Improving Stepping Stone Detection Algorithms using Anomaly Detection Techniques

Abhinay Kampasi
Department of Computer Sciences
The University of Texas at Austin
Austin, TX, USA
abhinay@cs.utexas.edu

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

Giovanni Di Crescenzo
Telcordia Technologies, Inc.
Piscataway, NJ, USA
giovanni@research.telcordia.com

Abhrajit Ghosh
Telcordia Technologies, Inc.
Piscataway, NJ, USA
aghosh@research.telcordia.com

Rajesh Talpade
Telcordia Technologies, Inc.
Piscataway, NJ, USA
rrt@research.telcordia.com

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 algorithms have been proposed to detect stepping stones, timing correlation based algorithms being one of them. 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. 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.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General - Security and protection

General Terms

Algorithms, Security

Keywords

Anomaly Detection, Stepping Stones, Intrusion Detection, Evasion, Jitter, Chaff

1. INTRODUCTION

Network attackers frequently use a chain of compromised intermediate nodes called stepping stones to attack a target machine because it helps them to maintain anonymity. This is a big concern for intrusion detection systems because 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. Over the last few years many stepping stone detection algorithms have been proposed. Some of the first algorithms were content-based [3] and created thumbprints of streams and compared them looking for good matches. Other techniques rely on the content not changing significantly between different streams. 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 is that much of the interactive traffic today is encrypted and hence content comparison is not possible. Other approaches have looked at comparing the number of packets between connections [1]. Considering the fact that most interactive traffic is encrypted and packet payload is not always accessible, one of the most successful techniques for stepping stone detection has been timing-based that tries to correlate timing of packets across different connections. The algorithm proposed by [8] splits connections into ON and OFF periods and then correlates the end of OFF periods. All references to a stepping stone detection algorithm in the rest of the paper refer to [8] but are generally applicable to any timing-based stepping stone detection algorithm.

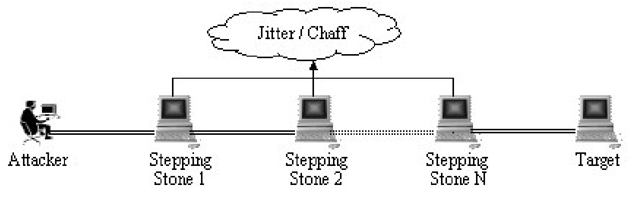

Figure 1: Stepping stone chain between attacker and target

The stepping stone chain between the attacker and target is shown in Figure 1. The attacker can evade the existing stepping stone detection algorithm if he injects sufficient amount of timing jitter or chaff packets at any of the intermediate stepping stones. We have leveraged *Netcat* [11] 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.

We have developed three anomaly detection algorithms to detect the presence of jitter and chaff in connections by correlating traffic in both directions of a connection. As far as we know this is the first approach to correlate traffic in both directions of an interactive connection to detect jitter and chaff anomalies. 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 used to detect chaff. If the attacker tries to obfuscate the traffic using chaff then this causality relation is broken resulting in the network traffic appearing anomalous.

We performed experiments on Deter test bed. Real-world 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* [2, 9] 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. These algorithms coupled with the stepping stone detection algorithm form an extremely robust detection framework that the attacker will find difficult to evade.

The main contributions of this paper are summarized below:

(1) The response-time based algorithm is able to detect jitter anomalies in interactive traffic

(2) The edit-distance and causality based algorithms are able to detect chaff anomalies in interactive traffic

(3) The timing based stepping stone detection algorithm combined with the three anomaly detection algorithms forms an integrated framework that is very difficult to evade

The rest of the paper is organized as follows. Section 2 covers the related work and Section 3 discusses how an attacker can evade the existing stepping stone algorithm. The anomaly detection algorithms are described in Section 4 and the custom server implementation is explained in Section 5. Section 6 outlines the new integrated stepping stone detection framework. We evaluate the algorithms in Section 7 before concluding in Section 8.

2. RELATED WORK

Over the past few years, many algorithms have been proposed for detecting stepping stones. Content-based algorithms [3] compare content over different streams looking for a high degree of correlation. The content-based techniques have many limitations like high computation costs, restricted access to packet payload and encrypted traffic. Much of the stepping stone detection research today focuses on timing based correlation.

The stepping stone detection algorithm proposed by [1] is based on monitoring the number of packets between connections. This paper uses computational learning theory and analysis of random walks to provide provable upper bounds on the number of packets one needs to observe to confidently detect a stepping stone. This paper also gives bounds on the amount of chaff that the attacker would need to inject in order to evade detection without delving into the issue of actually detecting chaff.

There have been some anomaly detection techniques [4, 5, 6] 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 with 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.

3. ALGORITHM EVASION

The ON-OFF timing based stepping stone algorithm can be evaded by injecting jitter or chaff in the connection chain.

3.1 Stepping Stone Detection Algorithm

The stepping stone algorithm 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 2.

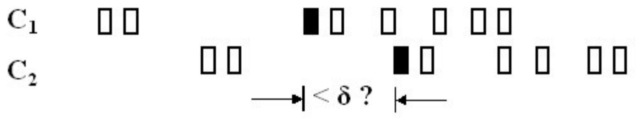

Figure 2: Correlation of packets for connections C1 and C2 based on timing

Let OFF_1 and OFF_2 denote the total number of OFF periods within two connections C_1 and C_2 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 $(OFF_{1,2}) / \min(OFF_1, OFF_2) > \gamma$ (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. A drawback of the algorithm is its inability to deal with jitter and chaff. 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.

3.2 Evasion using jitter

If the attacker injects timing jitter or delay of more than δ 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.

3.3 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.

4. ANOMALY DETECTION

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 algorithm together with the anomaly detection techniques forms a robust framework 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).

4.1 Response-time Based Anomaly Detection

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 + δ_{RT}) then we mark the ON period as anomalous as shown in Figure 3. The round trip time (RTT) is calculated using a smoothed version of Jacobson/Karel's algorithm. The value of δ_{RT} was selected as 50 msec after analyzing the typical response times of servers in many real-world traces.

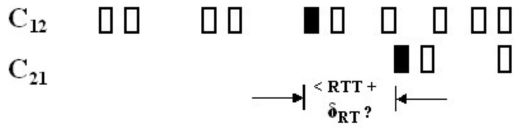

Figure 3: Response-time based anomaly detection algorithm

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 given in Figure 4.

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 $\delta_{RT.}$ 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.

```
Initialize ON_Packets = 0, Anomalous_Packets = 0
Split the packets on the forward direction $C_{12}$ of an interactive connection into ON and OFF periods using $T_{idle}$
For every ack sent on $C_{21}$ for a data packet sent on $C_{12}$
    Update RTT using Jacobson/Karel's algorithm
End
For every packet sent at ON period on $C_{12}$
   Increment count for ON_Packets
   If response packet on $C_{21}$ is sent within (RTT + $\delta_{RT}$)
      Packet is not anomalous
   Else
      Packet is anomalous
      Increment count for Anomalous_Packets
      Run procedure Check_for_Anomaly
   End
End
Procedure Check_for_Anomaly
   If number of ON_Packets > MIN_ON_PACKETS (set to 10)
      If Anomalous_Packets / ON_Packets >= $\gamma_{RT}$
         Connection is anomalous due to jitter
      End
   End
End
```

Figure 4: Pseudo code for response-time based anomaly detection algorithm

4.2 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 in Figure 5 explains the algorithm in greater detail.

```
Initialize Cumulative_Edit_Distance = 0
Split the packets on two directions $C_{12}$ and $C_{21}$ of an interactive connection into ON and OFF periods using $T_{idle}$
Let $C_{12}$_seq[ i ] be the sequence of intervals for subsequence i for forward direction of traffic
Let $C_{21}$_seq[ i ] be the sequence of intervals for subsequence i for reverse direction of traffic
For subsequences of length SEQ_LENGTH (set to 10) for $C_{12}$
    Let i be the current subsequence number
    Run procedure Edit_Distance ($C_{12}$_seq[ i ], $C_{21}$_seq[ i ])
    Let local_edit_dist[ i ] be the edit distance returned
    Cumulative_Edit_Distance += local_edit_dist[ i ]
    If Cumulative_Edit_Distance > $\beta$ * i
        Connection is anomalous due to chaff
    End
End
Procedure Edit_Distance (A[1..m], B[1..n])
    Initialize matrix[i, 0] = i  for i = 0 to m
    Initialize matrix[0, j] = j  for j = 0 to n
    For i = 1 to m
        For j = 1 to n
            x = 0 if |A[ i ] - B[ j ]| < $\alpha_{ED}$ (set to 50 msec) else x = 2
            matrix[i, j] = min {matrix[i-1, j-1] + x, matrix[i-1, j] + 1,
            matrix[i, j-1] + 1}
        End
    End
    Return matrix[m, n]
End
```

Figure 5: Pseudo code for edit-distance based anomaly detection algorithm

4.3 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 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 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 such that for every pair of consecutive ON periods on C_{12} there will be exactly one ON period on C_{21} as shown in Figure 6. Similarly, for every pair of consecutive ON periods on C_{21} there will be exactly one ON period on C_{12}.

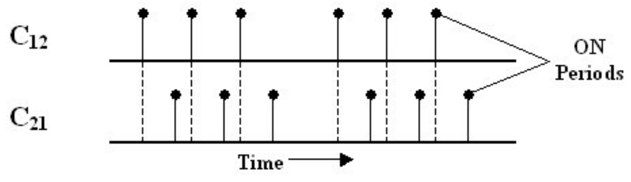

Figure 6: Exactly one ON period on C_{21} between two consecutive ON periods on C_{12} and vice versa

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 in Figure 7 explains the algorithm in greater detail.

```
Initialize ON_Packets_Forward = 0, ON_Packets_Reverse = 0, Anom_Packets_Forward = 0, Anom_Packets_Reverse = 0
Split the packets on the forward direction $C_{12}$ and reverse direction $C_{21}$ of an interactive connection into ON and OFF periods using $T_{idle}$
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 the next ON period on $C_{12}$ = 0 or > 1
        Packet is anomalous
        Increment count for Anom_Packets_Forward
        Run Procedure Check_for_Anomaly
    End
End
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 the next ON period on $C_{21}$ = 0 or > 1
        Packet is anomalous
        Increment count for Anom_Packets_Reverse
        Run Procedure Check_for_Anomaly
    End
End
Procedure Check_for_Anomaly
    If number of ON_Packets_Forward > MIN_ON_PACKETS (set to 25)
        If Anom_Packets_Forward/ON_Packets_Forward >= $\gamma_{forward}$
        OR Anom_Packets_Reverse/ON_Packets_Reverse >= $\gamma_{reverse}$
            Connection is anomalous due to chaff
        End
    End
End
```

Figure 7: Pseudo code for causality based anomaly detection algorithm

The 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 "bulk" packets would only constitute a single ON period and hence would not affect the overall outcome of the algorithm.

This algorithm may 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 integrated framework described in Section 6 consisting of the stepping stone detection algorithm and the anomaly detection algorithms is extremely difficult to evade.

5. Custom Server Implementation

In order to evaluate the anomaly detection algorithms, we first needed to evade the existing stepping stone detection algorithm. For this, we implemented a custom server that injects jitter and/or chaff during communication between two nodes. We have leveraged an open source tool *Netcat* for our custom server implementation.

5.1 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. Figure 8 summaries 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.

```
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
    End
    Send request to master
End
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
    End
    Send response to slave
End
```

Figure 8: Pseudo code for injecting jitter

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.

5.2 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. Figure 9 summarizes the pseudo code for injecting chaff.

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 received at port 9000 to port 23 on host nodeb and in doing so inject chaff at random intervals between 800 and 1000 msec in both forward and reverse directions.

```
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
    End
End
For every request received from slave
    If CHAFF_OPTION is enabled and data is chaff
        Do not send chaff
    Else
        Send request to master
    End
End
For every response received from master
    If CHAFF_OPTION is enabled and data is chaff
        Do not send chaff
    Else
        Send response to slave
    End
End
```

Figure 9: Pseudo code for injecting chaff

6. INTEGRATED FRAMEWORK

The timing based stepping stone detection algorithm and the three anomaly detection techniques form an integrated framework for detecting and tracing back to the source of an intrusion that is very difficult to evade. If the attacker uses a chain of intermediate nodes for malicious activity then the stepping stone algorithm will be able to trace back the attack. 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.

Figure 8 shows a scenario where the attacker uses jitter/chaff in one of the connections to evade detection. A-B-C-D form a part of a stepping stone chain used by the attacker. The attacker injects jitter/chaff at node B. Suppose that an intrusion is detected on node D 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. We describe two of the APIs that will assist us in demonstrating how the attack can be traced back to node A (and further downstream if required).

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 all the hosts in the connection chain containing 's'.

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 that 's' was part of.

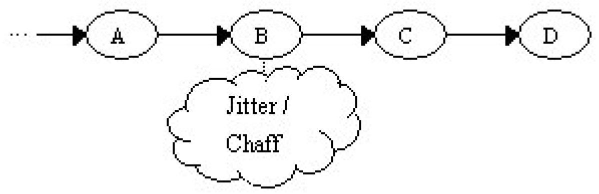

Figure 10: Stepping stone connection chain A-B-C-D with jitter/chaff being introduced at node B

Let us now analyse how the framework will trace back the attacker for this connection chain. Firstly the *analyzer_report_stepping_stones(D)* API will indicate that D is part of the stepping stone chain B-C-D. So we have traced back the attack till node B. Now *analyzer_report_anomalies(B) API* will indicate that B is part of an anomalous interactive connection A-B. Hence, we have traced back the attack to node A. Proceeding in this manner we can trace the intrusion all the way back to the attacker using these two APIs.

7. EVALUATION

All the experiments were performed on Deter test bed. In order to test the algorithms we wrote a custom server to inject jitter and chaff in connections. The algorithms were evaluated against real-world traces from Telcordia Technologies and University of Texas Computer Sciences department. The algorithms are online and were also tested on live traffic on Deter.

7.1 Experimental Setup

We used Deter test bed [10] for evaluating the algorithms. Figure 11 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.

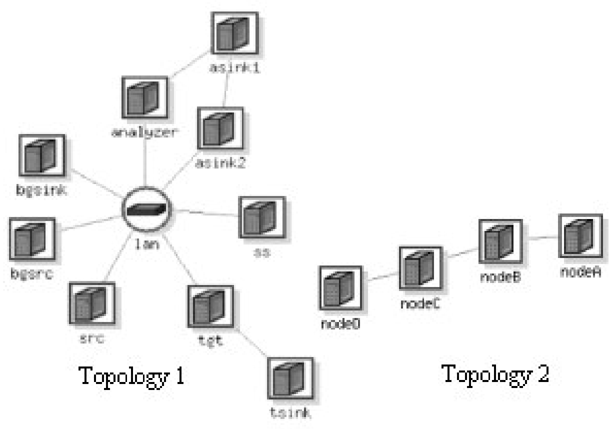

Figure 11: Deter network topologies

7.2 Bro

The algorithms were implemented in *Bro* [9] 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 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.

7.3 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 Table 1. 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.

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

Table 1: Evaluation of response-time based algorithm on real-world and custom server traces

The evaluation of the algorithm on live traffic is presented in Table 2. 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 the 50 connections with 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%***.

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

Table 2: Evaluation of response-time based algorithm on live traffic on Deter

7.4 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 Table 3. 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 chaff	17	17	0	0
Connections with jitter and chaff	6	6	0	0
Real-world traces – normal connections	4450	62	62	0

Table 3: Evaluation of edit-distance based algorithm on real-world and custom server traces

The evaluation of the algorithm on live traffic is presented in Table 4. 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%***.

Trace Details	Total number of Connections	Anomalies Detected	False Positives	False Negatives
Connections with different values of chaff	200	200	0	0
Connections with different values of jitter and chaff	250	250	0	0
Connections with no chaff	300	0	0	0

Table 4: Evaluation of edit-distance based algorithm on live traffic on Deter

7.5 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 Table 5. 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.

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

Table 5: Evaluation of causality based algorithm on real-world and custom server traces

The evaluation of the algorithm on live traffic is presented in Table 6. 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 different values of chaff	200	200	0	0
Connections with different values of jitter and chaff	250	250	0	0
Connections with no chaff	300	0	0	0

Table 6: Evaluation of causality based algorithm on live traffic on Deter

8. CONCLUSION

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. 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, the techniques are general in that they can be used to detect the presence of jitter and chaff in any interactive connection. 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 [7]. The anomaly detection scripts monitor all connections flagged as interactive by this script. As a result, the anomaly detection algorithms can detect anomalous interactive traffic on any port.

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.

9. REFERENCES

[1] Blum, A., Song, D. and Venkataraman, S. *Detection of Interactive Stepping Stones: Algorithms and Confidence Bounds*. International Symposium on Recent Advances in Intrusion Detection (RAID), Lecture Notes in Computer Science, vol. 3224, Springer, Jan 2004, 258-277.

[2] Paxson, V. *Bro: A System for Detecting Network Intruders in Real-Time*, Computer Networks, 31(23-24), 14 Dec. 1999, 2435-2463.

[3] Staniford-Chen, S. and Heberlein, L. T. *Holding intruders accountable on the internet*. Proceedings of IEEE Symposium on Security and Privacy, May 1995, 39-49.

[4] Yang, J. and Huang, S.-H. S. *A real-time algorithm to detect long connection chains of interactive terminal sessions*. InfoSecu '04: Proceedings of the 3rd international conference on Information security. New York, NY, USA: ACM Press, 2004, 198-203.

[5] Yang, J. and Huang, S.-H. S. *Matching tcp packets and its application to the detection of long connection chains on the internet*. AINA 2005: 19th International Conference on Advanced Information Networking and Applications, March 2005, 1005-1010.

[6] Yung, K. H. *Detecting long connection chains of interactive terminal sessions*. RAID 2002, Lecture Notes in Computer Science, vol. 2516, Jan 2002, 1-16.

[7] Zhang, Y. and Paxson, V. *Detecting Backdoors*. 9th USENIX Security Symposium, Denver, Colorado, USA, Aug 2000, 157-170.

[8] Zhang, Y. and Paxson, V. *Detecting stepping stones*. 9th USENIX Security Symposium, Denver, Colorado, USA, Aug 2000, 171-184.

[9] *Bro Intrusion Detection System*. http://www.bro-ids.org/

[10] *Deter - Network Security Testbed based on Emulab*. https://www.isi.deterlab.net/

[11] *The GNU Netcat project*. http://netcat.sourceforge.net/