Honeypot-based Forensics

F. Pouget, M. Dacier
Institut Eurécom
2229, route des Crêtes; BP 193
06904 Sophia-Antipolis ; France
Email: {pouget,dacier}@eurecom.fr

Abstract:
Some attacks on honeypots are very frequent and repetitive. In addition, such repetitive attacks generate a very large amount of data. In this paper, we show that it might be misleading to consider general statistics obtained on these data without carrying an in depth analysis of the various processes that have led to their creation. We show that such analysis can be done by means of a simple clustering approach. We present an algorithm to characterize the root causes of these attacks. This algorithm enables us to obtain precious and non trivial information to identify the various attacks targeting our environment. We use this algorithm to identify root causes of the data collected from our honeypot environment. We demonstrate that identifying the root causes is a prerequisite for a better understanding of malicious activity observed thanks to honeypots environments. Finally, we hope this work will open new avenues for the ongoing work related to honeynets.

Keywords:
Log analysis, attack forensics, alert correlation, quantitative risk assessment, honeypots, root cause analysis

1. Introduction

During the last two years, many different uses of honeypots have been proposed. Some of them are deployed to waste hackers time [LaBrea], others to reduce spam activity [THP03, Kraw04] or to deceive attackers [ChGK03, Cohe99] and some more to analyze hacker intrusion steps [GenH03]. Honeypot applications are quite diverse and they are studied with many details in [PDD03]. In the context of our own experiments, we have used a home-made platform of virtual machines specifically designed to observe malicious traffic in a long term perspective.
Our honeypot environment has been implemented in January 2003. A practical experience report has been presented in [DPDe04] in which we have shown data obtained by means of three honeypots being attacked over a period of four months. In addition, we have pointed out some regularity exhibited by the data which might indicate some stable processes running against our honeypots. This observation has then been confirmed in [DaPD04] where a ten month period of data was presented.
However, a closer look at those apparent regularities reveals some hidden phenomena that can hardly be observed when looking at the bulk of the data. Surprisingly enough, literature on that field presents new interesting design approaches on possible honeypot usages, but as far as we know, no work reports any in-depth honeypot data analysis.
As for illustration, some research activities are launched over the world to apply honeypots technologies. Most of them, including the authors, have joined the Honeynet Research Alliance, a common project grouping more than 15 different international teams [HonAll]. They have proposed interesting software architectures, such as GenI and GenII Honeynet [GenH03] but very few efforts have been made so far to help analyzing collected data. In general, the proposed architecture includes an intrusion detection system (also known as IDS), and the whole analysis consists in mining IDSs alarms [HoIe03, HonAll, HonNo, Nevi03,

Spit02]. However IDSs can only be seen as complementary analysis tools and a more specific attention must be paid to raw data: honeypots can bring more information than simply those provided by IDSs alarms.
In [DaPD04, DPDe04], we have presented information collected from enriched observation of honeypots data. In the present work, we intend to show that this global observation from raw data must be carefully analyzed at a lower and more precise layer. The apparent regularity we observe may sometimes be misleading. Thus, this work consists in analyzing and identifying such regular processes. Guided by our previous observations, we apply a clustering approach. Each cluster is supposed to represent one single regular process, or *attack tool*. However, we show in a second step that this approach must be refined in order to precisely identify such tools. As a result, we describe and apply a complementary technique to get valid clusters of *attack tools*.
The presented results confirm that a deeper and more specific analysis of honeypots data is required to have a good understanding of malicious activity. We hope this seminal work will help launching new research activities on honeypots data.
The paper is organized as follows: Section 2 states the problem resulting from observations made in [DaPD04] and [DPDe04]. Section 3 presents a simple clustering method to extract regular processes. We also check in this section that we have obtained coherent clusters and we refine our clustering technique. Section 4 describes some analyses and applications obtained from the defined clusters. Section 5 concludes the paper.

2. Problem statement

2.1. Environment

2.1.1. Honeypots: A short introduction

Honeypots have been used for more than fifteen years in computer systems, even if the use of this word is quite recent [Stol88, Bell92, Bell93]. In [DaPD04], we have pointed out some terminological issues in regards of that word: there is no common agreement on the definition of honeypot at this time writing. Indeed, a simple glance at some honeypots mailing lists shows that suggested definitions describe how to use honeypots instead of defining what they really are [Secu04]. As a result, we have proposed to take advantage of well defined concepts introduced by the dependability community to derive the following definition [Pow03]:

- *Definition:* “A **honeypot** consists in an environment where vulnerabilities have been deliberately introduced in order to observe attacks and intrusions.”

Put in other words, honeypots are a great environment to observe malicious traffic.
Our platform is described in details in [DaPD04]. It consists in a VMWare virtual environment with three virtual machines running various Operating systems (Linux RedHat, Windows 98, Windows NT) and services (ftp server, web server, etc). The environment is totally passive, and all incoming packets are most likely from malicious origins. Furthermore, virtual machines are built on non-persistent disks [VMware]. Thus, changes are lost when machines are powered off or reset. In other words, rebooting a compromised machine consists in a simple backward recovery mechanism. It is especially convenient in our situation as we are expecting long term data collection.

2.1.2. Data Collection: Enriched database

Data is sent every day through a secure connection from the honeypot environment to a data server and then, is used to feed a specific database. This data server was carefully designed in a preliminary step. Its entity-relationship scheme is quite complex, and we give in figure 1 a simplified overview of its structure.

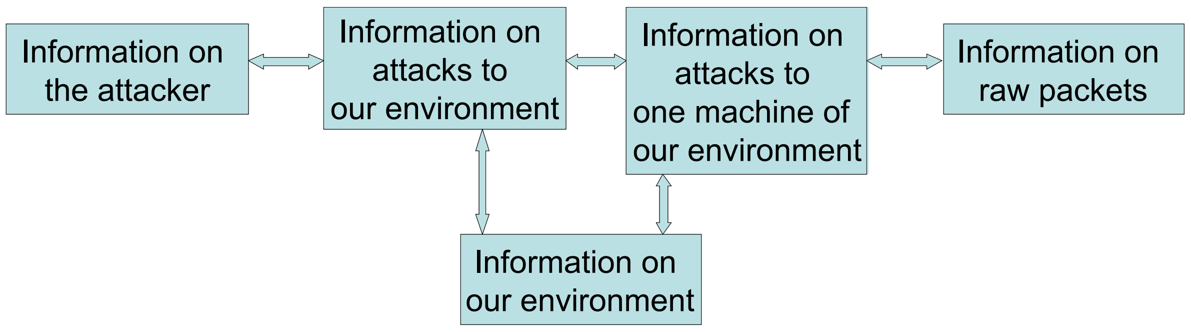
Information on the attacker
Information on attacks to our environment
Information on attacks to one machine of our environment
Information on raw packets
Information on our environment

Figure 1: Simple database structure scheme

Information is grouped into five main categories, and collected data is enriched by additional information such as:

- The IP geographical localization of packets' source addresses, obtained with tools such as Netgeo [Netgeo]
- Passive OS fingerprinting on tcpdump logs, thanks to tools such as p0f [Pof] and Disco [Disco]
- Well-known blacklists of IP addresses
- An enriched precision on attack date (correlation with working days, working hours, holidays, etc)
- A honeypot environment description
- An analysis of observed sources' ports
- Other attack features: For instance, are honeypot machines attacked in parallel or in sequence?

2.2. Preliminary results

2.2.1. Terminology

Let us first introduce two definitions:

- *Attack Source:* it defines an IP address that targets our honeypot environment within one day. This time constraint is arbitrary and based only on our observation [DaPD04]: so far, observed attacks have always been limited to short time periods (no more than 1 minute). Thus, if the same IP address is sending packets to one of our honeypots on the 12th of January and then on the 4th of February, we consider that they come from two distinct *attack sources*. In addition, this definition is motivated by the fact that some IP addresses are dynamically allocated and they change frequently, in terms of days (the ISP even often forces users to change IP addresses if their connection remains open for a long period). As explained in [DaPD04], only a very few IP addresses have been observed in two different calendar days during the whole year of the experiment.
- *Ports Sequences:* attack sources send packets to specific ports of one honeypot machine. A *Ports Sequence* defines the specific order according to which ports have been targeted on a given honeypot machine. For instance, if source A sends requests on port 80 (HTTP), and then on ports 8080 (HTTP Alternate) and 1080 (Socks), the associated *ports sequence* will be {80; 8080; 1080}.

Within our environment, an *attack source* cannot be associated with more than three *ports sequences* since we have three target machines. The latter case is observed when a source attacks our three virtual machines with different *ports sequences*. For instance, the famous worm Blaster has many variants but always behaves the same way: it first scans port 135, and never goes further if the port is closed [Grah03, Sym03]. Otherwise, it scans port 4444. If only one machine has port 135 open, then the corresponding *ports sequence* from the *attack source* is {135, 4444}. If the same attack source targets other virtual machines, the associated ports sequences become {135}.

2.2.2. Former results

The database we have developed offers a large panel of information types. We report the interested reader to [DaPD04] for some results we have obtained by means of simple queries against the database. One major observation we have made was the following: very few ports were targeted over the experiment period. From February 2003 to February 2004, only 195 different ports have been probed. Each attacking machine probes one or more targeted ports following a given *Ports Sequence*. The number of different observed sequences is limited to 485 distinct sequences. Furthermore, we have observed two important points:

- Each sequence is often limited to one port.
- A given set of ports almost always uniquely identifies a *ports sequence*, as we have very rarely observed two sequences differing only by the order of ports being probed.

Table 1 taken from [DaPD04] represents, per month, the top 8 *ports sequences* performed by the attack sources observed during one month. For instance, one can see that in March, 45.5% of the attack sources have sent packets to the sole port 445, while 2.7% have scanned sequentially ports 80, 57 and 21. These 8 sequences characterize the activity of about 75% of the attacks every month.

March 2003	**April 2003**	**May 2003**	**June 2003**
445 (45.5%)	445 (43.5%)	445 (40.1%)	445 (30.6%)
80 (15.9%)	80 (15.4%)	80 (15%)	80 (15%)
1433 (8.8%)	1433 (7.6%)	1433 (8.8%)	139 (9.9%)
139 (5.8%)	139 (7.5%)	139 (7.1%)	1433 (8.2%)
21 (3.5%)	21 (3.3%)	21 (3.3%)	445,139 (4.4%)
135 (2.9%)	135 (2.2%)	135 (3.2%)	139, 445 (3.7%)
80, 57, 21 (2.7%)	80, 57, 21 (1.4%)	80, 57, 21 (2.2%)	21 (3.3%)
443 (2.1%)	443 (1.4%)	139, 445 (2%)	135 (3.2%)
Others (12.8%)	Others (17.7%)	Others (18.3%)	Others (21.7%)
July 2003	**August 2003**	**September 2003**	**October 2003**
445 (29.5%)	445 (23.6%)	80 (15.8%)	80 (15.6%)
80 (20.5%)	80 (17%)	1433 (12.6%)	1433 (11.9%)
1433 (9.6%)	139 (11%)	445 (12.6%)	139 (10.6%)
139 (8.1%)	1433 (9.5%)	139 (11.6%)	135 (9.8%)
21 (3.1%)	135 (5.8%)	135 (7.8%)	445 (8.4%)
139, 445 (2.9%)	139, 445 (4.1%)	554 (5.1%)	27374 (6%)
135 (2.5%)	17300 (3.4%)	139,445 (4.2%)	139,445 (5.2%)
443 (1.6%)	21 (3.2%)	21 (4.2%)	135, 4444 (5.2%)
Others (22.2%)	Others (22.4%)	135, 4444 (3.8%)	21 (4.4%)
		Others (22.3%)	Others (22.9%)
November 2003	**December 2003**	**January 2004**	
135 (28.9%)	135 (28.4%)	135 (23.5%)	Others ≈ 185 distinct sequences each month
135, 4444 (13.6%)	135, 4444 (17.3%)	6129 (14.6%)	
80 (9.2%)	80 (9.4%)	135, 4444 (14.5%)	
1433 (8.7%)	1433 (8.5%)	1433 (8.4%)	
139 (8%)	139 (8.4%)	139 (6.3%)	
445 (6.4%)	445 (5.4%)	445 (5.7%)	
21 (3.5%)	139, 445 (2.6%)	80 (4.3%)	
554 (2.9%)	21 (2.1%)	139, 445 (2.5%)	
Others (20.8%)	Others (17.9%)	Others (20.2%)	

Table 1: Percentage of ports sequences per month

2.2.3. Open issues

These results show that only a few *ports sequences* are observed each month and that they represent a large volume of traffic. However, it is important to determine if they have the same *root causes* [Para88]. Indeed, they may cover up more subtle and rare attacks because of the volume of data they represent in the honeypots logs.
We call a *root cause* the most basic cause that can be reasonably identified as the origin of the attack on a given *ports sequence*. A *root cause* can be reasonably associated to one *attack tool*, or at least to one of its configuration.
Thus, we propose in Section 3 to identify the most basic factors that characterize these *root causes*. Julisch et al. have called such an approach a *Root Cause Analysis* (RCA) in [Juli03]. The idea here is the same but the context is different: their goal was to eliminate semi automatically clusters of false alarms while we are interested in identifying attack tools. Furthermore, they were looking at intrusion detection alarms while we consider raw network packets sent against honeypots. Last but not least, the clustering algorithms used differ: they used hierarchical tree classifications while we choose association rules [Juli03, Agra93].
The *root causes* we get are grouped into clusters: one cluster is supposed to characterize one *root cause.* Section 3.3 shows that this statement is not totally exact. As a consequence, we propose a refinement of the algorithm to get a better *root cause* identification.
Finally, if such *root causes* are efficiently identified, we can imagine new techniques to enrich their identification and to ease correlation of alerts issued by intrusion detection systems in production networks. Indeed, if frequent attacks are correctly identified, we can eliminate the ambient noise they produce in log files to put all efforts in rarer phenomena which require special attention.
This paper aims at showing that *ports sequences* do not describe one and only one *attack tool* in a univocal way. In addition, we propose a simple technique that efficiently identifies such *attack tools* (or *root causes).*

3. Root Causes Identification

3.1. Accurate parameters

We have observed many repetitive attacks in our honeypot logs. As explained in 2.2.3, we propose to identify them by means of simple features. Our database offers many kinds of information as presented in 2.1.2. We have made several tests to find the most relevant ones. For the sake of concision, we only present in the following three of them. They give condensed and interesting results, in regards of the experiments we made with other parameters. They are listed below:

- T: the number of machines in our environment targeted by one source
- n_i: the number of packets sent by one source to machine i. $i \in \{1,2,3\}$.
- N: the total number of packets sent by one source to the whole environment. In other words, $N = \sum n_i$, with $i \in \{1,2,3\}$.

As said in Section 2.2.3, we want to determine the *root causes* of frequent processes observed in our honeypot environment. Many techniques exist to do *Root Cause Analysis* [Lati02, Liv01, Para88]. At this stage, the algorithm choice is not as important as the results we expect to get from it. Therefore, we first propose to apply a simple one. If results are encouraging, more complex techniques will be applied to refine them, if needed.
The method we apply is described in the following subsection and results are given in subsection 3.2.3.

3.2. Association Rules

3.2.1. AR Motivations

We determine T, n_i and N for each attack source associated to a given *ports sequence*. They are then inserted in a new database table. To make a long story short, there is one table per *ports sequence*, with each column being one parameter {T, n_i, N} and each line being information on one *attack source*. The idea consists in searching for interesting relationships between the items contained in each table. One simple but widespread solution consists in applying association rule (AR) mining.

3.2.2. AR Applications

Association rules are powerful exploratory techniques which have a wide range of applications in many areas of business practice and research - from the analysis of consumer preferences or human resource management, to the history of language [Agra93]. For instance, these techniques enable analysts and researchers to uncover hidden patterns in large data sets for so-called *market basket analysis*, which aims at finding regularities in the shopping behavior of customers of supermarkets. With the induction of association rules one tries to find sets of products that are frequently bought together, so that from the presence of certain products in a shopping cart one can infer (with a high probability) that certain other products are present.
The usefulness of this technique to address unique data mining problems is best illustrated in a simple example: "If a French customer buys wine and bread, he often buys cheese, too". It expresses an association between (set of) *items*. This association rule states that if we pick a customer at random and find out that he selected certain items, we can be confident, quantified by a percentage, that he also selected certain other items. The standard measures to assess association rules are the *support* and the *confidence* of a rule, both of which are computed from the *support* of certain item sets. Some additional rule evaluation measures are sometimes considered, but they are out of the scope of this work. Moreover Han et al. have classified association rules in various ways, based on some criteria (type of values handled in the rule, dimensions of data involved in the rule, levels of abstractions involved in the rule set, etc) [Han01]. We report the interested reader to the seminal paper [Agra93] for more information on AR evaluation methods. The *support* and *confidence* indices are presented below. Many algorithms currently exist to restrict the search space and check only a subset of rules, but if possible, without missing important rules. Generally speaking, association rules mining is a two-step process:

- Find all frequent itemsets: by definition, each of these itemsets will occur at least as frequently as a pre-determined minimum support count.
- Generate strong association rules from the frequent itemsets: By definition, these rules must satisfy minimum support and minimum confidence.

We use the algorithm presented by Agrawal et al. in [Agra94], which is called *the Apriori algorithm*.
Thus, for a given rule R: A and B -> C, we consider two parameters:

- *Support(R)* = the support is simply the number of transactions that include all items in the antecedent and consequent parts of the rule. (The support is sometimes expressed as a percentage of the total number of records in the database.) $$Support(R) = \frac{\#transactions(containingA, B, C)}{\#transactions}$$

- *Confidence(R)* = is the ratio of the number of transactions that include all items in the consequent as well as the antecedent (namely, the support) to the number of transactions that include all items in the antecedent. $Confidence(R) = \frac{\#transactions(containingA, B, C)}{\#transactions(containingA, B)}$

3.2.3. AR Interesting Outputs

Association rule mining consists in finding frequent itemsets (set of items, satisfying the *minimum support threshold*), from which strong association rules are generated. These rules also satisfy a *minimum confidence threshold*.
The A*priori* algorithm is applied to each *ports sequence* table in our database. Table 1 gives some resulting frequent itemsets with their respective support. We only give some of them for conciseness concerns. We provide for 11 sequences some frequent itemsets as well as their corresponding support. These *ports sequences* are the most frequent ones, as observed in [DaPD04]. The 'Total Number of clusters' columns provides the number of obtained clusters, with support higher than 1%.
For instance, we find 5 important clusters for *ports sequence* {445}. Three of them represent attacks on the sole port 445, having the following common features:

- *Cluster Nb 1*: Three virtual machines are targeted. One packet is sent to the first machine and three to the two others.
- *Cluster Nb 2*: One machine only is targeted and it receives strictly one packet.
- *Cluster Nb 3*: Three machines are targeted and the attack source sends a total of eight packets

These three clusters stem for almost 90% of attacks with *ports sequence* equal to {445}. Furthermore they are mutually exclusive (an attack cannot belong to two different clusters).

Ports Sequences	**Some frequent itemsets**	**Support (%)**	**Total number of clusters (support> 1%)**
{445}	Cluster 1: T = 3 & N = 9 (n1=3 & n2=3 & n3 = 3) Cluster 2: T= 1 & N = 1 Cluster 3: T = 3 & N = 8	65.5% 18.4% 6.1%	5
{135,4444}	Cluster 4: T = 3 & N = 13 (n1 = 3 & n2 = 8 & n3 = 2) Cluster 5: T = 3 & N = 26 (n1 = 6 & n2 = 16 & n3 = 4)	66.3% 33.6%	3
{80}	Cluster 6: T = 1 & N = 3 Cluster 7: T = 3 & N = 11 (n1 = 3, n2 = 5, n3 = 3) Cluster 8: T =1 & N = 1	55.8% 18.7% 5.6%	6
{1433}	Cluster 9: T = 3 & N = 9 (n1=3 & n2=3 & n3 = 3) Cluster 10: T = 3 & N = 3 (n1=1 & n2=1 & n3 = 1) Cluster 11: T = 3 & N = 12 (n1=4 & n2=4 & n3 = 4)	53.9% 18.7% 18.1%	5
{139, 445}	Cluster 12: T = 3 & N = 38 (n1 = 18 & n2 = 18 & n3 = 2) Cluster 13: T =3 & N = 145 Cluster 14: T = 1 & N = 74	41.1% 7.6% 6.8%	18

{135}	Cluster 15: T = 3 & N = 13 (n1 = 3 & n2 = 7 & n3 = 3) Cluster 16: T = 3 & N = 21 (n1 = 3 & n2 = 15 & n3 = 3)	50.1% 15.3%	4
{17300}	Cluster 17: T = 3 & N = 3 (n1 = 1 & n2 = 1 & n3 = 1) Cluster 18: T = 3 & N = 6 (n1 = 2 & n2 = 2 & n3 = 2) Cluster 19: T = 3 & N = 9 (n1 = 3 & n2 = 3 & n3 = 3)	39.8% 34.6% 25.5%	9
{21}	Cluster 20: T = 3 & N = 16 (n1 = 1 & n2 = 1 & n3 = 1) Cluster 21: T = 3 & N = 11 (n1 = 1 & n2 = 1 & n3 = 1) Cluster 22: T = 3 & N = 7 (n1 = 1 & n2 = 1 & n3 = 1) Cluster 23: T = 1 & N = 11 (n1 = 1 & n2 = 1 & n3 = 1)	23.4% 13.7% 10.6% 9.1%	25
{80, 57, 21}	Cluster 24: T = 3 & N = 40	65.1%	5
{554}	Cluster 25: T = 3 & N = 8 Cluster 26: T = 1 & N = 1 Cluster 27: T = 3 & N = 9	50.8% 29.8% 7.4%	6
{139}	Cluster 28: T = 3 & N = 9 (n1=3 & n2=3 & n3 = 3) Cluster 29: T = 2 & N = 8 Cluster 30: T = 1 & N = 4	49.2% 9.5% 6.3%	6

Table 2: Some frequent itemsets for 11 ports sequences

By choosing a support higher than 1%, we have built 92 clusters for the presented *ports sequences*. They represent 87 % of the attacks we have observed during the one year experiment. We have applied this method for all ports sequences, and we have selected relevant rules to create exclusive clusters.

A first major observation is that there can be several clusters associated to a single *ports sequence*. Thus, *ports sequences* cannot be directly used to identify *root causes*.

Furthermore, this result clearly validates that most of the collected data is redundant and predictable insofar as a few clusters can be associated with a large number of observed attacks.

There are some other phenomena that must be clarified. First, for one given attack, we sometimes observe that virtual machines receive the same number of packets from the attack source. This is due to blind and simple scans on IP ranges. Secondly, there is a clear correlation between packets numbers and OSs/services running on our honeypot machines. Obviously enough, attacks on open ports correspond to more important packets traffic (see *ports sequence* {135} for instance). Finally, some *ports sequences* such as {139, 445} have a larger number of associated clusters. The reason is that such attacks often correspond to high packets traffic: they target Windows machines and try to obtain a lot of information from these talkative netbios services. There might be some tcp retransmissions that make parameters value somehow different in terms of packets numbers. A clustering refinement would be possible in this case.

The previous clusters are obtained based on very simple parameters. They first show that basic *ports sequences* do not identify *root causes* in a unique way. Secondly, each cluster is now supposed to represent one *root* cause, or *attack tool*. We show in the following section this is partially correct by checking clusters' coherency. We then introduce another membership property to refine clusters.

3.3. Levenshtein distances

3.3.1. Introduction

We have obtained interesting clusters in Section 3.2. They cover a large percentage of attacks, but we cannot be sure if they are *good* clusters or not. One can argue that there might be two different attacks generating the same amount of packets against the same *ports* sequences. Clusters are built on very simple parameters, and their consistencies must be validated.
In order to validate clusters consistency, we consider packet data contents. The payloads of all packets sent from the same source are concatenated to form a simple text phrase, thanks to the tethereal utility [Tethe04]). Tethereal is the command line version of the popular network traffic analyzer tool ethereal. It allows examining data from a capture file and browsing detailed information for each packet in a text format. Thus, we consider each phrase as a tethereal line, with '||' separators. Figure 2 gives a short phrase of an ftp attack for illustration.

EXTRACTED FROM ETHEREAL LOG: Source X attack (2 packets received):

Packet 1-> File Transfer Protocol (FTP) || Request: USER || Request Arg: anonymous
Packet 2-> File Transfer Protocol (FTP) || Request: PASS || Request Arg: Agpuser@home.com

EXTRACTED FROM ETHEREAL LOG: Source Y attack (2 packets received):

Packet 1-> File Transfer Protocol (FTP) || Request: USER || Request Arg: anonymous
Packet 2-> File Transfer Protocol (FTP) || Request: PASS || Request Arg: Mgpuser@home.com

Phrase distance: 1
One substitution **Agpuser@home.com** $\longrightarrow$ **Mgpuser@home.com**

Figure 2: examples of phrases from an ftp attack

Each cluster gathers all *attack sources* that are assumed to be due to a single *root cause*, i.e. to the use of the same attack tool against our honeypots. We define for each *attack source* its associated *'attack phrase'*. Then, we compare for each attack of one given cluster the phrase distances to all others phrases of the same cluster. This technique is based on the Levenshtein edit distance which is explained below.

3.3.2. Phrase distance approach

The Levenshtein distance (LD) algorithm has been used in many domains, such as spell checking, speech recognition, DNA analysis or plagiarism detection. It is a measure of the similarity between two strings, which we will refer to as the source string (*s*) and the target string (*t*) [Borg03]. The distance (sometimes called edit distance) is the number of deletions, insertions, or substitutions required to transform *s* into *t*. For example,

- If *s* is "Agpuser@home.com" and *t* is "Agpuser@home.com", then LD(*s*,*t*) = 0, because no transformations are needed. The strings are already identical.
- If *s* is "Agpuser@home.com" and *t* is "Mgpuser@home.com", then LD(*s*,*t*) = 1, because one substitution (change "A" to "M") is sufficient to transform *s* into *t*.

In general the two components of the phrase distance (i.e. the string distance and the positional distance) can have a different cost from the default (that is 1 for both) to give another type of phrase distance. There is a third component: a cost that weights on the phrases that have less

exact matches. It is described in details by Roger et al. in [Borg03]. This third component is disabled by default (i.e. it has a 0 cost), but it can be enabled with custom cost.
The method we apply sums the phrase distance from the words from the set (i.e. formed by the defined set of characters) and the phrase distance calculated from the "words" belonging to the complementary set. Moreover, the algorithm used to find the distance is the "Stable marriage problem" one [Gus89, Ksm01, Mathw] This is a matching algorithm, used to harmonize the elements of two sets on the ground of the preference relationships (in this case the string distance of single "words" plus the positional distance plus eventually the exact match weight).

3.3.3. Results

The algorithm has been applied to clusters built in Section 2.2.3. For a given cluster, we compare all *attack phrases* and calculate the average distance. More concretely, let C be the set of *attack phrases*, with |C| = n. Let D(a,b) be the phrase distance of a and b and i any *attack phrase*. Then, the average distance from i to other elements of C is:

$$d_i = \sum_{j \in C} \frac{D(i,j)}{n-1}$$

Thus, the average distance is defined as:

$$D_C = \sum_{i \in C} \frac{d_i}{n} = \sum_{\substack{i,j \in C \\ i<j}} \frac{2.D(i,j)}{(n-1).n}$$

Some results are presented in Table 2. The ‘phrase distance’ column gives the average distance D_c after having computed all intermediate values.
The standard deviation (square root of the *variance*) is a frequent measure to describe variability in data distributions. It is a measure of the average amount by which observations deviate on either side of the mean. Unlike the variance, the standard deviation describes variability in the original units of measurement. Its value is given in the ‘Standard deviation’ column.

Cluster	Ports sequence	Phrase distance	Standard deviation
Cluster 1	{445}	0	0
Cluster 2	{445}	0	0
Cluster 3	{445}	0	0
Cluster 4	{135, 4444}	0	0
Cluster 5	{135, 4444}	22	84
Cluster 9	{1433}	39	311
Cluster 10	{1433}	0	0
Cluster 11	{1433}	641	1213
Cluster 20	{21}	88	258
Cluster 21	{21}	0	0
Cluster 22	{21}	0	0

Table 3: Cluster coherency with Levenshtein distance

We observe from Table 3 that seven out of eleven clusters have a null phrase distance. They contain very similar *attack phrases*. We say that they are coherent and correspond to one *root cause, i.e. to a single attack tool.* The null value can be explained by the fact that these attacks are limited to simple and repetitive scans.
A deeper analysis of other clusters is required to understand the phrase distance value. It suffices to extract different patterns to obtain some information. There are two options: Either we observe very strong similarities and non zero phrase distances are simply due to packets

random fields. In this case, we consider the cluster corresponds to one *root cause,* and the random fields are *attack tools* features. Or we find different patterns that have nothing in common. In this situation, we build new sub-clusters and reevaluate the new distance phrase. In our examples, we have:

- Cluster 20 of *ports sequence* {21}: there are two anomalous attacks which bias the average phrase distance value. They are totally different from all the others. Once eliminated, we get a new average phrase distance of two. This is due to the random login name used by the attack tool on port 21 (see figure 2).
- Cluster 5 of *ports sequence* {135, 4444} is not null because of data payload differences. This *ports sequence* is associated to MBlaster attacks and some variants contain data payload in the first packets [Sym03].
- Cluster 9 and 11 of *ports sequence* {1433}: both can be split into two sub-clusters: those which contains data payload of 8 bytes (all zeros) and of 24 bytes (all zeros). These sub-clusters have a null phrase distance. The initial cluster was not totally correct.

We get similar results with other clusters. Very few of them have required a small refinement (sub-clustering). This validates that with simple parameters, we have built interesting clusters that characterize main attack types our honeypot environment is facing.

3.3.4. Conclusion

We have presented a method to find root causes of frequent traffic met in the honeypots logs. The *root cause analysis (RCA)* steps are:

- Find frequent *ports sequences* (table 1 in our example)
- Build clusters on simple parameters (such as those used in our example on Section 3.1)
- compute the average distance phrase and its standard deviation to validate the clustering phase
- Split the cluster in sub-clusters, if necessary, by recursively applying the phrase distance approach (see Cluster 9 and 11 of ports sequence {1433})
- Get one cluster per *root cause*. The *root cause* corresponds to a specific configuration of an *attack tool*.

This method validates that an in-depth analysis is a prerequisite to a better understanding of malicious activity collected by our honeypots. The apparent regularity we observed in [DAPD04, DPDE04] is *de facto* due to various *root causes* which are precisely identified by our method.

Moreover, the obtained results can be exploited in different ways. First, they can permit to filter honeypot logs and detect newly observed *root causes*. Secondly, they allow us to track *root causes* evolution. Finally, they may characterize some specific *attack tools* and help finding some new signatures of them. Three applications are presented in the following Section 4.

4. Clusters applications

4.1. Approximate number of attacks

We have obtained data reduction with simple clusters. Each of them groups attacks which can be associated to one *attack tool*. If we now create all possible clusters (same operation than in 3.2.3 but with min(Support)=0%) for the observed *ports sequences*, we obtain the –pessimistic- number of *attack tools* (tools fingerprints) we have observed during the experiment year.

# attack tools = ∑ #clusters(*i*), for each *ports sequence i* = 753.
This number is very conservative. Indeed, we consider all observed attacks. However, many *ports sequences* are not exactly due to attack tools on our machines. They can come from other phenomena such as backscatters [MoVS01].
Table 3 presents the quantitative number of obtained clusters. The first column takes into account the Association Rules (AR) clustering only, while the second column presents the two steps of our algorithm (Association Rules –AR- and phrase distances –LD-).

Support value	AR	AR + LD
> 0%	491 clusters	753 clusters
> 1%	92 clusters	152 clusters

Table 4: Quantitative number of clusters

Finally, this result is conservative insofar as we did not consider cases of packet losses and retransmissions. Clusters are built on received packet numbers, so attacks can be placed in two distinct clusters if losses occur despite their very strong similarity.

4.2. Tools fingerprinting

By construction, each cluster can be potentially associated to one *attack tool*. This requires either a good knowledge of blackhat tools, or a short lookup in security advisories pages or incidents mailing lists [ISC03, Secu04]. Moreover, clusters must be analyzed in depth thanks to parameters described in Section 1.2.3. From the clusters presented above, we get the following information:

- Three clusters of *ports sequence* {135, 4444}: three variants of MBlaster worm. Others seem to exist (see [Sym03]), but we did not observe them in our environment.
- Two clusters of *ports sequence* {554}: They correspond to two different configurations of an RTSP scanner [Rtsp]
- Cluster 1 of *ports sequence* {21}: it corresponds to a famous ftp scanner, named Grim's Ping [GrimP] .
- Cluster 2 of *ports sequence* {21}: it can be associated to another ftp scanner running on Windows machines, named Roadkil's FTP Probe [Road04]
- Cluster 2 of ports sequence {139, 445}: it corresponds to an enumeration and penetration-testing scanner called the Leviathan Auditor [Lev04].
- Cluster 1 of ports sequence {1433}: this is due to another worm named SQLSnake – or any of its variants. It scans for open SQL Server 2000 [Snake].
- Cluster 2 of ports sequence {1433}: this corresponds to another scanner, named Sfind.exe [Sfind].

It suffices to test these tools in our environment to find their 'signatures'. If one attack has been observed from this tool, a cluster with similar 'signature' exists.
This method is all the more efficient that main attackers do not even take time to modify such tool 'signatures' (for instance, the default logins & passwords of ftp scanners).
Finding that attacks have characteristic signatures is not a breakpoint by itself. Rule-based intrusion detection systems are built on such observations. However, a honeypot environment permits to identify frequent observed signatures. Such attacks trigger lots of 'true positive alerts' and, when identified, are not of real interest for the busy administrator. It would be interesting to extract such alerts, so that more specific alerts remain. This is even more interesting that administrators are often overwhelmed with alerts from many security systems. In addition, a similar work must be performed on the honeypots data. After having extracted well-known observed attacks, a deeper look at rare and strange ones is compulsory to

understand what happened. Are they due to a new tool? Are they specific to the honeypot environment? Analysis of data collected by honeypots is then reduced to the observation of a few phenomena. In practice, this makes honeypots more interesting to administrators that already have a large amount of data to deal with.

4.3. Root causes evolution

In this section, we consider the *root cause* evolution over time. Indeed, identified *root causes* are tightly associated to *attack tools*. These tools are often used by script kiddies for a while and then abandoned for newer ones.
Figure 3 represents the evolution of some *root causes* over that year 2003, for attacks coming from Australia. The choice of Australia is justified by the fact most of the observed attacks on our honeypot platform in the year 2003 were coming from this country [DPDe04, DaPD04].
We have shown in [DaPD04] that the number of Australian attacks is quite constant at the beginning of year 2003. A closer look shows that some curves cross between March and June. Thus, this phenomenon validates that apparent constant phenomena are misleading and do not take into account microscopic, yet important, (attack tools) variations.
Furthermore, this analysis enables us to clearly identify new phenomena, such as Mblaster, represented by cluster 4. Finally our *root cause* analysis presents some limitations that we intend to correct: generally speaking, one cluster clearly represents one unique *root cause*, but we cannot guarantee that one *root cause* is represented by only one cluster. We have presented some discriminatory parameters to build *root causes* clusters. Other parameters must be specified to refine these clusters and merge a unique *root cause* into one single cluster.

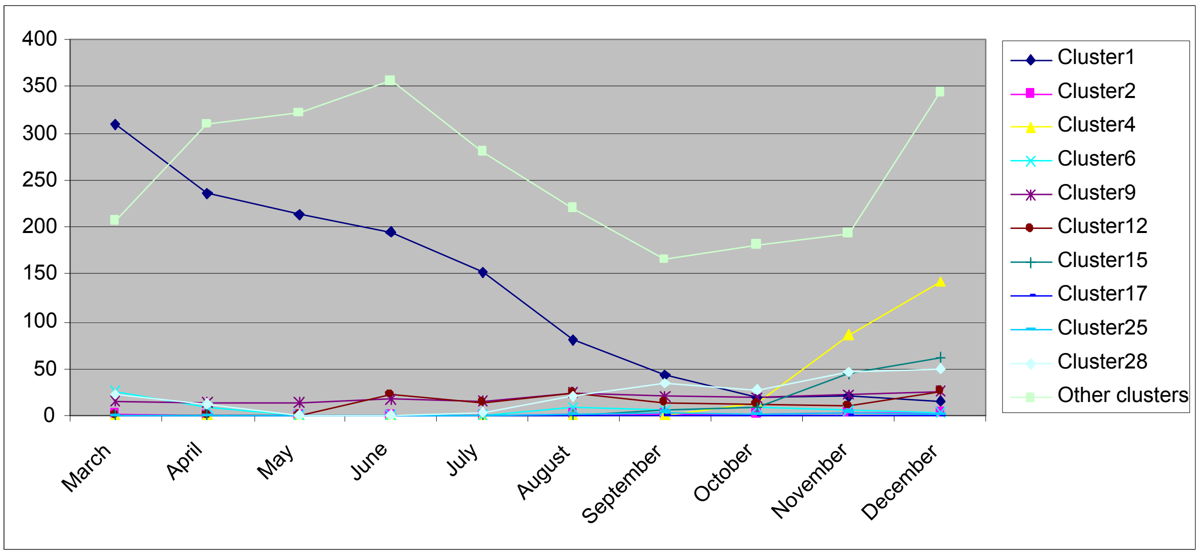

Figure 3: Root causes evolution over months for Australian attacks

5. Conclusion

Honeypots are very interesting observation environments that enable us to easily collect malicious data. However, very few efforts are currently made to take advantage of these rich information sources. Much of the effort is devoted to honeypots design.
We show in this paper that simple clustering techniques can be applied to obtain more in-depth information on observed attacks. We have proposed one solution based on association rules and phrases distance. Results are very promising and confirm that such an analysis is

meaningful. Indeed, we obtain clusters representing *root causes* of attacks. We have shown with this approach that a large amount of malicious traffic corresponds to a few number of *root causes*. Such *root causes* are important information on the *attack tools* in use. Other applications such as tool fingerprinting or analysis of tools evolution were also presented. Future work will consist in refining the clustering method and reusing such clusters to help correlating alerts in the Intrusion Detection process.

6. Bibliography

[Agra93] R. Agrawal, T. Imielinski, A. Swami. “Mining Association Rules between Sets of Items in Large Databases”. *In Proc. of the 1993 ACM SIGMOD International Conference on Management of Data*.

[Agra94] R. Agrawal, R. Srikant. “Fast Algorithms for Mining Association Rules”. *IBM Almaden Research Center*, 1994.

[Bell92] S. Bellovin, “There Be Dragons”, *Proc. of the Third Usenix Security Symposium*, Baltimore MD. Sept. 1992.

[Bell93] S. M. Bellovin, “Packets Found on an Internet”, *Computer Communications Review* 23:3, pp. 26-31, July 1993.

[Borg03] C. Borgelt, “Finding Association Rules/Hyperedges with the Apriori Algorithm”. *Dpt of Knowledge Processing and Language Engineering,* Germany, 2003.

[Cohe99] *Deception Tool Kit*, DTK, Fred Cohen & Associates. http://www.all.net/dtk/dtk.html

[CLPB01] F. Cohen, D. Lambert, C. Preston, N. Berry, C. Stewart and E. Thomas, “*A Framework for Deception*”, Tech. Report, July 2001, http://all.net/journal/deception/Framework/Framework.html

[DaPD04] M. Dacier, F. Pouget, H. Debar, “Attack Processes found on the Internet”. *NATO Symposium IST-041/RSY-013*, Toulouse, France, April 2004.

[DPDe04] M. Dacier, F. Pouget, H. Debar, “Honeypots, a Practical Mean to Validate Malicious Fault Assumptions”. *Proc. Of the 10th Pacific Ream Dependable Computing Conference (PRDC04)*, February 2004.

[Disco] *The Disco tool* home page: http://www.altmode.com/disco/

[GenH03] “*Know Your Enemy: GenII Honeynets Easier to deploy, harder to detect, safer to maintain*”, by the honeynet Project members, June 2003. Available on line: http://project.honeynet.org/papers/gen2/

[Grah03] R. Graham, *“MS-RPC/DCOM/Blaster case study”*, ISS presentation available on line at: http://csiannual.com/classes/v12.pdf

[GrimP] Grim’s Ping home page: http://grimsping.cjb.net/

[Han01] J. Han, M. Kamber, *“Data Mining: Concepts and Techniques”*, Morgan Kaufman Publishers, 2001.

[HoIe03] Irish Honeynet Alliance members. Collected data available on line: www.honeynet.ie/results.htm

[HonAll] Honeynet Alliance home page: http://www.honeynet.org

[HonNo] Norvegian Honeynet Alliance members: Collected data available on line: www.honeynet.no/reports/

[ISC03] *Internet Storm Center*, home page: http://isc.incidents.org/

[Juli03] K. Julisch, “Using Root Cause Analysis to Handle Intrusion Detection Alarms”. *PhD dissertation*, IBM Research laboratory of Zurich, Switzerland, 2003.

[Kraw04] N. Krawtez, “Anti-Honeypot Technology”, *IEEE Security and Privacy.* Vol 2, Nb 1, p. 76-78, 2004.

[Ksm01] K. Smani lectures available at: http://www.csee.wvu.edu/~ksmani/courses/fa01/random/lecnotes/lecture5.pdf

[Labrea] Labrea Tarpit Project, http://labrea.sourceforge.net/

[Lati02] R.J. Latino, K. latino, “Root Cause Analysis: Improving Performance for Bottom Line Results”. *CRC Press, LLC,* 2002.

[Lev04] Leviathan Auditor open source project, home page: http://leviathan.sourceforge.net/

[Liv01] A.D. Livingston, G. Jackson, K. Priestley, “Root Causes Analysis: Literature Review”. *HSE Books*, 2001. Available at: http://www.hsebooks.co.uk

[Mathw] The Stable Marriage Problem: http://mathworld.wolfram.com/StableMarriageProblem.html

[MoVS01] D. Moore, G. Voelker et S. Savage. "*Inferring Internet Denial-of-Service Activity*", 2001 USENIX Sec. Symp. www.caida.org/outreach/papers/2001/BackScatter/usenixsecurity01.pdf

[Netgeo] *Netgeo Utility*, available online at http://netgeo.caida.org/perl/netgeo.cgi

[Nevi03] A. Neville, “*IDS Logs in Forensics Investigations: An Analysis of a Compromised Honeypot*”, March 2003. Available on line: http://www.securityfocus.com/infocus/1676

[Para88] M. Paradies, D. Busch. "Root Cause Analysis at Savannah River Plant". *In Proc. of the IEEE Conference on Human Factors and Power Plants,* page 479-483, 1988.
[PDD03] F. Pouget, M. Dacier, H. Debar: "Honeypot: a comparative survey". *Eurecom Report, RR-03-81*, September 2003.
[P0f] p0f passive fingerprinting tool home page: http://lcamtuf.coredump.cx/p0f-beta.tgz
[Pow03] "Conceptual Model and Architecture of Maftia", D. Powell, R. Stroud (editors), MAFTIA Project (IST-1999-11583), Deliverable D21, January 2003; available on line at http://www.maftia.org
[Road04] Roadkil's FTP Probe home page: http://homepages.ihug.com.au/~roadkil/ftpprobe.htm
[Rtsp] RTSP Scanner available at: http://iperl.homelinux.org/haxor/scanner.pl
[Secu04] Honeypots mailing list archives from SecurityFocus: http://www.securityfocus.com/popups/forums/honeypots/faq.shtml
[Sfind] Sfind scanner home page: http://www.force5web.com/articles/sql_scan.htm
[Snake] SQLSnake presentation available at: www.giac.org/practical/Christopher_Short_GCIH.doc
[SoMS01] D. Song, R. Malan and R. Stone, "*a global snapshot of internet worm activity*", November 2001. Technical Report, http://research.arbor.net/downloads/snapshot_worm_activity.pdf.
[Stol88] C. Stoll, "Stalking the Wiley Hacker", *Communications of the ACM*, Vol. 31 No 5. May 1988.
[Spit02] L. Spitzner, "*Honeypots: Tracking Hackers*", Addislon-Wesley, ISBN from-321-10895-7, 2002.
[Sym03] Symantec MBlaster advisories available at: http://securityresponse.symantec.com/avcenter/venc/data/
[Tethe04] tethereal utility home page: http://www.ethereal.com/tethereal/
[THP03] *Tiny Honeypot* home page: http://www.alpinista.org/thp/
[VMware] *VMWARE*, User's manual. Version 3.1, home page: http://www.vmware.com