SEVENTH FRAMEWORK PROGRAMME

Theme ICT-1-1.4 (Secure, dependable and trusted infrastructures)

WORLDWIDE OBSERVATORY OF
MALICIOUS BEHAVIORS AND ATTACK THREATS

D15 (D4.5) Intermediate Report on Contextual Features

Contract No. FP7-ICT-216026-WOMBAT

Workpackage	WP4 - Data Enrichment and Characterization
Author	Engin Kirda
Version	1.0
Date of delivery	M24
Actual Date of Delivery	M25
Dissemination level	Public
Responsible	Eurecom

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n°216026.

SEVENTH FRAMEWORK PROGRAMME
Theme ICT-1-1.4 (Secure, dependable and trusted infrastructures)

The WOMBAT Consortium consists of:

France Telecom	Project coordinator	France
Institut Eurecom		France
Technical University Vienna		Austria
Politecnico di Milano		Italy
Vrije Universiteit Amsterdam		The Netherlands
Foundation for Research and Technology		Greece
Hispasec		Spain
Research and Academic Computer Network		Poland
Symantec Ltd.		Ireland
Institute for Infocomm Research		Singapore

Contact information:
Dr. Marc Dacier
2229 Route des Cretes
06560 Sophia Antipolis
France

e-mail: Marc_Dacier@symantec.com
Phone: +33 4 93 00 82 17

Contents

Abstract

The objective of this Workpackage 4 is to develop techniques to characterize the malicious code that is collected in the previous workpackage. The main idea is to enrich the collected code thanks to metadata that might reveal insights into the origin of the code and the intentions of those that created, released or used it. This deliverable provides a preliminary discussion of possible contextual features of malware, and for each feature, an estimate on its effectiveness and the difficulty to obtain it. Some of these features can be used to analyze potential threats and discriminate collected samples that are mere variations of already known threats.

1 Introduction

The Internet has become an essential part of the daily lives of many, as more and more people are making use of services that are offered on the Internet. It evolved from a basic communication network to an interconnected set of information sources enabling, among other things, new forms of (social) interactions and market places for the sale of products and services. Online banking or advertising are mere examples of the commercial aspects of the Internet.

Just as in the physical world, there are people on the Internet with malevolent intents that strive to enrich themselves by taking advantage of legitimate users whenever money is involved. Malware (i.e., software of malicious intent) helps these people accomplish their goals.

To protect legitimate users from these threats, security vendors offer tools that aim to identify malicious software components. Typically, these tools apply some sort of signature matching to identify known threats. This technique requires the vendor to provide a database of signatures. Then, these manually created signatures are compared against potential threats. Once the security vendor obtains a sample of a new potential threat to study, the first step for a human analyst is to determine whether this (so far unknown) sample poses a threat to users by *analyzing the sample*.

If so, the analyst attempts to find a pattern that allows to identify this sample (i.e., the signature). This pattern should be generic enough such that it allows matching with variants of the same threat, but not falsely match on legitimate content. The analysis of malware and the successive construction of signatures by human analysts is time consuming and error prone. At the same time, it is trivial for malware authors to automatically generate a multitude of different malicious samples derived from a single malware instance. An antivirus vendor that receives thousands of unknown samples per day is not extraordinary nowadays. This substantial quantity requires an *automated* approach to quickly differentiate between samples that deserve closer (manual) analysis, and those that are a variation of already known threats. This automatic analysis can be performed in two ways. *Dynamic* analysis refers to techniques that execute a sample and verify the actions this sample performs in practice, while *static* analysis performs its task without actually executing the sample.

This deliverable provides a preliminary discussion of possible contextual features of malware and malware-based threats. In this preliminary report that aims to serve as a starting point, we focus and discuss the contextual features that are found in the datasets of the WOMBAT partners. Some of these features can be used to analyze potential threats and discriminate collected samples that are mere variations of already known threats.

In the following sections, we give a brief description of each specific dataset that a WOMBAT partner maintains, provide a list of contextual features that are stored in the dataset, describe each feature, and give some examples of the data that we have collected.

2 Dataset: SGNET (Eurecom)

As described in D3.2, SGNET is a distributed honeypot deployment that takes advantage of protocol learning techniques to achieve at low cost a level of interaction that is sufficient to fully emulate code injection attacks and collect malware samples. SGNET is able to generate in depth information on the characteristics of the code injection attacks that led to the download of a given malware sample, and adds on top of this additional contextual information on the observed event. The SGNET dataset is therefore able to provide a variety of contextual features on all the malware samples that were successfully downloaded during its operation.

2.1 Propagation dynamics

Honeypot deployments such as SGNET offer valuable information on the temporal evolution of malware propagation. By looking at the localization of the infection sources and of their target, and at their evolution over time, we can get insights on the malware propagation strategy.

For instance, Figure 2.1 shows the observed behavior for different malware samples known to be variant of the same bot family. The different variants are plotted along the x axis, while the y axis represents, from top to bottom:

- The location over the IP space of the hosts seen by SGNET as pushing such malware sample
- The number of weeks in which such malware sample was seen as active by SGNET
- The period of activity of each sample, plotted over a 13 months timeline

The temporal evolution exposes the expected bot-like coordinated behavior. For instance, the variant with longest activity evolves over time in the following way:

- 15/7 - 16/7: observed hitting network location A
- 18/7: observed hitting network location B

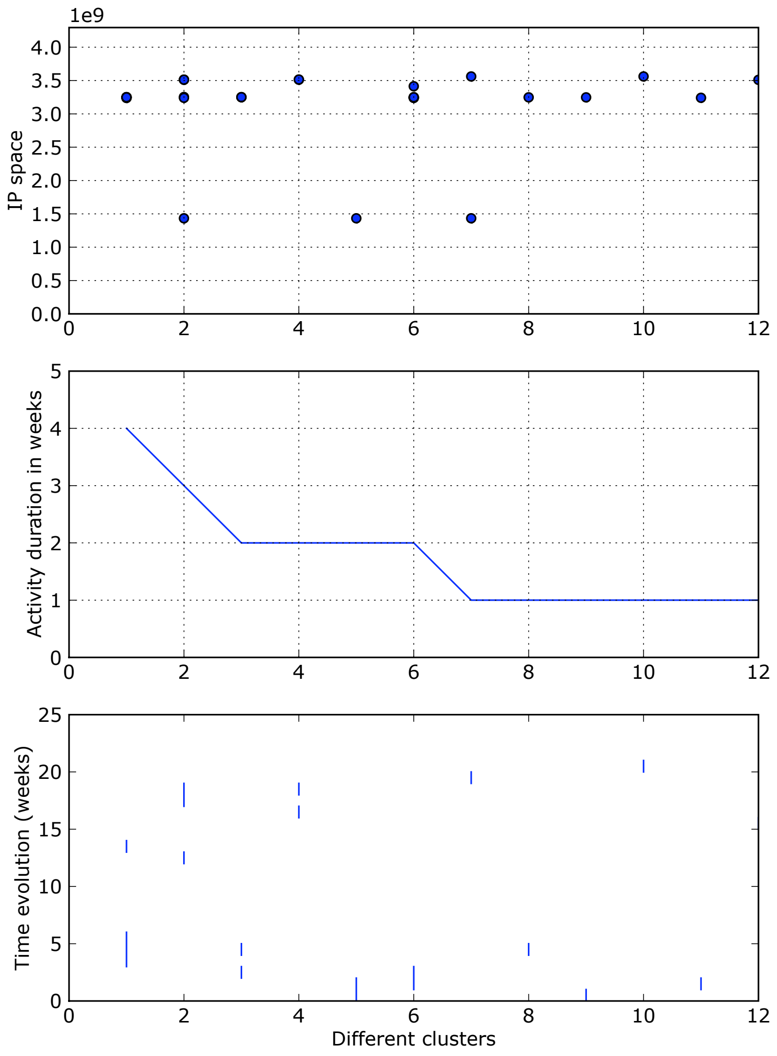

Figure 2.1: Propagation dynamics for different bot variants observed by SGNET honeypots

- 26/7: observed hitting network location B
- 2/8-4/8: observed hitting network location A
- 27/9: observed hitting network location B

Propagation dynamics is a potentially very interesting source of information on the malware evolution, but is also highly dependent on SGNET's coverage of the IP space. Such challenge is actually clear from the previous example. SGNET honeypots observed the propagation of the different malware variants for relatively small timespans, after which the associated botnet was probably pointed towards an area of the IP space not covered by any SGNET honeypot. It is therefore very difficult to make any exact estimation of the botnet lifespan and temporal evolution using solely honeypot data.

2.2 Geolocation

Every IP address observed in SGNET is mapped to its geographical location taking advantage of the Maxmind IP geolocation database [**?**]. Information on the geographical location of the sources involved in the propagation of a given malware variant can give interesting insights on the nature of that malware sample, and on the modus operandi of the attackers.

There are different reasons for which the spreading of malware samples can be affected by geographical boundaries.

- The infection localization can be a result of the malware propagation strategy. For instance, malware samples coordinated by means of a C&C channel might propagate only towards specific subnets specified by the bot herder. In these cases, the localization of the infected hosts might provide information on the modus operandi of the botnet creator.
- Malware might be specifically designed to specifically target, or avoid, a specific population. For instance, malware such as Conficker.A is designed to suicide whenever the keyboard language is detected as ukrainian[1].
- Malware propagation might be biased by the characteristics of the exploit and the associated vulnerability. An exploit might be successful only under certain localizations of the vulnerable application, preventing infection of devices located outside a specific geographical area.

[1] `http://mtc.sri.com/Conficker/`

2.3 Additional address information

Additional information can be extracted on the IP addresses of the attackers involved in the propagation of a given malware family.

Reverse DNS resolution can be used to infer information on the nature of the host. For instance, while testing the SGNET deployment we detected a considerable amount of code injection attacks generated by a single IP address that repeatedly pushed the same malware type. When looking at the reverse DNS resolution, we have been able to see that the specific IP address corresponded to the gateway of a university campus, probably hit by a worm infection and masquerading an large population of infected hosts. The identification of the nature of an IP address starting from its reverse name resolution is however mainly a manual process, and is very difficult to automate.

Similarly, information generated by the Emerging Threats project and by the Spamhaus block lists are helpful in finding intersections between the observed threats and other malicious activities (e.g. spam generation) identified by third parties.

2.4 Passive OS fingerprinting information

While propagating to SGNET honeypots, the compromised sources provide in their network conversation useful hints that can be used to identify the TCP/IP stack responsible the interaction, and therefore get information on the associated operating system. P0f [2] is a passive OS fingerprinting tool able to identify the operating system by looking at specific packets of the network conversation. By applying this tool to the network logs generated by SGNET, we can infer information on the OS version of the sources, and therefore on the configurations that are vulnerable to the propagation of a specific malware type.

Passive OS fingerprinting is based on databases or features, that are often maintained on a voluntary basis and are often imprecise. Information that can be obtained by tools such as p0f is therefore often rather generic, but offer a high level overview of the profile of a population of attackers. For instance, one of the bot variants considered in Figure 2.1 is associated to the following configuration profile:

```
{'cpe://microsoft:windows:2000': 2,
 'cpe://microsoft:windows:2000:sp2': 9,
 'cpe://microsoft:windows:2000:sp3': 31,
 'cpe://microsoft:windows:2000:sp4': 24,
```

[2]http://lcamtuf.coredump.cx/p0f.shtml

```
'cpe://microsoft:windows:xp': 2,
'cpe://microsoft:windows:xp:pro:sp1': 22,
'cpe://microsoft:windows:xp:sp1': 24,
'unknown': 33}
```

2.5 Code injection characteristics

As described in D3.2, the different components of the SGNET deployment generate information on the propagation strategy of the malware samples. Information on the propagation strategy used by different malware samples can provide interesting insights on the modus operandi of malware writers.

The protocol learning techniques employed by SGNET model the network conversation preceding the code injection attacks under the form of Finite State Machines. These Finite State Machine are traversed during honeypot operation in order to carry on the conversation with the attackers, and the path followed by the attacker during the traversal can be used to classify the interaction type. By construction, two different instances of the same exploit implementation will in fact follow the same path in the FSM traversal.

Looking at the SGNET dataset we can see that a total of 17651 code injection attacks, that led to the download of 11281 different malware binaries, are the result of the traversal of only 51 different FSM paths. From these numbers we can have a first high level idea on the amount of code sharing in the malware writing community.

Once an exploit has been successfully emulated, SGNET is able to extract the shellcode and take advantage of a set of heuristics to understand and emulate its behavior, ultimately downloading the malware sample. During this process, a variety of information is logged and stored in the dataset:

1. Shell interaction. In some shellcodes, malware download is triggered by a set of shell commands that are either embedded in the shellcode itself or pushed through a separate TCP connection.

2. File name. When available, the file name of the malware samples once pushed to the victim.

3. Download/upload protocol. The type of protocol ultimately used to transfer the malware content, either by uploading it to the victim or by forcing the victim to download it.

4. Download source. The host providing the malware content. In most cases, the malware is provided by the attacker itself, but we have witnessed a limited number of cases in which the malware was downloaded from an external repository.

In the totality of cases observed by SGNET, a given malware sample was always seen as propagating using the same propagation vector. That is, we have never observed cases of multi-headed worms. Most interestingly, some propagation vectors proved to be unusually frequent, and shared by a variety of completely different malware families.

3 Dataset: ANUBIS (TUV)

3.1 The Anubis Malware Behavior Database

As discussed in Deliverable D06 (D3.1) "Infrastructure Design", Anubis is a dynamic analysis environment for malicious code. When analyzing a binary, Anubis executes it in an instrumented sandbox environment and produces a human-readable *Behavioral Analysis Report* describing the system- and network-level actions performed by the binary. This report generally contains enough information for a human expert to establish whether the suspicious binary is in fact malicious, and to get a high-level overview of some of the malicious functionality it implements.

As part of the WOMBAT project, the Anubis service was enhanced by implementing a Malware Behavior Database that is used to store the information available in *Behavioral Analysis Reports* and allow efficient searching, data mining, and analysis of the collected data. The Anubis Malware Behavior Database was briefly discussed in Deliverable D08 (D4.1) "Specification language for code behavior".

The purpose of this database is primarily to store *behavioral* features of the analyzed malware samples. However, some of the malware's behavior, and particularly its network activity, can reveal important information on the context of the malware infection, such as the command and control infrastructure it employs. In some cases, this information can be directly compared and correlated with contextual information available from other sources. As an example, the same HTTP server may be observed serving malware binaries to executables analyzed by Anubis, and to exploits captured by the SGNET distributed honeypot (as reflected in the "Download Source" feature discussed in Section 2.5).

3.2 Anubis and the Malware Life-Cycle

> "Adding to this confusion is the tendency of the botnet owners to frequently change Pushdo's functionality and code. *It is perhaps better to think of Pushdo as a criminal operation, rather than a single piece of malware*"
>
> Trend Micro report on the Pushdo/Cutwail botnet, May 2009 [3].

Malicious software is a crucial component of criminal activity on the internet. Because of the increasing complexity of the techniques employed by cyber-criminals, malware samples cannot simply be analysed in isolation but must be considered as components of the infrastructure deployed by cyber-criminals to support their fraudulent activities. Compared to honeypots such as SGNET (discussed in Chapter 2) or HSN (discussed in Chapter 8, Anubis provides information on different stages of the malware life-cycle. Client-side and server-side Honeypots observe the propagation mechanisms with which hosts are initially infected with malware, such as exploits against network services, and drive-by-download attacks from malicious web servers. Anubis cannot provide this information: Samples are directly submitted by end-users, network administrators, or other security researchers. In most cases, we do not have information on how a binary that is submitted to Anubis for analysis was originally obtained by the submitter. In fact, in some cases the submitter himself may not know: end users sometimes submit a binary to Anubis precisely because they do not know how it found its way into their systems.

On the other hand, Anubis provides a wealth of information on the behavior of malicious code *after* the host has been compromised. While malware behavior has been discussed in D08 (D4.1) "Specification language for code behavior", and is outside the scope of this document, the network-level activity performed by a malicious binary reveals valuable information on the context of the malware infection, such as the command and control infrastructure it relies on, or the modus operandi employed for malware propagation. These network-related contextual features can in many cases directly be compared with other data sources, as well as provide additional insignt into suspicious traffic observed at the network level.

In the following sections, we discuss contextual features available in the Anubis database. Before introducing the network-level features, we discuss contextual features related to Anubis sample submission.

3.3 Submission Features

As discussed in the previous Section, Anubis can provide limited information on the origin of a malware sample. Nonetheless, some contextual data is available that may provide additional insight.

Number of submissions. A single binary sample may be submitted to Anubis by several users over time. The number of submission provides an indication of the prevalence of the binary in the wild. Furthermore, polymorphic malware generally produces unique binaries for each infection. Therefore, binaries that are submitted multiple times by different users are extremely unlikely to be polymorphic.

Time of submission. For each submission of a binary, the date and time that it was submitted. This can provide some information on the period of time over which this malicious binary was active in the wild. In particular, the first time of submission for a binary provides a useful lower bound for the ”age” of the malware sample.

Submitter. The Anubis user(s) that submitted the binary to Anubis. Since registration is not required to submit samples to Anubis, this information is not always available.

Submitter IP. The IP address(es) from which the binary was submitted to Anubis. This provides some information on the submitters of samples sent to Anubis by anonymous users.

Captcha solved. Whether the submitter solved the Anubis CAPTCHA when sending in the sample. This indicates whether the submitter is a human being that manually submits individual samples or an automated program. Users are encouraged to solve the CAPTCHA, since it causes the submitted samples to be analyzed with higher priority.

Submitted file name. The name of the submitted file. A single binary may be submitted by different users under different names. The name is often the name under which the malware appears in the wild. Sometimes, users give sample meaningful names, such as ”keylogger.exe”, before submitting them to Anubis. Finally, honeypots typically give conventional file names to binaries they collect. For instance, the nepenthes honeypot typically stores collected files with names starting with ”nepenthes-”.

User comments. Anubis users can leave comments on the files analyzed. This is sometimes used to inform us of issues with Anubis, such as binaries that detect the Anubis environment and do not perform their intended behavior. Additionally, some users inform us of the origin of the sample, or share information on the malware’s purpose.

3.4 Network Features

Anubis collects all the network traffic generated by the analyzed samples. Information from the IP/TCP/UDP headers and from DNS packets is stored into the Malware Behavior Database. Furthermore, scan detection heuristics are employed, and a number of application-layer protocols that are frequently used by malicious code are detected and analyzed. Specifically, features related to the HTTP, IRC, SMTP and FTP protocols are stored.

UDP Traffic Features For each UDP conversation, the destination IP address and port, as well as the number of packets sent and received.

TCP Traffic Features For each TCP connection, the destination IP address and port, as well as the number of packets sent and received. Additionally, information is available on whether the TCP connection was successfully established and terminated.

Queried Domains For each DNS query, the queried domain, the query type, and the result of the query. While malware also performs queries for legitimate domains, many of the queried domains are related to botnet command and control. Furthermore, the result of the query at the time of analysis is useful because malicious infrastructure is frequenly hosted using fast-flux hosting techniques. Therefore, resolving the domain name at a later date will lead to different IP addresses.

Opened Ports TCP and UDP ports that the binary opened for listening. Opening ports is an indication of backdoor functionality. The use of certain port numbers may be characteristic of certain malware variants. This information can therefore allow attribution of suspicious traffic observed at the network level to a specific malware.

Port Scans The target IP address and number of scanned ports for each host that is scanned. Patterns in scanning behavior can be used to correlate malware analyzed in Anubis with attacks observed by honeypots such as SGNET.

Address Scans The scanned subnet and the number of scanned hosts.

HTTP Traffic Features These include the type of HTTP request (GET,POST,..), the requested Host and URL, the client and server headers, the User-Agent, and the content type of the requested page. Furthermore, the entire body of POST requests and of HTTP replies is available. The IP address, DNS domain name and TCP port used by the contacted server are of course also available, as they are for all TCP connections. HTTP is currently the main protocol used by malware for Command and Control, therefore many of the HTTP servers contacted by Anubis malware are malicious.

IRC Traffic Features These include the name, password and topic of IRC channels joined by the malware, as well as the nickname and username it employs. IRC servers contacted during Anubis analysis are invariably used for Command and Control, but they may in fact be legitimate, publicly accessible servers. Malware authors create their own channels on these servers and use them to control their botnets.

SMTP Traffic Features Emails sent by malware inside Anubis are invariably SPAM or phishing emails. SMTP features include the recipient name and email address and the subject, sender address, and full content of the email. Furthermore, the name and file type of all attachments is available. This data can help identify the origin of spam campains that send messages similar to those observer in Anubis.

FTP Traffic Features These include the username and password used to connect to the FTP server.

4 Dataset: Virustotal (HISPASEC)

The VirusTotal WAPI Dataset was conceived as a means of querying information on files uploaded to the free online service (http://www.virustotal.com) and retrieving metadata about the submissions themselves (number of times a file was sent, names with which the files were uploaded, etc.).

VirusTotal was initially developed as an online multiantivirus scanning tool, as such, the obvious piece of information that can be included in its dataset are the antivirus engine results themselves. Having said this, it is not the sole data that is present in the VirusTotal database. Soon, the web application started to integrate file characterization tools such as PEiD (identification of packed files), Sigcheck (identification of files that are digitally signed), PDFiD (characterization of PDF files), etc. The output of these tools is or will be part of the dataset, hence, this dataset can be described as a static file characterization database. The word file must be emphasized, many of the other WAPI datasets work only with confirmed malware, given the public nature of VirusTotal, all kinds of files are submitted on a daily basis. Since many of its massive users are malware research related teams (universities, antivirus labs, honeypot and honeyclient deployments, etc.), a noticeable portion of the received files are indeed malware. Nonetheless, also freeware, shareware, all kinds of software, legitimate music files, documents and other innocuous files are present in the VirusTotal store and can be queried via its dataset.

4.1 VirusTotal dataset contextual features

The VirusTotal dataset is made up of three basic objects: source, file and analysis, the file object being the keystone of the dataset. A brief description of each type is provided below:

- Source: represents a given VirusTotal user. In the context of VirusTotal a user is considered to be a unique IP or email. The fact that IPs might be reused is ignored.
- File: represents a file submitted to VirusTotal. The reader should note that the term malware is not used since files of any nature are received at VirusTotal.

- Analysis: is associated to a particular scan request on a given file by a specific user, in other words one file may be subjected to several analyses, either because they have been performed at different moments in time or because different users have requested it. The analysis displays all the antivirus engine information returned by VirusTotal on a particular file, the output of other tools is considered to be static and are introduced in the dataset as an intrinsic property of the file object.

All of these objects have several attributes, methods and reference methods that provide useful information to the user and allow for data correlation. A deeper glance at the different types is now given, justifying, where appropriate, why certain pieces of information were included in the dataset.

4.2 Source

Currently this object only has only one attribute, source_type, which identifies, whenever it is known, the nature of the source that sent the file to VirusTotal. A (growing) discrete number of category labels are used: security company, antivirus vendor, telco company, anonymous, etc. Under particular circumstances this feature could be interesting for law enforcement (determination of a malicious origin), yet it may proof itself even more useful when trying to determine whether repeated submissions of a same file are product of the research activity of a particular user or due to the predominance of a given sample in the wild.

Certain users repeatedly submit a same set of samples to VirusTotal in order to detect signature updates and measure antivirus industry reaction times. For example, the file with md5 hash dceb3b61d06e4a96ea13cbd0b0225dfe was sent 100 times, yet only three different sources account for these submissions, one of them being the SGNET deployment with 97 uploads. This user would appear in the dataset with the label Honeypot/Honeyclient.

This object could also include an attribute identifying the country of origin of the source, making use of tools like Maxmind this is a simple task. This information is actually present in the VirusTotal database but has not yet been included in the WAPI dataset, once included, it will be very useful in identifying country based targeted attacks. For example, we have noticed that certain banker samples that affect Peruvian banks are uploaded to VirusTotal mostly by Peruvian users.

4.3 File

This is the keystone object of the dataset, its attributes can be grouped according to the goal they pursue:

- File identification: md5, sha1, sha256. These are the main hashes currently being used by the malware research community, they are the means for data correlation with other databases and datasets.
- Preliminary file characterization: size (file size in bytes), file_description (magic number descriptor) , portable_executable (determines whether the file is a portable executable), dll (determines whether the file is a dynamically linked library). These attributes provide a quick overview of the type of file that is being dealt with.
- Portable executable immediate information: whenever a file is a Windows Portable Executable, there are a series of file properties that can be immediately retrieved from the database at low cost. These attributes are entry_point (initial execution offset), pe_timestamp (date-time indicating the compilation time of the executable, can be faked), ep_surroundings (first 256 bytes that follow the entry point), machine_type (target architecture of the executable). All these attributes allow for a preliminary morphological characterization of an executable file, it is information that might be used for clustering of malware based on static properties rather than behavioural analysis. This is information is extended by certain object methods that will be later described.
- VirusTotal service related metadata: there are a number of interesting parameters that can be extracted from the submissions to VirusTotal that are not dependant on the binary properties of the file. first_seen and last_seen represent the first and last submission date-times of the file to VirusTotal, it allows the analyst to know how old, at least, a file is and may give a notion of whether it is still found in the wild. last_detection_ratio gives the fraction of antivirus engines that detected the file in its most recent analysis. num_submissions and num_diff_submission_sources are, respectively, the number of times that the file was sent to VirusTotal and the number of these that were submitted by different sources. Finally, num_hash_requests indicates the number of times a particular file was queried for analysis information via the hash request feature in VirusTotals service web site. These last pieces give an idea of how prevalent a given sample is in the wild.

The file object then has a series of methods that may also be categorized:

- Tool analysis: get_packers returns the packer signatures that match a given file according to tools like PEiD or the antivirus engines themselves, this allows the analyst to know whether the sample is packed and what unpacker (if any) he should use to get to the original entry point. get_tools_information returns the results of the rest of tools integrated in VirusTotal, for example, thanks to TRiD a WAPI user can get to know what is the most probable file type of a given sample.

- Portable executable advanced information: whenever a file is not packed, get_pe_imports and get_pe_exports are methods that may proof themselves useful in order to have a very top level idea of what a sample is doing based on static inference. On the other hand, get_pe_sections lists the sections of the executable along with basic information about them (entropy, offsets, size), this information can be used when clustering samples according to portable executable properties and to determine whether a sample is potentially packed.

- VirusTotal service related metadata: the dataset stores the file names with which samples are submitted for analysis, sometimes these names will simply be the md5 the samples, this gives no interesting information. However, very often the submission names will be the actual file names with which a given sample is executed in the wild, hence, the get_submission_names method may proof itself very useful when undertaking forensic investigations. In the context of other projects related with VirusTotal, over 3000 spanish home user PCs are scanned for malware, the wild paths and wild names of any files that are present in the VirusTotal database is also part of the dataset and can be queried via the get_wild_names and get_wild_paths methods. Finally, another useful element for forensic analyses is the names with which a file is found hosted in web servers whenever its propagation strategy is via drive-by-downloads or web-based social engineering. Thanks to a honeyclient setup these names are available through the get_wild_names method.

- Antivirus engine analyses: get_detection_ratio_evolution retrieves the detection ratio of all analysis performed on the given file. This may allow researchers to detect signature improvements and signature failures. If something interesting is observed, the dataset user may then get a deeper understanding of what happened using the WR_get_most_recent_analyses method, which returns the latest 100 analyses performed on the sample. Similarly, WR_get_last_analysis and WR_get_first_analysis return, respectively, references to the last and first analysis of the sample.

4.4 Analysis

As already mentioned, the analysis entity only includes antivirus engine results, this gives a hint about the malicious nature of a file. Redundant data such as the detection ratio of the analysis is also offered as an attribute. The analysis allows identifying the source that requested it and the moment in time in which it ended.

As it was previously mentioned, these features are a useful resource to statically characterize samples, in conjunction with the rest of the datasets that make up the WOMBAT api it might be a very valuable resource for forensics investigations and malware research. The main difficulty with VirusTotals dataset is its size, with over 25 million samples at the present moment in time, not all pieces of information on a given sample can be returned at once since it would be very costly in terms of api response time. For example, when querying for all the analyses performed on a given sample, only the latest 100 are returned, if this was not the case, certain samples could have thousands of analysis that would end up in relatively high network transfer times.

5 Dataset: Shelia (VU)

Shelia is a Windows-based honeypot for the client side. The current version is available from `http://www.cs.vu.nl/~herbertb/misc/shelia`. It will detect exploits of client applications and consists of two main parts: a management/control part and an intrusion detection engine. Among other things, the management/control part feeds the detection engine with things to check - URLs, potentially malicious attachments, etc. It allows for different ways to get these items to the detection engine.

A particularly interesting feeder is what we call the *client emulator*. The emulator connects to an IMAP server and emulates a naive user reading email. It will read the email in a folder, dig out all the links and attachments from emails and feeds these to the detection engine. The detection engine blindly follows all the links, opens all attachments, etc.

The intrusion detection engine determines whether the website, or the attachment, is malicious and if so, it raises an alert. Unlike most other systems, Shelia creates virtually no false positives (although there may be false negatives).

The contextual and other features collected by Shelia include the set listed below. We divide the features in categories, specify how they can be useful, and for some them, print an example.

1. Target context

   a) *Target application.* Which application was exploited? For analysis, we need as much information as possible about the victim process. For instance, if we want to be able to say that attackers move from server applications to client applications, or from web browsers to instant messengers, we need to know what sort of process was exploited. In addition, we may be interested in more detailed questions, such as: "Which version of an IE running on what type of host is attacked most". For this purpose, we need more information (e.g., the version number). Shelia currently provides fairly limited information about the victim process - the name of the binary and the path. Example:

   ```
   Target application = C:\Program Files\Internet Explorer\iexplore.exe
   ```

   b) *System address width.* We collect a flag that indicates whether the system on which the malware was collected was 32 or 64 bits. This will help us assess the vulnerability on different platforms (does it work on 32 *and* 64 bits, or only one of the two. In addition, it will help us repeat the experiment.

c) *Pid.* The process identifier of the victim process. The pid is not directly useful in most security analyses (except if the process that is attacked is one with a well-known pid, which seems unlikely). However, it may be helpful locally to perform further forensics.

2. Capture context

   a) *Timestamp.* When was the alert generated? Time stamps are an important bit of contextual information. They allow us to correlate attacks at different sites, evaluate attack intensity in time periods, and spot attackers' patterns in time (e.g., do attack campaigns start at specific time of the day?). Example:
   `Timestamp = 2009-08-24 10:51:53`

3. Exploit context

   a) *Attack object.* The object identifies the URL or filename that was used to compromise the target. It is important to identify URLs, so as to correlate the URLs with other databases that black list URLs and to assess the lifetime of a malicious site. Example:
   `Object = http://azadars.com`

   b) *Payload.* As accurately as possible, we store the exploit (in binary). The exploit may need further analyses (e.g., to see what methods are used to get the value of the program counter, or to classify the exploit according to the system/API calls that it makes).

   c) *Payload address.* Shelia stores the location of the malware in the victim's memory. The address may help us identify the attack in broad terms. For instance, we may be able to classify the attack as heap overflow. Example:
   `Address = 202571238`

4. Malware identifiers and attributes

   a) *File name.* Shelia stores the name of the malware file that is downloaded on the victim system. While file names may vary as malware spreads, sometimes the filename provides a hint. Example:
   `File name = C:\DOCUME~1\user\LOCALS~1\Temp\update.exe`

b) MD5 and SHA1 and sha-256 hashes: uniquely identify a specific malware binary. They allows to check whether different samples are exactly the same, and to look up the malware in other repositories. Example:

```
Md5    = 00b23b08657a153fcde4e0891e2484bb
Sha1   = 522674387e1a8e2d3ab5f7c11ecd9db7e5904dc4
Sha256 = b851756487f055bb746cae506e5ffc016f88a07177ab7bfc5b8be7208cbc8156
```

c) *Length.* The length of the malware binary in bytes. Often, the hashes of a binary are different, because the malware samples different in a few bytes. In that case, the length of the binary may be a first useful hint to see if binaries 'look' similar.

d) *Binary.* The binary can be analysed, run, tested, etc.

5. Activity context

   a) *Calls and call arguments.* For the victim process, we store all calls to a carefully selected subset of the Windows API. This is interesting, as it allows us to assess the attack's *behaviour.* In addition, we may use the calls and the call arguments to whether certain files or registry key values have been modified, etc. As such, they can also be classified as context. Example:

   ```
   Call = 'ReadFile', Args = 'T Y P E L I B'
   ```

   b) *Description.* To further help the security analysis, we store a description of the function that is called. Besides helping out non expert analysts, the description does not play a further role in the contextual analysis.

6 Dataset: NoAH (VU and FORTH)

As described earlier in WOMBAT deliverable D3.2 ("Design and prototypes of new sensors") NoAH is an architecture for large-scale threat monitoring. In a nutshell, NoAH clients, namely Honey@home, forward traffic to unused IP addresses or unused ports to a honeypot farm and forward the replies provided by the honeypots back to the attackers. Honey@home is designed to be simple and lightweight, as its main target audience is home users or administrators unfamiliar with honeypot technologies. The software package that comes with it needs no special configuration to run, and is ported to run on both Linux and Windows platforms. Also, it is non-intrusive as it runs in the background with minimal CPU, memory, and network overhead.

The contextual features that make sense to maintain in the NoAH infrastructure are the IP addresses, timestamp, TCP ports, TCP flags and payload size.

6.1 Geolocation

The geolocation of IP addresses provide us information about the attacker's topology. The geographic location is retrieved by querying a local MaxMind database. Based on this we can categorize the attackers according to their geographical position. That led us to the creation of *Attack maps*. These maps display the geographic distribution of distinct source IP addresses. Each country is colored based on how many IP addresses are hosted in that country. Countries that host no attackers are colored as white, low activity countries are colored as green while countries that host lots of attacking IP addresses are red.

6.2 Packet Characteristics

The timestamp the packet was captured is important for our analysis. Based on the packet timestamps, we can identify attack patterns such as diurnal cycles in malware propagation and host scanning. Furthermore, we can gather information about the attacking behavior by examining the inter-arrival of packets. For example, many scanning tools send constant number of packets per second. Timing analysis can reveal such information.

Furthermore, we hold information about TCP ports. TCP and UDP port information help us understand what services are targeted the most so as to configure the NoAH honeypots. By emulating heavily attacked ports, we are able to capture more attack instances and malware. For example, over the last year we observed a significant increase in sniffing activity on TCP Port 445 that signaled an impending mass malicious code attack targeting a vulnerability over the SMB protocol.

Finally, each NoAH sensor receives unsolicited traffic, that is traffic that comes in response to spoofed attacks. It is trivial to identify such traffic by inspecting the TCP flags of each packet.

6.3 Malware Characteristics

The honeypot farm maintained by the NoAH infrastructure runs multiple instances of medium and high-interaction honeypots. These honeypots emulate/run a number of services and are able to capture exploitation attempts, shellcode injections and malware samples.

For the shellcode and malware samples, the information stored in the NoAH database is:

- *Timestamp* is essential to record the date and time the exploitation attempt was made or the malware sample was downloaded. Timestamp allows us to identify the lifetime of a malware sample and the persistence of attack vectors.
- *IP addresses and TCP/UDP ports.* This is the basic information required to identify the attacker source as well as the targeted honeypot and service.
- *Shellcode type.* Based on the stages of the exploitation attempt, we are able to identify what kind of exploit was sent by the attackers. For example, in the case of port 445 we can identify various types of shellcodes, such as DCOM, ASN1, PNP or LSASS vulnerabilities that were triggered.
- *Shellcode* itself is also recorded. The hexdump of the shellcode can be later used for post-analysis.
- *Malware sample.* If the shellcode vector leads to a malware fetch, we download it. For the malware we store the binary data itself as well as the md5 sum of the sample.

7 Dataset: HARMUR (Symantec)

As described in Deliverable 3.2, HARMUR is a repository of information on the evolution of client side threats, with special focus on their temporal dimension. Differently from HoneySpider or Shelia, HARMUR is not, per se, a honeyclient, and does not crawl the web searching for vulnerabilities. HARMUR is an aggregator for information generated by other security services, information that is classified by HARMUR into two orthogonal categories:

- Site feeds. Lists of sites that have been judged by a honeyclient as containing malicious or suspicious content.
- Analysis feeds. Information on the current state of each site, on its security state (e.g. type of threats associated to its content) and on its current context (e.g. nature of the web server hosting the content).

By iterating the analysis of each site over the time, HARMUR is able to collect information on the temporal evolution of security threats, and add important information blocks to the understanding of nowadays threats economy.

HARMUR specifically focuses on the security and network properties of sites rather than malware samples. Still, whenever a malware sample can be mapped back to a set of sites responsible for its propagation, HARMUR can provide information instrumental to the definition of the malware propagation context, and its dynamics.

7.1 Naming information

For each domain name, HARMUR tries to collect as much information as possible on its association to physical resources such as its authoritative nameservers, as well as the address of the servers associated to the known hostnames for such domain. DNS information is retrieved by HARMUR by directly contacting the authoritative nameserver for each domain, avoiding the artifacts normally generated by DNS caching.

On top of the basic DNS information, HARMUR takes advantage of the whois protocol to retrieve information on the registration information for each domain name. Such information can be extremely valuable in understanding the modus operandi of the attackers. For instance, a single registrant registered on ONLINENIC 71 distinct domains exactly on the same day. The domain names were the result of the permutation of a few dictionary words associated to antivirus software, and the all the hostnames known to HARMUR as belonging to these domains resolved to a single physical web server. A more in depth analysis revealed that all these domains were ultimately used for the distribution of rogue AV software.

Despite its usefulness in getting insights on the dynamics of domain registrations, whois information proved to be very difficult to extract in practice. RFC 3912, describing the WHOIS protocol specification, clearly states that *the protocol delivers its content in a human-readable format.* In the context of an automated retrieval and dissection of the whois information, this is a non-negligible problem: every registrar uses different notation and syntax to represent the managed registration records. We have, therefore, implemented parsers for the 17 most common registrars, but despite of this effort, only 311,245 domains out of a total of 1,509,101 domains tracked by HARMUR at this date (20.6%) are correctly associated to complete registration information. While this number can be certainly improved in the future by adding support for additional registrars, it clearly underlines the underneath problem.

7.2 Server information

HARMUR extracts a variety of different information on each web server known to be hosting one or more of the tracked sites.

Geolocation. Similarly to many other WOMBAT datasets, HARMUR tries to take advantage of geolocation databases such as Maxmind to collect information on the geographical location of the servers hosting the tracked content. As previously said in this document, the geographical context of a certain threat can provide useful hints on its root cause.

AS information. In parallel to geolocation, HARMUR collects information on the Autonomous System hosting the tracked threats. As shown by FIRE [**?**], online criminals have been often masquerdading behind disreputable Internet Service Providers known to have very loose control on the activity of their customers. Collection of AS information in HARMUR is possible thanks to Team Cymru's IP to ASN mapping project[1]. Interestingly, while using this service we have detected a limited number of cases in which the server was mapping a single IP address to multiple AS numbers. To disambiguate

[1]`http://www.team-cymru.org/Services/ip-to-asn.html`

such cases, we integrate the Team Cymru's information with that provided by the Route Views project (`routeviews.org`).

Web server state. HARMUR was designed to be able to scale to an extremely large number of domains, and is therefore unable to perform expensive analyses on the tracked domains. Still, it is possible to retrieve from web servers very valuable information on the operation of the server. More specifically, HARMUR probes each server with an HTTP HEAD request for the root page of the site. Such action allows us to obtain statistics on the current reachability of the server, and associated downtimes. Also, by looking at the HTTP headers of the response, we are able to collect information on the version of HTTP server that is currently running on the server. As of today, we have discovered 21202 different version strings on 303715 different servers. Some very specific version strings, have been seen on a large number of servers, suggesting the usage of standard, "out of the box" configurations in the deployment of malicious domains. For instance, the following version string was witnessed on 4510 different servers:

```
Apache/2.2.11 (Unix) mod_ssl/2.2.11 OpenSSL/0.9.8i
DAV/2 mod_auth_passthrough/2.1 mod_bwlimited/1.4
FrontPage/5.0.2.2635
```

7.3 Dynamic information

HARMUR data collection mechanism aims at generating information on the evolution of the tracked domains. All the features previously described are therefore repeatedly extracted throughout the lifetime of a malicious domain.

Figure 7.1 provides a practical example of the importance of this dynamic traffic for the study of fast flux networks. In each rectangle, we have plotted the set of web server IP addresses associated to a specific domain at a given point in time. By looking at the evolution of this association, we can see how domains apparently not correlated end up in intersecting each other over time. Most probably, these different domains are managed by the same individual or organization and are hidden by the same fast-flux network. Without taking into consideration these dynamics, the relationship between these domains would not have been trivial to identify.

A similar example of "dynamic correlation" among domains can be seen in Figure 7.2. Different domains, apparently unrelated and initially hosted on different servers, have been all moved within a few days to another web server.

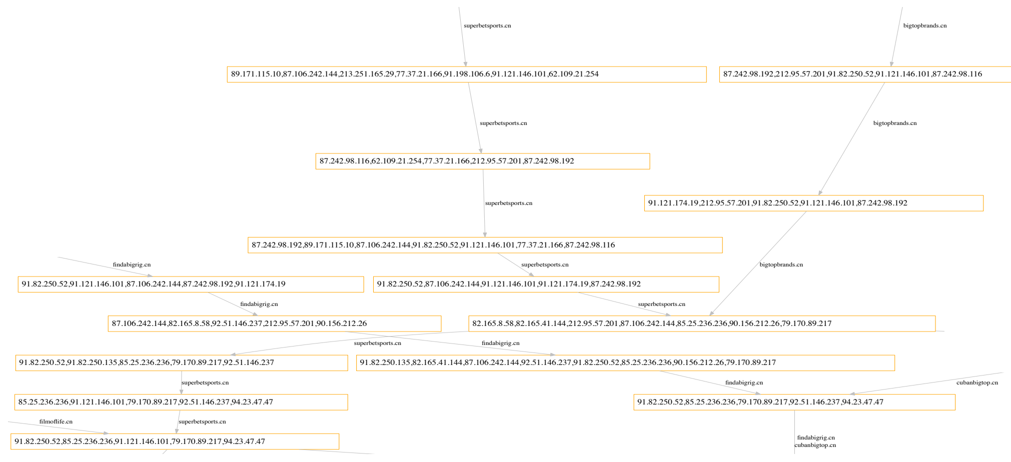

Figure 7.1: Evolution of DNS information for fast-flux networks

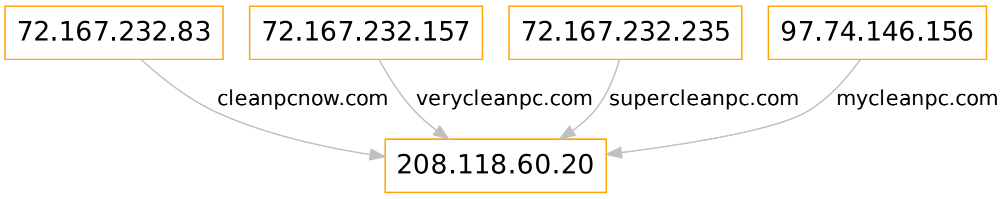

Figure 7.2: Rogue site dynamics

8 Dataset: HoneySpider Network (NASK)

8.1 Introduction

The HoneySpider Network (HSN) is a hybrid client honeypot aimed at fast and accurate classification of web-pages. The main goal of HSN is to identify whether the visited web-page is malicious or not in the shortest time possible. For this purpose, the system design employs two types of client honeypots. Low-Interaction client honeypot is used to filter out malicious and suspicious web-pages and is the first line of detection. To confirm its results a High-Interaction client honeypot is used. The layered approach is used to improve the detection rate of the system, as the scope of low and high interaction honeypots is different.

As HSN is concerned primarily with URLs, not malware, we understand context in terms of the circumstances that surround the collection and assessment of a URL and thus determine its role in an event. This context can then contribute to a better understanding of the attack infrastructures and players behind client-side attacks i.e. improving the "threats intelligence" and "attack attribution". We have identified a number of contextual features behind the usage of a URL for malicious infection purposes. Not all of these are collected by HSN, but will be in the future. What follows is a list of features, along with their short explanation, whether they are currently being collected or not, and their importance and usefulness for the tasks of attack attribution.

8.2 Features identified

URL name The naming scheme behind URLs can uncover possible links between URLs, demonstrating that they can be part of the same attack. TLD usage (e.g. .cn or .com) can also contribute to additional information about an attack. HSN stores all the URL names discovered, including those that are generated as requests when visiting the initial parent URL. An addition of heuristics that look for some basic similarities between URL naming is not difficult.

Geolocation information The origin of an attack is a useful piece of contextual metadata. This includes information like country code (i.e. the country where the malicious web page is hosted), autonomous system number, operator name. HSN collects all this information, not just for the initial URL but for all the further requests. This is done in real-time, by querying the Team Cymru Whois database through DNS requests.

IP characteristics IPs that the URL resolves to can also provide insight: is there a limited group of IPs being used, are these on a similar subnet or network class, is there overlap between seemingly unrelated URLs in terms of IP usage etc. HSN stores all IPs associated with a URL when the URL is being checked. Overlap between these URLs is part of the work we are currently attempting for our threats intelligence.

Fast flux assessment Whether a URL is fast-flux or not is a useful piece of information that can show relationships between collected URLs as well as give additional insight into an attack infrastructure. HSN makes such an assessment, based on an algorithm that takes into account the TTL of a domain, amount of A records returned, network distance between the A records, and the existence of certain keywords in reverse DNS records for the A records returned (a sort of an "individual subscriber" count).

Whois information WHOIS database information about a domain name can supply information such as DNS Registrant (ie. the entity that owns the domain), DNS registrar (ie. the entity – organization – through which the domain was registered), date of registration. Information about the registrants e-mail address is also an example of information that can be gained. Currently, HSN does not collect whois information.

Referer used The referer that was used when visiting a web site is an important piece of contextual information. Many malicious web sites behave differently depending on how they are accessed. For instance, a URL may serve malicious code when accessed from the Google Search results page and differently when accessed directly. HSN stores the referer headers used.

Browser profile Browser headers, such as the User-Agent, Accept, Accept-Language, Accept-Encoding headers etc, and their ordering can also influence the behavior of a URL. Basic browser headers are supported by HSN.

Server type information Information about the HTTP server version behind a URL can also be a useful feature. This information is stored by HSN.

URL source URLs can be obtained in different ways. They can be collected from spamtraps (e-mail, instant messenger), dynamically created from Google queries, obtained from proxy logs, or reported by some external party. How they are obtained can also give insight into how a malware group searches for possible targets. It is not only the source type that is of interest, but the specific source as well. This may help to infer targeted attacks. Information relating to the origin of URL submission is stored in HSN.

URL categorization Initial URLs that are obtained for checking are most often content serving URLs. They normally do not serve exploits directly or serve malware. Usually, they have iframes injected into them that serve as redirects to the actual exploit or malware download site. There can be multiple redirect, exploit and malware download URLs involved, forming an attack tree. Recreating such attack trees of URLs and finding relationships between them can be very useful in identifying different groups of attackers. Recreating such attack trees and looking for relationships between them is the main area of interest for HSN.

Exploits identified Exploit information about a URL can be useful in understanding an attack infrastructure. This can be information relating not only to a specific exploit, but to a more generic identifier, such as whether the URL is a drive by download or not. HSN does not currently provide information about the exact exploits being used, but will do so in future versions.

Malware collected Malware collected from a URL can also give an insight about the relationships between sites. HSN can collect malware from URLs. However, this is not the main area of operation of the system. Malware is stored by the system along with MD5/SHA1 hashes, which can be used to further identify the roles and relationships of URLs being investigated.

In summary, there is a wealth of contextual information that can be obtained about a URL and its nature. Most of the above features are currently part of the HSN dataset. Key to the dataset is the "path of exploitation" from the compromised server (ie. the content site or the landing site) to the one serving malware. Building this types of relation-graphs is the goal of the threat intelligence WP of the WOMBAT project. The goal, from the HSN perspective, is to discover global network of relations which will allow for the recognition of the servers of main interest to the CERT teams – not just exploit servers or servers the malware is downloaded from or servers which only redirect further but are located in the CERT's constituency.

9 Dataset: Bluebat (Polimi)

BlueBat is an ad hoc device based on the GNU/Linux OS, which is primarily designed to collect samples of the latter type of malware, using a set of software based on the pybluez framework [2].

We remind to the reader that BlueBat is an experimental Bluetooth honeypot sensor, which we described in earlier WOMBAT deliverables D06 and D13, as well as in a publication [5]. As discussed in previous deliverable D06, Bluetooth exhibits a number of security issues in various specific implementations of the stack. Such attacks are well known, and allow different degrees of data access, communication interception, up to and including running any AT command taking full control of the phone. However, viruses for mobile device primarily rely on simple *social engineering* to propagate, sending copies of themselves to any device which comes into range through an OBEX push connection.

Given this specific setting, contextual features are both peculiar and limited.

9.1 Features collected

Basic features Basically, the entries logged by a BlueBat sensor contain a description of the interaction with the peer, the address of the peer, and if the transaction pushed a file (likely, a malware file), the file itself and its MD5 hash. The latter feature can be used to cross-search for the same binary on other datasets that may likely contain it (specifically, Shelia and Virustotal).

The address of the peer and the filename of the pushed file are simple to gather and they can be considered contextual features. Their usefulness is similar to what happens in TCP/IP based exploit collection, so we will not need to elaborate on this, except to note that while an IP address is logical, the MAC address of a Bluetooth peer is a physical and unique identifier of a device. Also, since most of the other datasets use IP addresses as contextual information, this information cannot be effectively cross-linked.

Geolocation Things become more interesting when considering geolocation, as this is an extremely important parameter in our collection setting. Geolocation for Bluetooth devices does not need to be inferred by databases (as it happens for IP addresses), but is directly inferred by their proximity to a specific honeypot. Honeypots in our current deployments are static, so the collected data can be very simply geographically tagged. For mobile honeypots, we have developed a connection with the gpsd [1] GPS daemon for tagging with location data.

Blueprinting In parallel to the data collection, we perform a continuous scanning for devices, and we fingerprint the ones we find, using the process called "blueprinting", which is described in [4]. Basically, and similarly to what happens with hosts on the Internet, by mapping the different services exposed by a device and several Bluetooth stack parameters, it is sometimes possible to determine remotely the type of hardware and software of the device.

This is actually difficult to do in extremely open settings, as device scanning is very slow. So even if we use extremely powerful antennas for running the OBEX server, we need to scale down to less powerful ones for additional scanning. This means that we will not always be able to have the fingerprinting results for a given device. Additionally, Blueprinting has less and less reliable databases as the number of variations of hardware and software on Bluetooth enabled devices grows, and this is a shortcoming that cannot be addressed.

10 Conclusion

The ob jective of Workpackage 4 is to develop techniques to characterize the malicious code that is collected in the previous workpackage. The main idea is to enrich the collected code thanks to metadata that might reveal insights into the origin of the code and the intentions of those that created, released or used it.

This intermediate deliverable provides a preliminary discussion of contextual features of malware and malware-based threats in the datasets that are maintained by the WOMBAT partners. Some of these features can be used to analyze potential threats and discriminate collected samples that are mere variations of already known threats.

Bibliography

[1] Gpsd website. http://gpsd.berlios.de/.

[2] Pybluez website. http://org.csail.mit.edu/pybluez/.

[3] A study of the pushdo/cutwail botnet. http://us.trendmicro.com/imperia/md/content/us/pdf/threats/securitylibrary/study_of_pushdo.pdf, 2009.

[4] L. Carettoni, C. Merloni, and S. Zanero. Studying bluetooth malware propagation: The bluebag project. *Security & Privacy, IEEE*, 5(2):17–25, March-April 2007.

[5] A. Galante, A. Kokos, and S. Zanero. Bluebat: Towards practical bluetooth honeypots. In *Communications, 2009. ICC '09. IEEE International Conference on*, pages 1–6, June 2009.