time Based Anomaly Detection

Our response-time based anomaly detection algorithm is based on the fact that in an interactive session, a packet on the forward leg of a connection (e.g. from a client to a server) must be followed by a response on the backward leg within a certain amount of time.

Let C be an interactive connection where C_{12} indicates the flow of packets from client to server and C_{21} indicates the flow of packets from server to client. The response-time based algorithm is formulated on the fact that a Send packet on C_{12} should be followed immediately by a response packet on C_{21} in the form of an Echo. Packets on C_{12} are split into ON and OFF periods using parameter T_{idle} (set to 300 msec) similar to the stepping stone detection algorithm. Splitting the connection into ON and OFF periods drastically reduces the amount of packet processing without affecting the results. We tested the algorithm with different values of T_{idle} ranging from 300 to 500 msec and did not observe any change in results indicating that the algorithm is not very sensitive to this parameter. If for every packet sent on C_{12} at start of an ON period, we do not see a response packet on C_{21} in the form of an echo within (RTT $+\delta_{RT}$) then we mark the ON period as anomalous as shown in Figure 2, where RTT denotes the round trip time, which is calculated using a smoothed version of Jacobson/Karel's algorithm [27]. The value of δ_{RT} was selected as 50 msec after analyzing the typical response times of servers in many real-world traces.

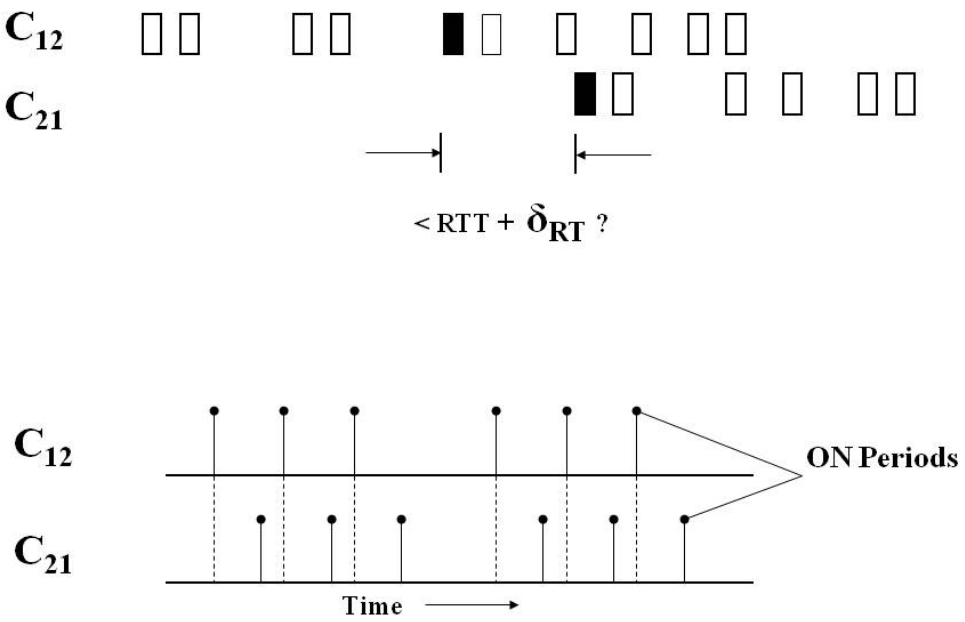

Figure 2: (Top): Response-time based anomaly detection algorithm. (Bottom): Exactly one ON period on C_{21} between two consecutive ON periods on C_{12} and vice versa.

If the ratio of anomalous ON periods to total ON periods is greater than γ_{RT} (set to 0.67) then we flag the connection as anomalous. A very small value for γ_{RT} may lead to many false positives while a large value for γ_{RT} may lead to many false negatives. We decided to select a conservative value for this

parameter, which may result in some false negatives. However, it does not affect the effectiveness of the overall framework because in order to evade the stepping stone algorithm, the attacker needs to inject jitter for more than 70% of the packets (as indicated by parameter γ in Section 3.1) and in doing so will be flagged as anomalous by this algorithm. The pseudo code of the algorithm is as follows.

Pseudocode for response-time based anomaly detection algorithm:

1. Initialize ON_Packets = 0, Anomalous_Packets = 0
2. Let C_{12} (resp., C_{21}) be the forward (resp., reverse) direction of an interactive connection
3. Split the packets on C_{12} into ON and OFF periods using Tidle
4. For every acknowledgement sent on C_{21} for a data packet sent on C_{12}
   Update RTT using Jacobson/Karel's algorithm
   For every packet sent at ON period from C_{12}
   Increment count for ON_Packets
   If response packet from C_{21} is sent within (RTT + δ_{RT}) msec
   Packet is not anomalous
   Else
   Packet is anomalous
   Increment count for Anomalous_Packets
   If procedure Check_for_anomaly returns yes
   Return: connection is anomalous due to jitter
5. Return: connection is not anomalous

Procedure Check_for_anomaly:

1. If number of ON_Packets $>$ MIN_ON_PACKETS (set to 10)
   If Anomalous_Packets / ON_Packets $>= \gamma_{RT}$
   Return: yes
2. Return: no

The algorithm may suffer from false positives if the server is extremely loaded and takes more than δ_{RT} (50 msec) time to respond. However, typical server response times are much smaller than δ_{RT} . Also, if the connection is part of a stepping stone chain then the response time will increase proportionally to the length of the chain and may exceed δ_{RT} for sufficiently long chains. Marking such connections as anomalous will not change the outcome of the stepping stone detection framework as shown in Section 5. The algorithm may suffer from false negatives if the attacker types at a very fast speed so that all packets on C_{12} are sent within T_{idle} and are not split into ON and OFF periods. However, for interactive traffic, typing so fast may be impractical for the attacker. Also, if the jitter injected is less than δ_{RT} then it will not be detected. However, such low values of jitter will not allow the attacker to evade the stepping stone detection algorithm.

3.2 Edit-distance Based Anomaly Detection

The edit-distance based algorithm relies on the fact that if an interactive connection is normal then the sequences of time durations of the associated ON and OFF intervals for two directions of this connection C_{12} and C_{21} are identical or at least very similar. Two identical sequences have an edit-distance of zero and similar sequences have an edit-distance close to zero. If the attacker injects some chaff in the connection then these sequences become dissimilar and start having a positive edit distance that increases proportionally to the amount of chaff injected. This criterion can be used to detect chaff in interactive connections.

Given streaming sequences of packets along the two directions of a connection as C_{12} and C_{21}, we use the methodology described before to split the packets into ON and OFF periods taking T_{idle} as 300

msec. The time difference between two ON periods is used to form a sequence of intervals for C_{12} and C_{21}. The streaming sequences are broken into multiple subsequences and the local edit distance of each subsequence is measured. Given that the permissible edit distance for a subsequence is β, a connection is flagged as anomalous if the cumulative edit distance of the different subsequences is greater than β times the number of subsequences. After analyzing many normal interactive connections, the value of β was set to 10. The packet stream is processed as a collection of subsequences in order to support online analysis. The pseudo code below explains the algorithm in greater detail.

Pseudocode for edit-distance based anomaly detection algorithm:

1. Initialize Cumulative_Edit_Distance = 0
2. Split the packets on two directions C_{12} and C_{21} of an interactive connection into ON and OFF periods using T_{idle}
3. Let C_{12}seq[i] be the sequence of intervals for sequence i for forward direction of traffic
4. Let C_{21}seq[i] be the sequence of intervals for sequence i for reverse direction of traffic
5. For subsequences of length SEQ_LENGTH (set to 10) for C_{12}
   Let i be the current sequence number
   Run procedure Edit_Distance (C_{12}seq[i], C_{21}seq[i])
   Let local_edit_dist[i] be the edit distance returned
   set Cumulative_Edit_Distance += local_edit_dist[i]
   If Cumulative_Edit_Distance $> \beta * i$
      Return: Connection is anomalous due to chaff
6. Return: Connection is not anomalous.

Procedure Edit_Distance (A[$1,\ldots,m$],B[$1,\ldots,n$]):

1. Initialize matrix[$i,0$] = i for $i = 0,\ldots,m$
2. Initialize matrix[$0,j$] = j for $j = 0,\ldots,n$
3. For $i = 1,\ldots,m$
   For $j = 1,\ldots,n$
      set $x = 0$ if |A[i] - B[j]| $< \alpha_{ED}$ (set to 50 msec) else set $x = 2$
      matrix[i,j]= min(matrix[$i-1,j-1$]+x, matrix[$i-1,j$]+1,matrix[$i,j-1$] +1)
4. Return: matrix[m,n].

3.3 Causality Based Anomaly Detection

The causality based algorithm is used to detect the presence of chaff in interactive connections. Given streaming sequences of packets along the two directions C_{12} and C_{21} of a connection, we use the methodology described before to split the packets into ON and OFF periods taking T_{idle} as 100 msec. For interactive traffic, it can be expected that a user types a command, waits for its output and then types another command. This typing behaviour gives rise to a pattern of ON periods with the following property (also shown in Figure 2): for every pair of consecutive ON periods on C_{12} there will be exactly one ON period on C_{21}. Similarly, for every pair of consecutive ON periods on C_{21} there will be exactly one ON period on C_{12}.

An ON period on C_{12} is defined to be anomalous if there is either zero or more than one ON periods on C_{21} before the next ON period on C_{12}. Let $\gamma_{forward}$ be defined as the ratio of the anomalous ON periods to the total ON periods on C_{12}. Similarly $\gamma_{reverse}$ is defined for C_{21}. Normal interactive connections will have low values for both $\gamma_{forward}$ and $\gamma_{reverse}$. If either of these correlation metrics has a value greater than 0.67 then the connection is flagged as anomalous due to chaff. The pseudo code below explains the algorithm in greater detail.

Pseudocode for causality based anomaly detection algorithm:

1. Initialize ON_Packets_Forward = 0, ON_Packets_Reverse = 0
2. Initialize Anom_Packets_Forward = 0, Anom_Packets_Reverse = 0
3. Split the packets on the forward direction C_{12} and reverse direction C_{21} of an interactive connection into ON and OFF periods using T_{idle}
4. For every packet sent at ON period on C_{12}
   Increment count for ON_Packets_Forward
   If number of ON periods on C_{21} before next ON period on C_{12} is $= 0$ or > 1
   Return: Packet is anomalous
   Increment count for Anom_Packets_Forward
   If Procedure Check_for_Anomaly returns yes
   Return: Connection is anomalous due to chaff
5. For every packet sent at ON period on C_{21}
   Increment count for ON_Packets_Reverse
   If number of ON periods on C_{12} before next ON period on C_{21} is $= 0$ or > 1
   Return: Packet is anomalous
   Increment count for Anom_Packets_Reverse
   If Procedure Check_for_Anomaly returns yes
   Return: Connection is anomalous due to chaff
6. Return: Connection is not anomalous

Procedure Check_for_anomaly:

1. If number of ON_Packets $>$ MIN_ON_PACKETS (set to 25)
   If Anomalous_Packets / ON_Packets $>= \gamma_{forward}$ Return: yes
   If Anomalous_Packets / ON_Packets $>= \gamma_{reverse}$ Return: yes
2. Return: no

This algorithm may suffer from false positives for interactive connections used for bulk file transfer or for commands with extremely large outputs because they involve a large number of packets sent in only one direction. However, for most such cases the stream of packets would constitute only a single ON period and hence would not affect the overall outcome of the algorithm.

The algorithm may also suffer from false negatives if the attacker injects chaff at a rate greater than T_{idle}. The attacker uses a custom server to inject chaff and this can be done very fast. In order to counter this and reduce false negatives, the value of T_{idle} is chosen to be much smaller than that considered for the other algorithms. We tested the algorithm by varying values of T_{idle} from 0 to 500 and did not observe significant changes in results and set T_{idle} as 100 msec for our experiments. The algorithm may also suffer from false negatives if the attacker injects chaff alternately in both directions of the connection.

While a sufficiently aggressive attacker can defeat any of the proposed algorithms, there is always utility in raising the bar for the attacker. The attacker traceback methology described in Section 4 consisting of the stepping stone detection algorithm and the anomaly detection algorithms is extremely difficult to evade.

4 Attacker Traceback Methodology

The timing based stepping stone detection algorithm and the three anomaly detection techniques can be efficiently combined to form an integrated methodology for detecting the source of an intrusion and tracing back to the attacker, as follows. If the attacker uses a chain of intermediate nodes for malicious activity then this methodology consists of iterating the combination of the timing based stepping stone

detection algorithm and the three anomaly detection techniques. Each execution of this combination helps detecting a new stepping stone even in the presence of active traffic perturbation like jitter and chaff, and adds a new node on the path from the target to the attacker, until tracing back to the attacker is completed. In this process, any attempts by the attacker to evade detection using jitter or chaff will cause the traffic to appear anomalous and the anomaly detection algorithms will flag the connections as anomalous.

Consider again the scenario in Figure 1 where the attacker uses jitter/chaff in one of the connections to evade detection. The attacker uses a chain of N stepping stones and injects jitter/chaff at stepping stone 2. Suppose that an intrusion is detected on stepping stone N and we want to trace back the attack. We have written an API for the detection framework that will enable us to do the same (for N= 4). We describe two of the APIs that will assist us in demonstrating how the attack can be traced back to node 1 (and can be similarly traced down to the attacker node).

The analyzer-report-stepping-stones API is used to report stepping stones detected by the framework. Given a host name or an IP address ‘s’ we can iterate through the output of this API to get the downstream hosts in the chain corresponding to ‘s’. The analyzer-report-anomalies API is used to report jitter/chaff based anomalies detected by the framework. Given a host name or IP address ‘s’ we can scan the output of this API to get all the anomalous connections ‘s’ was part of.

Let us now analyze how the framework will trace back the attacker for this connection chain. Firstly the analyzer-report-stepping-stones(4) API will indicate that node 4 is part of the stepping stone chain 2-3-4. So we have traced back the attack till 3. Now the analyzer-report-anomalies(2) API will indicate that node 2 is part of an anomalous interactive connection 1-2. Hence, we have traced back the attack to node 1. Proceeding in this manner we can trace the intrusion all the way back to the attacker using these two APIs.

Our algorithms make a number of assumptions that are more appropriate for Intranets and thus our overall solutions achieve a much higher level of practicality when facing active stepping stone attacks from network insiders. Specifically, a first assumption is that the associated software implementing our solutions is actually deployed on all nodes that have the potential of later becoming stepping stones. While this can be realized within an Intranet at a reasonable cost, it would be require major architecture and protocol re-design and re-organization in wider contests like the Internet. A second assumption is that all such nodes, in the presence of traffic from a potential attacker, need to necessarily interact and honestly cooperate in order to successfully complete the three anomaly detection algorithms. Here, this assumption is much more likely to be satisfied in a network of nodes belonging to the same organization, as opposed to nodes belonging to mutually distrusting organizations. Finally, we note a third assumption on the adversary model, in that we assume that the active attacks, such as in the above jitter mode, chaff mode and jitter+chaff mode, have a noticeable impact on the exchanged network traffic. Even in this case, this assumption is much more likely to hold in a network that is easy to map for an insider, as opposed to a less explored network like the Internet.

5 Evaluation

Our evaluation experiments were performed on Deter test bed, using two network topologies (as described in Section 5.1). In order to test the anomaly detection algorithms we wrote a custom server to inject jitter and chaff in connections (this is described in Section 5.2). The detection algorithms were implemented in Bro language (discussed in Section) and evaluated against real-world traces from Telcordia Technologies and University of Texas Computer Sciences department (this is described in Sections 5.1). The algorithms are online and were also tested on live traffic on Deter.

5.1 Experimental Setup

We used Deter test bed [28] for evaluating the algorithms. Figure 3 shows the two network topologies used for our experiments. The nodes in these topologies had Linux 2.4.20 as the operating system with ssh, telnet, tcpdump, bro and netcat installed.

We used traces from Telcordia Technologies and University of Texas Computer Sciences department to evaluate the algorithms. The interactive traffic (SSH and Telnet connections) was extracted from these traces. Topology 1 was used to evaluate the algorithms on live traffic. Topology 2 was used to generate traces with jitter and chaff using the custom netcat server. We installed the custom server on nodeB and nodeC and ran it in jitter, chaff and jitter+chaff modes.

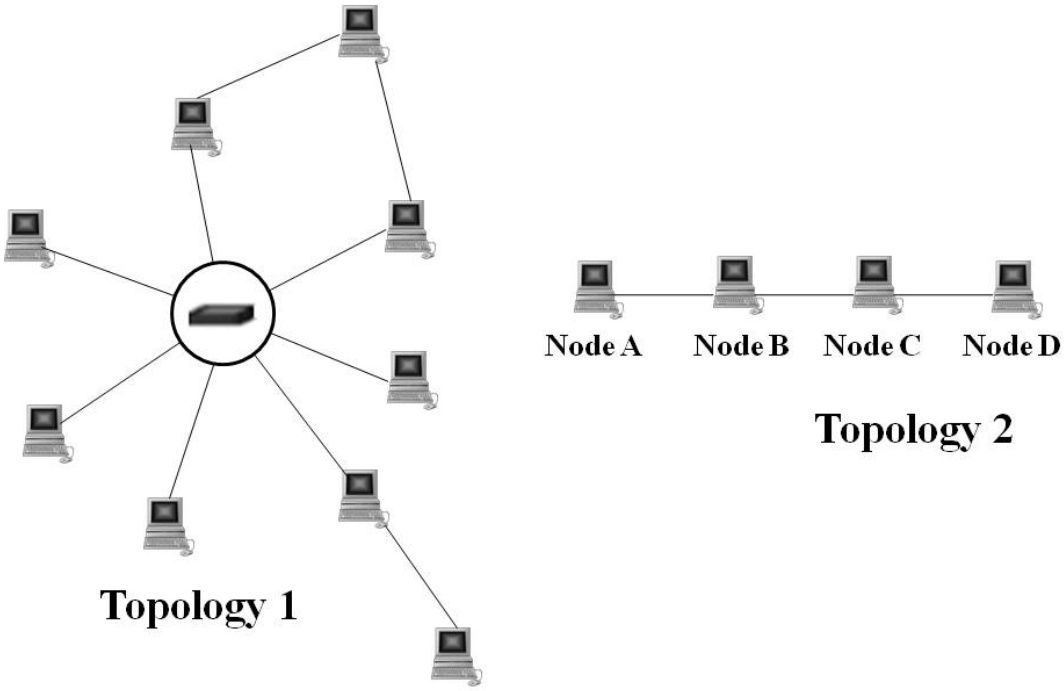

Figure 3: Deter network topologies

5.2 Custom Server Implementation

In order to evaluate the anomaly detection algorithms, we first needed to evade the timing-based stepping stone detection algorithm from [2]. Towards this goal, we implemented a custom server that injects jitter and/or chaff during communication between two nodes. We have leveraged Netcat, an open source tool, for our custom server implementation.

Injecting Jitter. One can inject jitter in customized netcat using the *–i* option. A random delay is introduced depending upon the lower and upper bound specified.

Below we summarize the pseudo code for injecting jitter. The syntax for this option is: $-i$ lower-bound:upper-bound where lower and upper bounds are specified in msec.

Pseudocode for injecting jitter:

1. For every request received from slave
   If JITTER_OPTION is enabled
   Randomly select a number *r* between lower and upper bound
   Inject a delay of *r* msec
   Send request to master
2. For every response received from master
   If JITTER_OPTION is enabled
   Randomly select a number *r* between lower and upper bound
   Inject a delay of *r* msec
   Send response to slave

A typical netcat command using this option would be: netcat $-L$ nodeb:23 $-p$ 9000 –*i* 80:100 which indicates that netcat has been setup in tunneling mode to tunnel all data received at port 9000 to port 23 on the host nodeb and in doing so introduce a random delay between 80 and 100 msec.

Injecting Chaff. One can inject chaff in netcat using the –*C* option. Chaff can be introduced in either direction at random intervals depending upon the lower and upper bound specified. We summarize below the pseudo code for injecting chaff.

Pseudocode for injecting chaff:

1. If CHAFF_OPTION is enabled
   Randomly select a number *r* between lower and upper bound
   If no data received from either slave or master for time *r*
   If both f and *r* options are specified
   randomly select the direction to send chaff
   else select the specified direction
   send chaff in the selected direction
   Else do data processing
2. For every request received from slave
   If CHAFF_OPTION is enabled and data is chaff
   Do not send chaff
   Else send request to master
3. For every response received from master
   If CHAFF_OPTION is enabled and data is chaff
   Do not send chaff
   Else send response to slave

The syntax for this option is: $-C$ lower-bound:upper-bound:fr where lower and upper bounds are specified in msec, f indicates chaff will be sent in forward direction and *r* indicates chaff will be sent in reverse direction. At least one of f and *r* options should be specified. A typical netcat command using this option would be: netcat $-L$ nodeb:23 $-p$ 9000 –*C* 800:1000:fr which indicates that netcat has been setup in tunneling mode to tunnel all data at port 9000 to port 23 on nodeb and in doing so introduce chaff at random intervals between 800 and 1000 msec in both forward and reverse directions.

5.3 Bro

The anomaly detection algorithms were implemented in Bro [4] language. Bro is an open-source, Unix-based Network Intrusion Detection System (NIDS) that passively monitors network traffic and looks for suspicious activity. The algorithms were written as policies in Bro. Each policy script can implement

Trace Details	Total number of Connections	Anomalies Detected	False Positives	False Negatives
Connections with jitter	10	10	0	0
Connections with jitter and chaff	6	6	0	0
Real-world traces – normal connections	4450	2	2	0

Figure 4: Evaluation of response-time based algorithm on real-world and custom server traces

Trace Details	Total number of Connections	Anomalies Detected	False Positives	False Negatives
Connections with jitter	250	200	0	50
Connections with jitter and chaff	200	200	0	0
Connections with no jitter	250	0	0	0

Figure 5: Evaluation of response-time based algorithm on live traffic on Deter

event handlers that are invoked when certain events are fired by the event engine. The tcp-packet event was used by our policy scripts for packet level analysis of network traffic. Various filters can be defined to limit the amount of packet processing; in our case we only monitored interactive traffic like telnet/ssh.

5.4 Response-time based algorithm evaluation

The response-time based algorithm is used to detect jitter anomalies in interactive traffic. We evaluate the algorithm in terms of its ability to correctly identify all instances of jitter – low false negatives, and not wrongly identify connections as anomalous that have no jitter – low false positives. The evaluation of the algorithm on traces is presented in Figure 4. In traces generated using the custom server running in jitter mode and jitter+chaff mode, the algorithm detected all jitter instances correctly with no false negatives. In real-world traces, the algorithm had 2 instances of false positives.

The evaluation of the algorithm on live traffic is presented in Figure 5. The algorithm did not have any false positives for connections with no jitter. The custom server was used to inject varying amounts of jitter from 20 msec to 320 msec. For each of these different jitter values 50 connections were established. The algorithm had 50 false negatives for jitter values of 20 msec because this value is less than δ_{RT}. However all these connections were identified as part of a connection chain by the stepping stone detection algorithm. The algorithm had no false negatives for connections with a combination of jitter and chaff. We define percentage of success of an algorithm as the ratio of the total number of connections with no false positives/negatives to the total number of connections. Using this criterion the overall success rate of the response-time based algorithm is 98.99%.

5.5 Edit-distance based algorithm evaluation

The edit-distance based algorithm is used to detect chaff anomalies in interactive traffic. We evaluate the algorithm in terms of its ability to correctly identify all instances of chaff – low false negatives, and not wrongly identify connections as anomalous that have no chaff – low false positives. The evaluation of the algorithm on traces is presented in Figure 6. In traces generated using the custom server running in chaff mode and jitter+chaff mode, the algorithm detected all chaff instances correctly with no false negatives. In real-world traces, the algorithm had 62 instances of false positives.

Trace Details	Total number of Connections	Anomalies Detected	False Positives	False Negatives
Connections with jitter	17	17	0	0
Connections with jitter and chaff	6	6	0	0
Real-world traces – normal connections	4450	62	62	0

Figure 6: Evaluation of edit-distance based algorithm on real-world and custom server traces

Trace Details: Connection Features	Total number of Connections	Anomalies Detected	False Positives	False Negatives
Different values of chaff	200	200	0	0
Different values of jitter and chaff	250	250	0	0
No chaff	300	0	0	0

Figure 7: Evaluation of edit-distance based algorithm on live traffic on Deter

The evaluation of the algorithm on live traffic is presented in Figure 7. The custom server was used to inject chaff at varying intervals ranging from 100 to 800 msec. The algorithm was able to detect all anomalies in connections with chaff and a combination of jitter and chaff. Also, no false positives were detected for connections with no chaff. Using the criterion mentioned above, the overall success rate of the edit-distance based algorithm is 98.81%.

5.6 Causality based algorithm evaluation

The causality based algorithm is used to detect chaff anomalies in interactive traffic. As with the edit-distance based algorithm, low false positives/negatives are used to evaluate the algorithm. The evaluation of the algorithm on traces is presented in Figure 8. In traces generated using the custom server running in chaff mode and jitter+chaff mode, the algorithm detected all chaff instances correctly with no false negatives. In real-world traces, the algorithm had 49 instances of false positives.

The evaluation of the algorithm on live traffic is presented in Figure 9. The algorithm did not have any false positives for connections with no chaff. The custom server was used to inject chaff at varying intervals ranging from 100 to 800 msec. All instances of anomalies were detected in connections with chaff and a combination of jitter and chaff. Using the criterion mentioned above, the overall success rate of the causality based algorithm is 99.06%.

Trace Details	Total number of Connections	Anomalies Detected	False Positives	False Negatives
Connections with chaff	17	17	0	0
Connections with jitter and chaff	6	6	0	0
Real-world traces – normal connections	4450	49	49	0

Figure 8: Evaluation of causality based algorithm on real-world and custom server traces

Trace Details: Connection Features	Total number of Connections	Anomalies Detected	False Positives	False Negatives
Different values of chaff	200	200	0	0
Different values of jitter and chaff	250	250	0	0
No chaff	300	0	0	0

Figure 9: Evaluation of edit-distance based algorithm on live traffic on Deter

6 Conclusions

We have successfully designed and implemented algorithms that can detect the presence of jitter and chaff in interactive traffic. This strengthens the existing timing-based stepping stone detection algorithms that can be evaded by an aggressive attacker using jitter and chaff. The anomaly detection algorithms coupled with the stepping stone detection algorithm provide an integrated framework that is robust and difficult to evade, especially for insiders of networks like an Intranet, where certain assumptions made on software deployment, node cooperation and adversary model, are much more likely (in fact, expected) to hold. All the three algorithms have very low false positives/negatives and a high success percentage of about 99%.

Although we designed these algorithms to strengthen existing stepping stone detection algorithms in the presence of active insider attacks, the techniques are general in that they can be used to detect the presence of jitter and chaff injected by external attackers in any interactive connection within the general Internet. Any intrusion detection system that uses timing of packets to correlate interactive connections can use the techniques described here to detect anomalous activity.

The attacker can evade detection by installing the custom server on any random port because the Bro tcp filter will only monitor interactive traffic on known ports like 22/23. We have performed some work on identifying anomalous activity on any port. Bro provides a policy script interconn.bro that can identify interactive connections. The anomaly detection scripts monitor all connections flagged as interactive by this script. As a result, the anomaly detection algorithms can detect any anomalous interactive traffic.

The existing algorithms use magic numbers for various parameters critical to the success of the algorithm. These parameters have been defined by comprehensive testing on real-world traces and live traffic on Deter. Dynamically determining the values of these parameters is a possible avenue for future work and may help in further reducing the number of false positives/negatives.

References

[1] S. Staniford-Chen and L. T. Heberlein, “Holding intruders accountable on the internet,” in *Proc. of 2005 IEEE Symposium on Security and Privacy, Oakland, California, USA*. IEEE, May 2005, pp. 39–49.

[2] Y. Zhang and V. Paxson, “Detecting stepping stones,” in *Proc. of the 9th USENIX Security Symposium, Denver, Colorado, USA*, August 2000, pp. 171–184.

[3] V. Paxson, “Bro: a system for detecting network intruders in real-time,” *Computer Networks*, vol. 31, no. 23-24, pp. 2435–2463, 1999.

[4] “Bro intrusion detection system,” http://www.bro-ids.org/, accessed Mar 2011.

[5] X. Wang, D. S. Reeves, S. F. Wu, and J. Yuill, “Sleepy watermark tracing: An active network-based intrusion response framework,” in *Proc. of IFIP TC11 16th International Conference on Information Security (IFIP/Sec’01), Paris, France*. Kluwer, June 2001, pp. 369–384.

[6] K. Yoda and H. Etoh, "Finding a connection chain for tracing intruders," in *Proc. of the 6th European Symposium on Research in Computer Security (ESORICS'00), Toulouse, France, LNCS*, vol. 1895. Springer-Verlag, October 2000, pp. 191–205.

[7] X. Wang, D. S. Reeves, and S. F. Wu, "Inter-packet delay based correlation for tracing encrypted connections through stepping stones," in *Proc. of the 7th European Symposium on Research in Computer Security, Zurich, Switzerland (ESORICS'02), LNCS*, vol. 2502. Springer-Verlag, October 2002, pp. 244–263.

[8] W. T. Strayer, C. E. Jones, I. Castineyra, J. B. Levin, and R. R. Hain, "An integrated architecture for attack attribution," in *BBN Technical Report No. 8834*, 2003.

[9] D. L. Donoho, A. G. Flesia, U. Shankar, V. Paxson, J. Coit, and S. Staniford, "Multiscale stepping-stone detection: Detecting pairs of jittered interactive streams by exploiting maximum tolerable delay," in *Proc. of the 5th International Symposium on Recent Advances on Intrusion Detection (RAID'02), Zurich, Switzerland, LNCS*, vol. 2516, October 2002, pp. 17–35.

[10] X. Wang and D. S. Reeves, "Robust correlation of encrypted attack traffic through stepping stones by manipulation of interpacket delays," in *Proc. of the 10th ACM Conference on Computer and Communications Security (ACM CCS'03), Washington, DC, USA*. ACM Press, October 2003, pp. 20–29.

[11] A. Blum, D. Song, and S. Venkataraman, "Detection of interactive stepping stones: Algorithms and confidence bounds," in *Proc. of the 7th International Symposium on Recent Advances on Intrusion Detection (RAID'04), Sophia Antipolis, France, LNCS*, vol. 3224. Springer-Verlag, September 2004, pp. 258–277.

[12] A. Belenky and N. Ansari, "On ip traceback," *IEEE Communications Magazine*, vol. 41, no. 7, pp. 142–153, 2003.

[13] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys*, vol. 41, no. 3, Article No. 15, 2009.

[14] M. V. Mahoney and P. K. Chan, "Learning nonstationary models of normal network traffic for detecting novel attacks," in *Proc. of the 8th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'02), Edmonton, Alberta, Canada*. ACM Press, July 2002, pp. 376–385.

[15] W. W. Streilein, D. J. Fried, and R. K. Cunningham, "Detecting flood-based denial-of-service attacks with snmp/rmon," in *Proc. of 2003 Workshop on Statistical and Machine Learning Techniques in Computer Intrusion Detection, Fairfax, Virginia, USA*, September 2003.

[16] J. B. D. Cabrera, L. M. Lewis, X. Qin, C. Gutiérrez, W. Lee, and R. K. Mehra, "Proactive intrusion detection and snmp-based security management: New experiments and validation," in *Proc. of IFIP/IEEE the 8th International Symposium on Integrated Network Management (IM'03), Colorado, USA*. IEEE, March 2003, pp. 93–96.

[17] L. P. Gaspary, E. Meneghetti, and L. M. R. Tarouco, "An snmp agent for stateful intrusion inspection," in *Proc. of IFIP/IEEE the 8th International Symposium on Integrated Network Management (IM'03), Colorado, USA*. IEEE, March 2003, pp. 3–16.

[18] R. S. Puttini, J. Percher, L. Mé, and R. T. de Sousa Jr., "A fully distributed ids for manet," in *Proc. of the 9th IEEE Symposium on Computers and Communications (ISCC'04), Alexandria, Egypt*. IEEE, June-July 2004, pp. 331–338.

[19] H. L. J. Yu, M. Kim, and D. Park, "Traffic flooding attack detection with snmp mib using svm," *Computer Communications*, vol. 31, no. 17, pp. 4212–4219, 2008.

[20] W. Ehrlich, A. Karasaridis, D. Liu, and D. Hoeflin, "Detection of spam hosts and spam bots using network flow traffic modeling," in *Proc. of the 3rd USENIX Workshop on large-scale exploits and emergent threats (LEET'10), San Jose, CA, USA*, April 2010, pp. 7–7.

[21] J. Yang and S. H. S. Huang, "A real-time algorithm to detect long connection chains of interactive terminal sessions," in *Proc. of the 3rd international conference on Information security (InfoSecu'04), Shanghai, China*, November 2004, pp. 198–203.

[22] J. Yang and S. H. S. Huang, "Matching tcp packets and its application to the detection of long connection chains on the internet," in *Proc. of International Conference on Advanced Information Networking and Applications (AINA'05), Taipei, Taiwan*. IEEE, March 2005, pp. 1005–1010.

[23] K. H. Yung, "Detecting long connection chains of interactive terminal sessions," in *Proc. of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID'02), Zurich, Switzerland, LNCS*, vol.

2516. Springer-Verlag, October 2002, pp. 1–16.

[24] A. Kampasi, Y. Zhang, G. D. Crescenzo, A. Ghosh, and R. Talpade, "Improving stepping stone detection algorithms using anomaly detection techniques," in *Univ. of Texas at Austin Technical Report No. 28*, 2007.

[25] V. Paxson and S. Floyd, "Wide area traffic: the failure of poisson modeling," *IEEE/ACM Trans. Netw.*, vol. 3, no. 3, pp. 226–244, 1995.

[26] "The gnu netcat project," http://netcat.sourceforge.net/, accessed Mar 2011.

[27] V. Jacobson and M. Karels, "Congestion avoidance and control," *ACM SIGCOMM Computer Communication Review*, vol. 25, no. 1, pp. 157–187, 1988.

[28] "Deter - network security testbed based on emulab," https://www.isi.deterlab.net/, accessed Mar 2011.

Giovanni Di Crescenzo is a senior scientist at Telcordia Technologies with 15+ year research experience. He received a Ph.D. in Computer Science from University of California San Diego, USA, and a Ph.D. in Applied Mathematics from University of Naples, Italy. His main research activity has been in cryptography, computer/network/information security, computational complexity and algorithms, where he has been involved in more than 15 research projects funded by government agencies and has produced 100+ scientific publications in refereed conferences and journals, including 1 book, 1 book chapter, and 3 proceedings of conferences or workshops for which he was a technical program chair. He has received 10 awards or patents, has given more than 20 invited talks, and regularly referees papers for major conferences and journals in his areas of expertise. Giovanni has also been extensively involved in technical consulting in various areas within cryptography, security, and intellectual property for several information technology customers.

Abhrajit Ghosh is Director of the Autonomic Systems Research group at Telcordia. He has extensive R&D experience in the area of cyber security including network intrusion detection. He has worked on several programs relating to various aspects of network security, including IARPA's Network Attack Traceback and P2INGS programs. He currently leads a team developing a network security monitoring solution for a Tier-1 commercial ISP.

Abhinay Kampasi graduated with a Bachelor's degree in Computer Engineering from Pune Institute of Computer Technology, University of Pune in 2003. He then worked for a software company, Persistent Systems for two years. Abhinay graduated with a Master's degree in Computer Science from University of Texas at Austin in 2007 during which he performed research in network traffic analysis under Prof. Yin Zhang. He is currently working on server related technologies at Microsoft Corporation.

Rajesh Talpade has 13+ years of experience in new product and market development, innovation, sales, and consulting in IP network, security and wireless areas. He is currently Executive Director for Product Management at NIKSUN. Prior to NIKSUN, Rajesh was at Telcordia Technologies where he most recently led all phases of transition of innovative technology to deployed product in new market segment, growing business from concept to six customers in 2 years. While at Telcordia, he won over $10M in US Government R&D funds, several in highly competitive settings with under 5% win-rate. Rajesh has also been CTO of a wireless security start-up founding team that secured $2M in Series A funds from Venture Capital and private investors. Rajesh has received 3 patents, 10

patents-pending, 20+ published papers, 2 guest-editorships, 1 book-chapter and 1 IETF RFC. He was awarded the PhD in Computer Science by Georgia Institute of Technology, and an MBA by Columbia Business School.

Yin Zhang is an Associate Professor in the Department of Computer Science at the University of Texas at Austin. His research interests span several areas of computer networks, including network measurement, network management, and network security. Dr. Zhang received his Ph.D. in Computer Science from Cornell University in 2001. Between 2001-2004, he was a member of the Networking Research Department at AT&T Labs – Research. He received NSF CAREER Award in 2006 and is a Senior Member of IEEE.