Stepping Stone Detection Analysis

Steve Bono

1 Introduction

In an effort to conceal her true source, an attacker will often connect through one or more remote machines before launching an attack on an unsuspecting victim. Using tools such as ssh, telnet, or any other remote login application she can chain connections between numerous networks all over the world. This greatly complicates the ability to determine the true source of an attack and can falsely implicate an organization in a computer crime it had no knowledge of.

This motivates the need for a method of determining if a host on a given network is being used for such an attack. A host such as this is referred to as a stepping stone and the series of remote logins utilizing stepping stones is termed a connection chain. Any two hosts in a connection chain is referred to as a stepping stone connection pair. This paper analyzes an approach for detecting stepping stone connection pairs presented by Zhang and Paxson in [10]. By correlating the timing characteristics between inbound and outbound connections on a network it is possible to detect stepping stone connection pairs.

The next section briefly covers alternative methods to the one analyzed here. Section 3 describes the Zhang and Paxson technique in detail including problems it faces and some solutions. Section 4 offers numerous ways an attacker may avoid detection as well as attacks that may be directed at the stepping stone monitor itself. Section 5 describes test scenarios for analyzing these attacks as well as test scenarios for analyzing the performance of this technique in large networks, high speed networks and unstable networks. Finally, section 6 summarizes the conclusions that have been drawn from this analysis.

2 Alternate methods

Detection methods for identifying stepping stone connection pairs come in two flavors, passive and active, each of which has significant downfalls. Passive detection relies primarily on traffic analysis for detection while active detection utilizes real time protocols for tracing back to the source of an actual stepping stone connection chain.

One of the pioneer methods for stepping stone detection was presented in [6] by Staniford and Heberlein that statistically analyzed the content of telnet sessions entering and leaving a network. They located connection pairs by charting the character frequencies occurring in inbound and outbound sessions by time and were then able to map connection pairs together. However, this and all other content based detection
methods are thwarted by the use of encryption when using a remote login application such as SSH.

A method that does not rely on packet contents, but instead on TCP sequence numbers and inter-packet delay is presented by Yoda and Etoh in [11]. This method relies on recordings from multiple sources across the Internet. The deviation between traces is computed and if there is little deviation, the pairs being considered are probable connection pairs.

Donoho et al. in [2] examine the maximum tolerable delay that can be withstood by an attacker in an attempt to un-correlate her traffic between stepping stones in a connection chain. They theorize that as this delay increases, correlations can still be experienced between inbound and outbound traffic. However their results are not justified with an actual implementation and may not feasibly be justified as such due to the complexity requirements as the network increases in size and speed.

Wang, Reeves and Wu present in [8] yet more passive methods for detection. They utilize a number of functions to correlate inter-packet delay times. This involves multiple points of recording and is not affected by unsynchronized clocks at these locations. Therefore, this potentially provides a solution which the on/off timing analysis does not address, which is that there can be multiple points of entry and exit for a network and still accurately detect stepping stone connection pairs.

Active approaches typically involve tracing back to the original source given a connection chain. For a connection chain $H_1 \leftrightarrow H_2 \leftrightarrow \ldots \leftrightarrow H_n$, active approaches attempt to discover the identity of $H_1$ by first identifying $H_{n-1}$ and then by recursively identifying hosts in reverse order to the source. A system proposed by Jung in [3] requires that each host in the connection chain knows each connection chain leading up to it. Another technique proposed by Staniford and Heberlein in [5] involves subverting each previous host in the connection recursively to the source. Another method presented by Wang, Reeves and Wu in [9] injects a type of watermark into the reverse path of a connection chain intended to notify routers and hosts along its path that an intrusion has been detected.

Active approaches fail where passive do not in that they are only reliable as far back as the most reliable host. A host may become unreliable if it has been completely compromised or there are different technologies and policies between unrelated networks a connection chain may traverse. Passive methods fail where active do not in that they are generally only useful after the fact. That is, since connection data over a period of time is required to accurately detect a connection pair, it is sometimes already too late to stop it. In both situations, there is still a question of how to differentiate between a legitimate stepping stone and one being used in an attack.

3 Detecting stepping stones

To detect stepping stone connection pairs using the on/off timing method it is necessary to first make the following assumptions and considerations.

(1) There is a single point of entry/exit to the network and all traffic coming or going must pass through the monitor. This is necessary to prevent a connection chain from entering and leaving the network through separate monitors, since this detection method is not distributed.
A connection is considered idle if it has not shown activity within $T_{idle}$ ms. The number of times a connection $A$ comes off of an idle period is denoted $off_A$.

A coincidence occurs when two connections end an idle period within $\delta$ ms of each other. The number of coincidences between two connections $A$ and $B$ is denoted $off_{A,B}$.

Two connection pairs are considered a stepping stone connection pair if and only if

$$\frac{off_{A,B}}{\min(\text{off}_A, \text{off}_B)} \geq \gamma$$

where $\gamma$ is a constant.

### 3.1 Problems with detection

This method has a number of downfalls. Most daunting is the $O(n^2)$ storage requirements. Each known connection must be tested with every other known connection. In large networks, under this extremely basic model it would be nearly impossible to detect stepping stone connection pairs after a certain level of network congestion. This complexity can be significantly reduced using various filters and storage methods that will be shown later.

Another major set back with this process is the extraordinary number of false positives it yields. The value $\gamma$ is typically chosen as a value between 0 and 1, therefore if a new connection appears, the moment there is a coincidence with any other connection it is detected as a stepping stone (most likely this is a false positive). This is because the ratio in (4) will always equate to 1 after a single coincidence with a new connection, which is greater than $\gamma$.

There are also network oddities that can disrupt the detection process. Packet loss and retransmissions can skew the off idle times of connections at the monitor and heavily congested links can cause delays exceeding the specified constants $T_{idle}$ and $\delta$. We will see that filters can be added to account for these perturbations as well.

It would seem that there should be a concern that IP fragmentation or incoming connections via an encrypted tunnel (such as a VPN tunnel) could possibly disturb the detection process as well. This is not the case. IP fragmentation requires that the first 8 bytes of the data portion of the IP datagram be present in the first fragment. Since our detection relies on TCP port numbers for determining the connection parameters, this is sufficient to extract the necessary information and the remaining fragments may be ignored. Also, the off idle times of a fragmented stream will still be correlated with another connection in its connection chain.

A virtual private network connection can be thought of as a connection that joins two networks into one. Each end is usually controlled by friendly organizations if not one in the same. Therefore, data arriving over such a connection is not necessarily considered an “inbound” connection. Note however that should the other network have an entry or exit point, this disobeys assumption (1) and stepping stone connection pairs cannot be detected to the fullest extent.
3.2 Some solutions

To help relax the storage requirements, in my implementation the connections are only paired after they have encountered a coincidence. Therefore, the complexity becomes $O(n \times k)$, where $k$ is the average number of connections that come off of an idle period within $\delta$ ms of another connection at some point in the detection process. More importantly, a termination filter and timeout filter can be added to further reduce the value of $n$. A termination filter can remove connections from memory whenever a connection is seen as being terminated (such as by seeing a RST or FIN packet). Similarly, the timeout filter can be used to remove connections from memory that have not shown any activity for a specified time duration. The timeout value should be used carefully, if an attacker is willing to wait for the duration of the timeout value, and no keep alive messages are sent in the mean time, she can fool the monitor into removing the connection before showing activity again. In either case, the value of $n$ is reduced from the number of connections occurring over the entire duration of the detection process to the currently open $n$ connections, which will naturally always be less.

Avoiding false positives involves adding another filter that adds another consideration to the previous four:

(5) In addition to (4), two connections are only considered a stepping stone connection pair if and only if

$$c_{soff_{A,B}} \geq \min csc \quad \text{and} \quad \frac{c_{soff_{A,B}}}{\min (off_A, off_B)} \geq \gamma'$$

where $c_{soff_{A,B}}$ is the number of consecutive coincidences between $A$ and $B$, $\min csc$ is a constant that is the minimum value allowed for $c_{soff_{A,B}}$, and $\gamma'$ is another constant much smaller than $\gamma$.

This additional consideration greatly reduces the number of false positives while negligibly increasing the number of false negatives. The recommended values for all constants in [10] is listed in figure 1.

<table>
<thead>
<tr>
<th>$T_{idle}$</th>
<th>$\delta$</th>
<th>$\gamma$</th>
<th>$\min csc$</th>
<th>$\gamma'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5000 ms</td>
<td>40 ms</td>
<td>0.3</td>
<td>5</td>
<td>0.04</td>
</tr>
</tbody>
</table>

This additional consideration greatly reduces the number of false positives while negligibly increasing the number of false negatives. The recommended values for all constants in [10] is listed in figure 1.

Another filter used to eliminate false positives is the direction filter. This filter ignores coincidences where connections come off of an idle period over the same interface. Typically, a connection chain will pass data from $H_1$ to $H_n$ and afterwards a response is passed from $H_n$ back to $H_1$, or vice versa. The behavior of $H_1$ sending data to $H_n$ and $H_n$ responding before the data is ever received is irregular and can be ignored, since this is not the behavior of a connection chain.

Yet another filter for eliminating false positives is the server filter. This filter ignores coincidences between connections that share a common identical end point. That is, one end of each connection has the same address and port number. When this is the case, almost always it tells us that the duplicate end point is a server being used by both connections. It would require significant modifications of the remote login applications.
used to be able to share a port in a connection chain, and is therefore not considered. The case that is accepted is that \( H_i \) connects to a server on \( H_{i+1} \).

In both the direction and server filters, if a connection chain traverses the monitor more than twice, both of the cases above do become valid. Even if filters ignore some connection pairs in these situations there must be at least one connection going in the opposite direction of the others, meaning it cannot share a direction or address and port, and will therefore be identified as coinciding with the other connections.

Finally, to handle the problems of network perturbations and the like causing inconsistent detection of stepping stone connection pairs, the following two filters may be added: a length filter to ignore all coincidences involving packets with no data, including control packets, and a sequence filter to ignore coincidences involving packets that are being retransmitted.

4 Attacks on stepping stone detection

In this section, I examine in detail each aspect of the stepping stone detection process, including each constant and each filter, and suggest attacks that may be used to exploit them. But first, I will describe some attacks on the monitor itself.

4.1 The monitor as an IDS

The monitor being used is only as strong as the IDS technology it is built upon. The monitor will most likely be susceptible to insertion and evasion attacks described in [4]. By inserting data into the monitor that will not actually be acknowledged by its destination an attacker can generate false positives as well as mask a true stepping stone connection. If the monitor ignores data that it believes will be rejected by the end system but is not, an attacker can successfully evade detection all together.

In many cases it is more difficult to fool a stepping stone detection algorithm that it is too fool an IDS.

4.2 DoS attacks against the monitor

If the monitor is running in real time, it is necessary to keep all connection data and information in memory and be able to quickly access it. We have already discussed the unrealistic storage requirements in the worse case. It is then up to the attacker to provide such a worse case situation. By sending spoofed packets each with different connection information, a monitor can be DoS-ed since it must maintain some state for each of the connections it analyzes.

A more devastating DoS attack can be performed if the attacker can identify packets leaving the monitor on the external interface (or internal, given an inside attacker). The attacker has only to send large numbers of spoofed data packets within short period of time after seeing an outgoing packet in order to register the spoofed connections as coincidences with the real connection. This increases the amount of state required for each connection more so than the first attack and is more powerful.

4.3 Attacks on \( T_{idle} \)

Here we examine how to exploit the value and purpose of \( T_{idle} \). Notice that as \( T_{idle} \) gets big, the number of false negatives increases due to fast paced interactive traffic being overlooked. As \( T_{idle} \) gets small, the number of false positives and false negatives
increases. The number of false positives increases because the monitor will detect non-
interactive traffic as interactive.

We can exploit $T_{idle}$ first by never actually coming off of an idle period. After a
connection is established, an attacker can continuously send a stream of data through the
connection chain to avoid an increasing off count. This can be done in a number of ways.

One such attack is to perform an insertion attack on the monitor. Send packets at
regular intervals that will not actually reach their destination to fool the monitor into
never considering the connection idle.

Another attack is to connect from $A$ to $B$ through the monitor using an application
like telnet with options set to send each character as it is typed, and then connect from $B$
to $C$ back through the monitor with options set to send entire lines only. In this way, the
traffic can be structured so that the off count of the first connection is inflated by the
number of characters in the line, but the other connection’s off count is only incremented
once.

4.4 Attacks on $\delta$

The value $\delta$ is one of the most value to exploit. Its purpose is to be the threshold
for coincidences. As $\delta$ grows, the number of false positives increases, since more and
more traffic will come off of idle periods within the coincidental threshold. As $\delta$ shrinks,
the number of false negatives increases, since it becomes less and less likely that
connections in an actual stepping stone connection chain will coincide within the
threshold.

Exploiting $\delta$ is a matter of exceeding its value in order to avoid appearing as a
coincidence, or coming off of an idle period within $\delta$ ms in order to falsify a coincidence
with another connection.

Delaying a connection chain long enough after entering a network before again
leaving through the monitor can be done in several ways. If $\delta$ is small enough, simply
connect repeatedly to a large number of hosts within the network, in hopes that the
normal transmission delay when applied enough times will eventually add up and exceed $\delta$.
If the network contains trans-oceanic links (or if the attacker is inside the network and
wants to appear as though he is outside) they may be used to delay the coming off of an
idle period by more than $\delta$ ms. Yet another approach is to flood a network point such as
a router or host being used as a stepping stone in hopes of hampering the speed at which
packets are processed. Finally, another less likely approach is to find an extraordinarily
slow host (or a host that can be made extraordinarily slow by bogging it down with
computations) and use this host as a stepping stone before leaving the network.

If an attacker can identify traffic exiting the monitor, she can create false
connection data to be correlated with the identified traffic by sending it within $\delta$ ms. This
is the premise again for the DoS attack listed above.

4.5 Attacks on $\gamma$

The purpose of $\gamma$ is to regulate the number of coincidences with respect to the
number of off times each connection being compared has actually had. As $\gamma$ increases,
the likely hood of a connection pair being deemed a stepping stone connection pair is
less. Conversely, this likelihood increases and $\gamma$ decreases. The important point to note
is that this is determined by the values of $off_A$, $off_B$ and $off_{A,B}$.
Inflating \(off_A\) or \(off_B\) in hopes of preventing a connection seems an ideal starting point. Given connections \(A: A_1 \rightarrow A_2\) and \(B: B_1 \rightarrow B_2\), both passing through the monitor and the connection chain \(A_1, A_2, B_1, B_2\), by periodically (but outside of the \(\delta\) threshold) inserting data from \(A_1 \rightarrow A_2\) and \(B_2 \rightarrow B_1\), the values for both \(off_A\) and \(off_B\) will be increased.

We have already shown that inserting false positives is fairly easy and the reason lies in the value of \(\min(off_A, off_B)\) resulting in a value of 1 after a single insertion of a coincidence. Then with an \(off_{A,B}\) value of 1, the ratio is also 1 and is therefore detected as a stepping stone connection pair.

Later we will show that all three values \(off_A, off_B\) and \(off_{A,B}\) can be reset by the attacker by exploiting different filters.

### 4.6 Attacks on \(\text{mincsc}\)

The required number of consecutive coincidences for a connection pair to be regarded as a stepping stone connection pair, \(\text{mincsc}\), greatly decreases the amount of naturally occurring false positives, but gives an attacker a particularly powerful attack against avoiding detection by the monitor.

The addition of \(\text{mincsc}\), does little to prevent the intentional insertion of false positives by an attacker. Rather than inserting a single packet to cause a false alarm, the attacker must insert a minimum of \(\text{mincsc}\) correlated connections to raise an alarm. Since the recommended value of \(\text{mincsc}\) is 4, this hardly complicates matters for an attacker, though an attack may require slightly more ingenuity.

\(\text{mincsc}\) can be exploited since it requires a consecutive coincidence count, \(\text{csoff}_{A,B}\), regardless of the actual coincidence count. If the attacker were to insert a packet and come off of idle time for connection \(A\), out of sync with connection \(B\), the value for \(\text{csoff}_{A,B}\) will be reset and no detection will be reported by the monitor.

### 4.7 Attacks on \(\gamma'\)

The purpose of \(\gamma'\) is to regulate the number of consecutive coincidences with respect to the minimum of the \(off\) counts for the two connections being compared. As \(\gamma'\) increases and decreases it has similar characteristics to the increase and decrease of \(\gamma\). Attacks on \(\gamma'\) are equivalent to attacks on \(\text{mincsc}\) and \(\gamma\) and are discussed in those sections above.

### 4.8 Attacks on the direction filter

The direction filter was added to reduce the number of false positives by ignoring connections that do not match the connection chain signature of inbound and then outbound, or outbound and then inbound. The filter ignores all connections coming off of an idle time over the same interface. This can be exploited by the attacker with an insertion attack.

Assume the attacker has created a connection chain \(A, B, C\), where \(B\) is within a network and the connections \(A \rightarrow B\) and \(B \rightarrow C\) traverse the monitor in opposite directions. Just before the attacker passes a message through the connection chain she forges and inserts a message from \(C \rightarrow B\) to bring the connection off of idle time in the inbound direction, then proceeds with the normal message. Since the connections both came off of idle time in the same direction, they are ignored.
4.9 Attacks on the termination filter

The termination filter was added to reduce the storage requirements to enough to keep state for active connections instead of all connections ever seen. When a RST or FIN packet is seen, the connection and all coincidences are removed. If the monitor is susceptible to insertion attacks, an attacker only has to forge and insert a RST packet to reset all data associated with a connection including all off counts and coincidence counters.

Assuming the attacker cannot carry out an insertion attack, she can still cancel her connection traversing the monitor between each interactive action. This may increase the frustration involved in carrying out an attack, or may make some commands impossible, but none the less it is a way of exploiting this filter.

4.10 Attacks on the timeout filter

The timeout filter was meant to remove connections after they have not shown activity for a given period of time. However, this filter should not be used except in rare circumstances, since it is trivial to defeat by simply waiting longer than the timeout value before transmitting again.

4.11 Attacks on the sequence filter

To ignore retransmissions it was necessary to add a filter that allowed the monitor to keep additional state information for each connection involving sequence numbers. Whenever a packet is seen that has an old sequence number it is ignored and whenever a packet is seen that has a sequence number that has not been reached yet it is also ignored, since it will be ignored by the destination and later resent.

This gives the attacker the ability to inflate the sequence numbers artificially without actually sending information. Once the sequence number has been inflated, the attacker can transmit a command through the connection chain, but the data for one of the links will be deemed a retransmission and ignored.

5 Testing

Testing was broken down into two categories, real-time simulations using Simnet and analysis of the detection algorithm using offline packet logs. The main purpose for using Simnet was to demonstrate the feasibility of the attacks and to demonstrate the monitors ability in a (somewhat) real-time environment. The analysis of the algorithms performance under pressure of increased network size and traffic as well as the effectiveness of the different filters was done offline using packet dumps as input to the monitor.

5.1 Simnet

The environment used in Simnet is represented in figure 2. The node GATE is a gateway router serving all of the nodes H1 through H8 to the outside world. The nodes G1 through G8 are outside of the network and should be considered any node on the Internet. Notice that GATE provides a chokepoint between the H-network and the Internet.

Simnet provided an ideal environment for all of the attacks presented in section 4, and they have been implemented as script files that can be used for demonstration of each
attack. In some cases, delays and loss rates were added to individual links to demonstrate the attacks and to demonstrate the resilience of the detection algorithm.

An application was installed on GATE to monitor all traffic between the H-network and the Internet, with the purpose of detecting interactive stepping stone connection pairs. Telnet client and server applications were also created to allow connection chaining between hosts.

![Figure 2. Simnet test network.](image)

Using Simnet to test the scalability of the detection algorithm proved to be an unrealistic task. Java does not (to my knowledge) allow for querying the system clock in smaller intervals than milliseconds. Also, there was a significant delay (approx. 200 ms) experienced even when bandwidth was unlimited and the latency for each link was 0. The delays seemed to vary occasionally, probably due to other processes on the test system running along side of the Simnet process. Since the minimum possible transmission delay that could be achieved was normally around 200 ms, it required scaling the monitor’s constants for $\delta$ and $T_{idle}$ up by a factor of 10 or more. This frustratingly delayed the time it took to run each script.

Simnet was also unable to handle more than 700 nodes on the test system. After considering creating subnet-like nodes, that would represent a multitude of connections, I realized that this would represent nothing more than one node pushing packets through the monitor and into another node that would simulate responses, as seen in figure 3. There wasn’t any difference between doing this and using an offline packet reader.

![Figure 3. Simulation with fake networks.](image)

### 5.2 Offline detection

This section analyzes the detection algorithms ability to scale as the network increases in size and speed, as well as the effectiveness of each of the filters. The trace
files that were used were taken from “http://moat.nlanr.net/Traces/” and were of various sizes. The traces were recorded on OC-3 links and were each 90 seconds in duration. However, during this 90 seconds, over 1.6 million packets traversed the router, offering a sound starting point for scalability testing.

The trace files that were acquired only contained header information and did not contain packet contents. Therefore it was impossible to know whether or not any of the connection pairs detected were actually part of the same connection chain. Since the packet logs only contained 90 seconds worth of data, it is unlikely that there are large numbers of interactive sessions taking place that both enter and exit the network in the same connection chain during that time. Therefore, I assumed in all of the results below that none of the traces contained valid connection pairs, all pairs detected are considered false positives. Regardless, the huge numbers allow for good testing.

In order to gauge the performance of this algorithm as the network became congested, the same data files were used but the constants of the detection algorithm were scaled accordingly. To simulate a network with twice as many connections, the log file was treated as a 45 second trace and the constants $\delta$ and $T_{idle}$ were doubled. This fooled the detector into considering roughly twice as many connection pairs as coincidences than the previous run. Different levels of congestion were scaled in this manner.

As seen in figures 4 and 5, the time required to analyze the trace files increased linearly as the networks congestion level increased. The results are clearer for the larger trace file used in figure 4. More coincidences will occur in a congested environment, and therefore the tables maintained by the monitor will grow in size and take longer to locate the correct entries. Clearly, on the system being tested on, any traffic more congested than normal traffic would have been impossible to monitor in real time. Even a designated system for monitoring such traffic would be exceedingly stressed as the duration of the traces exceeded 90 seconds.

![Figure 4. Congestion in large network.](image)

![Figure 5. Congestion in small network.](image)

Figures 6 and 7 show us that the memory requirements as the network becomes congested remain relatively static. They show a minimal increase over the levels of congestion. This is interesting because it would seem at first that the tables of connections would be increasing in size as the number of coincidences increases. This is most likely due to the fact that the trace files do not actually contain a larger number of connections, they are just scaled to seem to take place in a shorter time frame.
Interesting results were obtained when analyzing the effectiveness of the individual filters as well. The filters that were analyzed were the directional filter, the server filter, the length filter, and a custom filter that includes steps (4) and (5) of the properties listed in section 3, instead of just (5) which is used for each test.

As can be seen in figures 8, 9 and 10, each an independent test, the directional filter eliminated the most false positives followed by the server filter. The custom filter including steps (4) and (5) did not improve much over the normal step (5) method for detecting stepping stone connection pairs.

Figure 6. Congestion in large network.

Figure 7. Congestion in small network.

Figure 8. Filter effectiveness test one.

Figure 9. Filter effectiveness test two.

Figure 10. Filter effectiveness test three.
Each test was run with the length filter on and off. Remarkably, when the length filter was turned on more false positives were detected than when turned off. Since we are considering all connections paired to be false positives here, this may imply that without the length filter there will be significantly more false negatives.

The charts represent the number of false positives reported as the first bar, the number of reported stepping stones retracted (that is, the detector realized later than in fact what was detected was not a stepping stone connection pair), and the third bar represents the difference between the first and second bars representing the remaining false positives detected.

6 Conclusions

We have just shown that it is reasonable to employ one of these detection algorithms after the fact. Real time detection will present a problem as network congestion increases. Often times, on/off time monitoring is conducted offline and is considered to be an offline algorithm. It can be performed in real time as seen in the Simnet scenarios, but this will still only detect an intrusion after the intrusion has occurred, and therefore it will always be after the fact.

The robustness of the algorithm is very sufficient in detecting interactive sessions by ignoring network anomalies and perturbations as well as ignoring control packets. The on/off timing itself provides an advantage over an attacker trying to fool an IDS by using multiple protocols or encryptions.

Most importantly, the stepping stone detection algorithm is only as strong as the IDS it is running on top of. It has been shown that a single packet can cause false negatives or false positives if it can be inserted into the monitor. It has also been shown that the algorithm may be evaded by exploiting time delays to exceed different constants held by the monitor.
7 References


