HONEYNETS: FOUNDATIONS FOR THE DEVELOPMENT OF EARLY WARNING INFORMATION SYSTEMS

F. Pouget, M. Dacier, V.H. Pham
Institut Eurecom
2229, route des Crêtes ; BP193
06904 Sophia-Antipolis Cedex ; France
Email: {pouget,dacier,pham}@eurecom.fr

H. Debar
France Telecom R&D[1]
42, rue des Coutures; BP 6243
14066 Caen Cedex 4 ; France
Email: herve.debar@francetelecom.com

Abstract This paper aims at presenting in some depth the "Leurré.com" project and its first results. The project aims at deploying so-called low level interaction honeypot platforms all over the world to collect in a centralized database a set of information amenable to the analysis of today's Internet threats. At the time of this writing, around two dozens platforms have been deployed in the five continents. The paper offers some insight into the findings that can be derived from such data set. More importantly, the design and the structure of the repository are presented and justified by means of several examples that highlight the simplicity and efficiency of extracting useful information out of it. We explain why such low cost, largely distributed system represents an important, foundational element, towards the building of early warning information systems.

Keywords Honeynet, Internet Attacks, Database, Malware, Cybercrime

1 INTRODUCTION

The mere existence and success of workshops such as WORM, DIMVA and SRUTI ([1],[2],[3]) indicate that Internet-wide infectious epidemics have emerged as one of the leading threats to information security and service availability. Important contributions have been made in the past that have proposed propagation models ([4],[5],[6]) or that have analyzed, usually by reverse engineering specific worms, their modus operandi [7],[8],[9],[10]. Several initiatives exist to monitor real world data related to worms and attacks propagation. The Internet telescopes, the DShield web site, the work reported by John McHugh in [11] are among them. These approaches are extremely valuable and have in common the wish to collect very large amount of data collected thanks to a large amount of monitored addresses.

By launching the Leurré.com project in the course of 2004, we have decided to take a different, yet complimentary, approach. Instead of collecting aggregated information, such as Netflow records, about a very large number of connections and hosts, we have decided to keep in one centralized database very precise information concerning a very limited number of nodes under close scrutiny. Furthermore, we have proposed to several partners to join the project by hosting such data collection platform. Concretely speaking, we do use low interaction honeypots, based on the honeyd software [12], emulating 3 vulnerable machines on a single home computer. We do record all packets sent to or from these machines, on all platforms, and we store the whole content of all the packets into a database. Special care must be taken in the design of the database to offer an easy and intuitive way to retrieve interesting information.

Before launching the Leurré.com project, we have investigated the usefulness of honeypots to

[1] This research is supported by a research contract with France Telecom R&D, Contract Number 46127561

analyze Internet threats. Therefore, a first platform has been deployed for almost two years and its results analyzed. Experience gained led to the publications of several papers and to several iterations in the design of the database used to store the information. The most important results can be summarized as follows:

- In [13], we present early results based on a couple of months of data highlighting the potential of the data set we were in the process of building. The apparent limited number of attacks as well as the regularity of their origins is presented and explained.
- In [14], we introduce a clustering algorithm to group together traces of attacks likely due to the same attack tool. The method is presented, a technique to test the consistency of the obtained clusters is offered and experimental results, based on 16 months of data are detailed.
- In [15] we highlight, based on concrete cases, the limitations of the other approaches such as the internet telescopes or Dshield. We discuss the aftermath of a very atypical worm, the Deloder one, as we see it from our honeypot point of view. We also describe an experimentation protocol, as well as its results, aiming at validating the assumption that several groups of attackers are exchanging information on the Internet in a non trivial way.

All these results have been obtained thanks to a single platform collecting data from one environment only. Encouraged by these results, we have decided to collect the very same kind of data from a diversity of places. This led to the creation of the Leurré.com project. A key element for the success of such project is its ability to store large amount of data and, more importantly, to offer its users a simple way to retrieve meaningful pieces of it efficiently. This is where the design of the centralized database plays a key role. In the next Sections, we will present not only the structure of this database but also how it can be used to bring to light some ongoing attack processes in the Internet. Using several examples, we will present in parallel the structure of the database and the results it delivers to us.

The structure of the paper is as follows. Section 2 introduces the design of the platform we are using. Section 3 details the raw data we can get from the platforms we monitor and how this information can be stored in and retrieved from the database. Section 4 explains the need for enriching this data set with several kinds of external related information such as, for instance, the geographical locations of the attacking sources. For each new type of added attribute, we explain how it fits into the general database structure and, by means of examples, how one can take advantage of it. Section 5 explains why, from a usability point of view, it is also important to build what we call meta-data tables. These are tables that contain redundant information that could be retrieved by means of SQL queries on the database but that, for some efficiency reasons, we explicitly include as part of its design. Examples are given. Section 6 concludes the paper.

2 HONEYPOTS

2.1 Initial Set Up

Average daily tcpdump file size	1.5 Mbytes
Maximum file size	9.6 Mbytes
Number of files collected	516 days
Average number of packets per tcpdump file	13600 packets

Tableau 1: VMWare-Based Environmnent, a big picture

A detailed description of our initial platform as well as a thorough treatment of the state of the art in honeypots is given in [16]. This first platform is a so-called high interaction honeypot. It consists in a VMWare virtual environment with three virtual machines running various Operating systems (Linux RedHat, Windows 98, and Windows NT) and services (ftp server, web server, etc). Virtual machines are built on non-persistent disks [17] which means that changes are lost when machines are powered off or reset. In other words, rebooting a compromised machine offers us a new, clean, environment. This platform has been deployed in February 2003. We collect every day all traffic coming from or to the three virtual machines in tcpdump pcap files [18]. Table 1 summarizes the files characteristics we have obtained from February 2003 until now. The total number of collected files represents the

amount of days during which the platform was up and running.

2.1.1 Distributed Honeypots

The results obtained with this initial platform and summarized here above have shown that most of the attacks are caused by a few numbers of attack tools and that there are very stable processes occurring in the wild. Furthermore, as discussed in [15], the fingerprinting capability of the attack tools appears to be very limited. Therefore, one can reasonably decides to use low, instead of high, interaction honeypots despite the fact that they can be easily identified by a remote attacker. A low interaction honeypot being much cheaper to implement in terms of software and hardware, our hope is to see many institutions volunteering to deploy such a platform on their premises, whereas they wouldn't have been eager to pay the price of a full fledged high interaction honeypot.

As a consequence, we have implemented a new platform similar to the one presented before, but with emulated operating systems and services. We have developed such a platform based on open source software[2]: it emulates three different Operating Systems, Windows 98, Windows NT Server and Red Hat 7.3 respectively. The platform only needs a single host station, which is carefully secured by means of access controls and integrity checks. Every day, we connect to the host station machine to retrieve traffic logs and to check its security logs.

In the context of the Leurré.com project, we have started deploying these platforms all over the world [19]. At the time of this writing, we have deployed 25 platforms, in 5 continents and 12 different countries. We invite the interested reader to look at [20] for a first analysis of this distributed honeypot architecture.

Table 2 gives an overview of the data which is collected every day for these 25 platforms. We only consider here the tcpdump pcap files. Volumes vary greatly between platforms. For instance, log files collected on one German platform can be as twenty times bigger than those of one Lithuanian platform, whereas the number of collected packets can be almost thirty times higher. Therefore, this Table does not fairly represent any of the platforms but gives a rough idea of the amount of data we deal with.

Average daily dump file size	114 Kbytes
Maximum file size	13 Mbytes
Number of collected files	1492 files

Table 2: LEURRE.COM project, the big picture

3 RAW COLLECTED DATA

3.1 Raw Data

As explained in Section 2, we collect from the different honeypot platforms tcpdump files which contain observed suspicious packets. We also collect other information, such as application log files to verify the integrity of the platforms but these lie outside the scope of the paper as they are not used to analyze the attacks we face. It is worth noting that broadcast and multicast traffic is filtered out from the tcpdump files we collect (e.g. arp traffic, Cisco Discovery Protocol CDP, Spanning-Tree Protocol STP, etc). In other words, we only are interested in packets from/to the honeypot virtual machines specifically. At the network level, these are mainly IP and ICMP packets. At the transport protocol, they are mainly UDP and TCP ones.

3.2 New Definitions

This data needs to be properly organized as it will be used for further analysis and experiments. One major remark is that it is not obvious, *a priori*, to define what the best structure of the database could be as we do not know the most frequent queries that we will have to run on the dataset. As explained before, this is the reason why we went through several iterations in its design, as we were progressing with our research. The result presented here appears to have reached a relative stability, based on several months of work with it.

[2] The platform implements a modified version of Honeyd at this time.

In theory, no traffic should be observed to or from the machines we have set up. As a matter of fact, many packets hit the different virtual machines, coming from different IP addresses. Typically, if an attacker decides to choose one of our honeypots as his next victim, he tries to establish direct TCP connections or to send UDP packets against it. We group all these attempts into what we call a *"Tiny Session"* by contrast to the notion of "*Large Session*" which includes all *Tiny Sessions* that a given attacker might have launched against a given platform. In our setup, as we have three honeypots, a *Large Session* can be made of 1 to 3 *Tiny Sessions*. This idea of *Tiny* and *Large Sessions* is at the core of the design of the database. Therefore, the five most important tables in the database are the following ones:

- ***Host:*** this table contains all attributes (or links to other tables containing attributes) required to characterize one honeypot virtual machine.
- ***Environment***: this table contains all attributes (or links to other tables containing attributes) required to characterize one honeypot platform, i.e. a group of three *hosts*.
- ***Source:*** this table gathers all attributes (or links to other tables containing attributes) required to characterize one attacking IP within one day.
- ***Large_Session:*** this table contains all attributes (or links to other tables containing attributes) required to characterize the activity of one *Source* observed against one *Environment*.
- ***Tiny_Session:*** this table contains all attributes (or links to other tables containing attributes) required to characterize the activity of one *Source* observed against one *Host*.

It is worth noting that, according to these definitions, we consider that we have two distinct *Sources* of attacks when a given attacking IP address is observed twice on the same Environment with more than 24 hours between the two observations. The reason for this doing is experimental. We have found out that it was extremely rare, not to say impossible, to see Large Session lasting more than a couple of seconds. It is also extremely rare to observe the same IP address in multiple days. Last but not least, we do know that most of the attacks come from personal PCs which, usually, use temporary addresses [15],[16]. For all these reasons, it is quite likely that the same IP address observed in two different days is not linked to a single physical machine. Therefore, it makes sense to separate, in the database, the activities of the first from the activities of the second one by giving them distinct *Source* identifiers.

The Entity-Relationship diagram presented in Figure 1 exhibits the respective roles of these tables. The relationship between *Source* and *Environment* is called *Large_Session*. The relationship between one *Source* and one *Host* is called a *Tiny_Session*.

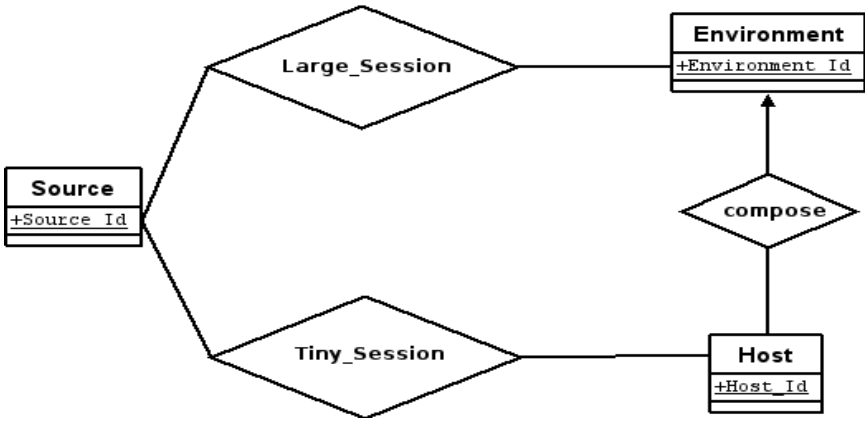

Figure 1: Entity6Relationship Diagram

3.3 Database Construction

The purpose of the Entity-Relationship model is to allow the description of the conceptual scheme. It is normal to turn it into a practical relational model upon which the real database is built. There exists a vast amount of knowledge and a rich literature explaining how to optimize this process, with respect to various types of constraints (speed, memory consumption, table sizes, number of

indices, etc.). We invite the interested reader to see [21],[22], for instance, for more on this topic.

Typically, the theory on the 'good' design of relational databases has been looking at problems such as: data redundancy in the tables, *update and insertion* anomaly issues. These problems are classically solved by transforming non optimal entity relationships models into their so-called "normal forms" (*Third Normal Form, Boyce-Codd Normal Form).* In our case though, the problem is slightly different and offers us the freedom not to follow that path. There are two important reasons for that. Firstly, we do not care about update and insertion anomaly issues as we do not touch to the data once they is inserted into the database. Indeed, the stored information is something that we have no reason to modify as it represents a fact of the past. Secondly, we do care a lot about the efficiency of querying the database and are not too concerned by space efficiency (be it on disk or in memory) as the total amount of data we deal with remains rather modest, compared to what existing database systems can handle. Therefore, we do have consciously decided to introduce in our design tables that contain redundant information. In other words, these tables contain information that could be retrieved by querying other tables. However, having the results of such queries available at hand in ad-hoc tables proves to be extremely useful when using the database. As a consequence, we decide to keep them, acknowledging the fact that, without their presence, the database would be more 'optimal' according to the classical criteria.

These remarks have been carefully taken into account for the database implementation. Figure 2 represents the Unified Modeling Language (UML) class diagram we have chosen corresponding to the ER diagram in Figure 1.

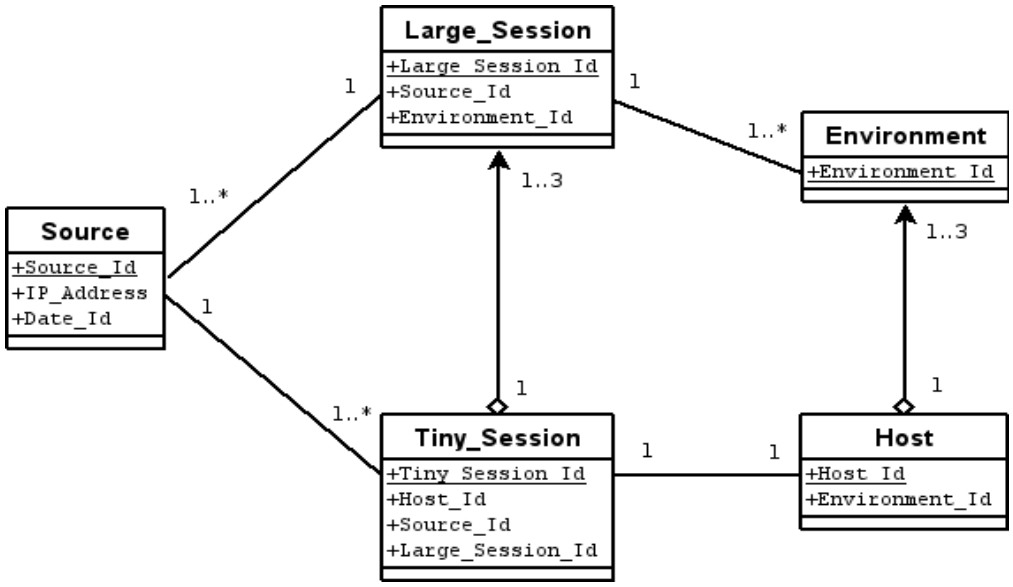

Figure 2: UML Class Diagram

Primary keys are underlined in Figure 2. For the *Source* table, the key *Source_Id* is equivalent to the expected double key *(IP_Address,Date_Id)*. For the *Large_Session* table, the primary key *Large_Session_Id* is equivalent to the expected double key *(Source_Id,Environment_Id)*. Finally, similarly to the two last cases, the primary key *Tiny_Session*_Id from table *Tiny_Session* is equivalent to the expected key pair *(Source_Id, Host_Id)*. Some redundancies have also been introduced on purpose: as an illustration, the *Large_Session_Id* attribute in the *Tiny_Session* table can be potentially removed. Indeed, it is always possible to identify the Large Session a given Tiny Session belongs just by knowing its Host_Id and Source_Id. We show in the following that this redundancy is however beneficial for performance improvements. Moreover, it greatly simplifies the process of writing new queries.

Raw packets were not presented in the previous figure for clarity concerns, but they are definitely stored. They are parsed and stored in tables similar to those used by the Snort Intrusion Detection System [23]. Packets are either coming from a Host or from a Source. They are linked to a given Tiny Session. Figure 3 presents the resulting new tables. Each packet has its unique identifier, called *cid*.

The reader can easily deduce the global graph from the two figures 2 and 3. In the following, we will keep introducing new tables, in an incremental way. We will represent only the relevant information as the representation of the whole schema appears to be rather difficult. We invite the interested reader to contact us if he wants to obtain the whole diagram.

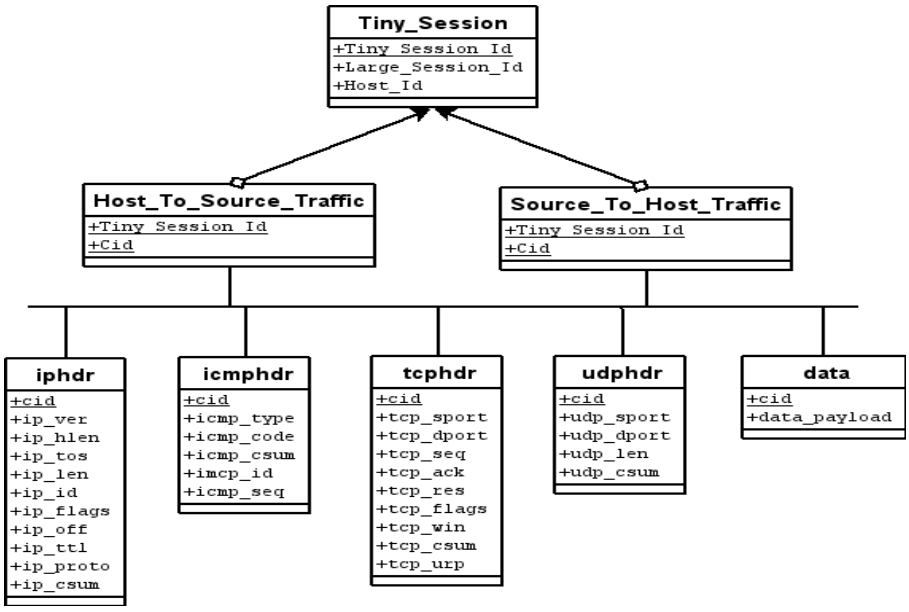

Figure 3: Raw packet Storage

3.4 Illustrative Examples

At the end of 2004, the database contains 17 Gbytes. The tables presented in Section 3.3 have been implemented in a MySQL [24] database running on a RedHat 9.0 server, with 2GHz, 80 GB ROM and 1GB RAM. The entries for the previously introduced tables are given in Table 3.

Table names	Number of entries
Source	511 415
Large_Session	525 152
Tiny_Session	933 536
Environment	25
Host	75

Table 3: Number of table entries in 11/04

From these tables, it is now very simple to answer questions like the four following ones:

1. How many *Sources* have been observed per *Environment*?
2. How many *Hosts* have been targeted by each *Source*?
3. How many *Sources* have been observed from multiple *Environment*s?
4. What is the percentage of IP addresses observed during more than one day?

The first question can be answered by means of the following SQL query:

- `SELECT Environment_Id, count(Source_Id) FROM Large_Session GROUP BY Environment_Id`

The output is a two-column table. It gives for each environment (left column) the corresponding number of observed Sources (right column). This is –partially- represented in the second and third column of Table 4 in which we see a large diversity in the number of hits against various Environments. Of course, this number is relative to the activity period of each platform as they did not start at the same time. A better output consists in dividing the total number of observed Sources by the

number of active days. This provides the average number of Sources observed each day on each platform. Even in this case, we observe on some platforms twenty times more Sources per day than on some others, in the average.

Platforms	Number of Observed Sources	Number of days the platform has been active	Average number of Source observed each day
Platform 1 (France, industry)	58791	70	840
Platform 2 (France, academy)	21781	121	180
Platform3 (Germany, academy)	109426	105	1042
Platform 4 (Lithuania, academy)	7841	156	50
Platform 5 (USA, industry)	22784	79	285

Table 4: Number of Observed Attack Sources per Platform

The clear difference between Large_Session entries and Tiny_Session entries in Table 3 indicates that many attacks target more than one virtual machine. This is verified when answering to our second question by means of the following SQL query:

- `SELECT Source_Id, count(Tiny_Session_Id) FROM Tiny_Session GROUP BY Source_Id`

The output is a two-column table which provides for each Source_Id (left column) the associated number of *Tiny_Sessions* (right column). As a reminder, all the exchanges of packets between one Source and one *Host* are part of a single *Tiny_Session*. We find out with this request that in average, 54% of the *Sources* have targeted the three virtual machines. A closer look also indicates that they always have targeted the three virtual machines in the same order, the sequential order. 40% of the observed *Sources* have targeted one and only one virtual machine. The remaining 6% *Sources* have targeted two out of the three honeypots.

The answer to the third question tells us if some IP addresses have been observed on multiple platforms the very same day. This is given by the following query:

- `SELECT Source_Id, count(Large_Session_Id) FROM Large_Session GROUP BY Source_Id`

This request reveals that 9995 out of the 511415 *Sources* (i.e. less than 2% of the *Sources*) have been observed on more than one platform the very same day, by definition of the notions of *Source* and *Large Session*.

The fourth question goes one step further than the previous one by looking at the percentage of IP addresses that have been observed on two different days. In other words, how many IP addresses are found under more than one *Source* identifier? This number can be found by dividing the result of this query:

- `SELECT count(distinct(IP_Address)) FROM Source`

By the result of this other one:

- `SELECT count(Source_Id) FROM Source`

The result is around 91%. This simply means that it is unlikely to observe the same attacking IP address twice on the same platform. The last two questions highlight the fact that, first, attacks are issued from a very large pool of IP addresses and, second, that it might not be worth the effort of implementing the notion of blacklists [25] since, apparently, a few of them are observed more than once.

4 ADDITIONAL INFORMATION

4.1 IP Geographical location

4.1.1 Various Information

The geographical location of IPs can represent interesting information to better understand where the attacks are coming from. We initially used one utility called Netgeo, developed in the context of the CAIDA project [26]. NetGeo is made of a collection of Perl scripts that map IP addresses to geographical locations. This software is open source and has been applied in several research papers, among which [27],[28],[29],[30]. However, as we show in [15], there are some differences with other tools like MaxMind, IP2location or GeoBytes [31],[32],[33]. The reason is that Netgeo returns in many cases the geographical location of the Autonomous System (AS) instead of the real location of the IP itself. Thus, we have decided to include into our database the geographical information provided by several tools and leave it up to the user to choose the one whom he felt more comfortable with. This offers us as well the opportunity to test and compare the results of these tools. We have subscribed to a commercial solution called MaxMind, and have made some comparisons with demo versions of commercial tools like IP2location and GeoBytes [32],[33].

4.1.2 Database Modifications

The geographical information has to be considered for one IP at a given point in time. Indeed, the location can change over months (see Maxmind updates presented in [31] for instance). For this reason, the geographical location is linked to the notion of *Source*. Two choices are possible here. The first solution consists in adding into the *Source* table one attribute for each utility which will be a pointer to its output. This method is not practical, as it requires modifying permanently the *Source* table for each tool application. On the other hand, the second solution considers that the geographical location characterizes a Source but also represents a new and important information type. Thus a new table called Geographical_Information is introduced. This enables us to leave the important *Source* table unchanged. Details are presented in Figure 4.

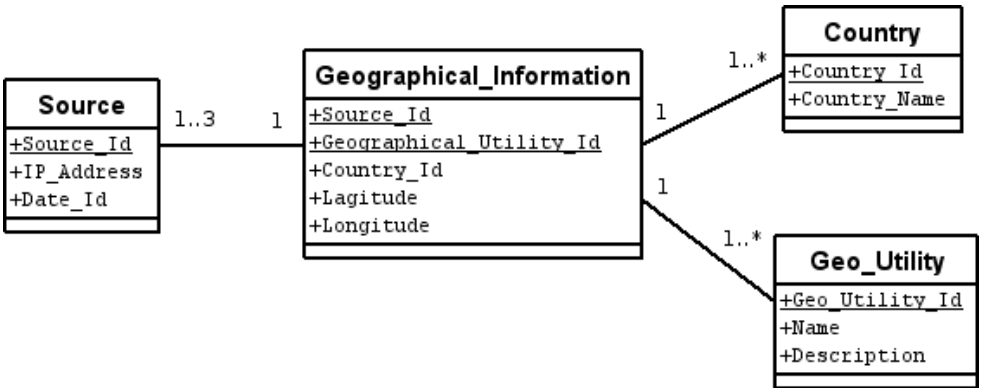

Figure 4: Geographical Information on the attacks

One *Geographical_Information* entry is defined by the primary key pair *(Source_Id, Geographical_Utility_Id)*. It returns the corresponding *Country_Id* provided by the *Geo_Utility* whose identifier is *Geo_Utility_Id*. The table *Country* maps the full country name associated to the Country_Id. Luckily enough, all tools are using the same Country_Id. It consists in 2 letters, as specified in the ISO 3166-1 alpha-2Code [34]. Thus, it is very easy to directly work on the *Country_Id* attribute, as shown with the examples given below.

4.1.3 Examples

A very simple SQL request enables us to get an idea of the amount of differences between the outputs provided by the various tools.

- `SELECT Source_Id, count(distinct(Country_Id)) FROM Geographical_Information GROUP BY Source_Id`

This gives a two-column output. If the values in the second column are strictly higher than one, it means that the same *Source* is seen as belonging to different countries by the various tools. As a matter of fact, we find out that Maxmind and Netgeo give a different result for 65% of the *Sources*. This result is obtained by looking at the number of cases where two different countries are assigned to the very same *Source*. The comparison with another tool called IP2location ([32]) confirms Maxmind results.
It is usually said that most of the attacks originate from China and the United States of America. This is confirmed by the following request which focuses on the results provided by Maxmind (identified by *Geographical_Utility_Id* = 2)

- `SELECT Country_Id, count(Source_Id) n FROM Geographical_Information WHERE Geographical_Utility_Id = 2 GROUP BY Country_Id ORDER BY n`

The `ORDER BY` command orders the results in increasing order. Thus, it is easy to pick up the ten most active countries, as presented in Table 5.

Top Ten attacking Countries for all Sources (given in decreasing number of importance)	Number of Observed Sources
US – The USA	77621
TW – Taiwan	48440
CN – China	40046
DE – Germany	34348
FR – France	26911
KR – Republic of Korea	24496
ES – Spain	22756
JP – Japan	20602
GB- The United Kingdom	19771
CA-Canada	18286

Table 5: Top 10 attacking countries for all Sources

More generally, 185 distinct countries have been observed since the beginning of this experiment. It is worth noting that there are 191 countries members of the United Nations and 192 countries are recognized by the United States State Department.

We also note that 10 countries are responsible for more than 66% of the attacks while 175 countries are for the remaining 34%. This analysis can also be performed per platform. This is done by introducing the notion of *Environment* into the previous request:

- `SELECT Country_Id, count(Source_Id) n FROM Geographical_Information, Large_Session WHERE Geographical_Utility_Id = 2 AND Large_Session.Source_Id = Geographical_Information.Source_Id AND Large_Session.Environment_Id = 21 GROUP BY Country_Id ORDER BY n`

We apply the same request than previously, but we only consider here Sources having targeted the Environment 21. An analysis of the evolution of the attacks per country has been performed in [20]. We refer the interested reader to this paper for more results on the analysis of the attacks based on geographical information.

4.2 Passive OS Fingerprinting

4.2.1 Various Utilities

Many worms propagate through Windows machines [9],[10],[35]. Others target specific Linux services [36],[37]. In all cases, it seems that there exists some correlation between the Operating System of the attacker and the attack type it is performing. Many techniques exist to

determine/fingerprint the operating system of machines, even if they are not all perfect. They are often classified as *active* when they send specific traffic to the machine to probe its OS. On the contrary, so-called *passive* fingerprinting methods rely on the observation of packets without interacting with the machine. This last approach is far more interesting in our case. Indeed, we want our honeypot machines to remain passive. Indeed, we do not want to show to the attacker that his activities are under observation.

A dozen of utilities implements passive fingerprinting techniques. Most of them compare packet fields to a given fingerprints database. A match means that the OS has been determined. Some tools differ by the answer provided in the case of uncertainty, others by their fingerprinting tables. As a consequence, we have decided to use three different passive fingerprinting utilities, with the possibility to add new ones later on if needed. They are respectively called Disco, p0f and ettercap [38],[39],[40]. They all use tcpdump pcap files as input. Their output is in text format which we parse to store the information into the database. Details of the tables are provided in the next Section.

4.2.2 Database Modifications

Supposedly, the OS fingerprinting qualifies a given attack *Source*. However, for practical reasons, we have decided to run the OS fingerprinting process on a per platform basis. As a consequence, we consider that the OS fingerprinting information is related to a given *Large_Session* and not to a given *Source.* Details on the database architecture are presented in Figure 5.

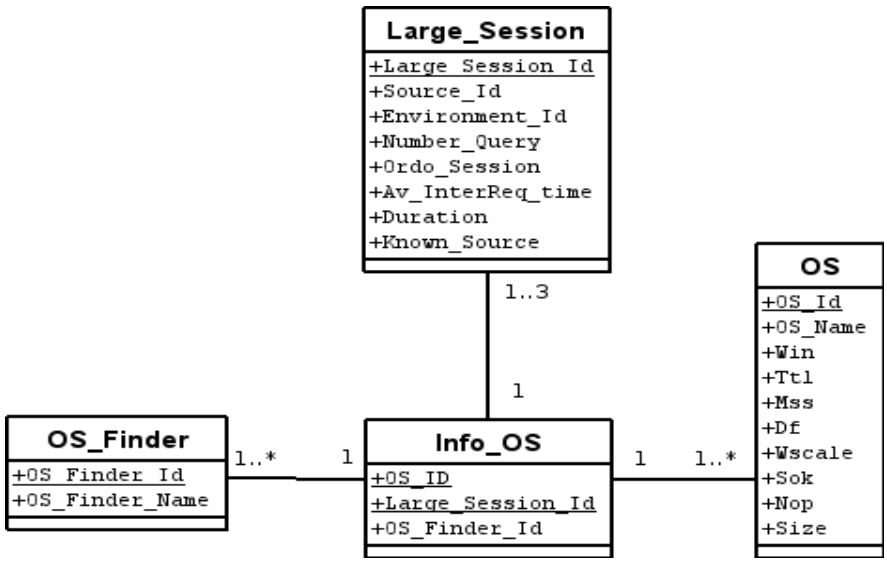

Figure 5: Passive OS Fingerprinting Information

There are two options here, as with the geographical location information. The first solution consists in introducing a new attribute in the *Large_Session* table for each fingerprinting tool. The second solution consists in having the OS information in a new table called *Info_OS*. Using a similar reasoning as before, we decide to use the second option. Each entry is totally determined by the triple key *(Large_Session_Id, OS_Finder_Id, OS_Id)*. The *OS* table contains all fingerprints, even those which are not currently determined.

4.2.3 Some Applications

It appears that a comparison between fingerprinting tools is not straightforward. Indeed, the tools rely on different fingerprints database. As a consequence, two same signatures might not be associated exactly to the same OS(s). For instance, the very same signature will be detected as "*Windows 2000 SP2+, XP SP1 (seldom 98 4.10.2222)*" by p0f and as "*Windows XP Pro,S*" by Disco. A naive comparison will claim that these are two different operating systems. On the other hand, they both point out that the attack *Source* runs on a Windows machine. This can be an interesting hint to better characterize the *Source* profile.

To avoid such a problem, we make use of regular expressions. They can also be easily

implemented with MySQL. Thus, if we are interested in looking at the number of attacking *Sources* considered to be Windows machines by p0f (OS_Finder=2), we write the following request:

- `SELECT count(distinct(Source_Id)) FROM Large_Session, Info_OS, OS WHERE Large_Session.Large_Session_Id = Info_OS.Large_Session_Id AND Info_OS.OS_Id = OS.OS_Id AND OS.Name REGEXP "Windows"`

We count all the distinct sources to avoid counting multiple times all sources having targeted multiples platforms. By repeating this query with various parameters, we obtain the comparison proposed in Table 6:

Passive Fingerprinting Tools	Detected "Windows" Machines	Detected "linux/unix/solaris" machines	Number of returned undetermined values
Disco	66%	0.3%	33.7%
P0f	81.8%	0.5%	17.7%
Ettercap	77,6%	0.4%	22%

Table 6: Comparison between three Passive OS Fingerprinting Tools

All tools agree that the majority of attacks are coming from Windows machines. P0f and Ettercap give very high percentage values for these machines. We also notice that Disco has a high rate of undetermined fingerprints. This is the less elaborated tool. It bases its fingerprints on TCP SYN and TCP SYN/ACK packets only. If we now compare how many Sources differ between Disco and p0f, we find out that Disco does not return any result when there is an ambiguity. Ettercap and p0f, however, tend to assign a default Windows value in many cases. This is all the more confirmed that we have learnt recently that Ettercap passive fingerprinting functionality is apparently based on the p0f_v1 (p0f first version) code. As a consequence, we have decided to use p0f as our default OS fingerprinting tool.

4.3 Domain Name Resolution

4.3.1 DNS and other network features

It is interesting to analyze the geographical location of the attack sources, as we have shown in Section 4.1. Another potentially interesting information can be the machine name resolution. The Domain Name Resolution (DNS) associates one name to one IP. It is based on a distributed database containing all name/address pairs. This database is organized in domains that form a hierarchy. The nslookup tool enables us to query a NS to find (among other information) the IP address corresponding to a host name. It is also possible to obtain the name corresponding to an IP address. The reason is that NS manages two databases: the direct zone and the inverse zone. We have implemented a simple Perl script based on the Perl Net-DNS library that recursively performs the reverse DNS request.

The network from which the attack occurs can also be of interest. This information can be retrieved with *Whois* queries. A Whois query against a registry's database identifies the registrar and the name servers for the domain name given in the query. A query against a registrar's database identifies the owner of the domain name, and the contacts associated with it. Some simple Perl scripts exist to directly perform WHOIS lookups automatically [41]. These lookups provide various information on the domain, as its registration ID, its creation date, its expiration date, etc.

Finally, each IP belongs to a given network. A network can be defined as an IP address with a Classless Inter-Domain Routing (CIDR) mask. The method is precisely described in RFC 1520 ([42]). The CIDR notation specifies an IP address range by the combination of an IP address and its associated network mask. CIDR notation uses the following format xxx.xxx.xxx.xxx/n where n is the number of (leftmost) bits in the mask. For example, 192.168.12.0/23 applies the network mask 255.255.254.0 to the 192.168 network, starting at 192.168.12.0. This notation represents the address range 192.168.12.0 - 192.168.13.255. We have decided for each *Source* to store its corresponding CIDR mask in the database. Both Netgeo and Maxmind provide information on the network the *Source* is coming from. We derive from the IP-range network values the CIDR network mask.

To summarize, we thus introduce into the database three new external important information types: the domain reverse resolution, the result of *Whois* queries and the network parameters. As a follow up, since these information characterize uniquely a given *Source*, we have decided to enrich the corresponding table with this information by adding attributes, the value of which point to new tables providing the imported information. This is represented in Figure 6. For instance, *Network_Id* is a pointer to an entry in the table *Network* where a network address can be found together with an estimated CIDR value.

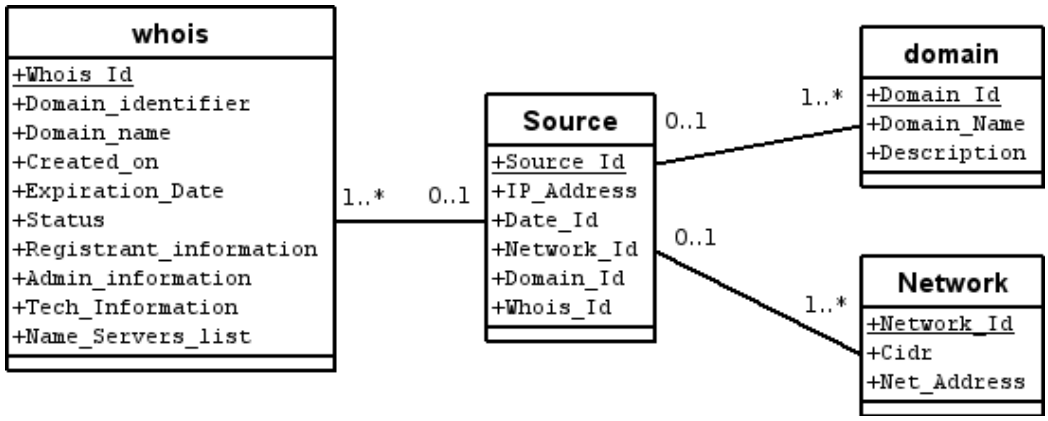

Figure 6: Network and Domain information

4.3.2 Examples

Some interesting statistics can be done on the domain names. We have developed some parsers to extract the different levels of the name. The top 5 most attacking domains are respectively the .net, .com, .jp, .fr and .de first name levels. Other statistics can be done quite easily by checking if the attacking machine belongs to a company network or is more likely to be a home computer. For instance, simple extractions of patterns like '%dsl%', '%cable%', '%dial%' are a good indicators of home computers. On the other hand, patterns like '%web%', '%mail%', '%serv%' in the machine name are likely to show up for machines belonging to some industrial professional environment. Several other analyses are possible and this is an ongoing task in our project.

5 META DATA

5.1 *Meta-Data* definition

We have explained here above the existence of redundant information in our database for the sake of querying efficiency. In the following, we call "meta-data" the content of these redundant tables. Meta data information is information that is saved in specific tables but that could also be derived by means of queries against the other tables. *Meta-data* tables are built by means of automated scripts that are run whenever new data are entered into the database. We provide in the following a non-exhaustive list of *meta-data* information that can be found today in the database:

- The duration of the observation of a given *Source*
- The average inter arrival time of requests sent to a given *Source*. This might help differentiate hand-written attacks from automated tools.
- An indicator on who starts the attack session (to efficiently verify that none of our honeypots has ever been seen initiating a connection outside).
- A simple attribute that indicates how many virtual machines are targeted
- The number of observed complete TCP connections in a given *Tiny Session*
- Some precisions on the observation date: working days, working hours, etc..;
- The sequence of ports that have been targeted during the attack on a virtual machine
- A Boolean value indicating if a given Source has already been observed or not
- An attribute to mark *Tiny sessions* that are likely due to backscatter activities
- Another Boolean value to indicate if attacks on multiple virtual machines were

performed in sequence or simultaneously.

- Etc.

The main idea is that we do not want to compute this *meta-data* information whenever we need it. It is considered to be useful enough to be part of the database information. The attentive reader will notice that, actually, the core notions of *Source*, *Tiny_Session* and *Large_Session* are already *meta-data.*They are attack representations we observe from our platforms based on the raw packets initially stored.

In conclusion to this Section, there are a multitude of *meta-data* we have found interesting to represent and analyze. Others will definitely appear along with our experiments. We explain in the following how they can be easily integrated into the database architecture.

5.2 Flexible Database Interface

As explained in Section 5.1, the *meta-data* list we present in this document is far from being exhaustive. There is a very easy way to figure out where to integrate new *meta-data* into the current architecture. It boils down to answering the following question:

☑ *What characterizes this new piece of Information?*

- The Source only?
- The attack session between one Source and one Environment, i.e. a *Large_Session*?
- The attack session between one Source and one virtual machine, i.e.; a *Tiny_Session*?

The following steps are simple updates of the appropriate tables. Once tables are created or modified, updating the whole database to reflect these changes is a matter of minutes, thanks to a couple of scripts. It is important to note that the updates concern only the addition of new data, thanks to new data structures. It does not imply changing some existing values in certain tables. If that was the case, adding new meta data could lead to update anomaly problems.

The *meta-data* described in Section 5.1 have all been treated this way. The resulting tables are presented in Figure 7.

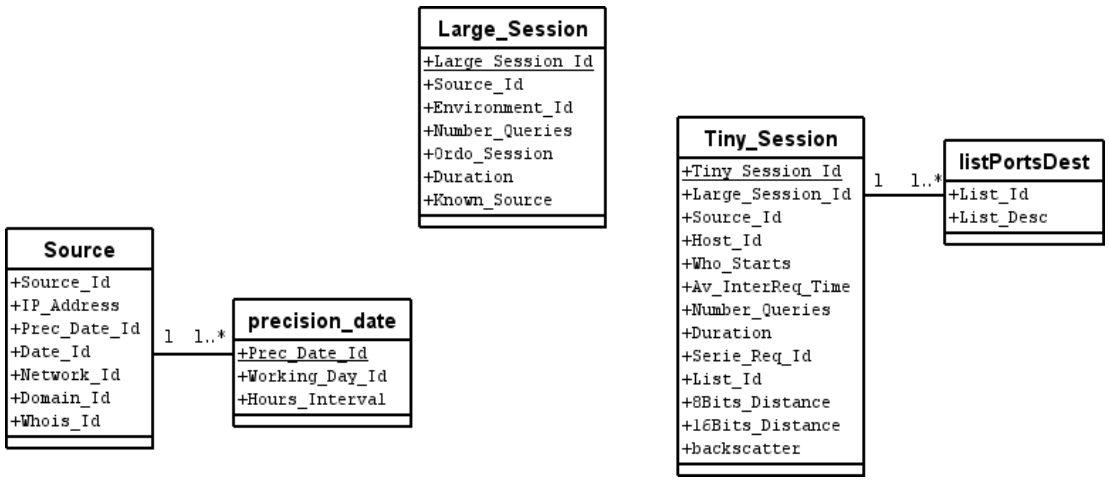

Figure 7 : Meta Data Insertion

For the *Source* table, one major attribute has been added.

- The *Precision_Date* attribute gives more detailed information about the date of the observation of an attack. This information can be used to determine if the attack occurred during working days (Monday-Friday), or working hours (9h-17h). The *Working_Day_Id* is a Boolean value. The *Hours_Interval* attribute indicates the hour at which the Source has first been observed. This way, it is now quite easy to answer questions like: Are we less attacked during working days? Are attacks more virulent during night time? Some elements of responses are presented in Figure 8 for the VMWare platform which has been up and running for almost 2 years. All *Sources* that have been first observed between 00:00 and 01:00 are presented by the '0' bar. The 24 bars stem for the 24 hour intervals. This curve has a very nice shape, interesting to look at. We do not observe the same amount of attacks every hour. The

curve clearly indicates that the attack time does not correspond to a random pattern. In addition, there is a weird peak of activities between 3pm. and 4pm.Thanks to the geographical location tables, we could identify if this shape is due to a particular country attacking at that specific time. We can also imagine many other possible investigations which lie beyond the scope of this paper.

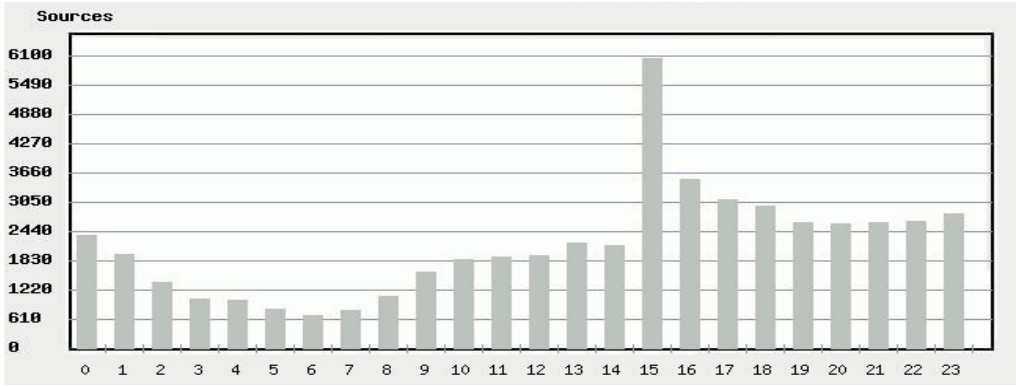

Figure 8: Attacks per Hour on our VMWare platform

For the *Large_Session* table, a few attributes have also been added:

- *Number_Queries*: this attribute gives for each *Large_Session* the total number of packets that have been sent by the *Source* to the whole honeypot platform, i.e. *Environment*. There is a similar attribute assigned for the *Tiny_Session* table. It corresponds to the total number of packets sent by the same *Source* to one virtual machine, i.e. *Host*.
- *Duration*: This attribute corresponds in table *Large_Session* to the time period from the date the *Source* has sent its first packet to the date it has sent its last packet. In a similar way, there is a *Duration* attribute in the *Tiny_Session* table for packets sent to one *Host* only.
- *Ordo_Session*: This attribute indicates if *Hosts* have been attacked sequentially or in parallel. To get this information, it suffices to look at packet timestamps. If the first packet sent to the second virtual machine is posterior to the last one sent to the first virtual machine, they have been attacked sequentially. In other cases, they have been attacked in parallel.
- *Av_InterReq_Time*: This attribute provides the average time that occurs between two packets sent by the same *Source*. This attribute has been introduced with the goal of chasing traces due to non automated attacks. Our assumption is that attacks launched manually from a console by attackers will be characterized by a large inter arrival time of packets. This is true, for instance, when a user replies in an interactive way to an ftp server asking him his username and password. So far, we only have observed a very few number of attacks like this within the 2 year experiments.

For the *Tiny_Session* table, three main attributes have been added:

- *Who_Starts*: This attribute is an indicator to check who initiated the connection between the *Source* and the *Host*. By design, the *Host* should never initiate a connection to a *Source* unless if it has been compromised. We use this information as another method to verify that none of our hosts gets hacked.
- We make use of the definition *Ports Sequence* as an ordered list of ports targeted by an attack *Source* on a given *Host*. 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}. The sequence of ports has been put in the *ListPortsDest* table. For all the *Tiny_Sessions* we have observed so far, the length of theses sequences of ports is usually very short. In this two-year experiment, we have observed only 9 scans on more than 500 ports. Thus, it seems more reasonable to store them once as strings and add in parallel a dedicated index in the *Tiny_Session* table.
- The *Backscatter* identification is quite straightforward. Indeed, backscatter packets are

responses to connections requests issued by spoofed IP addresses, typically in the case of a Denial of Service attack against a third party. If our addresses are used (spoofed) in the course of this attack, we will see the responses of the victim sent to us without us having talked to him first. These attacks have been very well-analyzed by Moore et al. in [43],[44]. Figure 9 summarizes the various types of responses (column 'response from victim') that can be sent "against" our honeypots. These packets hit a large variety of ports that are traditionally unused, such as 27374 (TCP RST), 11224 (TCP SYN ACK), 9026 (RST ACK), etc...

Packet sent	Response from victim
TCP SYN (to open port)	TCP SYN/ACK
TCP SYN (to closed port)	TCP RST (ACK)
TCP ACK	TCP RST (ACK)
TCP DATA	TCP RST (ACK)
TCP RST	no response
TCP NULL	TCP RST (ACK)
ICMP ECHO Request	ICMP Echo Reply
ICMP TS Request	ICMP TS Reply
UDP pkt (to open port)	protocol dependent
UDP pkt (to closed port)	ICMP Port Unreach
...	...

Figure 9: Backscatter Packet characteristics

5.3 Some Interesting *Meta-Data* Outputs

Insofar, the size of the database as described in the previous Sections is 27 Gbytes. Everyday, we insert around 13Mbytes of raw data. If we consider the global additional increase every day with all meta-data, this reaches 40 Mbytes per day. More specifically, the *Source* table is 36 Mbytes, the *Tiny_Session* is 97Mbytes, the *Large_Session* is 48 Mbytes, the *Source_To_Host_Traffic* table is 586 Mbytes, the *Host_To_Source_Traffic* table is 440 Mbytes, the *Geographical_Location* is 30 MBytes and finally the *Info_OS* table is 100 MBytes. These figures evolve every day but clearly indicate which tables are the largest.

Typically, it takes about 5 minutes to store the daily tcpdump file of one *Environment* This means that we need less than two hours to store every day all data from all platforms. This is perfectly acceptable. To conclude, we can easily show the gain in terms of speed obtained thanks to the meta data. For this example, we do imagine that the meta data *listPortsDest* does not exist in the database and we want to know all ports sequences observed against all platforms. This comes down to running the following SQL query:

- `SELECT group_concat(distinct(tcp_dport),distinct(udp_port)) FROM Tiny_Session, Source_To_Host_Traffic, tcphdr, udphdr WHERE Tiny_Session.Tiny_Session_Id = Source_To_Host_Traffic.Tiny_Session_Id AND Source_To_Host_Traffic.Cid = tcphdr.cid AND Source_To_Host_Traffic.Cid = udphdr.cid GROUP BY Tiny_Session.Tiny_Session_Id`

The answer is provided after 7 hours 48 minutes 21seconds during which almost 100 % of the CPU of the machine was devoted to this single request. The major reason of such an important delay is that the *tcphdr* table contains more than 17'700'000 entries. On the other hand, thanks to the integrated meta data, we can submit this Mysql query:

- `SELECT list_desc FROM Tiny_Session, listPortsDest where Tiny_Session.List_Id=listPortsDest.list_id`

Now, the time required to get an answer is reduced to 34seconds. This means that we gain almost two orders of magnitude in speed! In addition, the first SQL query does not give the exact sequence of ports with respect to the dates they were probed. It only gives the list of ports (TCP and UDP) that have been targeted. Asking for the *sequence* information would have made the request even more complex with an execution delay exceeding one day.

6 CONCLUSION

In this paper, we have presented in detail the design of the centralized database used in the context of the Leurré.com project. We have shown, step by step, the various tables that compose the databases, the reasoning beyond their creation as well as their usefulness to extract meaningful information easily from the database. We have explained why several tables have been created that do contain redundant information and we have motivated our choice for efficiency in the querying mechanism, at the cost of a greater storage need. Several examples have been given throughout the text to illustrate the various notions.

As shown in this paper, the richness of the database and the flexibility of its design are such that it enables a large diversity of analyses to be carried out on it. It is not the purpose of this paper to report on a specific analysis. Other publications have focused on some of these issues and some more work is ongoing. We have shown though by means of examples that this database helps in discovering trends in the attacks and in characterizing them. Being able to conduct such analysis in a systematic way is a prerequisite for establishing early warning information systems and, therefore, we believe our work constitutes a foundational element towards the creation of such centers.

It is our wish to share the data contained in this database with those interested in carrying some research on it. The authors can be reached by mail to get detailed information regarding how to join the project in order to gain access to the database.

7 REFERENCES

[1] WORM 2004, The 2nd Workshop on Rapid Malcode, held in Association with the 11th ACM Conference on Computer and Communication Security CCS, Oct. 2004, VA, USA. Home page at: http://www.acm.org/sigs/sigsac/ccs/CCS2004/worm.html

[2] DIMVA 2004, The Detection of Intrusions and Malware & Vulnerability Assessment, July 2004, Dortmund, Germany. Home page at: http://www.dimva.org/dimva2005

[3] SRUTI: Steps to Reducing Unwanted Traffic on the Internet, Usenix Workshop, July 2005, MA USA. Home page at: http://nms.lcs.mit.edu/~dina/SRUTI/

[4] S. Staniford, V. Paxson, N. Weaver. "How to Own the Internet in Your Spare Time". *In the Proceedings of the 11th USENIX Security Symposium*, pages 149-167. USENIX Association, 2002.

[5] Z. Chen, L. Gao, K. Kwiat. "Modeling the Spread of Active Worms". *In the Proceedings of the IEEE INFOCOM 2003*, April 2003, CA, USA.

[6] C.C. Zou, W. Gong, D. Towsley. "Worm Propagation Modeling and Analysis Under Dynamic Quarantine Defense". *In Proceedings of the 1st Workshop on Rapid Malcode (WORM'03)*, Oct. 2003, WA, USA.

[7] E. Spafford. "The Internet Worm Program: An Analysis". *Purdue Technical Report CSD-TR-823*, West Lafayette, IN 47907-2004, 1988.

[8] D. Moore, C. Shannon, G.M. Voelker, S. Savage. "Code Red, a Case Study on the Spread and Victims of an Internet Worm". *In Proceedings of the ACM/USENIX Internet Measurement Workhop*, Nov. 2002.

[9] McAFEE Security Antivirus. "Virus Profile: W32/deloder worm". Available at: http://us.mcafee.com/virusInfo/

[10] F-Secure Corporation. "Deloder worm analysis". Available at: http://www.f-secure.com

[11] J. McHugh. "Sets, Bags, and Rock and Roll Analyzing Large Data Sets of Network Data". *In Proceedings of the 9th European Symposium on Research in Computer Security USENIX'04*, Sept. 2004, Sophia-Antipolis, France.

[12] Honeyd Virtual Honeypot from N. Provos, home page: http://www.honeyd.org

[13] M. Dacier, F. Pouget, H. Debar. "Honeypots, A Practical Mean to Validate Malicious Fault Assumptions". *In Proceedings of the 10th Pacific Ream Dependable Computing Conference (PRDC'04)*, Feb. 2004.

[14] F. Pouget, M. Dacier. "Honeypot-based Forensics". *In Proceedings of the AusCERT Asia Pacific Information Technology Security Conference 2004 (AusCERT2004)*, May 2004, Australia.

[15] F. Pouget, M. Dacier, V.H. Pham. "Understanding Threats: a Prerequisite to Enhance Survivability of Computing Systems". *In Proceedings of the International Infrastructure Survivability Workshop (IISW 2004)*, Dec. 2004, Portugal.

[16] F. Pouget, M. Dacier, H. Debar. "Attack Processes found on the Internet". *In Proceedings of the NATO Symposium IST-041/RSY-013*, April 2004, France.

[17] VMWare Corporation. User's Manual version 4.1 available at: http://www.vmware.com

[18] TCPDUMP utility home page: http://www.tcpdump.org

[19] LEURRE.COM, the Eurecom Honeypot Project home page: http://www.eurecom.fr/~pouget/leurrecom.html

[20] F. Pouget, M. Dacier, V.H. Pham. "On the Advantages of Deploying a Large Scale Distributed Honeypot Platform". *To appear in the Proceedings ot the E-Crime and Computer Evidence Conference 2005*, Monaco, Feb. 2005.

[21] H. Garcia-Molina, J.D. Ullman, J.D. Widom. "Database Systems: he Complete Book". 2002.

[22] J.D. Ullman. "A First Course in Database Systems", 2nd Edition, 1989.

[23] SNORT Intrusion Detection Sytem home page: http://www.snort.org

[24] MySQL Open Source Database home page: http://www.mysql.com

[25] "Blacklist Scanner" in Security Focus Home Tools: http://www.securityfocu.com/tools/1962
[26] CAIDA Project. "Netgeo utility – the Internet geographical database" home page: http://www.caida.org/tools/utilities/netgeo/
[27] A. Rosin. "Measuring Availability in Peer-to-Peer Networks". Sept. 2003. Available at: http://www.wiwi.hu-berlin.de/fis/p2pe/paper_A_Rosin.pdf
[28] A. Zeitoun, C.N. Chuah, S. Bhattacharyya, C. Diot. "An AS-level study of Internet path delay characteristics". Technical report, 2003. Available at: http://ipmon.sprint.com/pubs_trs/trs/RR03-ATL-051699-AS-delay.pdf
[29] S.H. Hook, H. Jeong, A.L. Barabasi. "Modeling the Internet's large scale topology". *In PNAS –vol. 99*, Oct. 2002. Available at: http://www.nd.edu/networks/PDF/Modeling
[30] T.S. Eugene Ng, H. Zhang. "Predicting Internet Network Distance with Coordinates-based Approaches". *In Proceedings of INFOCOM 2002*. Available at: http://www-2.cs.cmu.edu/eugeneng/papers/INFOCOM02.pdf
[31] MaxMind GeoIP Country Database Commercial Product, home page: http://www.maxmind.com/app/products
[32] IP2location products, home page: http://www.ip2location.com
[33] GeoBytes IP Address Locator Tool, home page: http://www.geobytes.com/IPLocator.htm
[34] ISO 3166-1 alpha-2, Introduction to the 2-letter code for countries names. Available at: http://encyclopedia.thefreedictionary.com/ISO%203166-1
[35] Symantec Antivirus Corporation. Symantec Security Response w32.welchia.worm, 2004, available at http://response.symantec.com/avcentr/venc/data/w32.welchia.b.worm.html
[36] "Internet Worm squirms into Linux Servers". CNET Tech report available at: http://news.com.com/2100-1001-251071.html?legacy=cnet
[37] "Ramen Linux Worm seen in Wild". InfoWorld News available at: http://www.infoworld.com/articles/hn/xml/01/01/25/010125hnramen.html?p=br&s=3
[38] Disco Passive Fingerprinting Tool home page: http://www.altmode.com/disco
[39] P0f Passive Fingerprinting Tool, version 2.0 home page: http://lcamtuf.coredump.cx/p0f-beta.tgz
[40] Ettercap NG-0.7.1 Sourceforge Project available at: http://ettercap.sourceforge.net
[41] Comprehensive Perl Archive Network CPAN home page: http://www.cpan.org
[42] "Exchanging Routing Information Across Provider Boundaries in the CIDR Environment". IETF RFC 1520 available at: http://www.ietf.org/rfc/rfc1520.txt
[43] CAIDA Project: The UCSD Network Telescope. http://www.caida.org/outreach/papers/2001/BackScatter/
[44] D. Moore, G. Voelker, S. Savage. "Infering Internet Denial-of-Service activity". *In Proceedings of the 2001 USENIX Security Symposium*, Aug. 2001, CA, USA.