Journal of Information Assurance and Security.
ISSN 1554-1010 Volume 6 (2011) pp. 452–461

Detecting Multi-Hop Stepping-Stone Pairs with Chaff and Clock Skew

Ying-Wei Kuo and Shou-Hsuan Stephen Huang

University of Houston, Department of Computer Science
Houston, Texas 77204, United States
{ykuo, shuang}@cs.uh.edu

***Abstract*:** Stepping-stone attacks are attackers who use a sequence of stepping-stone hosts to initiate attacks in order to hide their origins. The goal of this paper is to find algorithms to correctly detect the attacks and have the ability to tolerate the clock skew or/and chaff while exhibiting low time complexity. We propose three novel algorithms for detecting correlation and similarity of two connections not only into and out of a single stepping stone host (adjacent streams), but also across multiple stepping-stone hosts. To evaluate the accuracy and efficiency, we conduct extensive experiments. We also evaluate how chaff packets and clock skew may affect these methods. We present a comparison of the algorithms in terms of false rates of detection, and identify one of the approaches that can efficiently achieve good performance under a variety of circumstances.

***Keywords*:** stepping-stone attack, intrusion detection, network security, connection chain, chaff evasion technique, clock skew.

I. Introduction

Cybercrime not only threatens the privacy of those who use the Internet, but also causes a serious data protection problem for the society. A popular technique of choice for cyber criminals to maintain anonymity nowadays is "stepping-stone attack" (see Figure 1 for an illustration), which uses a sequence of stepping-stone hosts to initiate attacks rather than directly linked from the intruder's origin. A "stepping-stone host" is a previously compromised intermediary host that allows an intruder to login and route the traffic to the destination. By routing the attack through several intermediate stepping-stone hosts, it becomes practically impossible to trace the route back to the originating host. The intruder can thus remain their anonymity. The goal of a stepping-stone detection system is to detect such attacks and protect the victim host from the intrusion.

A variety of techniques to detect stepping-stone attacks have been proposed previously [3],[4],[5]. Most of them studied the detection of a single stepping stone host by finding correlations of two consecutive connections in a connection chain (e.g. in Figure 1, determining if Host E is being used as a stepping-stone host by studying the correlation between connections 4 and 5). A generalization of the problem is to detect a stepping-stone chain across several hosts. Previous algorithms do not allow analyzing a correlation across multiple stepping-stone hosts (e.g. to determine whether Connections 2 and 5 are part of a connection chain). Since it is not realistic to have every host's information available, it may be required to detect the correlation between two traffic flows which are not necessarily consecutive in a stepping-stone connection chain. For example, if we suspect the attack originated from a particular host (by other means), we can try to correlate the connections on both ends to confirm such an attack.

Correlating two streams of packets in the multi-hop stepping-stone pair is very similar to the problems in Pattern Recognition. In this paper, we propose three Dynamic Programming (DP) based pattern recognition schemes for finding the correlation of two consecutive connections of two adjacent or non-adjacent connections across intermediate host(s). We examine which schemes make better performance under different circumstances such as clock skew and chaff involvement. The preliminary results of this work were published in [17].

The rest of this paper is structured as follows. In the next section we discuss the background of stepping-stone detections and the motivation of this paper. Section 3 provides the problem definition and the notations used. We proposed our novel schemes which inspired from pattern recognition approaches in Section 4. The experiments that we designed to compare several new and previous detection methods and the performance of the schemes are presented in Section 5. Finally, the conclusion is made in Section 6.

II. Background and Motivation

To date, most of the approaches for detecting encrypted stepping-stone connections are timing-based. There were two types of stepping-stone detection approaches. The first type was to distinguish the long stepping-stone connection chains from short, ones such as [1],[2]. Recently, Huang et al. [2] used an approximated round-trip time (RTT) to estimate the chain length with the intention of differentiating a long connection chain from a short one.

In this paper, we used the second type of approach, which was to find the correlation of a pair of connection flows so as

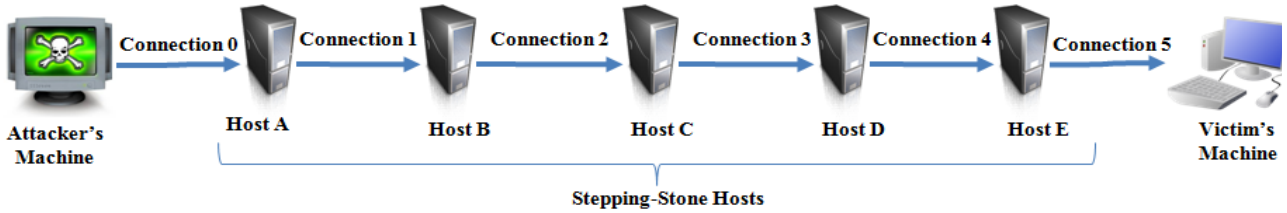


Figure 1. Example of Stepping-Stone Attack

to cut off the incoming flow from the suspicious host. There were some well known approaches [3],[4],[5] that find the correlation using the timestamps of the incoming/outgoing packets to/from a stepping-stone host. Donoho et al. [6] investigated the restrictions on the ability of intruders to hide their traffic through timing perturbation and packet padding. Blum et al. [7] proposed the methods that considered both delay and chaff insertion (existed separately) by calculating the amount of packets needed to confidently detect stepping stone flows. Moreover, Kuo and Huang [8] introduced a one-to-one order-preserving mapping algorithm DMIM which was proven to have close to 100% accuracy between a pair of incoming and outgoing flows. Since the 1-1 mapping does not always hold due to the traffic issues (such as packet loss), DM2 algorithm in [9] was later designed to solve this problem.

Recently, Kuo and Huang [16] proposed another algorithm named Optimal Subsequence with Merge (OSSM) with $O(n^2m^3)$ time complexity where n and m are the number of packets in two monitoring traffic streams. The main advantages of OSSM algorithm were: it avoided the possible clock skew on the host by mapping the intervals of two adjacent packets instead of the packets timestamps; and it allowed a limited amount of injected chaff packets.

In order to use single-hop stepping-stone detection algorithms to trace the intruder, it will require all the intermediate hosts to collaborate with the investigator. That is not practical when these stepping-stone hosts belong to different organization and maybe reside in different countries. If we suspect an intrusion coming from a particular host, we can compare the streams on both ends of the chain to confirm if they belong to the same chain without knowing much about the intermediate hosts. . One of our goals is to design a method of detecting the correlation between two traffic flows across multiple stepping-stone (not necessarily adjacent) hosts.

Clock skew is another issue that may cause inaccurate matching for which most of the single-hop detection algorithms did not have to take into consideration. On the Internet, clock skew is denoted as the difference in time shown by the system clocks at different hosts on the network. Although the Network Time Protocol (NTP) is widely used on the Internet for synchronizing the clocks of systems, not all Internet hosts synchronized the clock with it. Although OSSM algorithm [16] is able to deal with clock skew and some injected chaff, it has the drawback of high time complexity. Therefore, a second objective of this paper is to design an efficient algorithm that can avoid the clock skew issue.

Most of the recent timing-based approaches make an assumption that the intruder is willing to accept a maximum tolerable delay Δ (since humans are not willing to work over interactive connections with very long latencies) and the delay of each packet must be in the range of $[0, \Delta)$. Thus an important issue in the past is to find a suitable value for the maximum tolerable delay is a given environment. Our last objective is to design a method without making such an assumption.

III. Problem Definition

This paper considers the problem of detecting multi-hop stepping-stone stream pairs under the following assumptions:

1. Content of the packets are encrypted.
2. Only two streams of packets are available and there is no need to know other streams between them.
3. There may not be a one-to-one mapping between the two streams due to network issues, such as packet loss.
4. The timestamps on the two streams may have a clock skew.
5. Intruders may introduce chaff packets to evade detection. Thus one stream of the two under investigation may include some chaff packets and the other may not.

In this work, each connection includes a list of packet timestamps. Our strategy is to assign a dissimilarity score for a pair of connections, and if the dissimilarity value is below a user predefined threshold, an alarm indicating a potential stepping-stone attack is triggered. The reason to use a dissimilarity score instead of a similarity score is inherited from the Dynamic Time Warping algorithms that our algorithm was derived from. In those algorithms, there is a cost associated with the difference of time sequences. Here are some notations used in this paper:

- R: A reference stream $R = \{r_1, r_2, r_3, \ldots, r_n\}$ of length n which contains the timestamps of the packets in increasing order.
- T: A target stream $T = \{t_1, t_2, t_3, \ldots, t_m\}$ of length m which contains the timestamps of the packets in increasing order.
- d(p,q): The distance between points r_p and t_q typically measured in Euclidean distance.
- cost(p,q): The local cumulative difference between (up to) the aligned substreams $\{r_1 \ldots, r_p\}$ and $\{t_{1,\ldots,} t_q\}$ in the two time series.
- Dis(R,T): The dissimilarity score between stream R and stream T. The higher the score, the less similarity of two streams.

IV. Our Schemes

To overcome the obstacles mentioned in last section, we aim to find some methods that can intelligently avoid the disturbances. The idea in this paper is inspired by some DP based pattern recognition techniques [14] which are variations of Dynamic Time Warping (DTW) [10],[11], with application areas such as handwriting matching [12], speech recognition [15] and incomplete time series to post-stroke rehabilitation [13], etc. DTW technique not only measures optimal difference (dissimilarity) between two sequences but also solve the problem of time scaling (warps non-linearly by stretching or shrinking it) in time series. The basic idea is locally recovering the time axis in order to minimize the local cumulative difference between the aligned points in two time series R and T, and the possible alignments considered include all local compressions and shifts. DTW technique allowed one-to-many or many-to-one mapping, which tolerated an element on either stream to be mapped more than once. It is stated more formally in (1):

$$d(p,q) = |t_q - r_p|, \qquad p = 1, \ldots, n, q = 1, \ldots, m$$

$$cost(p,q) = d(p,q) + min \begin{cases} cost(p, q-1) \\ cost(p-1, q-1) \\ cost(p-1, q) \end{cases}$$

$$Dis(R,T) = cost(n,m) \tag{1}$$

It uses the local distance *d(p,q)* (Euclidean distance is used here) plus the minimum of three given values *cost(p,q-1)*, *cost(p-1,q)*, *cost(p-1,q-1)* to calculate *cost(p,q)* iteratively where $p = 1, \ldots, n$ and $q = 1, \ldots, m$. Dissimilarity score of two time-series R and T is always saved in *cost(n,m)*. The time complexity is *O(nm)*, because all the matching possibilities need to be considered to ensure the optimal answer found.

OSSM algorithm [16], on the other hand, uses the merge function to skip the outliers on the target stream, and uses a constant penalty (ρ) function to skip the outliers on the reference stream. During the one-to-one mapping, OSSM allows an interval on the target stream to merge with its previous intervals, one at a time until the smallest difference (the difference between an interval on reference stream and its corresponding interval(s) on target stream) among the merged intervals was found. The OSSM algorithm is defined in (2):

$$d(p,q) = |t_q - r_p|, \qquad p = 1, \ldots, n, q = 1, \ldots, m$$

$$\rho = mean_p\left(min_q\, d(p,q)\right) + std_p\left(min_q\, d(p,q)\right)$$

$$merge(p,q,v) = \left\{min\left(d\left(p, \sum_{s=a}^{v} s\right)\right), a = q, \ldots, v\right\}$$

Initialization:

$$cost(p,q) = \begin{cases} d(1,q), & if\ p = 1, \\ d(p,1), & if\ q = 1, \end{cases}$$

General case:

$$Given\ p = 1, \ldots, n\ and\ \ q = 1, \ldots, m, cost(u,v)\ is\ updated:$$

$$cost(u,v) = cost(p,q) + min\begin{cases} merge(p,q,v), & if\ u = p+1 \\ merge(p,q,v) + (u-p-1)\times\rho, & if\ u > p+1 \end{cases}$$

$$for\ \ n \geq u \geq p+1, m \geq v \geq q+1$$

$$Dis(R,T) = min\{cost(n,s) | s = 1, \ldots, m\} \qquad (2)$$

A. Dynamic Time Warping with Sliding Window (DTWW)

According to the assumption that the clock skew between each host is unknown, it is very likely that either the reference or target connection has a shifted timeline. Unfortunately, DTW technique returns higher dissimilarity score as the clock skew increases and the attack pair may be misidentified as normal pair. So, we present our first scheme named DTW with Sliding Window (DTWW) as a solution. The idea (shown in Figure 2) is to take a subsequence of timestamps with a fixed time window *w* from the end of the downstream connection as the reference stream, and do a sliding window operation on the upstream.

The sliding window starts from the beginning point and moves a packet at a time. Each window of the timestamps after a move is treated as the target stream. Then it uses (1) above to measure the dissimilarity score between each pair of the reference and target streams. The optimal (minimum) dissimilarity score of all pairs denotes the final dissimilarity for this connection pair. The time complexity is *O(nm(l-w/n))*, where *l* is the duration period. In the case it is unclear which stream is the upper stream, we can test both cases and one will work much better than the other one.

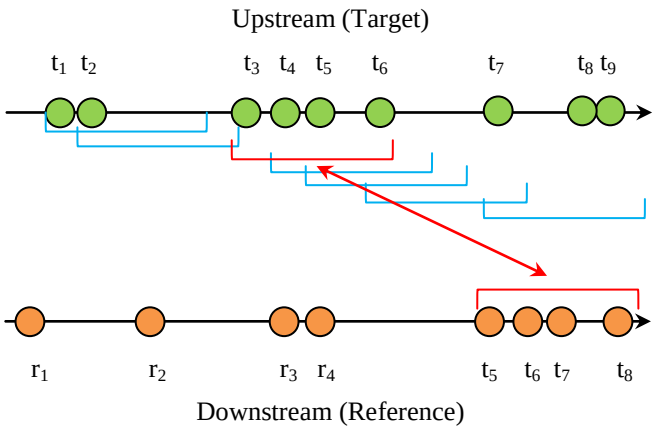


Figure 2. Example of sliding windows using DTWW algorithm

From our experience, the selection of the beginning point does affect the result. This method is the improvement of [17]. According to [17], the sliding window starts from the beginning point and moves a fixed time interval at a time (1 second has been used in [17]). In the current version, we move the window to directly the next packets. This improvement not only reduces the time required but also improves the accuracy of the algorithm. Approximately 3%~6% of improvement on the false negative rate can be achieved with the same clock skew.

An example to demonstrate the effectiveness of this algorithm can be seen in Figure 3. The reference stream tests the clock skew from -10 seconds to +10 seconds with an increment of 1 second at a time. According to the figure, the optimal dissimilarity score is 0.002 second at zero shifting time (no clock skew). For normal pairs of streams, the dissimilarity are generally much higher than those of the attack pairs. Not only the dissimilarities of the normal pairs are within a range they also fluctuate. On the other hand, the dissimilarity of the attack pairs has a distinct U-shape with a minimum in the middle. Thus our algorithm is set to look for the minimum point in such a U-shape curve and select the dissimilarity at the minimum point as the actual value.

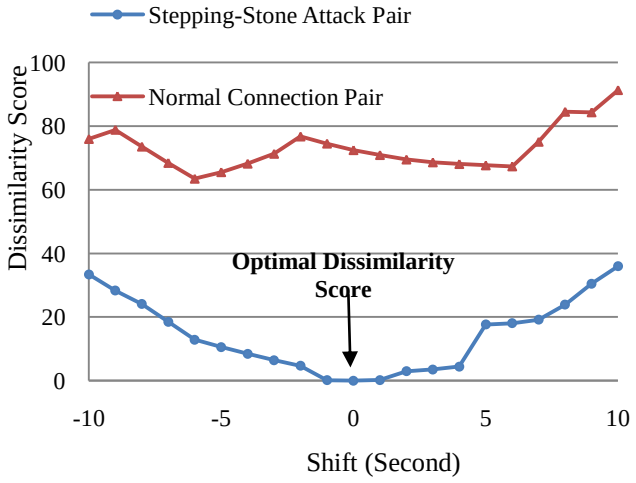


Figure 3. Example of using DTWW for a normal pair and an attack pair with the clock skews from -10 to +10 seconds.

B. Dynamic Time Warping with Slope (DTWS)

If we draw lines matching a packet in the reference stream to it's corresponding packet in the target stream (see Figure 4 for example), we will see that these lines have very similar slopes if the matching is a correct one. So our next algorithm will be trying to find all these "slope" as close to each other as possible. One of the ways to solve the clock skew issue is to use the *"slope"* or so-called *delay* between a reference packet and a target packet instead of using the packet timestamps or the interval between two consecutive packets. It is measured by the local distance *d(p, q)* without using absolute value (a slope may be positive or negative). Slope correlation then compares the difference of two slopes which will not be affected even though the time is shifted on one flow. Such a slope-based scheme is called Dynamic Time Warping with Slope (DTWS). This method has no restriction on which flow becomes reference stream and which becomes target stream. Since it considers all the possibilities for the optimal matching, its time complexity is *O(nm)*. DTWS algorithm is formally defined in (3):

$$d(p,q) = t_q - r_p, \qquad p = 1,\dots,n, q = 1,\dots,m$$
$$cost(p,q) = min\begin{cases} cost(p,q-1) + |d(p,q) - d(p,q-1)| \\ cost(p-1,q-1) + |d(p,q) - d(p-1,q-1)| \\ cost(p-1,q) + |d(p,q) - d(p-1,q)| \end{cases} \quad (3)$$
$$Dis(R,T) = cost(n,m)$$

DTWS scheme uses the minimum of three given slope correlations to calculate *cost(p,q)* repeatedly where $p = 1, \dots, n$ and $q = 1, \dots, m$. An example of using this algorithm is described in Figure 4. Based on the definition, cost(4, 5) is the minimum of $cost(5, 4)+|d(r_5, t_4)-d(r_5, t_3)|$, $cost(4, 3)+|d(r_5, t_4)-d(r_4, t_3)|$ and $cost(4, 4)+|d(r_5, t_4)-d(r_4, t_4)|$.

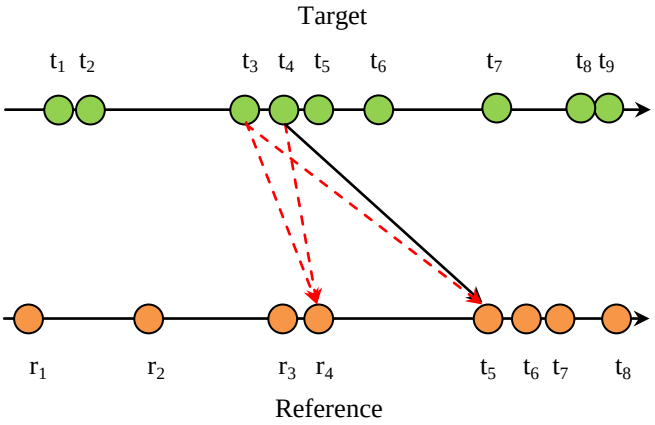


Figure 4. Example of comparing the slope correlation using DTWS algorithm

C. Optimal Slope Alignment (OSA)

Both DTWW and DTWS schemes have a main limitation, which is particularly sensitive to outliers. In other words, they always match the reference stream to the whole target stream no matter whether the target stream had a longer length (due to network problem or chaff) than the reference stream. Optimal Slope Alignment (OSA) scheme is designed to solve such potential problems. The goal of such scheme is to find the subsequences *R'* of *R* and *T'* of *T* such that *R'* best matches *T'*. Occasionally, an extra packet may appear on the reference stream which may cause serious misalignment of packets. Thus the unique feature of the next algorithm is that we allow some of the packets in the reference stream to be "skipped." Note that we have to do this carefully because clearly if we skip all packets the cost can go down to zero. So we have to add some type of penalty for skipping a packet.

Once more, OSA method compares the slope correlation,

ALGORITHM I. Optimal Slope Alignment (OSA)

```
Algorithm OSA(R,T,n,m,ρ)
  //initialization
  for (q=0; q<m; q++)
      cost(0,q) = 0;

  //calculate all possible costs
  for (p=1; p<n; p++){
      for (q=0; q<m; q++){
          for (k=0; k≤q; k++){
              currSlope = t_q-r_p;
              prevSlope = t_k-r_p-1;
              slopeDiff = |currSlope-prevSlope|;
              //give a penalty if it's many-to-one mappings
              if(k==q)   slopeDiff= ρ*slopeDiff;
              //update the cost, when there is a smaller one
              if(cost(p,q)>cost(p-1,k)+slopeDiff)
                  cost(p,q) = cost(p-1,k)+slopeDiff;
          }
      }
  }
  //find optimal solution
  minCost=∞;
  for (q=n-1;q<m;q++){
      if(cost(n-1,q)<minCost)   minCost= cost(n-1,q);
  }
  DisScore=minCost;
```

so the local distance $d(p, q)$ allows either a positive or negative value. It assumes that the reference stream is shorter than or equal to the target stream (allows to match partial target stream) and both reference and target streams may contain outliers. OSA scheme has the ability to skip outliers during the matching process, restricting that no points on the reference stream can be mapped more than once. It skips the outliers (chaff injected by intruders on a purpose to evade the detection) from the target stream without penalty, and introduces the penalty ρ for skipping on the reference stream (extra packets due to network issue). The optimal structure of OSA is formally defined in (4):

$$d(p,q) = t_q - r_p\,, \qquad p = 1,\dots,n, q = 1,\dots,m$$

$$cost(p,q) = \begin{cases} 0 & if\ p = 0 \\ min_{k=1}^{q} \begin{cases} cost(p-1,k) + |d(p,q) - d(p,q-1)|, & if\ k < q \\ cost(p-1,k) + \rho \times |d(p,q) - d(p,q-1)|, & if\ k = q \end{cases} \end{cases}$$

$$Dis(R,T) = min\{cost(n,s) | s = 1,\dots,m\} \qquad (4)$$

The penalty ρ is a user defined parameter between 0 and 1 if k equals q. Figure 5 illustrates an example of using such a slope correlation. This algorithm uses the minimum of all possible slope correlations to calculate *cost(p, q)* repeatedly where $p = 1, \dots, n$ and $q = 1, \dots, m$. Based on the definition, cost(5, 4) is the minimum of $cost(4, 4)+\rho\times|d(r_5, t_4)-d(r_4, t_4)|$, $cost(4, 3)+|d(r_5, t_4)-d(r_4, t_3)|$, $cost(4, 2)+|d(r_5, t_4)-d(r_4, t_2)|$ and $cost(4, 1)+|d(r_5, t_4)-d(r_4, t_1)|$.

The details of the algorithm are given in Algorithm I. For initialization, OSA scheme sets the distance between the first point in *R* and every point in *T* to 0 and infinity for all other possible matching. In the main loop, *p* goes over each point in R stream and *q* goes over each point in *T* stream while *k* goes over points in *T* stream up to the q^{th} point, respectively. Each node for OSA scheme is updated only when there is a smaller dissimilarity score from a connected node in the previous row, and guarantee to return the cost of the optimal (shortest) path leading to every node. In other words, the minimum value in the last row is the optimal score between two compared streams. Thus the complexity of *OSA* algorithm is $O(nm^2)$.

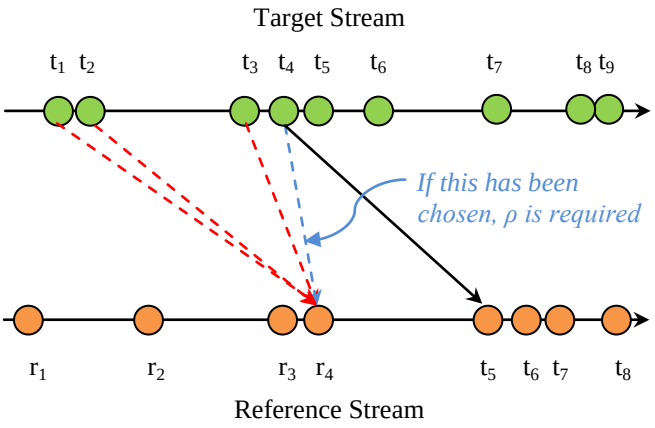


Figure 6. Example of comparing the slope correlation using OSA algorithm

V Comparison of the Schemes

The goal of the comparison is to inspect the accuracy and the performance of the various schemes. DTW and OSSM schemes are used to compare with our schemes. Whether an attack alarm is raised or not is based on the thresholds supplied by the respective schemes. Since the standards of how to calculate the dissimilarity scores are different for each scheme, the dissimilarly scores are not comparable across all of them. Hence, we use the false positive rate (FPR) and false negative rate (FNR) as the metrics to compare all the schemes.

A. Experiment Setup

In order to test the effectiveness of the correlation algorithms between two traffic flows (adjacent or not), we collect the traffic flows into and out of every host along this chain. Thus we first set up a stepping-stone connection chain with 7 intermediate stepping-stone hosts. This is done by using secured connection SSH to link different remote servers one followed another under various UNIX operating systems. An illustration of the connection chain setup is depicted in Figure 6 below.

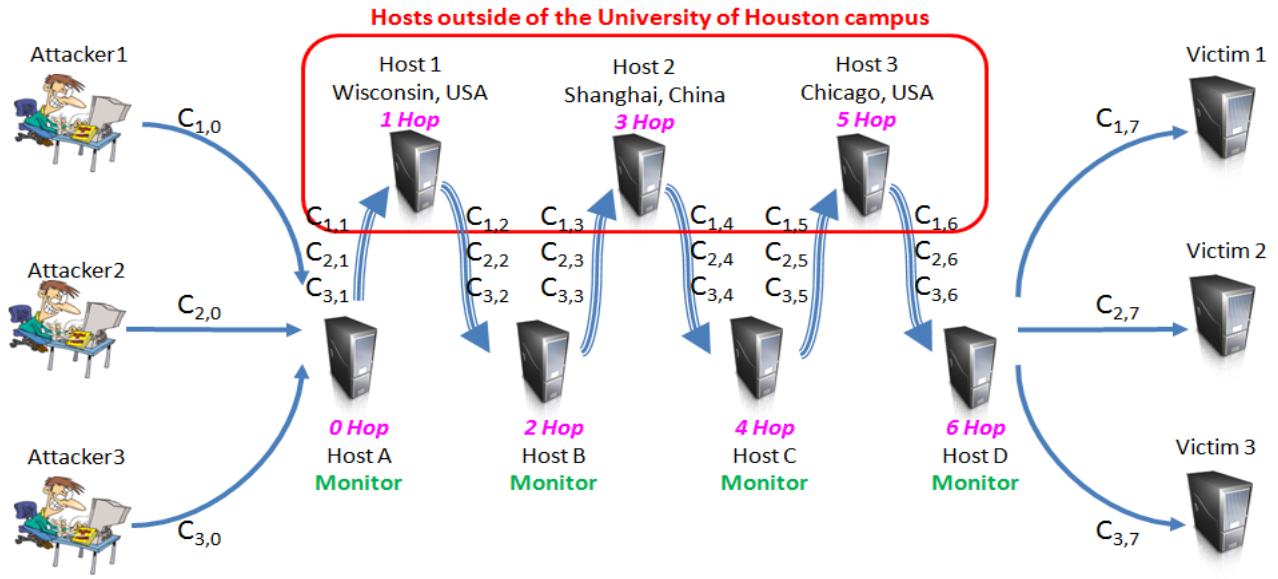


Figure 5. An example of the setup for the stepping-stone traffic flow collection

For the experiments, we have created several long distance flows. We have prepared three hosts far away from our campus (Host 1 in Wisconsin, USA, Host 2 in Shanghai, China, and Host 3 in Chicago, USA), and all others severs are on our campus network in Texas, USA. The chain is linked with a local host and a long distance host alternatively so that we can capture packets on all streams in the chain. For instance, one of the chains is linked in the order as follows: local → Wisconsin → local → Shanghai → local → Chicago → local. This setup allows us to analyze chains of different lengths with the same packet contents and timing information. All the flows are captured from live network connections. This alternating setup allows us to collect data of various lengths.

To generate traffics, we have three users (acted as three attackers) from three separate local hosts linked to the same stepping-stone chain and then connected to the corresponding victim's machine in different locations. The collection has been repeated 10 times on different dates, and the order of the three long distance hosts is different each time, even though the length of the chain remains the same. The users typed at their own pace and were only approximately synchronized at the beginning of the test.

B. Experiment Design

The collected data contains 30 (3 users times 10 collections) chains with 7 traffic flows (each from 0 hop to 6 hops). Of all combinations on each hop, there are 30 streams (attacker i to victim i) which serve as stepping-stone attack connections and the rest of 60 (attacker i to victim j where $i \neq j$) streams are acting as normal connections. The hop denotes the distance between the hosts or the number of intermediate hosts between the streams. For example, 0 Hop represents the comparison across a single stepping-stone host (Host A), which compares connection $C_{u,0}$ with connection $C_{u,1}$, where $u = 1, 2, 3$; and 1 Hop denotes the comparison across two stepping-stone hosts (Host A and B which compares connection $C_{u,0}$ with connection $C_{u,2}$, where $u = 1, 2, 3$). Of each stream, we took 60 seconds as the duration time for packet collection. For we set DTWW scheme's window w to 20 seconds and the starting point of each sliding window is increased at 1 second each time during the experiments.

Essentially, we try to classify stepping-stones flows into two groups in which one is the target named "attack" if there is strong correlation between streams and the other one is "normal". We conduct the study in a way that none of the investigators knows the locations or number of the stepping-stone hosts when the data are analyzed. We considered the following four scenarios on our stepping stone problem:

Scenario 1: Neither clock skew nor chaff exists,
Scenario 2: Only clock skew exists,
Scenario 3: Only chaff exists, and
Scenario 4: Clock skew and chaff exists simultaneously.

C. Dissimilarity Threshold and False Rate

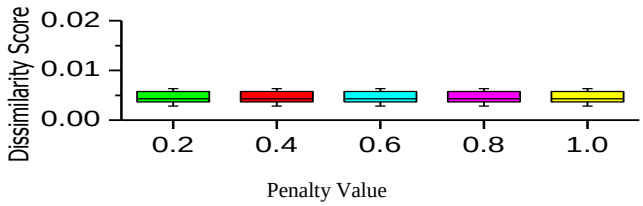
To validate our algorithms, we have to computer the accuracy of the classification. In all the correlation algorithms, we classify a pair of streams as attack if the dissimilarity is low (i.e., they are similar). Thus one issue that we have to deal with is to determine the threshold that separate attack from normal. Since each scheme computes the dissimilarity in their own way, they need to assign their own unique threshold. The threshold certainly depends on the network traffic and the testing environment such as operating systems etc. It is not practical to produce a number that can be used everywhere. Hence, we use the false rates, false positive rate and false negative rate, of the schemes to measure their performance. The false positive rate (FPR) represents the proportion of absent events that yield positive test outcomes, i.e., the ratio of normal connections falsely identify as the stepping-stone attack connections. On the other hand, the false negative rate (FNR) represents the proportion of present events that yield negative test outcomes, i.e., the ratio of stepping-stone attack connections misidentify as the normal connections.

Thus we shall describe our algorithm on selecting these thresholds. Our algorithm is based on machine learning where part of the data are used to train the system to derive a threshold and other data are used to test the effectiveness of the threshold. Each threshold is obtained via *Leave-one-out* cross validation. We split the original data into two sets. One sample is used as training data and all other data is used as the testing data. We use training data to generate the threshold and assign the similarity scores for the testing data. This operation is repeated n times (n is the number of samples). At each time, the training set cannot be redundantly assigned and a similarity score is generated. If the dissimilarity scores of both normal and attack are overlapped, the threshold is set to the 90 percentile (targeting 10% false positive rate) of the combined scores across the cross validation. Otherwise, if there is a gap between the dissimilarity scores of both classes, the threshold is set to the top 20 percentile of the gap. Note that the gap was the minimum of the normal dissimilarity scores minus the maximum of the attack dissimilarity scores.

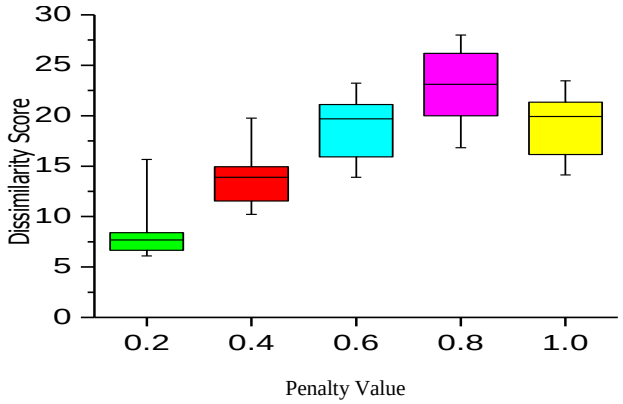
D. Penalty Selection of OSA Scheme

As we explained in the previous section, skipping (not matching) some packets from reference stream R and target stream T is necessary, because both streams may contain some dummy packets. However, skipping too many packets of sequence R reduces the distance cost between two streams and causes the false positive rate to increase. On the other hand, skipping not enough packets of sequence R increases a chance of accidental matches (tolerate some extra packets due to network issue) and causes the true positive rate to drop. To prevent this from happening and to find the best possible correspondence of subsequence R' of R and T' of T, a suitable penalty ρ for skipping packets is has to be determined.

We study the differences of dissimilarity score between various numbers of penalties (in 0.2 second increments) without making any assumptions of the underlying statistical distribution. The result is summarized in Figure 7 above. Each attack box-plot contains 30 samples and each normal box-plot contains 60 samples. The data used in this experiment was with consecutive connections. Note the bottom and top of the box are always the 25th and 75th percentile (the lower and upper quartiles), the band near the middle of the box is the 50th percentile (the median), and the ends of the box represent the minimum and maximum of all the data. Based on the figure, we can see setting penalty to approximately 0.8 can give us the maximum range between the separation of attack pairs and normal pairs. However, we need a more systematic way to find the optimal penalty.



a) Stepping-stone Attack Connections



b) Normal Connections

Figure 7. Box-Plot with various penalties. a) represents the results of stepping-stone attack connections; b) shows the results of normal connections.

The optimal penalty may vary because of the different network traffic environment. In order to define the optimal penalty for different data set, we extract five points with even interval (incremented by 0.2) between the possible range 0 and 1 and develop a distribution model for these points (2nd order, polynomial trend-line is suggested and used here). Then the trend-line can be used to compute the peak over penalty (extreme value). The result of finding such peak is displayed in Figure 9. We also investigate the difference of using points from the median ($y = -42.63x^2 + 69.72x - 4.685$), 25th ($y = -37.50x^2 + 61.68x - 3.792$) and 75th ($y = -31.75x^2 + 51.83x - 3.077$) percentiles and find the results of the peak over penalty are consistent (0.818, 0.822 and 0.816). A user may use either the median, 25th or 75th percentiles to find the optimal penalty of own set of data. We use 0.819 (the average of all three values) as the penalty throughout this paper.

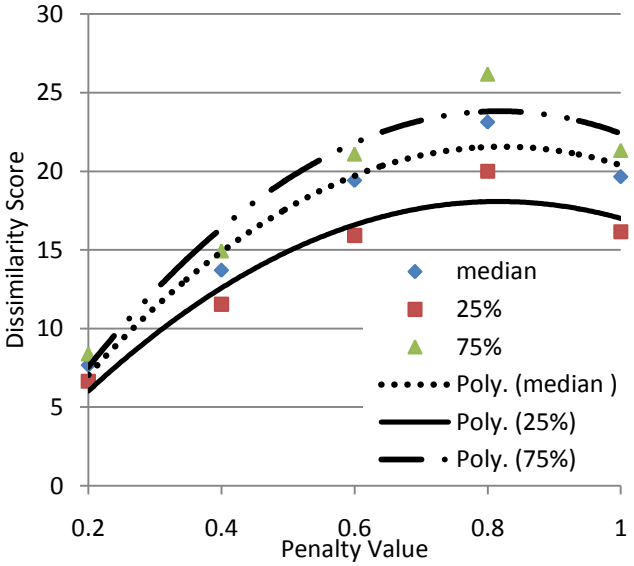
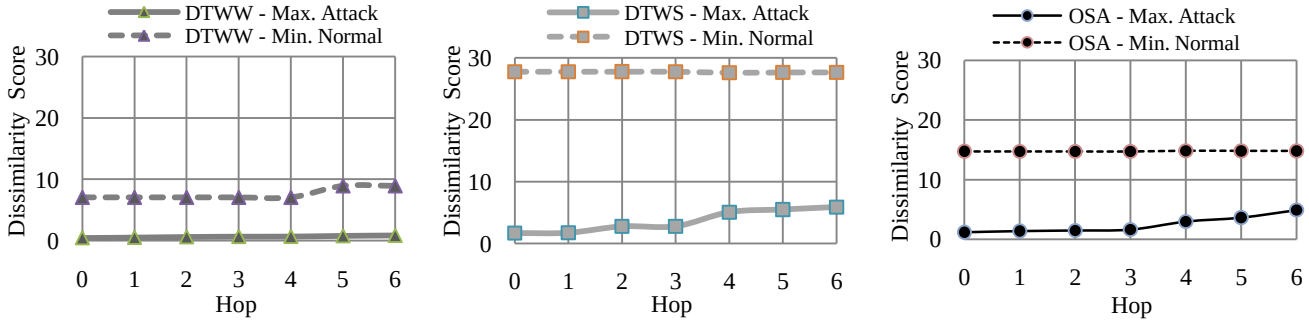
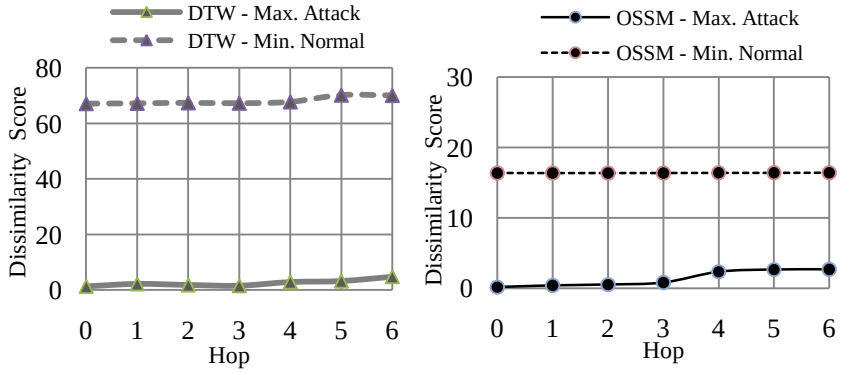


Figure 9. Trend-lines of median, 25th and 75th percentile based on the results from Figure 7 - Normal.



(a) DTWW Algorithm **(b)** DTWS Algorithm **(c)** OSA Algorithm



(d) DTW Algorithm **(e)** OSSM Algorithm

Figure 8. Dissimilarity Scores of each hop for algorithm DTWW, DTWS, OSA, DTW and OSSM

E. Performance without Clock Skew and/or Chaff

We firstly investigate the schemes' performance without involving any clock skew or chaff. Based on the results provided in Figure 6, we learned that the normal connection pairs stay constant with different hops, while the attack connection pairs slightly increase the dissimilarity score with the increasing number of hops between two connections. Hence, we conclude that when the distance of two connections is long (long stepping-stone connection chain), the accuracy of the detection may suffer.

On the positive side, all three of our schemes can find the gap of the dissimilarity score between attack and normal up to 6 hops. Figure 9 shows the worst case scenario, since each attack point in the figure represents the *maximum* dissimilarity score out of all 30 comparisons and each normal point represents the *minimum* dissimilarity score out of all 60 samples. All schemes return 0% false positive rate and 0% false negative rate among all the comparisons of up to 7 stepping-stone hosts (0 to 6 hops), while the attackers do not insert any chaff packets into the traffic flow.

F. Performance with Clock Skew Issue

To tolerate the clock skew in the detection, algorithm OSSM has used the *"interval"* along two adjacent packets to construct the detection. Our schemes use slope (delay) between a reference packet and a target packet to find the correlation. In this section, we investigate the impact of the clock skew on the discussed schemes. We use the same data as described previously and then inject a fixed length time delay for every packet in the target stream.

We present the performance results of having 3 seconds clock skew are illustrated in Table I. There are 100% FNR for DTW scheme and 17%~27% FNR for DTWW scheme. It indicates that using the sliding window operation does improve the performance of DTW scheme. However, because of the many-to-many operation and timestamp correlation, DTW and DTWW may falsely map a packet to several others and result in a high dissimilarity score. Although OSSM also uses the interval correlation, it supports one-to-one mapping operations. Therefore, it returns 0% false rate. Both DTWS and OSA schemes have 0% false rate, because they use slope correlation which does not change the difference when clock skew is present. Hence, using the slope correlation can resolve te clock skew issue well.

Table II. False Rates for DTW, OSSM, DTWW, DTWS and OSA algorithms. Data contains clock skew of 3 seconds.

HOP		ALGORITHMS				
		DTW	OSSM	*DTWW*	*DTWS*	*OSA*
	Corr.	*Time*	*Interval*	*Time*	*Slope*	*Slope*
0	FPR	0%	0%	0%	0%	0%
	FNR	93%	0%	17%	0%	0%
1	FPR	0%	0%	0%	0%	0%
	FNR	100%	0%	25%	0%	0%
2	FPR	0%	0%	0%	0%	0%
	FNR	100%	0%	17%	0%	0%
3	FPR	0%	0%	0%	0%	0%
	FNR	100%	0%	25%	0%	0%
4	FPR	0%	0%	0%	0%	0%
	FNR	93%	0%	27%	0%	0%
5	FPR	0%	0%	0%	0%	0%
	FNR	100%	0%	21%	0%	0%
6	FPR	0%	0%	0%	0%	0%
	FNR	100%	0%	21%	0%	0%

G. Performance with Chaff Involvement

It is very likely that intruders may inject chaff packets into the traffic and remove them later to decrease the possibility of being detected. Thus, we examine the tolerance of our schemes in the presence of chaff. We evaluate the performance of the algorithms when the chaff packets are involved by using the same data. The chaff packets obeying the Poisson distribution are inserted into the target stream at a chaff rate (*CR)*. As a restriction, the chaff packets can only be added on just one side of streams and the stream injected with chaff packets will be treated as the target stream. We assume adding chaff packets to either incoming streams or outgoing streams would not affect the overall results as long as only one side of streams is involved. The chaff rate is the number of chaff packets divided by the number of original packets on that particular injected stream. We assume that the chaff packets can only be added on just one side of streams.

The performance results are shown in Table II below. Unfortunately, DTW, DTWW and DTWS schemes are not able to tolerate the chaff packets even with a low 50% chaff rate (more than 80% false rate). OSSM is able to tolerate up to 150% chaff rate with a maximal of 4% FPR and 17% FNR.

Table I. False Rates with various chaff rate for the five algorithms.

HOP		ALGORITHM																	
		DTWS		OSSM				DTWW		DTWS			OSA						
	CR%	0	50	0	50	100	150	0	50	0	50	100	0	50	100	150	200	250	300
0	FPR	0%	0%	0%	3%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	3%	8%
	FNR	0%	100%	0%	0%	0%	0%	0%	93%	0%	83%	93%	0%	0%	0%	0%	0%	0%	0%
1	FPR	0%	0%	0%	3%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	3%	7%
	FNR	0%	100%	0%	0%	0%	0%	0%	93%	0%	83%	93%	0%	0%	0%	0%	0%	0%	0%
2	FPR	0%	0%	0%	3%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	3%	7%
	FNR	0%	100%	0%	0%	0%	0%	0%	93%	0%	83%	93%	0%	0%	0%	0%	0%	0%	0%
3	FPR	0%	0%	0%	3%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	2%	7%
	FNR	0%	100%	0%	0%	0%	0%	0%	93%	0%	90%	93%	0%	0%	0%	0%	0%	0%	0%
4	FPR	0%	0%	0%	3%	0%	3%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	1%	6%
	FNR	0%	100%	0%	13%	13%	7%	0%	92%	0%	90%	93%	0%	0%	0%	0%	0%	0%	0%
5	FPR	0%	0%	0%	4%	0%	4%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	1%	6%
	FNR	0%	100%	0%	25%	17%	17%	0%	92%	0%	93%	91%	0%	0%	0%	0%	0%	0%	0%
6	FPR	0%	0%	0%	4%	0%	4%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	1%	7%
	FNR	0%	100%	0%	25%	17%	17%	0%	92%	0%	93%	91%	0%	1%	0%	0%	0%	0%	0%

OSA scheme is able to tolerate up to 300% chaff rate with the maximal of 8% FPR and 0% FNR. This is because OSA was designed to skip chaff packets without penalty.

H. Performance Involved Both Clock Skew and Chaff

The previous experiments use only one threshold to compare. By varying the thresholds, we obtain different tradeoffs between false positive and false negative rates. We further investigate the performances of those schemes using The Receiver Operating Characteristic (ROC) curve. This curve could display the relative trade-offs between the true positive rate (benefits) and false positive rate (costs) for the schemes. Please note that true positive rate TPR = 1- false negative rate FNR. We know that the curves close to the upper left corner represent the better performance since the upper left corner represents 100% sensitivity (no false negatives) and 100% specificity (no false positives).

Figure 10 displays the ROC curves for DTW, OSSM, DTWS, OSA and OSSM. The results are taken from the average of all the hops (0 to 6 Hops) with a 0% to 200% chaff rate (50% CR increments) injected into the target streams. In addition, there are 3 seconds of clock skew. OSA scheme (0.96% FPR corresponds to 99.98% TPR) seems to uniformly dominate DTW, DTWW and DTWS schemes and it is somewhat better than OSSM scheme. Considering the problem's complexity, the result for OSA scheme is exciting.

VI. Conclusion

In this paper, we hypothesize that correlation detection between two traffic connections may exist across multiple hosts even with the presence of time skew and chaff packets. To validate it, we design three dynamic programming algorithms, inspired from pattern recognition techniques. These approaches rely on either repeatedly doing the comparison with a sliding windows, the interval correlation or the slope correlation. The connections analyzed in this work can be adjacent (as a special case) or across multiple hosts. The paper also aims to find ways to reduce the effect of clock skew and chaff issues while keeping relatively low time complexity.

Our experiments show interesting results under four different scenarios. If no clock skew or chaff packet exists, all three of our schemes gives 0% false rate in all of the tests up to a length of 7 stepping-stone hosts. When the clock skew is involved, DTWS and OSA schemes still remain 0% false rate by using slope correlation. Although DTWW scheme using sliding window operation shows a 27% FNR in our experiment, it does improve the performance of DTW scheme. We believe that this sliding window scheme is still valuable to be used in combinations with other pattern recognition schemes in order to deal with the clock skew issue. When the chaff packets are present, OSA scheme turns out to uniformly dominate other schemes. The similar result is concluded by looking at the ROC curve when involving both clock skew and chaff. In addition, Our novel scheme - OSA is a faster algorithm with a lower time complexity $O(nm^2)$, compared with the time complexity $O(n^2m^3)$ for OSSM scheme. Because OSA finds the slope correlations of two flows, not only clock skew and chaff packets are ignored but also the parameter of maximum tolerable delay can be omitted. Hence,

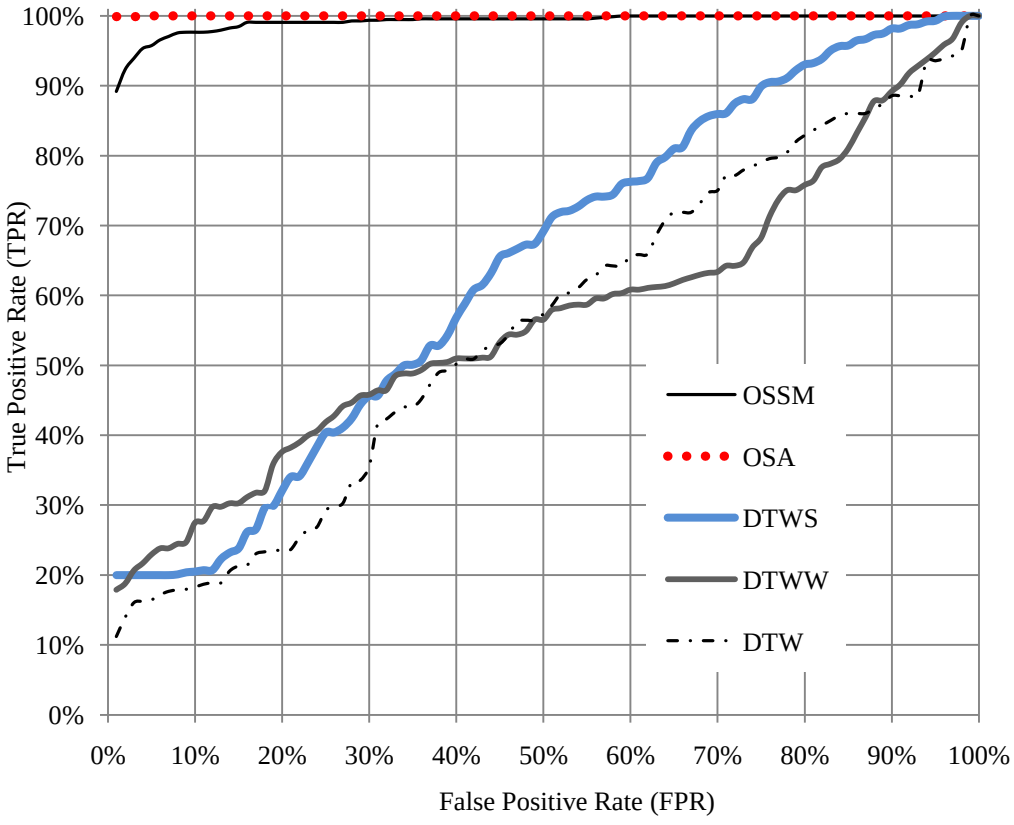


Figure 10. ROC curves for OSSM, OSA and DTWS schemes

OSA scheme can detect stepping-stone attacks with low time complexity, even in complex circumstances such as clock skew and chaff included.

These detection algorithms are useful in several situations. A large portion of intruders are actually insiders according to many studies. In such cases, the algorithm can be used to compare connections entering and exiting a local area network. The algorithm can also be used to confirm attacks from a host that we suspect (through other means). For example, if we suspect Host A is hacking into Host V, we can compare the suspected incoming stream to Host V and the outgoing packet streams of Host A. Our algorithm such as OSA can be used to confirm or reject such suspicion with reasonable accuracy. There is no need of collecting any packets on the intermediate hosts.

Acknowledgment

This research is supported in part by a grant from National Science Foundation (CNS- 0755500).

References

[1] K.H. Yung, "Detecting long connecting chains of interactive terminal sessions," In *Proceedings of of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID)*, pp. 1-16, 2002,.

[2] S. Huang, W. Ding, M. Hausknecht, Z. Riggle, "Detecting Stepping-Stone Intruders with Long Connection Chains, " *Journal of Information Assurance and Security (JIAS)*, Vol. 5, pp. 500-514, 2010.

[3] Y. Zhang and V. Paxson, "Detecting Stepping Stones", In *Proceedings of the 9th USENIX Security Symposium*, pp. 171-184, 2000.

[4] X. Wang, D. Reeves, and S. Wu, "Inter-Packet Delay-Based Correlation for Tracing Encrypted Connections through Stepping Stones", In *Proceedings of the 7th European Symposium on Research in Computer Security (ESORICS)*, pp. 244–263, 2002.

[5] X. Wang, and D. Reeves, "Robust Correlation of Encrypted Attack Traffic Through Stepping Stones by Manipulation of Inter-packet Delays", In *Proceedings of ACM Conference on Computer and Communications Security*, pp. 20-29, 2003.

[6] D. Donoho, A.G. Flesia, U. Shankar, V. Paxson, and S. Staniford, "Multiscale Stepping-Stone Detection: Detecting Pairs of Jittered Interactive Streams by Exploiting Maximum Tolerable Delay", In *Proceedings of the 5th Intl. Symposium on Recent Advances in Intrusion Detection*, pp. 244-263, 2002.

[7] A. Blum, D. Song and S. Venkataraman, "Detection of Interactive Stepping Stones: Algorithms and Confidence Bounds," In *Proceedings of the 7th Intl. Symposium on Recent Advances in Intrusion Detection (RAID)*, pp. 258-277, 2004.

[8] Y. Kuo and S. Huang, "Stepping-Stone Detection Algorithm based on Order Preserving Mapping", In *Proceedings of the 13th Intl. Conference on Parallel and Distributed Systems (ICPADS)*, pp. 1-8, 2007.

[9] Y. Kuo and S. Huang, "An Algorithm to Detect Stepping-Stones in the Presence of Chaff Packets", In *Proceedings of 14th IEEE Intl. Conference on Parallel and Distributed Systems (ICPADS)*, pp.485-492, 2008.

[10] D. Berndt and J. Clifford, "Using Dynamic Time Warping to Find Patterns in Time Series," In *Proceedings of AAAI-94 W. on Knowledge discovery and databases*, pp. 229-248, 1994.

[11] S. Chu, E. Keogh, D. Hart, and M. Pazzani, "Iterative Deepening Dynamic Time Warping for Time Series," In *Proceedings of the 2nd SIAM Intl. Conference on Data Mining*, pp. 148–156, 2002.

[12] T.M. Rath and R. Manmatha, "Word Image Matching uses Dynamic Time Warping," In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pp. 521–527, 2003.

[13] P. Tormene, T. Giorgino, S. Quaglini and M. Stefanelli, "Matching Incomplete Time Series with Dynamic Time warping: An Algorithm and An Application to Poststroke rehabilitation," *Artificial Intelligence in Medicine Journal*, Vol. 45, pp. 11-34, 2009.

[14] B. Burg, P.H. Missakian, and B Zavidovique, "Pattern Recognition through Dynamic Programming," In *Proceedings of SPIE's 29th Annual Internal Technical Symposium*, 1985.

[15] H. Sakoe and S. Chiba, "Dynamic Programming Algorithm Optimization for Spoken Word Recognition", *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol. 26, pp. 43- 49, 1978.

[16] Y. Kuo, S. Huang, W. Ding, R. Kern and J. Yang, "Using Dynamic Programming Algorithms to Detect Multi-Hop Stepping-Stone Connection Pairs in a Long Chain," In *Proceedings of the International Conference on Advanced Information Networking and Applications (AINA)*, pp. 198-205, 2010.

[17] Y. Kuo, S. Huang and C. Hill, "Detect Multi-Hop Stepping-Stone pairs with Clock Skew," In *Proceedings of the Sixth International Conference on Information Assurance and Security (IAS)*, pp. 74-79, 2010.

Author Biographies

Ying-Wei Kuo is currently a Ph.D. Student in the Department of Computer Science, the University of Houston, United States. She received a B.S. degree in computer science from the University of Otago, New Zealand and a M.S. degree in computer science from Oklahoma City University, United States. She was a Computer Engineer at the Inventec Corporation, and developed BIOS related software for Toshiba and HP laptops using C. Her current research focuses on designing efficient and highly accurate stepping-stone detection algorithms in order to detect the intruders.

Shou-Hsuan Stephen Huang is professor of Computer Science at the University of Houston, Houston, TX, USA. His research interests include data structures and algorithms, computer security, and intrusion detection. He received his BS degree in Mathematics from National Cheng Kung University, Tainan, Taiwan and his MS and Ph.D. degrees in Computer Science from the University of Texas-Austin. He is a member of the ACM and a senior member of the IEEE Computer Society.