Mining and Detecting Connection-Chains in Network Traffic

Ahmad Almulhem and Issa Traore

ISOT Research Lab,
ECE Department,
University of Victoria,
Victoria, CANADA

Summary. A *connection-chain* refers to the set of connections created by sequentially logging into a series of hosts. Attackers typically use connection chains to indirectly carry their attacks and stay anonymous. In this paper, we proposed a host-based algorithm to detect connection chains by passively monitoring inbound and outbound packets. In particular, we employ concepts from association rule mining in the data mining literature. The proposed approach is first explained in details. We then present our evaluations of the approach in terms of real-time and detection performance. Our experimentations suggest that the algorithm is suitable for real-time operation, because the average processing time per packet is both constant and low. We also show that by appropriately setting underlying parameters we can achieve perfect detection.

Key words: Connection chain, Stepping stone, Tracing, Traceback, Network forensics, Network security

1 Introduction

In order to provide a stronger level of security, most organizations use a mixture of various technologies such as firewalls and intrusion detection systems. Conceptually, those technologies address security from three perspectives; namely prevention, detection, and reaction. We, however, believe that a very important piece is missing from this model. Specifically, current technologies lack any investigative features. In the event of attacks, it is extremely hard to tie the ends and come up with a thorough analysis of how the attack happened and what the steps were. We believe the solution is in the realm of Network Forensics; a dedicated investigation technology that allows for the capture, recording and analysis of network events for investigative purposes [1]. The current practice in investigating network security incidents is a manual and brute-force approach. Experienced system administrators generally conduct it. Typically, investigation proceeds by processing various types of logs, which are located in a number of places. Brute force investigation however is a time consuming and error-prone process. It also can be challenging because the mentioned logs are not meant for thorough investigation. The logs may lack enough details or contrarily have lots of unrelated details. In this regard developing investigative tools that can assist and automate network forensics process is essential. In this paper, we present the foundation of a data mining tool that can assist network forensics analyst in automatically detecting connection-chains in network traffic data, which represent an important but challenging aspect of network forensics.

The term *connection-chain* refers to the set of connections created by sequentially logging into a series of hosts, known as *stepping-stones* [2, 3]. Attackers typically use connection chains to indirectly carry their attacks and stay anonymous. As such, several approaches have been proposed in the literature to detect them. We refer the interested reader to our review paper for a taxonomy and a detailed discussion of these approaches [4].

In this paper, we propose a host-based technique to detect connection-chains. In general, the main disadvantage of the host-based approaches proposed so far in the literature is that they are operating system specific [5, 6, 7]. Specifically, they are expected to be re-designed and re-implemented differently for different operating system. Also, it is not obvious if they can be applied to proprietary operating systems such as MS Windows.

To avoid being operating system specific, we adopt a black-box approach. In essence, inbound and outbound packets at a host are *passively* monitored to detect if there is a connection-chain. In particular, we employ concepts from association-rule mining from the data mining literature. Agrawal et al. were first to introduce association rules mining concepts, and demonstrate their usefulness in analyzing a database of sales transactions (*market basket transactions*) [8].

The rest of the paper is organized as follows. In section 2, we summarize and discuss related work on host-based approaches for connection-chains detection. In section 3, we give some background knowledge on association rule mining. In section 4, we present our detection framework by presenting our connection-chain mining approach and algorithm. In section 5, we describe the experimental evaluation of the proposed approach, and present and discuss the obtained performance results. Finally, in section 6, we make some concluding remarks.

2 Related Work

Several host-based detection techniques have been proposed in the literature. They can be broadly classified into two main classes. In the first class, processes at the concerned host are searched to find out if two connections are part of a connection chain [6, 7]. The idea is that if an outbound connection is created by an inbound one, then their corresponding processes should be "related". The main concern in this approach is that the search process may fail if the link is involved. For instance, this can be the case when the related processes are created through deeply nested pipes.

In the second class, an operating system itself is modified to support linking an outbound connection to an inbound one. Buchholz and Shields proposed special data structures and system calls to achieve the desired linking [5]. In particular, for each process, a new data structure `origin` is stored in its process table. For processes created by a remote connection, `origin` holds the typical 5-tuple information associated with that connection. For locally created processes, `origin` is undefined. When a process `forks` another one, `origin` is as usual inherited. The main concern in this approach is that modifying an operating system can be costly and might break already running software.

3 Background

3.1 Association Rules Mining

In the data mining field, *association rules mining* refers to a methodology that is used to discover interesting relationships in large data sets [9]. Specifically, the term *association rules* is used to denote the discovered relationships, while the process itself is called *mining for association rules.*

Formally, let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of *items.* Let $T = \{t_1, t_2, \ldots, t_N\}$ be a set of transactions, where each *transaction* t_i contains a subset of items from I, i.e. $t_i \subseteq I$. An *itemset* is also defined as a set of items. An *association rule* is an implication of the form $X \rightarrow Y$, where X and Y are disjoint itemsets, i.e. $X \bigcap Y = \phi$. The strength of an association rule is typically measured by its *support (s)* and *confidence (c).* The support implies that X and Y occur together in $s\%$ of the total transactions. On the other hand, the confidence implies that, of all the transactions containing X, $c\%$ also contain Y.

3.2 Connection Chains

A *connection chain* denotes a set of *tcp connections* [10], which are formed when one logs into one host, from there logs into another and so on. From a host perspective, a connection chain appears as a pair of connections through which packets flow back and forth. An important observation is that the time taken by packets inside the host has to be bounded for a connection chain to work [11]. Throughout this paper, we refer to this time bound as Δ.

Furthermore, a connection between two hosts is a bidirectional channel that enables both ends to send and receive data. For convenience, we refer to each channel as a *flow.* Further, an *inbound flow* refers to the flow of traffic from a remote host to the local host, while an *outbound flow* refers to the reverse direction. Similarly, *inbound* and *outbound packets* refer to packets in the corresponding flow.

4 Connection-Chains Detection

4.1 Connection-Chains Mining Approach

We adapt the traditional association rule mining framework, which originally was geared toward business transactions rules mining, for connection chains mining. In our approach, the items of interest correspond to a set of connections, and the desired association rules correspond to connection chains.

Formally, let $C = \{c_1, c_2, \ldots, c_n\}$ be the set of active *connections* at a given host. As packets flow in these connections, *transactions* are dynamically generated. For a given packet, *transactions* are restricted to be one of the following two types:

- *input transaction* $[c_i]$, where $c_i \in C$, or
- *chain transaction* $[c_i, c_j]$, where $[c_i, c_j] = [c_j, c_i]$, $c_i \neq c_j$ and $c_i, c_j \in C$.

An *input transaction* $[c_i]$ is generated when an *inbound* packet is received on the corresponding connection. On the other hand, a *chain transaction* $[c_i, c_j]$ is generated when an *outbound* packet in one connection follows an *inbound* packet in the other connection within a Δ amount of time. For a transaction of type $[.]$, the *support count* $\sigma([.])$ refers to how many times it has occurred.

A *connection-chain* is an association rule of the form $\{c_i, c_j\}$, with its *confidence* defined as follows:

$$confidence(\{c_i, c_j\}) = \frac{\sigma([c_i, c_j])}{\sigma([c_i]) + \sigma([c_j])} \qquad (1)$$

where $c_i \neq c_j$ and $c_i, c_j \in C$.

Note that a set notation is used to represent a connection chain instead of an implication ($\rightarrow$), in order to emphasize the fact that a connection chain does not imply a particular direction. Intuitively, the numerator of the confidence is a count of how many times a chain transaction has occurred; i.e. packets flow within Δ time unit in either directions: $c_i \rightarrow c_j$ or $c_j \rightarrow c_i$. The denominator represents a count of how may times an input packet is seen on the corresponding connection. Typically, a true connection chain is expected to have a high confidence close to 1, while a false one is expected to have a low confidence close 0.

4.2 Detection Algorithm

In figure 1, we summarize the detection algorithm as a pseudo-code. The input to the algorithm is a stream of packets P, which is either captured in real-time

```
 1: INPUT: $P$ a stream of packets
 2: $inboundPackets = \{\}$
 3: for all $p \in P$ do
 4:     if $d(p) = in$ then
 5:         generate an $[c(p)]$ transaction
 6:         add $p$ to $inboundPackets$
 7:     else if $d(p) = out$ then
 8:         for all $q \in inboundPackets$ do
 9:             if $t(p) - t(q) \leq \Delta$ then
10:                 if $c(p) \neq c(q)$ then
11:                     generate an $[c(p), c(q)]$ transaction
12:                 end if
13:             else
14:                 remove $q$ from $inboundPackets$
15:             end if
16:         end for
17:     end if
18: end for
```

Fig. 1. The detection algorithm.

or read from a saved capture file. Those packets are processed in the order of their timestamps.

For each packet $p \in P$, we define the following operators :

- $t(p)$: the time-stamp of p.
- $c(p)$: the connection to which p belongs.
- $d(p)$: the direction of p; either inbound (in) or outbound (out).

When the processed packet p is an *inbound* one, an input transaction of type $[c(p)]$ is generated. Also, the packet itself is added to the $inboundPackets$ set for later comparisons with *outbound* packets.

On the other hand, the processing of an *outbound* packet p is more involved. The packet is compared with all *inbound* packets that were stored in $inboundPackets$ set. Then, a chain transaction is generated of type $[c(p), c(q)]$, if $q \in inboundPackets$, $t(p) - t(q) \leq \Delta$ and $c(p) \neq c(q)$.

Although not shown in figure 1, support counts of the generated transactions are maintained in a special data structure. Then, the confidences are computed according to equation 1. Particularly, connection chains corresponds to any pair of connections with a *confidence* exceeding some user-defined threshold ($minconf$).

5 Experiments

5.1 Experimental Settings

We implemented the proposed approach in Java, and run various experimentations on a PC with the following specifications: a 1.3Ghz Intel Pentium m-processor, 2 GB RAM, and 80 GB 7200 RPM Hard drive. The experimentations were performed using a public network trace (*LBNL-FTP-PKT*) [12]. It was selected because it is reasonably large to assess the algorithm. Also, it only contains the interactive part (control stream) of FTP sessions. This means that the characteristics of the traffic in this trace is similar to those generated by applications such as `telnet` [13] and `ssh` [14] that are used in creating connection chains.

The trace contains a ten-day worth of traffic for the period of Jan 10-19, 2003. It contains 3.2 million packets flowing in 22 thousand connections. The connections are between 320 distinct FTP servers and 5832 distinct clients. Initially, we sliced the trace into 320 *subtraces* using the servers' ip addresses; i.e. each subtrace contains the packets exchanged with the corresponding server. In a way, running the algorithm on a subtrace is equivalent to running the algorithm in real-time on the corresponding server.

In the experimentations, we studied the effect of changing Δ. As such, we first analyzed the timing of inbound and outbound packets of those servers, and estimated the response time of the servers to be between 10-90 msec. We used this value as a guidance to set Δ in our test suite. Accordingly, we decided to use the following values of Δ: 1, 10, 50, 100, 200, and 500 msec. They were selected to investigate the effect of setting Δ, *below*, *around*, and *above* the true Δ value.

5.2 Real-Time Performance

To assess the algorithm's real-time performance, we evaluated the processing time per packet. For every subtrace (320 subtraces), we run the algorithm with a Δ of 1, 10, 50, 100, 200, and 500 msec; i.e. a total of $6 \times 320 = 1920$ cases. For a particular subtrace S_i, the processing time T_i is recorded in each case. The results are then plotted in figure 2.

As shown in figure 2, we notice that the processing time exhibits a linear trend as subtraces increase in size. Accordingly, the processing time per packet is almost constant, as it basically corresponds to the slope of these lines. Mathematically, it is given by $\frac{T_i}{|S_i|}$ *seconds/packet*, where $|S_i|$ is the number of packets. For this trace, the average processing time per packet is approximately 35 μsec/packet. Additionally, we noticed that varying Δ does not seem to have a significant effect on the processing time. Accordingly, we concluded that the algorithm is suitable for real-time operation, because the average processing time per packet is both constant and low.

5.3 Detection Performance

To assess the detection performance of the algorithm, we first picked the largest subtrace among the 320 subtraces, although other subtraces give similar results. The subtrace contains 1.7 millions packets that correspond to the traffic exchanged between the server (131.243.2.12) and 236 unique remote

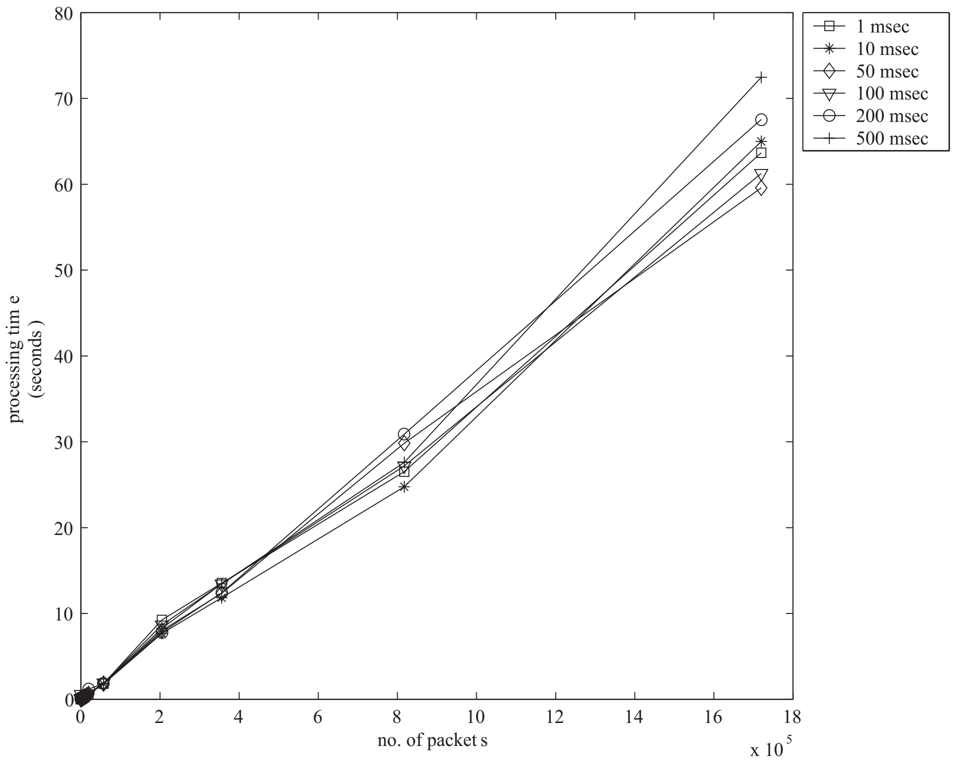

Fig. 2. The processing time of the 320 subtraces for different values of Δ. The subtraces are sorted in increasing order according to the number of packets. As shown, the processing time is approximately linear as subtraces increase in size.

hosts. Among these 236 unique remote addresses, we randomly picked 88 of them to create simulated connection chains as follows. Let L, R and R' respectively stand for the server (local), a remote host and a *fictitious* remote host. Then, the steps to create a simulated connection chain {R,R'} are as follows:

- For an inbound packet (R,L), create an outbound packet (L,R'). The time-stamp of the new packet is set to original time-stamp **plus** some random time t.
- For an outbound packet (L,R), create an inbound packet (R',L). The time-stamp of the new packet is set to original time-stamp **minus** some random time t.
- Merge those generated packets into the original trace.

For the random time t, we use a uniform random variable between 10-90 msec (an estimate of the server response time). Accordingly, the modified subtrace has $236 + 88 = 324$ remote addresses and $\binom{324}{2} = 52326$ possible connection chains. Only 88 out of the 52326 possible connection chains are *true* connection chains ($\approx 0.2\%$). Those are the ones that we actually simulated.

The modified subtrace is then used as an input to the algorithm. In order to study all connection chains detected by the algorithm regardless of their confidences, we compute confidence statistics for different values of Δ. The following values of Δ were considered: 1, 10, 50, 100, 200, and 500 msec. Note that a Δ of 100 msec is the ideal value in this case, because the server response time is estimated to be 10-90 msec.

Table 1. A summary of The Algorithm's output showing confidence statistics for different values of Δ under any non negative value for *minconf*.

		Confidence					
		Min	1st Quartile	Median	Mean	3rd Quartile	Max
$\Delta = 1$ ms	True	0.01429	0.01857	0.02389	0.02639	0.0317	0.04348
	False	0.0002823	0.0007423	0.0009671	0.001242	0.00151	0.0122
$\Delta = 10$ ms	True	0.02439	0.03584	0.06797	0.07214	0.08378	0.1923
	False	0.0003401	0.001433	0.002322	0.002967	0.003913	0.02817
$\Delta = 50$ ms	True	0.2581	0.4756	0.4093	0.4929	0.5595	0.8077
	False	0.0003804	0.002959	0.006042	0.009131	0.01292	0.07726
$\Delta = 100$ ms	True	1.0	1.0	1.0	1.0	1.0	1.0
	False	0.0003623	0.004518	0.01006	0.01599	0.02237	0.1467
$\Delta = 200$ ms	True	1.0	1.0	1.0	1.0	1.0	1.0
	False	0.0003623	0.008181	0.01796	0.03009	0.04302	0.2653
$\Delta = 500$ ms	True	1.0	1.0	1.0	1.0	1.0	1.0
	False	0.0004968	0.01471	0.03562	0.05958	0.08803	0.4173

A summary of the the algorithm's output is shown in table 1. For each value of Δ, we list several descriptive statistical quantities to show the confidences distributions of the true and false connection chains involved in the evaluation dataset.

We visualize the confidences of true and false connection chains in figure 3. In this figure, notice how the confidences of true and false connection chains overlap when Δ is set to very low values (1 and 10 msec). However, once Δ is set around or above the ideal value, true connection chains are clearly separated. In this case, by appropriately setting the confidence threshold (*minconf*) in the separation area, we achieve perfect detection rates. For instance, for $\Delta = 100$ msec, by setting $minconf = 0.5$ we obtain a true detec-

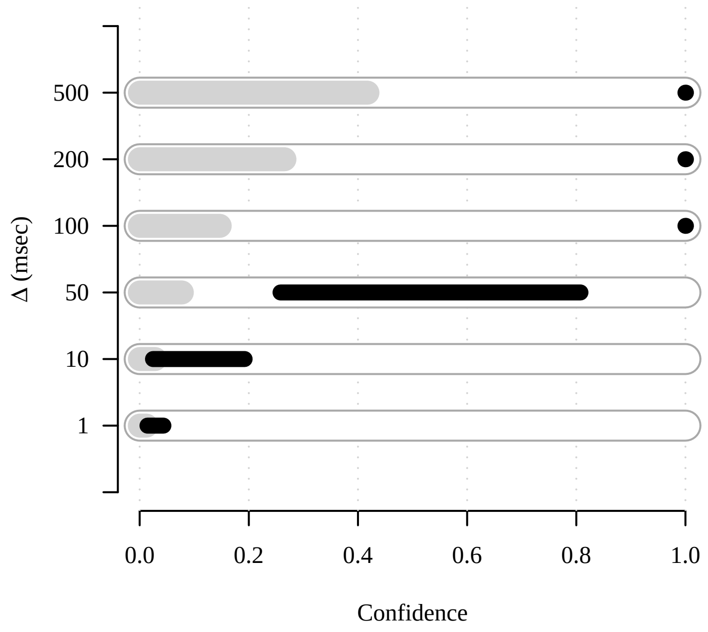

Fig. 3. The range (min-max) of confidences of true and false connection chains for different values of Δ. For each value of Δ, a grey region indicates the range for false connection chains, while a black region indicates the range for true ones.

tion rate = 100% and false detection rate = 0%. Also, notice that increasing Δ beyond the ideal value decreases the *separation* between the confidences of the true and false connection chains. In this case, the maximum separation occurs at the ideal value of Δ (100 msec). However, notice that this separation is reasonably large even when $\Delta = 500$ msec; i.e. 5 times the ideal value. In essence, large separation is desirable because it gives greater flexibility in setting the $minconf$ threshold. Such threshold is used to reduce (or eliminate) false connection chains.

6 Concluding Remarks

A *connection-chain* refers to the set of connections created by sequentially logging into a series of hosts. Attackers typically use connection chains to indirectly carry their attacks and stay anonymous. In this paper, we proposed a host-based algorithm to detect connection chains by passively monitoring inbound and outbound packets. We took advantage of the fact that the time taken by a packet inside the host has to be bounded for a connection chain to work. We refer to this time bound as Δ.

In the proposed approach, we employed concepts from association rule mining in the data mining literature. In particular, we proposed efficient algorithm to discover connection chains among a set of connections. Also, a confidence measure is proposed to measure the strength of a connection chain.

We implemented the proposed approach in Java, and run various experimentations to assess the real-time and detection performance. The experimentations were performed using a public network trace.

For processing time, our experimentations suggest that the algorithm is suitable for real-time operation, because the average processing time per packet is both constant and low. For the detection performance, our experimentations suggest that the algorithm is effective in detecting true connection chains. The setting of Δ seems to play an important role. In particular, we found that the confidences of true and false connection chains are clearly separated when Δ is set around or above (even 5 times) the true value. This gives greater flexibility in setting a confidence threshold ($minconf$) to reduce (or eliminate) false connection chains.

References

1. M. Ranum, "Network forensics: Network traffic monitoring," Network Flight Recorder, Inc., Tech. Rep., 1997.
2. S. Staniford-Chen and L. T. Heberlein, "Holding intruders accountable on the internet," in *Proceedings of IEEE Symposium on Security and Privacy*, May 1995, pp. 39–49.
3. Y. Zhang and V. Paxson, "Detecting stepping stones," in *9th USENIX Security Symposium*, Aug 2000, pp. 171–184.
4. A. Almulhem and I. Traore, "Connection-chains: A review and taxonomy," ECE Department, University of Victoria, Tech. Rep. ECE-05.4, 12 2005.
5. F. Buchholz and C. Shields, "Providing process origin information to aid in network traceback," in *Proceedings of the 2002 USENIX Annual Technical Conference*, 2002.
6. B. Carrier and C. Shields, "The session token protocol for forensics and traceback," *ACM Trans. Inf. Syst. Secur.*, vol. 7, no. 3, pp. 333–362, 2004.
7. H. W. Kang, S. J. Hong, and D. H. Lee, "Matching connection pairs," in *Lecture Notes in Computer Science*, vol. 3320, Jan 2004, pp. 642–649.
8. R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases," *SIGMOD Rec.*, vol. 22, no. 2, pp. 207–216, 1993.
9. P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining.* Addison-Wesley, 2006.
10. J. Postel, *Transmission Control Protocol*, RFC 793, sep 1981.

11. D. L. Donoho, A. G. Flesia, U. Shankar, V. Paxson, J. Coit, and S. Staniford, "Multiscale stepping-stone detection: Detecting pairs of jittered interactive streams by exploiting maximum tolerable delay," in *RAID 2002: Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection,*, october 2002, pp. 17–35.
12. "Lbnl-ftp-pkt," http://www-nrg.ee.lbl.gov/anonymized-traces.html.
13. J. Postel and J. Reynolds, *Telnet Protocol Specification*, RFC 854, May 1983.
14. C. Lonvick, *SSH Protocol Architecture*, Cisco Systems, Inc., December 2004.