# Correlating Temporal Thumbprints for Tracing Intruders

Jianhua Yang[1], Shou-Hsuan Stephen Huang[2]
[1]The Department of Mathematics and Computer Science, Bennett College
900 E. Washington Street, Greensboro, NC, 27401, USA
[2]Department of Computer Science, University of Houston
4800 Calhoun Rd. Houston, TX, 77204, USA

## ABSTRACT

The Design of TCP/IP protocol makes it difficult to reliably traceback to the original attackers if they obscure their identities by logging through a chain of multiple hosts. A thumbprint method based on connection content was proposed in 1995 to traceback attackers, but this method is limited to non-encrypted sessions. In this paper, we propose a thumbprint based on time intervals, T-thumbprint, to identify a connection. T-thumbprint is a sequence of time gaps between adjacent TCP 'Send' packets of an interactive terminal session. An algorithm is presented to correlate two T-thumbprints to see if they belong to the same connection chain. We also discuss how to use T-thumbprints to traceback an attacker on the Internet, and how to defeat attacker's manipulation. T-thumbprint has advantages of: (1) It can be applied to encrypt sessions; (2) It does not require tightly synchronized clocks; (3) It can defeat attacker's manipulation to some extent; and (4) It is efficient, can be used to trace attackers in real time.

## 1. INTRODUCTION

People depend on the Internet more and more in daily life and business activities. Research has shown that various attacks through the Internet have increased significantly with the growth of the Internet [7]. To hide their identities, most attackers log though a series of compromised hosts before launching their attacks, and the compromised hosts are called stepping-stones [3]. Attackers can avoid their responsibilities for their activities by using stepping-stones because of known spatial, political and cooperation reasons.

To prevent interactive connection attacks, many techniques have been developed. One of them is to prevent a machine from being used as a stepping-stone, that is to detect if there is a connection chain going through a particular host, so as to take some countermeasures to protect the victims, such as notify the administrator or directly cut off the chain [3, 4, 8]. Some other methods used to protect victims are to trace back to the attacker's host [1, 6, 9, 10]. Traceback techniques are classified into two categories: (1) IP traceback, and (2) connection traceback. Research on IP traceback has become rather active since 1999 because of DDOS attacks [11, 12]. IP traceback can't go beyond the hosts that send the spoofed IP packets. Typically, a proficient attacker often launches attack with the help of other hosts. So only identifying the source of IP packets is not sufficient to hold the attackers responsible for their actions. Techniques for traceback across stepping-stones have become more important under this situation. There are many papers published in traceback across stepping-stones [1, 3, 6, 13, 14, 15]. Most of these technologies are either network-based or host-based. The representatives of host-based approaches are DIDS (Distributed Intrusion Detection System) [14], CIS (Caller Identification System) [6], and Caller ID [1]. The typical network-based approaches are thumbprint [1], time-based approach [3], and deviation-based approach [13].

DIDS has two problems: (i) it can't be applied to large-scale network because of its centralized analyzing server; (ii) it fails to traceback if one host is unavailable to provide the user's audit log. The main problem with CIS is that the network load will be increased because the query incurs additional communication when the user tries to login one host from another. The drawbacks of Caller ID [1] include the possibility that counter-intruders cannot break into one of the compromised hosts, as well as running the risk of accidentally damaging intermediate systems.

With the help of thumbprints, which is a short summary of the contents of a connection, we can uniquely identify a connection and correlate those related connections. Because the thumbprint proposed in [1] requires only 24 bytes per minute per connection, and it is impossible to infer the content from the thumbprint, so it has the advantages: (1) efficiency; (2) secrecy; and (3) ease of implementation. But it faced a fatal problem that it cannot be applied to encrypted sessions (such as sessions established by SSH) because the thumbprint comes from the contents of a connection. Time based and deviation based approaches [3, 13] are techniques using the distinctive characteristics (such as packet size, time stamp, etc.) of interactive traffic, rather than using the contents of a connection. They address the content-dependency weakness of thumbprint method. They can be applied to encrypted sessions, and are more robust against retransmission variation.

This paper's work is to combine the previous two schemes together to propose a novel form of thumbprint, which is temporal thumbprint (T-thumbprint). T-thumbprint is a sequence of time gaps based on time stamps of the send packets in a connection. Compare to the methods in [1, 3, 6, 13, 14, 15], T-thumbprint has the advantages of efficiency, secrecy, robustness, and ability to defend against attacker's manipulation.

Section 2 will show a definition of T-thumbprint in details and its correlation algorithm. Section 3 addresses T-thumbprint's ability to identify connection chains. Section 4 will discuss how T-thumbprints may be used to traceback intruders and how to defend against intruder's manipulations. And finally in Section 5, some conclusions and future works will be given.

## 2. T-THUMBPRINT AND ITS CORRELATION ALGORITHM

Assuming that we are monitoring an interactive connection session between a client (Host 1) and a SSH server (Host 2), typically, we will observe a 'Send' packet from Host 1 to Host 2 followed by an acknowledgment and an 'Echo' packet [16] from the server. In T-thumbprint, we are interested in the Send packets originating from Host 1 only so as to not limit this approach on interactive sessions.

### 2.1 Temporal Thumbprints

Given a sequence of 'Send' packets $<p_1, p_2, \ldots, p_{n+1}>$ from Host 1 to Host 2, let $<t_1, t_2, \ldots, t_{n+1}>$ be the corresponding time stamps of $<p_1, p_2, \ldots, p_{n+1}>$. A temporal thumbprint (T-thumbprint) of the connection is defined to be the sequence $<t_2-t_1, t_3-t_2, \ldots, t_{n+1}-t_n>$. Each element represents a time gap between two successive Send packets. Essentially, we are characterizing a connection between two hosts by the time gaps of the packets. So we have an abstract problem of correlating two such T-thumbprints to see if they are close enough.

The length of a thumbprint depends on the number of send packets collected. We can certainly divide up the thumbprint into subsequences for each time unit (such as a 1-minute interval in [1]). The selection of n depends on the network. For local network, n can be relatively small because of less network fluctuation, usually 8. But for wide area network, n needs to be a little bit larger than on local network because of substantial network fluctuation, usually 32, 64, or 128. The larger the size, the easier it is to identify the connection, but less efficient. Similarly, we define T-thumbprint for incoming connection as well. We call the latter one incoming T-thumbprint, denoted as iT-thumbprint, and the former one outgoing T-thumbprint, denoted as oT-thumbprint. For convenience, we usually use T-thumbprint to represent both incoming and outgoing temporal thumbprint.

For the purpose of tracing intruders, the most important issue is how to determine one T-thumbprint is close to another one. For example, we would like to determine if a given iT-thumbprint is similar to an oT-thumbprint of the same host, or to determine if oT-thumbprint of one host is close to oT-thumbprint of another host so as to decide if the two hosts are in the same connection chain. We use MR (matching rate), which is the ratio between the number of matched elements and the number of elements of a T-thumbprint, to determine if two T-thumbprints match. In the following sections, we shall use array a[1..n], and b[1..m] to represent the two T-thumbprints. We define two elements x and y from two thumbprints are "close enough" if abs(x-y)/((x+y)/2) is less than a predefined threshold ε.

```
// A , and B are two T-thumbprints with elements a[1..n] and b[1..m] respectively.
// Range is a range to check within, T_Max, T_Min are thresholds
Initialize current position CurrPA, CurrPB for A, and B
    // get all the one-to-one match pairs
while (there are more elements in B and A) {
    // counter is the number of matching elements between A and B
    // NumA, NumB are the number of elements in A, B respectively
  lbA=CurrPA;        lbB=CurrPB;
  ubA=CurrPA+Range; ubB=CurrPB+ Range;
  Match=false;
  i=lbB;
    while(i<=ubB-2 && !Match){
      j=lbA;
      while(j<=ubA-2 && !Match){
      ra1=2*|b[i]-a[j]|/(a[j]+b[i]);
      ra2=2*|b[i+1]-a[j+1]|/(a[j+1]+b[i+1]);
      ra3=2*|b[i+2]-a[j+2]|/(a[j+2]+b[i+2]);
      if(ra1, ra2, ra3 are all less than ε ){
        MP[i] = j;  //save the matching in MP
        counter++;
        CurrPA=j+1; CurrPB=i+1;
        Match=true;
      }
      j++;
    }
    i++;
  }
  if (Match) {currPA=j; currPB=i;}
  else {currPA++; currPB++;}
}
    // Merge the unmatched segments
while(there are undefined elements in MP){
   if MP[i1]=j1 and MP[i2]=j2 and all MP[i] are
     undefined for i1<i<i2, merge all a[i] (i1<i<i2)
         into one
     element s1 and a[j] (j1<j<j2) into one element s2.
   if (2*|s1-s2|/(s1+s2)<ε) counter++;
}
matching rate MR=counter/min(NumA , NumB);
if(MR >T_Max) A is close to B
else if(MR<T_Min) A is not close to B
else undecided;
```

Algorithm 1: T-thumbprint correlating algorithm

### 2.2 Challenges of Correlating Sequences

We can't always assume that elements between two T-thumbprints match exactly. Fig 6 shows us a scenario that Host 1, Host 2 and Host 3 are connected by one SSH connection chain. If all the Send packets from Host 1 to Host 2 are forwarded exactly to Host 3, correlating such T-thumbprints would be trivial. But for most cases, especially on the Internet, it is more difficult than on a local network. Because even if in a same chain, the Send packets sent from Host 3 does not have a one-to-one relationship with the Send packets of Host 2. This makes correlating two T-thumbprints more complex.

To understand the scenario how a packet propagates on the Internet, we need to be clear how TCP and SSH protocol work [2, 16]. We assume there is a packet sent from Host 1 to Host 2, this packet would be decrypted first and then encrypted at Host 2, finally be forwarded to Host 3. This procedure is repeated until this packet reaches the end machine of the chain. There is a

possibility in packet delivery procedure that the packet would be divided into several packets or merged into a big packet.

The first reason is that after a certain period of connection time or each gigabyte of transmitted data between two adjacent hosts the encryption key will be re-exchanged, and this communication is happened only between two adjacent hosts, the packet is not forwarded to the downstream Host. The packets sent from upstream host should be more than the packets sent from downstream host. The second reason is lost packet retransmission. Suppose that a packet sent from Host 1 to Host 2 is lost during transmission for the reason either Host 2 does not receive that packet or Host 1 does not receive any acknowledge packet from Host 2, Host 1 would resend that packet until Host 2 acknowledges it. The third reason is that randomly ignore packet sent to server side from client side for security reason. Client side will randomly send some packets that are marked as ignore packets to server side. Once server side receives the ignore packet, it neither responds nor forwards, while what server side does is to acknowledge client side. So ignore packet transmission would result in T-thumbprint being not one-to-one. The fourth reason is that the packet may be fragmented during delivery process on the Internet. Either a packet size is more than the maximum size allowed on the Internet or travels from IPv6 to IPv4, the packet will be fragmented. The fifth reason is that the attacker may inject some characters (or random delay some packets) to the chain in order to defeat some traceback approaches, such as the approach in [3]. So attacker's manipulation on connection chain would also result in T-thumbprint difficult to match. Our algorithm is able to tolerate some of these problems stated. Experiment result showed that our algorithm can correlate two T-thumbprints if an attacker injects no more than 35% characters.

### 2.3 T-thumbprint Correlating Algorithm

Suppose we have two sequences A: a[1..n] and B: b[1..m]. Each sequence represents one T-thumbprint. We also assume that element a[1] matches with element b[1]. We cannot claim that each element in B is exactly matched with a element in A or each element in A is exactly matched with a element in B because of T-thumbprint asymmetry. We need to compute MR to determine if two T-thumbprints are matched.

The first easiest way to do this is to take one element from B, to compare to each element in A, to see if this element in B matches with any element in A by checking if they are 'close enough'. The problem with this approach is that there may be several elements in A matches with the same element in B. So in our algorithm three consecutive elements in A are checked if they are 'close enough' to three consecutive elements in B to determine one pair matching. It will largely decrease false positive rate. We already know a[1] matches with b[1], but it doesn't mean that b[2] must match with a[2]. The element b[2] in sequence B probably matches with a[2] or a[3] or some other elements. We divide the algorithm into two phases: the first phase matches elements between A and B one-to-one; the second phase matches the remaining unmatched elements.

In the first phase, if a[i] doesn't match with b[j], instead of moving to match b[j+1] with a[i+1], we continue to check if b[j] matches with elements a[i+1] etc. Symmetrically, we compare a[i] with b[j], b[j+1], etc. If there is no match after a number of comparisons (say 5), we increase both i and j and continue the process. If there is a match, we set up the current position for each sequence to the position next to the matching element in each sequence. In the second phase, we group all the unmatched elements between two matched elements and treat it as one and see if this new element will match with the corresponding one on the other sequence. For example, suppose in first scan, we have matched two one-one pairs, assuming they are <a[2], b[2]>, and <a[5], b[7]> respectively. In second phase, we just simply check if sum of a[3] and a[4] matches with sum of b[3], b[4], b[5], and b[6]. Algorithm 1 shows the correlating algorithm in details. The output of Algorithm 1 is the matching rate between the two sequences A and B.

## 3. IMPLEMENTATION

We set up the test environment in the Computer Science Department Lab at the University of Houston to collect network packets. Our objective is to test if we can use T-thumbprint to traceback to intruders and how is the performance of T-thumbprint.

### 3.1 Test Environment and software

We have two Linux hosts (Red Hat 9.0) acl08 and acl09 with administrative privileges. We need to monitor and capture all the packets going through the NIC of acl08 and acl09, where the network bandwidth is 100M. We use some other "source" machines to simulate several connections going through the two monitored machines. The source machines, called Host 1, … , Host 4, are located in computer science Lab running Linux operating system.

We have obtained access rights from other hosts without administrative privileges. One is "Mex", which is located in Mexico. The second is "Epic", which is located in California. The third one is Bayou, which is located on University of Houston campus. With the above hosts we formed connection chains Host i (i=1 to 4)→ Acl09 → Mex → Acl08 → Epic → Bayou by using SSH. In the above chains, Bayou is supposed to be the victim, and Host 1, … , Host 4 are the source machines, while other hosts in between are compromised stepping-stones.

We implemented a program called Temporal Thumbprint Traceback (TTT), which only supports Ethernet for simplicity, with Libpcap [5] to capture T-thumbprint. Libpcap is a free Packet Capture Library made by Lawrence Berkeley National Laboratory.

### 3.2 Test Procedure

We established four connections, Host i →Acl09 →Mex →Acl08 →Epic →Bayou, where i=1 to 4., while the four connections share the same path. We run the program TTT on Acl08 and Acl09 to capture the packets passing through them, and form T-thumbprints at Acl08 and Acl09. Our objective is to see if we can identify and match connections at the two monitored hosts. To make it more challenge, we set up several connections through the same chain of hosts. Four individuals operated at Host 1, Host 2, Host 3, and Host 4 respectively at the same time, and each person input the same content at his/her own typing speed. We captured all four T-thumbprints on each connection passing through Acl08 and Acl09. Consequently, what we need to do is to verify if we can use the four oT-thumbprints to pick up four connections. The results are showed in Section 3.3.

The reason why we make the connections we built shared the same path, the same location, the same contents, and the same time is to create the worst possible situation to find the separate identities for the four users. In the real world, most probably, the intruders on the Internet won't operate around same location and input same thing at same time as other users. We try to use the connections what we set up to simulate the worst case. If T-thumbprint could deal with this type of worst case, it can deal with all kinds of cases.

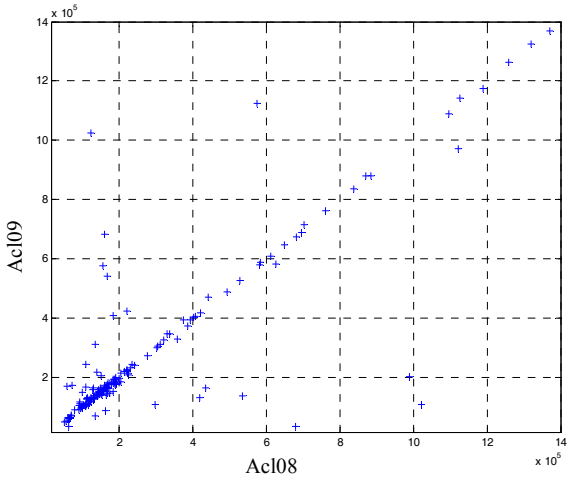


**Fig 1.** Matched T-thumbprints correlation before processing

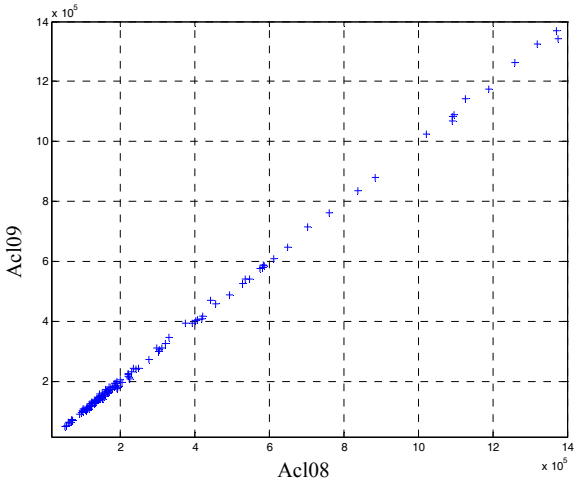


**Fig 2.** Matched T-thumbprints correlation after processing

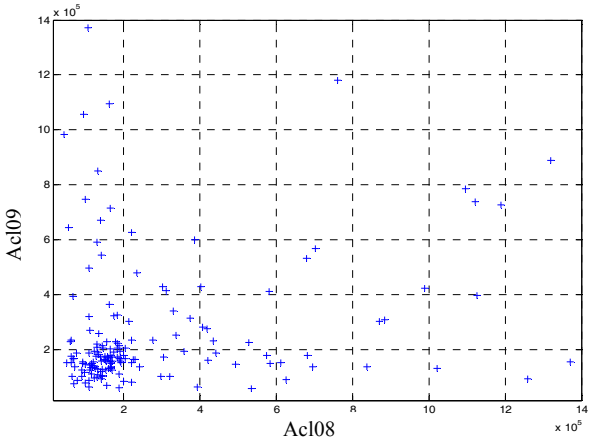


**Fig 3.** Unmatched T-thumbprints correlation before processing

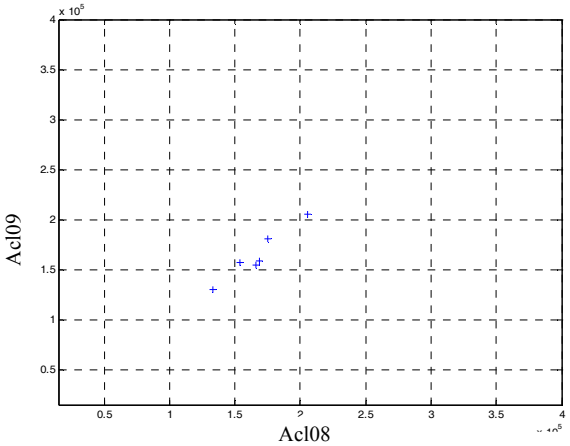


**Fig 4.** Unmatched T-thumbprints correlation after processing

**Table 1.** T-thumbprint correlating results between two hosts on the Internet.

| Connection at Acl09 | Connection at Acl08 | | | |
|---|---|---|---|---|
| | C0(%) | C1(%) | C2(%) | C3(%) |
| C0 | 92.37 | 0.00 | 0.57 | 0.52 |
| C1 | - | 84.00 | 0.57 | 0.00 |
| C2 | - | - | 87.42 | 0.62 |
| C3 | - | - | - | 89.00 |

## 3.3 Experiment Results and Analysis

We are going to show the results of (1) Given T-thumbprints, if we could identify the same chain among those connections; (2) The function of correlating algorithm; (3) The performance of T-thumbprint.

We got four oT-thumbprints at Acl08, and Acl09 respectively, and we use time interval that one packet travels from one host to

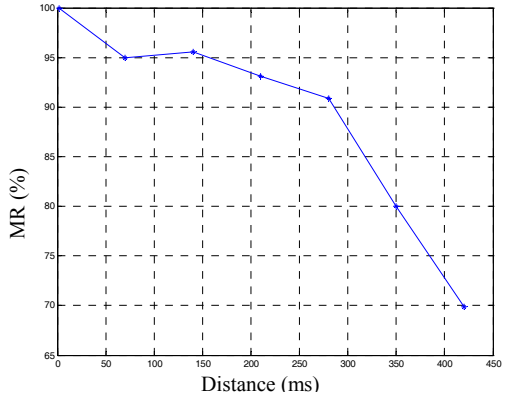


**Fig 5.** Performance of T-thumbprint

another to represent the distance between the two hosts. The distance between Acl08 and Acl09 in the experiment is about 70 ms long. Table 1 shows the result of MRs between different connections, it clearly shows the strong correlation between the connections. In Table 1, C0, C1, C2, and C3 represent outgoing connections of Acl08 and Acl09 respectively, and the values displayed in this table are MRs between one connection at Acl08 and another connection at Acl09.

Fig. 1, Fig. 2, Fig. 3, and Fig. 4 show us the performance of Algorithm 1. Fig. 1 and Fig. 2 show the scenarios of two matched T-thumbprints before and after processing with Algorithm 1, while Fig 3, and Fig 4 show the scenarios of unmatched case before and after processing by Algorithm 1, (unit used in these figures are microseconds). Comparing Fig. 1 to Fig. 2, and Fig 3 to Fig 4, it is not difficult to find that the ability of Algorithm 1 is to determine if two T-thumbprints are in the same chain.

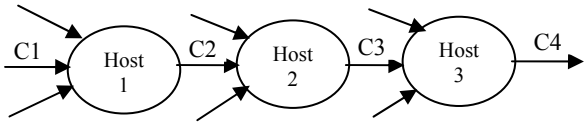


**Fig 6.** Illustrating how to use T-thumbprint to trace back

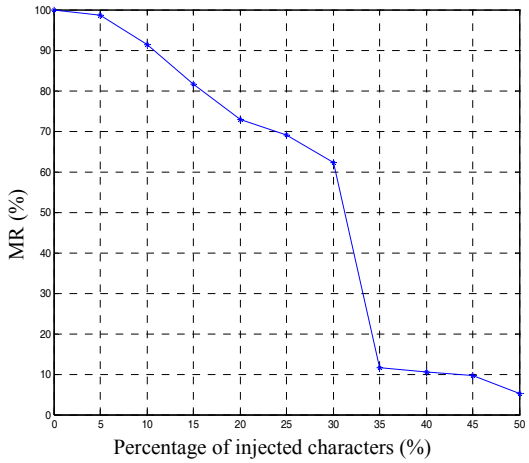


**Fig 7.** Scenario of using T-thumbprint to defeat attacker's chaff character manipulation

We use adaptability to measure T-thumbprint performance. The adaptability is defined as the variations of MR between two T-thumbprints over the distance between two hosts. Fig. 5 shows the performance of T-thumbprint, where Y axis stands for the MRs of two T-thumbprints, and X axis stands for the distance between two hosts. Fig 5 shows that if the distance is increased, the MR will be decreased for the T-thumbprints in the same chain. The method we used to increase the distance between two hosts is to connect another two hosts more times because there isn't any method to purely increase the distance between two fixed hosts without inserting any other hosts between them. So long as we introduce more hosts, it should affect the performance of the T-thumbprint negatively. In fact, the real performance of T-thumbprint is better than what is showed in Fig. 5, therefore Fig. 5 shows us only the lower bound performance of T-thumbprint.

# 4. TRACEBACK WITH T-THUMBPRINT AND DISCUSSION

## 4.1 Traceback with T-thumbprints

There are two ways to use T-thumbprints to traceback. One is to use oT-thumbprints only to do traceback. Another one is to use oT-thumbprints and iT-thumbprints together to trace back. Fig 6 shows the scenario that several hosts are connected by one chain <C1, C2, C3, C4> and used as stepping-stones, where Ci's are the connections and T-thumbprint$_i$ represents the corresponding thumbprint

The first way to do traceback is only to use oT-thumbprint. We have oT-thumbprint$_4$ at Host 3 for outgoing connection C4. What we need to do first of all at Host 3 is to request all outgoing T-thumbprints of each upstream host that connect to Host 3 directly. The second step is to correlate oT-thumbprint$_4$ with all other oT-thumbprints requested at Host3 to decide which connection is in the same chain with C4. Similarly, we can do the same thing in Host 2 as what we do in Host 3 to trace back which connection among all the incoming connections of Host 2 is in the same chain with C3. Recursively, we will eventually trace back to the intruder only with outgoing T-thumbprints. The problems with this way are inefficient, overloading the network and difficult to synchronize oT-thumbprint. To overcome the shortcomings of this way, we have another way to do traceback.

The second way is to combine incoming T-thumbprint with outgoing T-thumbprint to traceback intruders. Unlike the previous way, it is not necessary to transfer oT-thumbprint over network. What it needs to do in the first step is to correlate oT-thumbprint4 with all of the iT-thumbprint at Host 3 to decide which incoming connection is in the same chain with C4. The second step is to request Host 2 which incoming connection of Host 2 is in the same chain with C3. We do the same thing recursively and will eventually get where the intruder is. The main advantage of this way is we can guarantee the iT-thumbprint and oT-thumbprint are in the same time interval because we use the same process on the same host to collect the incoming and outgoing T-thumbprint at the same time. Another important issue is that we can traceback in real time. T-thumbprint is not large. Typically, it is 8x4 bytes long on local network, and 128x4 bytes long on the Internet. T-thumbprint could be generated and correlated within 1 second. So we could traceback attackers in real time by using T-thumbprints. Due to the space limitation, we will discuss the detail in another paper.

## 4.2 Attacker's Evasion

We have discussed the possibility to traceback intruders with T-thumbprint. Most probably, intruders who are aware of the risks of being traced try to evade the trace-back by modify their connections. To prevent the T-thumbprint detection, they may randomly delay the outgoing packets or randomly inject some characters into the connection so that the outgoing and incoming connections appear unrelated. T-thumbprint method depends on intruder's keystroke speed. V. Paxson [17] showed that user's keystroke should obey Pareto distribution. If more characters are inserted into the stream, it is very difficult to maintain Pareto distribution without carefully processing. We can detect the intruder's invasion by simply checking if the T-thumbprint

breaks Pareto distribution. This method doesn't always work because if the insertion is processed carefully, the intruder can still make the stream keeping Pareto distribution, however, at least T-thumbprint makes the intruder work harder. In the following, we are going to give the analysis that even if the manipulated T-thumbprints obey Pareto distribution, we still can correlate them but with some limitations.

One fact is that the intruders can only delay the outgoing packets, rather than accelerating them, and another fact is that intruders can't tolerate much long delay, which means there is an upper bound for intruder's packet delay. Suppose we have two time sequences N1(t), and N2(t) to represent two T-thumbprints respectively, where N1(t) is the original sequence, and N2(t) is the manipulated sequence. We make the following two assumptions: (1) The character emerges in the manipulated sequence N2(t) if and only if it has emerged in its original sequence N1(t). (2) If one character emerges in its original sequence N1(t), it must emerge in its manipulated sequence N2(t) within a certain time interval. David L., et al, [17] pointed out in theory that the two sequences are still related under the above two assumptions. That is, the scaling coefficients of the two sequences wavelet transform must be very close at long time scales. So even if in time domain, we couldn't correlate the two sequences (one is the original one, and the other one is the transformed one by packet delay), but we still can correlate them in frequency-time domain. The only problem is that we need to monitor the chain for a longer time. Algorithm 1 won't work on frequency-time domain because it is time domain based, and we are still working on frequency-time domain T-thumbprint correlating algorithm.

Suppose the attacker manipulate one connection by randomly injecting some characters into the chain. The experiment results in Fig. 7 showed that our approach could defeat attacker's injecting chaff attack in certain extant. Here we still assume that we have one time sequence N1(t) to represent the original T-thumbprint, and N2(t) to represent the injected T-thumbprint. N2(t) is generated by our simulating program with different injecting rate. At each certain injecting rate, we get one MR of N1(t) and N2(t) by using Algorithm 1. The results showed that when attacker add up to 35% characters to N1(t), Algorithm 1 is able to correlate N1(t) and N2(t).

## 5. CONCLUSIONS

We have proposed a new time-based thumbprint to correlate connections and traceback intruders on the Internet. The results showed it works on the Internet and has advantages of (1) It can be applied to encrypt sessions; (2) It does not require tightly synchronized clocks; (3) It can defeat attacker's manipulation in certain extent; (4) It is efficient, can be used to do trace-back in real time. Another important application of T-thumbprint is to detect stepping-stone, but with high false positive rate.

We don't need to install the software TTT in all hosts. For example, for the scenario that one intruder connects out from a local network and connects back to that local network finally, we just need the gateway of that local network to install this software. All the outgoing and incoming connections are monitored and for each connection one T-thumbprint is generated. If we correlate the incoming T-thumbprint and the outgoing T-thumbprint, it is not difficult to get the inside intruder. There are still some problems with the T-thumbprint traceback. It is vulnerable to intruder's manipulation even though we have shown some counter measures in the literature.

## 6. REFERENCES


1. Stuart Staniford-Chen, L. Todd Heberlein, "Holding Intruders Accountable on the Internet", **Proceedings of the 1995 IEEE Symposium on Security and Privacy**, Oakland, CA, May 1995, pp.39-49.
2. T. Ylonen, "SSH—Secure Login Connections Over the Internet", **In 6th USENIX Security Symposium,** San Jose, CA, USA, 1996, pp. 37-42.
3. Yin Zhang, Vern Paxson, "Detecting stepping-stone", **Proceedings of the 9th USENIX Security Symposium**, Denver, CO, 2000, pp. 67-81.
4. Kwong H. Yung, "Detecting Long Connecting Chains of Interactive Terminal Sessions", **RAID 2002**, 2002, pp. 1-16.
5. Lawrence Berkeley National Laboratory (LBNL), "The Packet Capture library",. **ftp://ftp.ee.lbl.gov/libpcap.tar.z,** accessed March 2004.
6. H. Jung (ed.), "Caller Identification System in the Internet Environment", **Proceedings of 4th USENIX Security Symposium,** 1993, pp. 17-32.
7. CERT, **http://www.cert.org**, accessed March 2004.
8. J. Yang, S. Huang, "A Real-Time algorithm to Detect Long Connection Chains of Interactive Terminal Sessions", **Proceedings of InfoSecu04**, Shanghai, China, 2004, pp.198-203.
9. S. Savage, D. Wetherall, A. Karlin, T, "Anderson: Network Support for IP Traceback", **ACM/IEEE Transactions on Networking**, Vol. 9, No. 3, 2001, pp. 226-237.
10. A. Snoeren, C. Partridge, L. Sanchez, C. Jones, F. Tchakountio, B. Schwartz, S. Kent, W. Strayer, "Single-Packet IP Traceback", **ACM/IEEE Transactions on Networking**, Vol. 10, No. 6, 2002, pp. 721-734.
11. J. Elliott, "Distributed Denial of Service Attack and the Zombie Ant Effect", **IP Professional**, March/April 2000.
12. Computer Emergency Response Team (CERT), "CERT Advison CA-2000-01 Denial-of-service developments", **http://www.cert.org/advisories/CA-2000-01.html**, January 2000.
13. K. Yoda, H. Etoh, "Finding Connection Chain for Tracing Intruders", **Proceedings of the 6th European Symposium on Research in Computer Security (LNCS 1985)**, Toulouse, France, October 2000, pp. 31-42.
14. S. Snapp (ed.), "DIDS (Distributed Intrusion Detection System)-Motivation, Architecture, and Early Prototype", **Proceedings of 14th National Computer Security Conference**, October 1991, pp. 167-176.
15. C. Ko, D.A. Frincke, T. Goan Jr.(ed.), "Analysis of An Algorithm for Distributed Recognition and Accountability", **Proceedings of the 1st ACM Conference on Computer and Communication Security**, 1993, pp. 154-164.
16. University of Southern California, "Transmission Control Protocol", **RFC 793**, September 1981.
17. D. L. Donoho (ed.), "Detecting Pairs of Jittered Interactive Streams by Exploiting Maximum Tolerable Delay", **RAID 2002**, 2002, pp. 45-59.