Profiling Distributed Connection Chains

Ahmad Almulhem* and Issa Traore

ISOT Research Lab
Electrical and Computer Engineering Department
University of Victoria, Victoria, B.C., V8W 3P6, Canada
Fax: (250) 721-6052, Phone: (250) 721-8697
E-mail: {almulhem, itraore}@ece.uvic.ca
(∗) Corresponding author

Abstract: A key challenge in network forensics arises because of attackers ability to move around in the network, which results in creating a chain of connections; commonly known as *connection chains*. They are widely used by attackers to stay anonymous and/or to confuse the forensic process. Investigating connection chains can be further complicated when several ip addresses are used in the attack. In this paper, we highlight this challenging problem. We then propose a solution through hacker profiling. Our solution includes a novel hacker model that integrates information about a hacker's linguistic, operating system and time of activity. It also includes an algorithm to operate on the proposed model. We establish the effectiveness of the proposed approach through several simulations and an evaluation with a real attack data.

Keywords: network security; connection chains; stepping stones; alert correlation; network forensics; spatial hacking; fingerprinting.

Reference to this paper should be made as follows: Ahmad Almulhem and Issa Traore (xxxx) 'Profiling Distributed Connection Chains', *Int. J. Communication Networks and Distributed Systems*, Vol. x, No. x, pp.xxx–xxx.

Biographical Notes: Ahmad Almulhem is a researcher in the area of network security with special interest in network forensics. He received his Ph.D. in 2007, from the department of electrical and computer engineering at the University of Victoria, Canada. He is currently an assistant professor in the department of computer engineering at King Fahd University of Petroleum and Minerals, Saudi Arabia.

Issa Traore received an Aircraft Engineer degree from Ecole de l'Air in Salon de Provence (France) in 1990, and successively two Master degrees in Aeronautics and Space Techniques in 1994, and in Automatics and Computer Engineering in 1995 from Ecole Nationale Superieure de l'Aeronautique et de l'Espace (E.N.S.A.E), Toulouse, France. In 1998, Traore received a Ph.D. in Software Engineering from Institute Nationale Polytechnique (INPT)-LAAS/CNRS, Tou-louse, France. From June - Oct. 1998, he held a post-doc position at LAAS-CNRS, Toulouse, France, and Research Associate (Nov. 1998 - May 1999), and Senior Lecturer (June-Oct. 1999) at the University of Oslo. Since Nov. 1999, he has joined the faculty of the Department of ECE, University of Vic-

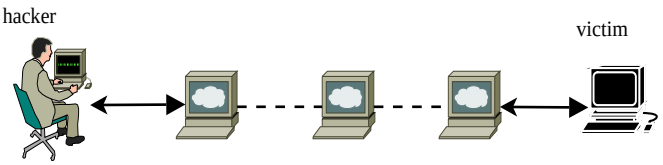

Figure 1 Using a connection-chain to hide an attacker's origin.

toria, Canada. He is currently an Associate Professor and holds since Oct. 2003 the position of Computer Engineering Programme Director. His research interests include Behavioral biometrics systems, intrusion detection systems, software security metrics, and software quality engineering. He is the founder and coordinator of the Information Security and object Technology (ISOT) Lab (`http://www.isot.ece.uvic.ca`).

1 Introduction

Computer networks give attackers the opportunity to attack their victims indirectly. In particular, it is common for a computer attack to originate from an attacker's computer, propagate through other intermediary computer(s), then attack a victim computer. This leads to what is called *connection chains* [1]. Attackers resort to connection chains to hide the origin of their attacks and/or to confuse the network forensic process.

A *connection chain* is created when someone recursively logs into a host, then from there logs into another host, and so on as shown in Figure 1. Due to the design of tcp/ip suite, the origin of the chain is effectively concealed as we move down the chain. As such, a connection chain actually provides an interactive channel to remotely manipulate a host without revealing someone's origin.

Connection chains can be further complicated to confuse the network forensics process. In Figure 2, two variations of connection chains are shown. In the first scenario, two independent attackers use the same intermediary host to attack a victim machine. In this case, a forensic process would mistakenly aggregate/correlate data from two different attackers and hence two different attacks.

In the second scenario, an attacker uses two different intermediary hosts for his attack. The victim will see two different ip addresses and has no way to tell that they are originating from the same hacker. Consequently, data and alerts from the two ip addresses would pass uncorrelated.

To address this problem, we need a way to profile hackers in order to differentiate one from another. Specifically, we need to look for invariant features that may distinguish one hacker from another, instead of only relying on his ip address. By examining received packets, we believe that it is possible to extract such features.

One of these features is a hacker's operating system (OS). OS *passive fingerprinting* is a well established technique that can be used for OS identification by examining the headers of certain packets (especially SYN packets) [2].

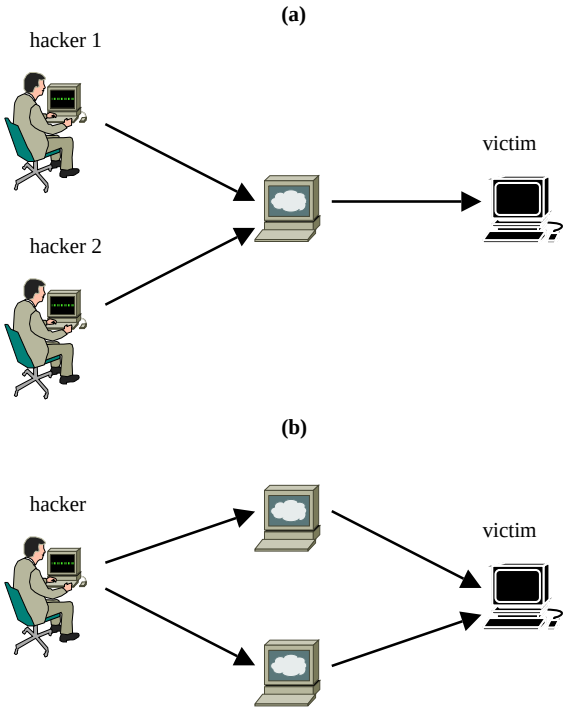

Figure 2 Two scenarios which confuse a network forensic process. (a) Two independent hackers use same intermediary node to attack a victim. (b) One hacker use two intermediary nodes to attack the same victim.

We propose a solution that enhances the OS passive fingerprinting [2] by using two additional features; namely a hacker's *linguistics* and *activity time*. For linguistics, the content of a packet is examined to look for clues about a hacker's language. For instance, a user/password combination may reveal a name that is known to belong to a certain language.

For time of activity, a hacker may have a preferred time to carry out his actions. By measuring his activities during a day, we may obtain a consistent pattern that further enhances the profiling process. We use arrival time of packets to determine this feature as we will show later.

The rest of this paper is organized as follows. In section 2, we present a hacker model that integrates information about a hacker's linguistic, operating system and time of activity. We then discuss our approach in applying the proposed model in section 3. We establish the effectiveness of the proposed approach through several simulations and an evaluation with real attack data in sections 4 and 5. Finally, we conclude in section 6.

2 Model Formulation

2.1 A Hacker Model

Our model for a hacker is defined as a 3-tuple $\mathcal{H}$ over the Cartesian product $\{\mathcal{O} \times \mathcal{L} \times \mathcal{T}\}$ where:

- $\mathcal{O} = \{x : x \text{ is the name of a known operating system}\}$
- $\mathcal{L} = \{x : x \text{ is the name of a known natural language}\}$
- $\mathcal{T} = \{Morning, Evening, Night\}$

$\mathcal{O}$ is basically a set of nominal values corresponding to different operating systems; such as {Windows98, Linux, ... }. Similarly, $\mathcal{L}$ is a set of nominal values corresponding to different natural languages, such as {English, French, ... }. Lastly, $\mathcal{T}$ is a tri-valued variable that indicates when a hacker is most active. Instead of recording detailed time information, we use this variable to classify a hacker as either a morning, evening, or night hacker. This provides an enormous reduction in data and a reasonable granularity for our purpose.

2.2 Model Application

Applying our model to a stream of incoming packets is essentially an abstraction process by which we intend to map every ip address into a hacker instance. Given a list of ip addresses P, this is formally stated as follows.

$$P \rightarrow \mathcal{H} : \{\mathcal{O} \times \mathcal{L} \times \mathcal{T}\}$$

In order to execute this mapping, we need the following three mapping functions:

$$\begin{cases} \psi \equiv header(p) \rightarrow \mathcal{O} \\ \lambda \equiv content(p) \rightarrow \mathcal{L} \\ \phi \equiv arrivalTime(p) \rightarrow \mathcal{T} \end{cases}$$

where, $header(p)$, $content(p)$ and $arrivalTime(p)$ are, respectively, the headers, contents and arrival times of packets originating from an ip address p.

The ψ function applies OS passive fingerprinting techniques in order to infer the OS of an ip address p [2]. Typically, the header portion of an ip packet is used for such task. Formally, this function is a trained classifier that operates by comparing certain header's fields to a database of known OS signatures. For example, the 8-bit *time-to-live* (TTL) field in the ip header is set with different values depending on the operating system. By default, for instance, TTL is set to 128 in Windows and 64 in Linux. Such features can be used to distinguish between different operating systems.

The λ function assigns an ip address p to one or more natural languages. This task is known as language guessing in the field of *text categorization* [3]. A popular technique, that accomplishes this task with almost-perfect accuracy, is due to Cavnar and Trenkle [4]. Briefly, the technique relies on analysing the n-grams[a] of a

[a] An N-gram is an N-character slice of a longer string [4].

document. The frequencies of different n-grams provide distinctive signatures that not only distinguish documents, but also discriminate between languages. There is already an efficient implementation of the algorithm as well as signatures for more than 70 different natural languages [5]. Formally, this function is also a trained classifier.

Finally, the ϕ function uses packets' arrival times to classify a hacker as either a morning, evening, or night hacker. Essentially, it is a 3-category histogram function of the packets' arrival times. In reference to local time, the function divides a day into three 8-hours time periods as shown below:

$$\overbrace{4am \ldots 12pm}^{\text{morning}} \ldots \overbrace{12pm \ldots 8pm}^{\text{evening}} \ldots \overbrace{8pm \ldots 4am}^{\text{night}}$$

Accordingly, it counts how many packets received in each period then yields the matching period with the maximum count. Note that this day division is in reference to the local time zone, rather than hackers' time zone.

2.3 Coherence

After performing the above mapping, an ip address falls into one of two categories: *coherent* ip addresses or *incoherent* ones. A coherent ip address is an ip address that assumes a single value for each variable in the hacker model. On the contrary, an incoherent ip address is an ip address which assumes multiple values for one or more of the variables in the hacker model.

To illustrate this concept, assume two ip addresses p_1 and p_2 are mapped as follows:

$$p_1 \rightarrow \{\{Windows98\} \times \{English\} \times \{Morning\}\}$$
$$p_2 \rightarrow \{\{WindowsXP, Linux2.4\} \times \{Arabic\} \times \{Night\}\}$$

In this example, p_1 is a coherent ip address because it assumes single values for each variable. However, p_2 is an incoherent ip address because the OS variable assumed two values; namely $WindowsXP$ and $Linux2.4$.

3 Approach

3.1 Overview

Using our model and the defined functions, we obtain a mapping for every single ip address into a hacker instance; i.e. $P \rightarrow \mathcal{H} : \{\mathcal{O} \times \mathcal{L} \times \mathcal{T}\}$. With this in hand, we can now address the two problems depicted in Figure 2.

For the first scenario (fig 2.a), the victim will receive packets from an incoherent ip address. For instance, the packets may reveal multiple values for the OS variable. Hence, the victim will be able to detect such ip addresses.

The second scenario (fig 2.b) is more challenging. In this case, we need a way to group ip addresses that share the same attributes. This task can be done using *clustering analysis* [6]. The details of this step is postponed to the next section.

A flowchart of the overall approach is shown in Figure 3. The algorithm iterates over each ip address. An ip address is first tested for coherence. Incoherent ip

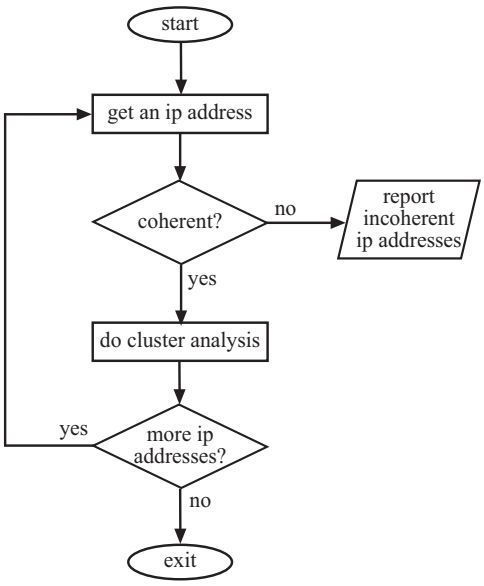

Figure 3 Flowchart of overall approach.

addresses are reported as being possible intermediary node shared by multiple different hackers. On the other hand, coherent ip addresses are passed to a clustering analysis algorithm. The algorithm determines groups of ip addresses that are likely used by one hacker.

3.2 Clustering Analysis Algorithm

In order to carry the clustering analysis, a measure of *dissimilarity* between observed objects is required. This measure is typically dependent on the type of variables that describe an object. In our case, the objects are coherent ip addresses which are described by the three variables of our hacker model. Additionally, the variables are *multistate nominal* ones. We now summarize the derivation of the measure following Gordon [6].

For each variable, a disagreement indices can be defined between each pair of states of the variable. Let $\delta_{klm}(\geq 0)$ be the disagreement between the lth and mth states of the kth variable. Accordingly, let $\delta_{klm} = 1$ if $l \neq m$ and $\delta_{kll} = 0$. The contribution to the dissimilarity d_{ij} between the ith and jth object that is made by the kth variable is defined by $d_{ijk} = \delta_{klm}$ if the kth variable is in state l for the ith object and state m for the jth object. The overall measure of dissimilarity between the ith and jth objects is then defined as

$$dis(i,j) \equiv \sum_{k=1}^{K} d_{ijk}$$

where K is the total number of variables.

After defining a dissimilarity measure, we need an algorithm to perform the clustering analysis [7, 6]. For our purposes, we devised an *incremental* algorithm

Algorithm: DoClustering(C_0, P)

Input: a list of initial clusters C_0.
Input: a list of ip addresses P.
Output: a list of resulting clusters C.

```
C ← C_0 ;
foreach p ∈ P do
    clustered ← false ;
    i ← 1 ;
    while (not clustered) and (i ≤ |C|) do
        if dis(p, C(i)) == 0 then
            C(i) ← C(i) + {p} ;
            clustered ← true ;
        end
        i ← i + 1 ;
    end
    if (not clustered) then
        |C| ← |C| + 1 ;
        C(|C|) ← {p};
    end
end
return C ;
```

Figure 4 A pseudocode of the clustering algorithm. Note that $|C|$ stands for the size of C, and $C(i)$ stands for the i^{th} element in C.

shown in fig 4. The algorithm incrementally classify every coherent ip address. Based on the dissimilarity measure, an ip address is either combined into an existing cluster or put in a new cluster. By the end, each cluster will contain one or more coherent ip addresses that share the same attributes; i.e. likely coming from the same hacker.

It should be noted that the number of clusters is dynamically updated as new clusters are formed. This departs from algorithms (*k-means* for instance) that require specifying the ultimate number of clusters for effective analysis.

3.3 Example

We conclude this section with an example to illustrate our approach. Suppose that a victim PC was attacked by 4 different ip addresses $P = \{p_1, p_2, p_3, p_4\}$. Also, suppose they are mapped as follows:

$$p_1 \rightarrow \{\{Windows98\} \times \{English\} \times \{Evening\}\}$$
$$p_2 \rightarrow \{\{WindowsXP\} \times \{Arabic, Chinese\} \times \{Night\}\}$$
$$p_3 \rightarrow \{\{FreeBSD\} \times \{French\} \times \{Morning\}\}$$
$$p_4 \rightarrow \{\{Windows98\} \times \{English\} \times \{Evening\}\}$$

The algorithm first identifies p_2 as an incoherent *ip* address because more than one language were detected. This means that p_2 is probably a node used by several hackers similar to the scenario depicted in fig 2.a.

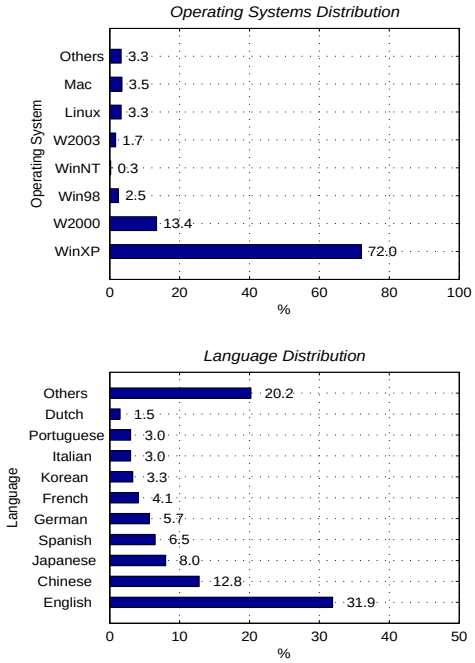

Figure 5 Empirical probability distributions of operating systems and languages on the Internet. (OS source: www.w3schools.com, Language source: www.internetworldstats.com.)

The other three ip addresses are coherent. Therefore, they are advanced for the clustering analysis algorithm. At the end of the analysis, we will have 2 clusters; i.e. $\{p_1, p_4\}$ and $\{p_3\}$. The $\{p_1, p_4\}$ cluster probably represents a situation similar to the scenario at fig 2.b. While, the $\{p_3\}$ cluster contains only one ip address, which means that it is probably a single hacker using a single ip address.

4 Simulations

We conducted various simulations to assess the proposed model and the clustering algorithm. For this purpose, we employed empirical probability distributions of operating systems and languages that reflect typical usage on the Internet. In particular, we used the freely available distributions shown in Figure 5. For the time variable $\mathcal{T}$, we used a discrete uniform distribution. We then generated many random samples of ip addresses that have characteristics following these distributions. All simulations were run using `Matlab` version 6.5 on a 1.3GHz laptop with 512MB.

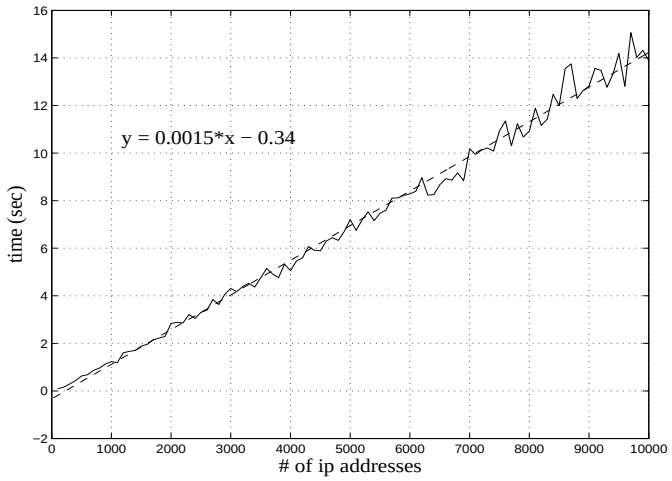

Figure 6 Execution time of the clustering algorithm as a function of the number of ip addresses.

4.1 Time Complexity

To start with, we evaluated the time complexity of the clustering algorithm. Specifically, we evaluated the running time of the algorithm in *batch mode* and in *incremental mode*. In batch mode, the algorithm is passed a set of ip addresses with no prior clusters. The algorithm then builds clusters starting from scratch. In contrast, in incremental mode, the algorithm is passed one ip address at a time along with a set of initial clusters.

In practice, the batch mode is intended for offline analysis, while the incremental mode is intended for realtime analysis. The distinction between the two modes depends on the parameters that are passed to the algorithm. In batch mode, the call is DOCLUSTERING$(\{\}, P)$, where the parameters are an empty set of initial clusters and a set of ip addresses P. In incremental mode, however, the call is DOCLUSTERING$(C_0, \{p\})$, where the parameters are an initial set of clusters C_0 and a set of only one ip address $\{p\}$. Typically, the algorithm is first run in batch mode to form an initial set of clusters C_0, then it is run in incremental mode to process new ip addresses as they arrive.

At a first glance, the algorithm may suggest a quadratic running time in terms of the total number of ip addresses because of the double loop. However, our simulations consistently showed a linear running time in batch mode, and a small running time in incremental mode.

For batch mode, we generated 100 samples with sizes that increase from 1 up to 10000 at an increment of 100. Each sample was then passed to the clustering algorithm. The time taken by the algorithm was then recorded. To ensure the validity of the results, 5 reruns of the same procedure were performed. In Figure 6, we show a plot of a typical run. As shown, the execution time increases linearly as the number of ip addresses increases. For reference, the linear fitting (dashed line) is also shown in the figure. It should be noted that 10000 is relatively very

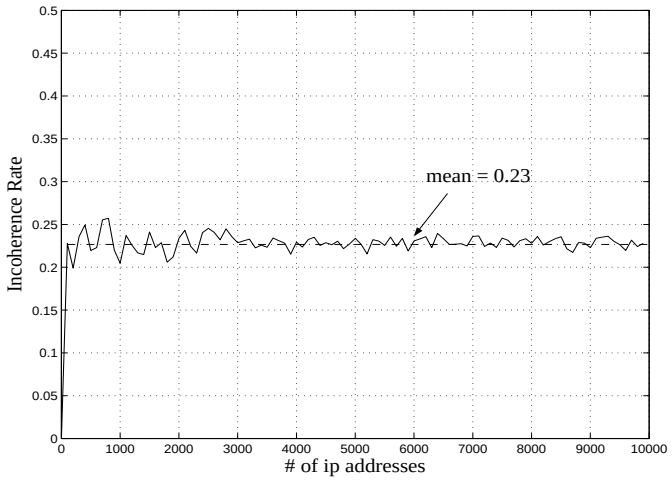

Figure 7 The rate of incoherent ip addresses relative to the total population.

large, because this number is supposed to be the number of unique "ip addresses" attacking a victim at one time.

In incremental mode, the outer loop is invoked only once because there is only one ip address to process. Therefore, the running time is basically the time taken by the inner loop to scan the initial clusters, i.e. $O(C_0)$. In our experiments, we found this time to be very small. For instance, the running time for clusters with up to 10000 ip addresses never exceeded 0.01 seconds.

4.2 Effectiveness

To a great extent, the effectiveness of our approach relies on the underlying distributions of the ip addresses. In this section, we present simulations related to this aspect.

At first, we studied incoherent ip addresses and their expected rates relative to the total population. We generated 100 samples with sizes that increase from 1 up to 10000 at an increment of 100. Each sample was then analyzed for incoherent ip addresses. An ip address is regarded incoherent when either the operating system or the language is set to "others". We then used the following metric:

$$incoherence\ rate = \frac{i}{N}$$

where i is the number of incoherent ip addresses in the population, and N is the total number of ip addresses in the population.

A plot of the incoherence rate is shown in Figure 7. It is interesting to note that the rate is almost constant as the size of the population increases. In this simulation, the average rate is about 0.23. In other words, about 23% of the population is expected to be incoherent and dropped from the clustering analysis.

We next considered the clustering of coherent ip addresses. It should be obvious that the number of possible clusters depends on the *resolution* of each variable in

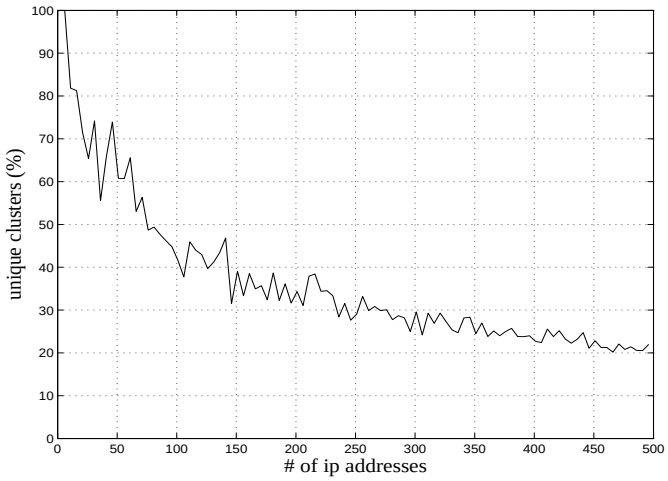

Figure 8 The ratio of the number of unique clusters as the population of ip addresses increases.

our model. We use the term "resolution" here to denote the number of possible states that a variable may assume. To show this graphically, we used the following metric:

$$clusters\ rate = \frac{nc}{N}$$

where nc is the number of clusters and N is the total number of ip addresses.

We generated 50 samples with sizes that increase from 1 up to 500 at an increment of 10. A plot of the clusters rate is shown in Figure 8. As shown, the ratio decreases rapidly (almost exponentially) as the number of ip addresses increases. Therefore, the algorithm will always deal with limited number of clusters. Also, this confirms that the algorithm has a low overhead as was shown in the previous section.

5 Evaluation using Real Attack

In this section, we evaluate the proposed framework using a real attack that was posted as a forensic challenge by the Honeynet project [8]. The attack was presented as challenge#19 in a series of 34 challenges called *scan of the month challenges*. We carefully picked this particular attack, because it is the only one that fits the attack scenarios addressed by our framework. In particular, it resembles the scenario depicted in Figure 2 (b).

5.1 Attack Overview

For this attack, Honeynet project provided 2 files in *pcap* format [9]. One file (newdat3.log) contains captured network traffic to/from a honeypot. The second file (slog2.log) contains generated syslog traffic. In total, there were 24440 packets communicated between the honeypot and 16 remote ip addresses. Among the

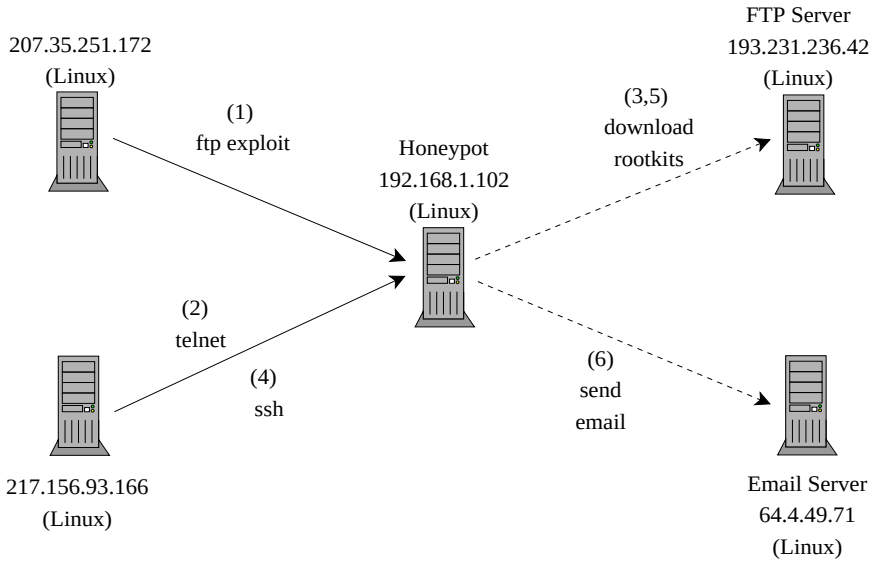

Figure 9 A diagram showing the ip addresses involved in the attack, and the attack steps.

16 ip addresses, four ip addresses were involved in the attack. Specifically, two ip addresses were actually used to compromise the honeypot, one ip address was accessed to download rootkits, and one was used to send an email.

The steps of the attack and the involved ip addresses are shown in Figure 9. Briefly, the attack proceeded as follows:

1. 207.35.251.172 connected (ftp) to the honeypot, run a successful exploit, and obtained a root shell. He then deleted the password of user "nobody", and added a new user (dns) with root privilege.

2. 217.156.93.166 connected (telnet) to the honeypot using user "nobody", and obtained a shell.

3. Via the obtained telenet shell, the attacker connected (ftp) to an ftp server on 193.231.236.42, and downloaded/installed a rootkit. Among other things, the rootkit started a modified ssh server on port 24 (backdoor).

4. 217.156.93.166 connected (ssh) to the honeypot, and obtained a shell.

5. Via the obtained ssh shell, the attacker again connected (ftp) to the ftp server on 193.231.236.42, and downloaded/installed more rootkits.

6. An email is sent through the email server at 64.4.49.71.

To see how this attack fits the scenario in Figure 2 (b), notice that the attack was concurrently carried from two ip addresses; namely 207.35.251.172 and 217.156.93.166. Apparently, the attacker controlled the computers at these ip addresses. The other two ip addresses (193.231.236.42 and 64.4.49.71), however, were

mainly used to download rootkits and send an email. As such, their links are indicated as doted lines.

5.2 Analysis

In order to apply our framework, each ip address has to be mapped into a hacker instance; i.e. a 3-tuple {OS, Language, Activity Time }. The mapping was done using an automated script as follows.

- For OS mapping, we used an open source tool called p0f [2]. P0f is a versatile *passive* OS fingerprinting tool, which can identify the operating system using a number of techniques.
- For the activity time, we built a small script that classifies each ip address as either morning, evening or night. The script simply counts the packets exchanged with each ip address and reports when they occur most.
- For Language mapping, we extracted readable text using a Unix tool called *strings*. We, then, examined the text manually to determine the language. We had to resort to manual examination, because we could not find a suitable tool. In future, we intend to build such tool.

We show the resultant mapping in Table 1. The ip addresses involved in the attack occupy the 4 bottom rows of the table, and are indicated with asterisks (*). For each ip address, we also show the packet count, and explaining remarks. These remarks are mainly to explain why the language was (was not) detected. Finally, we indicate missing values (OS, Language) as "-".

For some ip addresses, it was not possible to detect the OS fingerprints. The reason is related to the techniques used by the tool p0f. In particular, p0f detects OS by examining only TCP packets (SYN, SYN+ACK and RST/RST+ACK). Other packets, for instance UDP and ICMP, are ignored. In our case, the undetected ip addresses actually exchanged non-TCP traffic. To the best of our knowledge, there is no tool that can OS fingerprint using non-TCP traffic. This could be due to technical difficulty or due to the prevalence of TCP traffic. Creating such a tool, however, is definitely a welcome contribution.

For language detection, Romanian language was associated with two of the ip addresses involved in the attack (217.156.93.166 and 193.231.236.42). In particular, a Romanian name (gunoierul) was used as a password in an ftp session. For other ip addresses, it was not possible to detect the language because of the lack of payload, or because of having a payload which contains only commands and their responses.

5.3 Discussion

In the proposed framework, clustering analysis is used to aggregate mapped ip addresses into groups that share the same attributes. In reference to Table 1, we can identify two groups that share 2 or more attributes:

- Group 1: {208.179.195.130, 210.114.220.46}.
- Group 2: {64.4.49.71, 193.231.236.42, 217.156.93.166, 207.35.251.172}.

Table 1 Mapping of the ip addresses in challenge#19 from Honeynet project. (*) indicates the ip addresses involved in the attack.

ip address	packets count	OS	language	Activity Time	Remarks
66.51.200.115	1	Windows	-	evening	No payload.
208.179.195.130	2	Linux	-	night	No payload.
24.248.173.56	2	Linux	-	morning	No payload.
138.86.152.104	6	-	-	night	No payload.
128.175.106.247	10	-	-	night	Port scan only. No payload.
207.50.37.225	10	-	-	morning	No payload.
63.168.30.92	10	-		morning	No payload.
210.114.220.46	11	Linux	-	night	No payload.
206.75.218.84	11	FreeBSD	-	evening	No payload.
64.58.76.226	12	-	-	evening	No payload.
24.17.45.29	28	-	-	morning	No payload.
207.245.82.221	88	-	-	evening	Port scan only. No payload.
64.4.49.71*	29	Linux	-	evening	Payload contains commands/responses.
193.231.236.42*	543	Linux	Romania	evening	Romanian name (password) detected.
217.156.93.166*	2455	Linux	Romania	evening	Romanian name (password) detected.
207.35.251.172*	21222	Linux	-	evening	Payload contains commands/responses.

For group 1, the two ip addresses have the same OS and activity time; i.e Linux and night respectively. However, this grouping can not be confirmed, because only few packets are communicated (2 and 11 packets respectively). Also, the communicated packets carry no payload! Therefore, the relationship in this group is inconclusive and requires further investigation.

On the other hand, the ip addresses in group 2 are, indeed, related. They are all involved in the attack as explained earlier. We also can be certain this time, because many packets were communicated by ip addresses in this group.

On the downside, we notice that OS and Language fingerprinting generate many missing values. In this case, the language was detected in two out of 16 ip addresses (12.5%), and OS detected in 9 out 16 ip addresses (56.25%). Technically, this is due to the used tools, rather than the proposed framework. However, we believe that adding more attributes into the proposed hacker model can further enhance the profiling process.

6 Conclusions

Investigating connection chains can be complicated when *several* ip addresses are used in the attack. In this paper, we highlighted this challenging problem which affects both alert correlation and network forensics approaches. We also proposed a simple yet extensible hacker model to address this problem. The model integrates information about a hacker's linguistic, operating system and time of activity. The mentioned model is then applied within a framework. We finally conducted several simulations and an evaluation with real data to assess our approach.

References

[1] Stuart Staniford-Chen and L. Todd Heberlein. Holding intruders accountable on the internet. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 39–49, May 1995.

[2] Michal Zalewski. P0f v2: A versatile passive os fingerprinting tool. `http://lcamtuf.coredump.cx/p0f.shtml`. (Last visited: May 26, 2007).

[3] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1):1–47, 2002.

[4] William B. Cavnar and John M. Trenkle. N-gram-based text categorization. In *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175, Las Vegas, US, 1994.

[5] Frank Scheelen. Libtextcat: Language guessing library. `http://software.wise-guys.nl/libtextcat/`. (Last visited: May 26, 2007).

[6] A. D. Gordon. *Classification*. Chapman & Hall/CRC, 2nd edition, 1999.

[7] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Comput. Surv.*, 31(3):264–323, 1999.

[8] Lance Spitzner. The honeynet project. `http://www.honeynet.org`. (Last visited: May 26, 2007).

[9] Van Jacobson, Craig Leres, and Steven McCanne. Packet capture library. `http://www.tcpdump.org/`. (Last visited: May 26, 2007).