



**WORLDWIDE OBSERVATORY OF
MALICIOUS BEHAVIORS AND ATTACK THREATS**

D13 (D3.3) Sensor Deployment

Contract No. FP7-ICT-216026-WOMBAT

Workpackage	WP3 - Data Collection and Distribution
Author	-
Version	0.1
Date of delivery	M24
Actual Date of Delivery	M24
Dissemination level	Public
Responsible	FORTH
Data included from	POLIMI, NASK, VU, SYMANTEC, I2R

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 Routes des Cretes
06560 Sophia Antipolis
France

e-mail: Marc_Dacier@symantec.com

Phone: +33 4 93 00 82 17

Contents

1	Introduction	8
1.1	Overview	8
1.2	WAPI	10
2	SGNET	12
2.1	Introduction	12
2.2	Deployment and experiences	13
2.3	Current and future work	19
3	HARMUR	20
3.1	Introduction	20
3.2	Deployment and experiences	21
3.3	Current and future work	24
4	Shelia	26
4.1	Introduction	26
4.1.1	Shelia recap	26
4.2	Deployment and experiences	27
4.3	Current and future work	27
5	Paranoid Android and Multi-level intrusion detection	30
5.1	Introduction	30
5.2	Deployment and experiences	31
5.3	Current and future work	32
6	HoneySpider Network	35
6.1	Introduction	35
6.2	Deployment and experiences	35
6.2.1	Experiences relating to the architecture of the system	35
6.2.2	Experiences relating to the detection methods used	37
6.2.3	Experiences relating to the behaviour of malicious web sites	38
6.2.4	Experiences using HSN WAPI	39

6.3	Current and future work	40
7	BlueBat	42
7.1	Introduction	42
7.2	Deployment and experiences	42
7.3	Current and future work	44
8	NoAH	46
8.1	Introduction	46
8.2	Deployment and experiences	47
8.3	Current and future work	54
9	WAPI and WOMBAT Workshop Scenarios	55
9.1	Introduction	55
9.2	Preliminaries	56
9.3	Investigation of a Banking Fraud	58
9.3.1	Malware identification	58
9.3.2	Infection analysis	61
9.3.3	The real culprit	64
9.3.4	Conclusions	66
9.4	Monitoring of our Own Networks	67
9.4.1	Searching for infections	67
9.4.2	Looking for similar malware samples	71
9.4.3	Looking more in depth at zief.pl	73
9.4.4	Conclusions	75
10	Conclusions	76
11	APPENDIX	79

Abstract

This deliverable reports the deployment of all types of sensors implemented in the WOMBAT project and includes descriptions of experiences with the sensors from several months of deployment and experimentation. The sensors that are deployed are the SGNET, HARMUR, Shelia, Paranoid Android, HoneySpider Network, Bluebat and NoAH. The early experiences show that the WOMBAT Project [10] is fulfilling our preliminary expectations about having powerful tools for collecting data. These data are useful for categorizing attackers and malware behaviors. Moreover our experiments reveal that the sensors can cooperate with each other, enriching in this way the information offered for analysis.

1 Introduction

1.1 Overview

The purpose of this document is to present the deployment of the sensors in the context of the WOMBAT project and the experiences that were acquired during their implementation and usage.

In the previous deliverable D3.2 (“Design and prototypes of new sensors”), we presented a detailed overview of the sensors design. After a few months of designing we were able to develop the sensors and to conclude with experiences of their use.

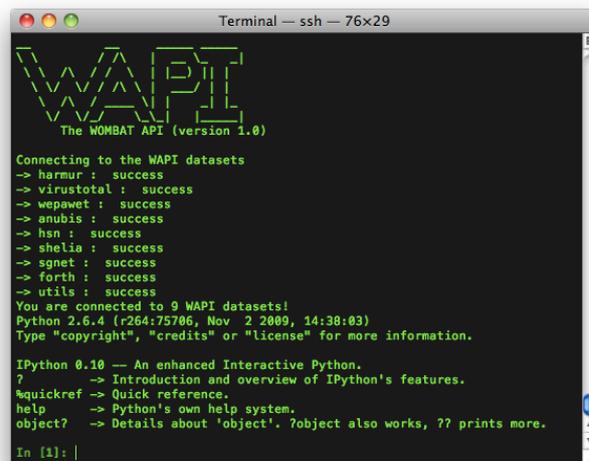
First, we present the deployment of SGNET (Section 2 in page 12), which is a distributed honeypot deployment for the collection of data on the evolution of Internet code injection attacks. It takes advantage of protocol learning techniques that address the previously introduced trade-off between the need to retrieve rich information about the observed activities and the need to reduce the resource and maintenance costs inherent in a distributed deployment.

HARMUR (Section 3 in page 20) (Historical ARchive of Malicious URLs), is an initiative aiming at the collection of detailed information on the nature, the structure and the evolution of Web threats. It positions itself as a consumer of the information generated by existing honeyclients.

In Section 4 (page 26) we present Shelia, a Windows-based intrusion detection system for the client side, originally developed as a design study in the context of the EU FP6 Noah project. The main idea behind Shelia is that it emulates a naive user: someone who will follow all links and open all attachments in spam email, and who clicks all links received via other means (say, instant messaging).

Next in Section 5 (page 30) we present a multi-level intrusion detection for smart phones in an architecture known as Paranoid Android. This project aims at protecting new smart phones. It uses what we called multi-level intrusion detection in previous deliverables. For this reason, we will discuss them together.

In Section 6 (page 35) we present HoneySpider Network, a honeyclient that is being developed under a joint venture called the HoneySpider Network project, together with GOVCERT.NL and SURFnet. The goal of this effort is to develop a complete client honeypot system, the HoneySpider Network (or HSN for short), based on existing state-

A terminal window titled "Terminal — ssh — 76x29" displays the WAPI client interface. At the top, the letters "WAPI" are rendered in a large, green, dashed font. Below this, it says "The WOMBAT API (version 1.0)". The terminal then shows a list of datasets being connected to, all with "success" status: harmur, virustotal, wepawet, anubis, hsn, shelia, sgnet, forth, and utils. A message states "You are connected to 9 WAPI datasets!". The terminal is running Python 2.6.4 (r264:75706, Nov 2 2009, 14:38:03). Below the Python version, it shows the IPython 0.10 help text, including options like "?", "quickref", "help", and "object?". The prompt "In [1]: |" is visible at the bottom.

```
Terminal — ssh — 76x29
WAPI
The WOMBAT API (version 1.0)
Connecting to the WAPI datasets
-> harmur : success
-> virustotal : success
-> wepawet : success
-> anubis : success
-> hsn : success
-> shelia : success
-> sgnet : success
-> forth : success
-> utils : success
You are connected to 9 WAPI datasets!
Python 2.6.4 (r264:75706, Nov 2 2009, 14:38:03)
Type "copyright", "credits" or "license" for more information.

IPython 0.10 -- An enhanced Interactive Python.
?                -> Introduction and overview of IPython's features.
?quickref       -> Quick reference.
help            -> Python's own help system.
object?        -> Details about 'object'. ?object also works, ?? prints more.

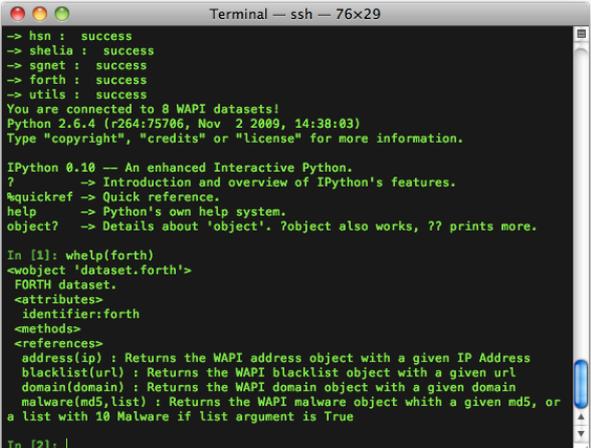
In [1]: |
```

Figure 1.1: WAPI Client

of-the-art client honeypot solutions and a novel crawler application especially tailored for the bulk processing of URLs.

Section 7 (page 42) presents the deployment of Bluebat. BlueBat is an experimental Bluetooth honeypot sensor. Bluetooth exhibits a number of security issues in various specific implementations of the stack. Viruses for mobile devices primarily rely on simple *social engineering* to propagate, sending copies of themselves to any device which comes into range through an OBEX push connection. Bluebat designed, in its first working prototype (BlueBat v.1.0), as an ad hoc device based on the GNU/Linux OS to collect malicious samples.

Finally, in Section 8 (page 46) we present NoAH's deployment. NoAH focuses on honeypots that listen to unused IP address space and analyze and/or interact with malicious traffic. The architecture of NoAH presents a flexible design for deployment and collaboration of honeypots. NoAH is not restricted to a single type of honeypot but tries to combine the good characteristics of low-, medium- and high-interaction honeypots. Its modular architecture permits the construction of a network of honeypots with minimal overhead and an affordable administrative one.

A terminal window titled "Terminal — ssh — 76x29" showing a successful connection to WAPI datasets. The output includes connection status for hsn, shelia, sgnet, forth, and utils. It then displays Python 2.6.4 and IPython 0.10 information. The user enters 'whelp(forth)' and receives a detailed help message for the 'forth' dataset, listing attributes like 'identifier' and methods like 'address(ip)', 'blacklist(url)', 'domain(domain)', and 'malware(md5, list)'.

```
Terminal — ssh — 76x29
-> hsn : success
-> shelia : success
-> sgnet : success
-> forth : success
-> utils : success
You are connected to 8 WAPI datasets!
Python 2.6.4 (r264:75706, Nov 2 2009, 14:38:03)
Type "copyright", "credits" or "License" for more information.

IPython 0.10 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's features.
?quickref -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object'. ?object also works, ?? prints more.

In [1]: whelp(forth)
<object 'dataset.forth'>
FORTH dataset.
<attributes>
  identifier:forth
<methods>
<references>
  address(ip) : Returns the WAPI address object with a given IP Address
  blacklist(url) : Returns the WAPI blacklist object with a given url
  domain(domain) : Returns the WAPI domain object with a given domain
  malware(md5, list) : Returns the WAPI malware object with a given md5, or
  a list with 10 Malware if list argument is True

In [2]: |
```

Figure 1.2: whelp function

1.2 WAPI

WAPI, the WOMBAT Application Programming Interface (API), aims at improving the ease of use of the various data sources that is available in the WOMBAT infrastructure. Each data owner is responsible for deciding what he is eager to share, in which format and to whom. WAPI provides the primitives to easily specify some sort of access controls. The role of the WAPI is to hide all implementation details from the programmer (naming schemes, querying methods etc.) and to provide her with an interface for retrieving and querying data across WOMBAT data sources.

The last year we have successfully implemented and tested the WAPI. The most important test occurred in the second WOMBAT workshop where the WAPI was used to investigate two real-case scenarios: (a) Investigation of a Banking Fraud and (b) Monitoring of Own Networks.

It is very easy to access each dataset. For example, if one wants to ask NOAH's dataset if it has any information about a specific malware (malware with md5 = bb8d199099a07e022fe03895d703fdda), they just has to write the below line of code:

```
malware = forth.malware(md5 = "bb8d199099a07e022fe03895d703fdda")
```

Figure 1.1 shows how a WAPI client looks like when it is successfully connected to harmur, virustotal, wepawet, anubis, hsn, shelia, sgnet and forth datasets. After the

connection with the datasets has been established, the client is ready to accept user input. Whelp function is responsible for revealing the information about what object types each dataset supports. Figure 1.2 shows the result of the whelp function in FORTH's dataset.

2 SGNET

2.1 Introduction

The collection of information on Internet malware scenario is a challenging task. The challenge arises from the need to cope with the spatial and quantitative diversity of malicious activities. The observations need to be performed on a broad perspective, since the activities are not uniformly distributed over the IP space. At the same time, the data collectors need to be sophisticated enough to extract a sufficient amount of information on each activity and perform meaningful inferences. How can the simultaneous need to deploy a vast number of data collectors be combined with the need of sophistication required to make meaningful observations? Addressing such a challenge is the ultimate goal of the SGNET deployment.

SGNET is a truly collaborative work within the WOMBAT project. As explained in the detailed design presented in Deliverable 3.2, SGNET incorporates technologies and research efforts from EURECOM, Symantec, VU Amsterdam, TU Vienna and Hispasec.

SGNET takes advantage of protocol learning techniques in order to address the previously introduced trade-off between the need to retrieve rich information about the observed activities and the need to reduce the resource and maintenance costs inherent in a distributed deployment. By using ScriptGen [13, 14], SGNET honeypots are able to model protocol conversation through a Finite State Machine (FSM) model and use such models to respond to clients for well-known activities. Whenever a new/unknown activity is encountered, SGNET honeypots are able to dynamically proxy the conversations to a honeyfarm, and take advantage of the real service implementation to handle them.

Figure 2.1 shows the main components of the SGNET deployment. SGNET is composed of multiple low-cost sensors whose FSM model is kept in sync by a central entity, the gateway. Whenever a new activity is encountered, SGNET honeypots require the instantiation of a new *sample factory* to the central gateway. The *sample factory*, based on Argos [15], acts as an oracle and provides to the sensors the required protocol interaction and, through memory tainting, detects and provides information on successful code injection attacks. Such information is used by the gateway to apply the ScriptGen algorithm and refine the FSM knowledge. After having seen a sufficient number of samples of

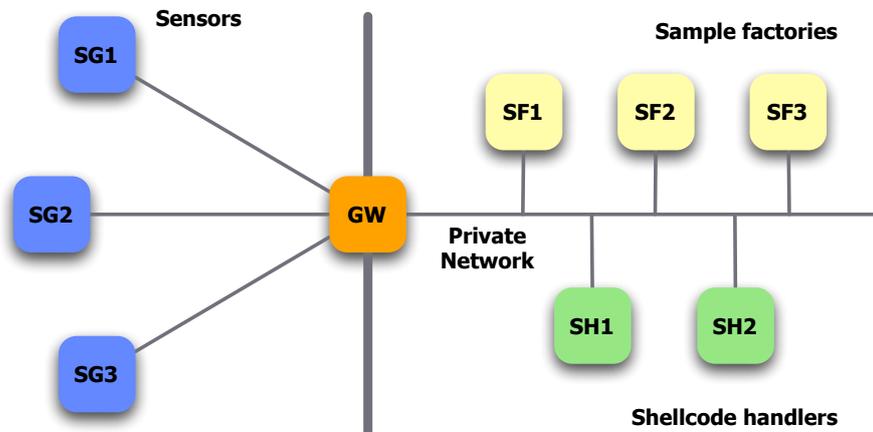


Figure 2.1: SGNET architecture

the same type of interaction, SGNET sensors are therefore able to handle autonomously future instances of the same activity leveraging the newly built FSM refinement.

The memory tainting information generated by Argos, combined with simple heuristics, allows SGNET honeypots to identify injected shellcodes. SGNET takes advantage of part of the Nepenthes [11] modules to understand the intended behavior of the observed shellcodes and emulate the network actions associated to it.

All the information collected during the interaction of the different SGNET entities is stored in a database, and fed to an information enrichment component described in detail in charge of adding additional metadata on the attacking sources, and on the collected malware. Among the different information sources, the most relevant to this work are the behavioral information generated by Anubis [12], and the AV detection statistics generated by VirusTotal [18]. Every malware sample collected by the SGNET infrastructure is, in fact, automatically submitted to these two services, and the resulting analysis reports are stored in the SGNET dataset to enrich the knowledge regarding the sample.

2.2 Deployment and experiences

The SGNET infrastructure has been fully implemented and gradually deployed in multiple platforms distributed over the whole IP space. The deployment has followed the strategy already successfully applied in the past by the Leurré.com project, and has



SGNET deployment

Query by Selection

Select	Group by
Generic parameters	
<input type="checkbox"/> Date Interval	From: 2009 11 6 To: 2009 11 20 Step: 1 DAY <input type="checkbox"/>
<input type="checkbox"/> Environment:	<input type="text"/> clear ---select---
<input type="checkbox"/> Destination address:	<input type="text"/> clear ---select---
Information on the attacker	
<input type="checkbox"/> OS:	<input type="text"/> clear ---select---
<input type="checkbox"/> Country:	<input type="text"/> clear ---select---
<input type="checkbox"/> ISP:	<input type="text"/> clear ---select---
<input type="checkbox"/> AV signature:	<input type="text"/> clear ---select---
Network parameters	
<input type="checkbox"/> IP protocol:	<input type="text"/> clear ---select---
<input type="checkbox"/> Destination port:	<input type="text"/> clear ---select---
<input type="checkbox"/> SG Path:	<input type="text"/> clear
<input type="checkbox"/> Snort alert set:	<input type="text"/> clear
Sequences	
<input type="checkbox"/> Compact Port Sequence ID:	<input type="text"/> clear
<input type="checkbox"/> Extended Port Sequence ID:	<input type="text"/> clear
Activity type	
<input type="checkbox"/> Backscatter only	
<input type="checkbox"/> Code injection only	
<input type="checkbox"/> Recognized shellcode only	
<input type="checkbox"/> Malware downloaded only	
Epsilon-Gamma-Pi-Mu model	
<input type="checkbox"/> Epsilon:	<input type="text"/> clear
<input type="checkbox"/> Gamma:	<input type="text"/> clear
<input type="checkbox"/> Pi:	<input type="text"/> clear
<input type="checkbox"/> Mu:	<input type="text"/> clear

according to Rate threshold

Figure 2.2: SGNET query interface

Location	Port 139	Port 445	Port 135	Port 80	Others	Total
US	616.5	89.3	639.3	20.0	65.3	1430.4
Poland	982.8	222.8	104.1	3.4	15.4	1328.4
France	619.3	372.8	155.7	55.7	20.1	1223.7
Ireland	556.4	545.1	33.0	52.3	13.9	1200.7
Belgium	546.9	44.3	105.9	61.3	26.9	785.4
France	154.8	177.2	219.8	10.3	14.6	576.7
Canada	322.4	17.5	187.0	1.9	16.9	545.7
UK	0.2	57.8	6.0	42.0	3.1	109.2
Spain	2.7	13.6	61.4	20.7	7.2	105.7
France	68.3	9.0	12.6	4.4	5.3	99.6
Australia	53.0	6.9	0.4	11.2	8.0	79.5
Australia	26.2	4.4	13.0	9.8	7.4	60.9
Greece	3.1	9.9	18.3	18.2	4.6	54.1
France	3.4	2.9	39.5	0.9	5.5	52.2
Germany	0.0	0.1	0.0	47.0	2.3	49.4
Ireland	4.9	5.3	8.3	2.7	19.5	40.7
Australia	11.4	6.2	1.7	13.3	2.9	35.6
Lithuania	13.5	2.9	2.8	3.8	10.1	33.0
Italy	7.7	1.6	0.0	11.2	3.7	24.3
Norway	1.4	5.6	1.1	2.5	5.4	16.0
Japan	0.0	0.0	10.0	0.0	0.0	10.0
US	0.0	0.0	0.0	8.2	0.4	8.7
Portugal	0.0	0.1	0.6	1.5	6.1	8.3
Italy	1.6	2.5	0.2	2.3	1.2	7.8
Total	3997.0	1598.0	1621.1	406.6	273.1	

Table 2.1: Average daily load (TCP sessions per day) for the most attacked ports

proposed win-win partnerships to any research entity interested in taking advantage of the information collected by the deployment. The partnership offers advantages to both parties: on the one hand, the partner gets full database access to the SGNET data, while on the other hand the partner is required to contribute to the deployment by installing a sensor. To avoid privacy and legal issues, all the participants have been asked to sign a non-disclosure agreement that prevents them from disclosing information on the identity of the attackers and on the identity of the other participants to the initiative.

As of the 1st of November 2009, a total of 37 institutions from all the 5 continents have joined the initiative and have started the process for the installation of a honeypot. At such date, 24 sensors have contributed to the deployment for at least one month.

Throughout its life, the deployment has observed a 186,315 distinct IP sources that generated a total of 3,808,760 TCP sessions, 64,331 of which resulted in being successful code injection attacks. 14,581 of these successful code injection attacks have been fully emulated by the deployment, and led to the collection of 9,350 distinct malware samples.

Table 2.1 provides an overview of the traffic load observed by the deployment for these sensors, and expressed in terms of average number of TCP sessions per day. It's interesting to see the considerable difference in the profiles of the different sensors: despite of the fact that all the sensors share exactly the same configuration and behave completely equally, different types of activities are observed and in different ratios. Activities that are globally dominant, such as the activity on port 139, disappear almost completely in a few platforms that are instead characterized mainly by other types of traffic, such as HTTP.

To allow the participants to monitor the status of their sensors and graphically visualize the collected information, we have offered them a simple Web tool that automatically generates requests to the SGNET dataset, and that interactively visualizes the results by means of a Java applet. The query interface, represented in Figure 2.2, allows to define a set of constraints over the visualized data, and choose the dimensions of interest for the visualization.

Figure 2.3 shows an example of the output of the tool, with a high level overview of the evolution of the exploit scenario observed by the SGNET deployment throughout the year 2008. The plot shows the weekly number of sources that have been witnessed performing successful code injection attacks against the deployment. Each plot line represents a different traversal of the ScriptGen Finite State Machines, and therefore represents a different activity type. Interestingly, the deployment observes a relatively stable amount of exploits for some high volume, highly visible activities. Such stable activities are then coupled with more bursty activities, that generate high activity peaks within a single week and seem to disappear subsequently.

An interesting example of such burstiness is represented in Figure 2.4. In this specific case, the deployment witnessed the sudden appearance in April 2008 of a new activity type on port 135, associated to a successful exploitation of the Microsoft DCE RPC service. The activity, previously unknown, triggered the sample factories and led to the generation of a new traversal in the ScriptGen FSM. Figure 2.4 represents the activity associated to that specific traversal, and breaks it down by country of origin of the attackers. While the initial wave of attacks in April 2008 comes primarily from Germany, France, Russia, Portugal and Spain, we can notice the rise of a less noisy and more steady rate of activities originated from the US that continues throughout the year 2008. This seems to suggest the reuse of the same exploitation code in what could possibly be a successive generation of the botnet, targeting a different portion of the IP space. The

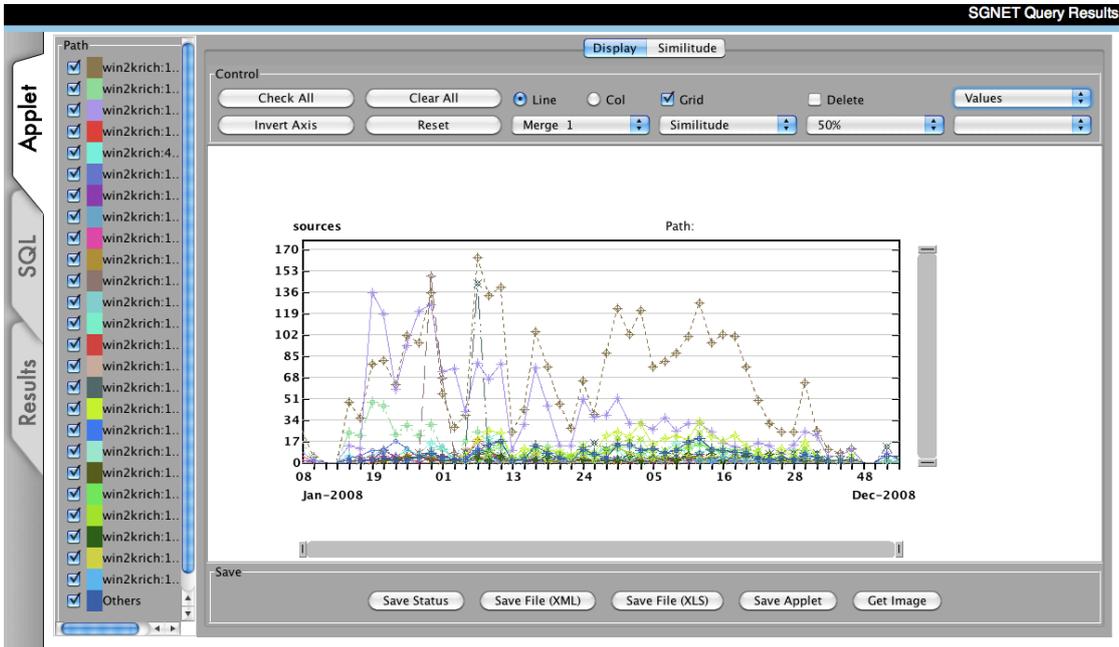


Figure 2.3: Exploit scenario evolution during 2008

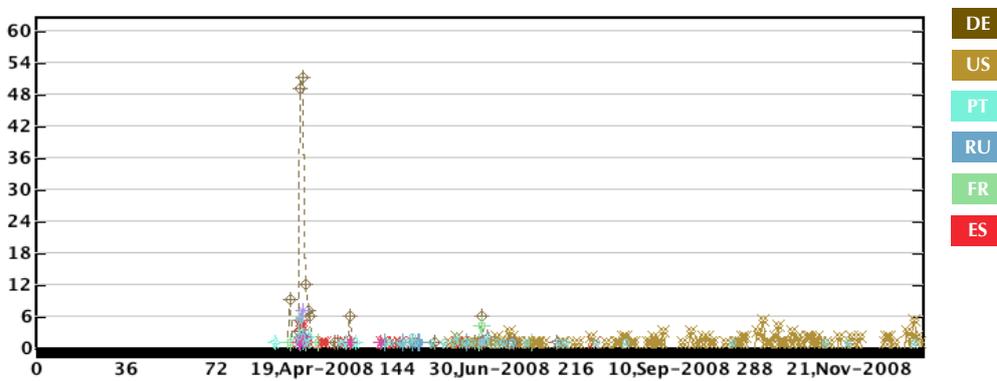


Figure 2.4: Generation of a new activity type, broken down by country of origin

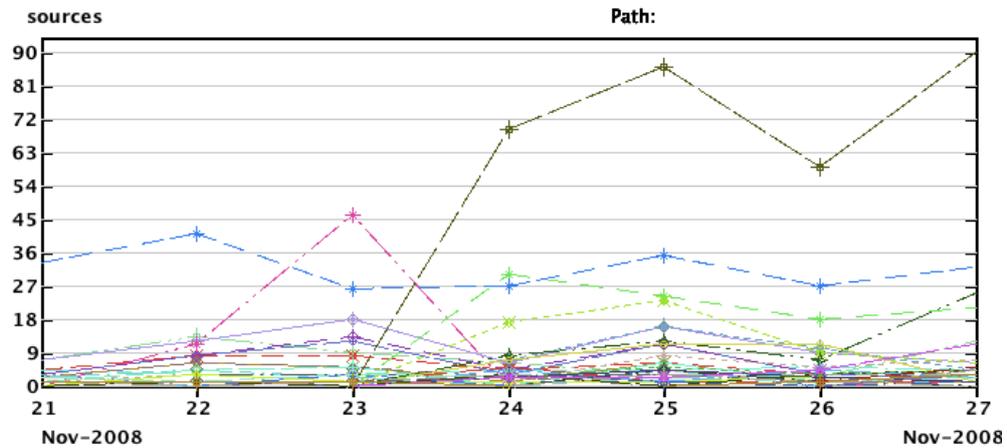


Figure 2.5: The raise of Conficker on the 24th of November 2008

shift in origin of the attackers is in fact coupled by a shift of the observing platforms: while the initial wave was witnessed mainly by a sensor located in France. The successive activity from US is witnessed by a honeypot hosted in the US. Despite the geographical distance of the attacking population and of its targets, the generated network activity is classified by the ScriptGen FSMs as identical. Such a simple example shows the value of the SGNET dataset in providing information on the modus operandi of the malware writers.

The ScriptGen learning process can be extremely helpful for focusing the attention of the security analyst in anomalous, and therefore interesting, events. We have seen in Figure 2.3 that most of the code injection attacks observed by the SGNET deployment are generated by stable and noisy activity types. An analysis of these activities and the malware associated with them shows that they are mostly associated to well known malware families such as Allapple [16]. The FSM classification performed by the SGNET deployment during its activity allows us to clearly pinpoint events associated to these well studied trends and focus instead on the more rare events associated to the detection of a new activity type. An example of such event is represented in Figure 2.5. Figure 2.5 represents the temporal evolution of the different FSM traversals in the week between the 21st and the 27th of November 2008. Differently from previous figures, Figure 2.5 shows the daily number of sources traversing all FSM paths and not only those leading to successful code injections. On the 24th of November 2008, SGNET generated three new traversals corresponding to different activities associated to the Conficker

worm, that received considerable attention from the press for the sophistication of its code. This new trend would have been impossible to identify through higher granularity statistics, such as destination ports, since it would have been hidden by other more noisy activities that are regularly appearing on a daily basis. The ScriptGen learning techniques employed by SGNET allow us instead to detect structural differences in the specific network conversations generated by this worm and associated to the first witnessed attempts to exploit the MS08-067 vulnerability.

2.3 Current and future work

The SGNET deployment is continuously expanding and is likely to continue to increase the number of participants to the initiative and the number of deployed sensors.

The return on experience in the deployment of the SGNET sensors has underlined its strengths, but also the weaknesses of the current deployment. More specifically, work can still be done in improving the ability of the deployment to study code injection attacks. As of now, only a fraction of the total number of potential exploits is correctly emulated by the infrastructure and leads to the successful download of a malware sample. The ability of the SGNET deployment to emulate vulnerabilities is strictly linked to the high interaction profiles supported by the sample factories. As of now, the whole SGNET infrastructure is based on a single sample factory profile, corresponding to an unpatched Windows 2000 system running IIS services. Exploits that do not target or work correctly on this specific configuration cannot be emulated by SGNET, limiting our ability to study them. But also, the emulation and identification of the shellcode are based on a set of heuristics that can potentially fail when facing new exploitation techniques. Joint work is being carried on by the WOMBAT participants to improve the generality of the shellcode handling algorithms in order to overcome these limitations.

3 HARMUR

3.1 Introduction

HARMUR, the Historical ARchive of Malicious URLs, is an initiative aiming at the collection of detailed information on the nature, the structure and the evolution of Web threats. In the recent years, we have witnessed a partial shift of attention from server-side attacks to client-side ones, with a specific focus on Web applications. According to [17], over half of the patched medium- and high-severity vulnerabilities in the second half of 2007 were browser and client-side vulnerabilities.

The interaction pattern of client-side attacks profoundly differs from that of server-side ones, and this difference is propagated to the detection methods. While server-side honeypots can be considered as “passive” components reacting to any connection attempts from randomly scanning attacking hosts, client-side honeypots (or *honeyclients*) need to actively scan the Internet to discover malicious sites. This is achieved often by crawling websites starting from a set of URL feeds or by acting as proxy for legitimate clients and analyze the generated interaction.

HARMUR positions itself as a consumer of the information generated by existing honeyclients. HARMUR has been designed and presented in D3.2 as a purely passive aggregator of information generated by third parties, ranging from honeyclients to standard Internet services such as *whois*. HARMUR combines together URL feeds, that provide on a regular basis streams of new, potentially interesting URLs worth being monitored, and Analysis modules that aim at generating information on these URLs. The whole system has been designed in order to allow the URL analysis to repeat in a periodic fashion over each of the monitored resource locators. This attention to the temporal evolution of the state of a URL and of its hosting site allows to study the dynamics of this specific category of threats, and better understand the correlation between their lifetime and its relation to operational network conditions. For instance, we want to understand whether malicious sites have ever moved along multiple hosting providers in order to maximize survivability, and the speed of these dynamics.

3.2 Deployment and experiences

A prototype of the HARMUR information tracker has been running since June 2009 and has collected information on the following site characteristics:

- **Norton safeweb information.** Detailed information on known threats generated by Symantec's Norton Safeweb initiative (<http://safeweb.norton.com>).
- **Google Safebrowsing information.** Blacklisting information generated by the Google Safebrowsing service.
- **DNS relations.** Analysis of the DNS relation between domain names, corresponding authoritative nameservers, and server IP addresses.
- **Whois information.** Registration information for the domain names.
- **Geolocation and AS information.** Information on the location of each name-server and HTTP server.
- **HTTP server status.** Information on the reachability of the web servers, and on their version as advertised in the HTTP headers.

In this period, HARMUR has collected information on a total of 10,120,758 distinct URLs belonging to 1,728,644 domains and hosted on 615,651 web servers. These domains can be broken down as follows:

- 62,502 domains are currently believed to contain security threats
- 19,676 domains have been associated in the past to security threats but now result clean
- 77,197 domains are not reachable
- 485,011 domains have not been analyzed by any honeyclient and have unknown security status
- 1,083,877 domains are not directly associated to any security threat

The information collected by HARMUR allows us to study and represent the relationships between the malicious domains and the physical servers on which they are hosted. Figure 3.1 provides an overview of the DNS relationships tracked by HARMUR for a specific threat class. Each point of the figure represents a domain name and is colored

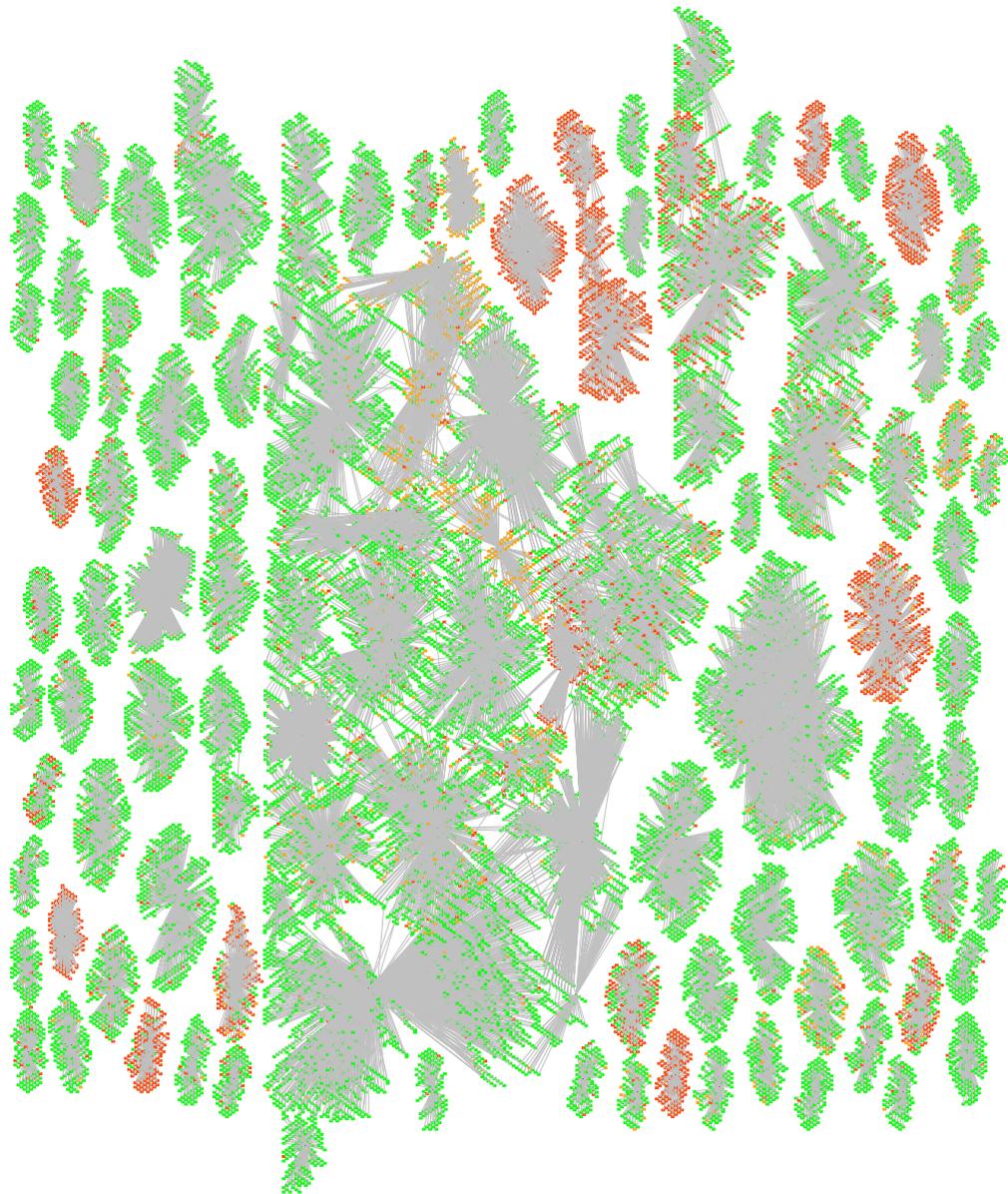


Figure 3.1: DNS relations for a specific threat class

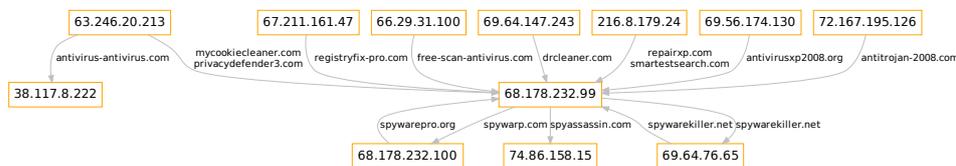


Figure 3.2: Domain expiration

in red if belonging to the threat class, in orange if belonging to another threat class, and in green if benign. The different domains are grouped and connected together by grey edges according to the DNS resolution of their names, that link them together to the same web server. Figure 3.1 shows the complexity of the problem of the identification of malicious domains, and underlines the challenges of protecting the web clients from potentially dangerous sites through blacklisting methods. More specifically, Figure 3.1 shows how both IP-based blacklisting and name-based blacklisting approaches are deemed to fail in the long run.

On the one hand, blacklisting malicious websites by blocking access to specific web server IPs is impossible. While we can clearly identify in Figure 3.1 cases in which a web server is solely used for hosting malicious content, and appears graphically as a completely red cluster, we have many more cases in which a server hosts both benign and malicious sites. Blacklisting these IP addresses would prevent users from visiting malicious domains, but would also prevent them from visiting benign domains that are hosted, for instance, by the same hosting service.

For the above reason, most of today's web security approaches follow the opposite approach and generate blacklists for domain names. Unfortunately, this approach is not effective on the long term. Registering batches of domain names, based on variations and permutations of a set of words, is an extremely easy process. This is exactly what HARMUR has enabled us to observe: a single webserver is often associated to hundreds of different names, probably automatically generated through the permutation of a few words. Because of this, the cost of maintaining reliable name filters is much higher than that of registering new ones.

In summation, Figure 3.1 represents the importance of the HARMUR dataset and its usefulness in building realistic pictures of the web threat space, that can help in better understanding the modus operandi of the attackers and the challenges of countering these threats.

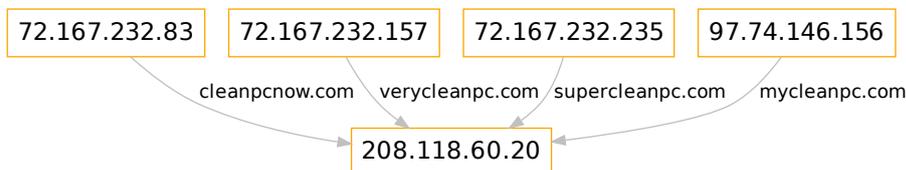


Figure 3.3: Domain dynamics

While the static picture enables us to have a first high level overview of the problem, the temporal evolution of the state of a domain can be extremely helpful in understanding the dynamics of the threat landscape. For instance, Figure 3.2 graphically represents the temporal evolution of a set of domain names. The edges represent modifications in the DNS resolution of a name, and therefore their transitions across multiple web servers. In the specific case of Figure 3.2, a set of domains apparently unrelated transits towards the same web server, 68.178.232.99. More detailed analysis for this IP address shows that it corresponds to a parking page for expired domains whose registration has not been renewed to the hosting provider. This gives us important information about these domain names: their maintainers consider them expendable, and have no interest in investing money to ensure their activity over long periods of time.

Figure 3.3 shows instead a slightly different scenario. The four represented domains, *cleanpcnow.com*, *verycleanpc.com*, *supercleanpc.com* and *mycleanpc.com* were registered after the activation of the HARMUR tracker and have moved from different known parking pages associated to different registration services to a single webserver previously hosting a single domain. This very simple example shows the value of the site dynamics in inferring relations among apparently unrelated phenomenon, and justifies the interest in carrying on this type of analysis.

3.3 Current and future work

Despite the relatively short life of the dataset, HARMUR shows very promising results in generating intelligence on the evolution and structure of web threats. This is accomplished by aggregating different information sources together in a central dataset and correlating them. The ongoing work on WAPI, the data-oriented API developed by the

WOMBAT project, is therefore crucial to HARMUR to allow an easy extension of the dataset by integrating other information sources developed within the project. Within the WAPI deployment, work is currently ongoing on the definition of WAPI interactions allowing to easily share information on web threats between HARMUR and the other WOMBAT-developed honeyclients, HoneySpider and Shelia.

4 Shelia

4.1 Introduction

Shelia is a Windows-based intrusion detection system for the client side, that detects attacks which arrive via e-mail, or through web browsing.

4.1.1 Shelia recap

The main idea behind Shelia is that it emulates a naive user: someone who will follow all links and open all attachments in spam email, and who clicks all links received via other means (say, instant messaging). Whenever Shelia detects a malicious website or attachment, it raises an alert.

What sets Shelia apart from most other client honeypots is the way in which it decides that something is malicious. Unlike most other systems, Shelia creates virtually no false positives¹ (although there may be false negatives²). Engrained in this design philosophy is that false positives are much more important than false negatives, since a high false positive rate means that you cannot act on alerts in an automated way.

False positives are avoided by detecting intrusions not by looking at changes to the file system after visiting a website (a common way in such honeypots), but by tracking who calls the sensitive operations. More precisely, whenever a call is made to change the registry, the file system, or network activity, Shelia tracks whether the call is coming from an area that is not supposed to contain code. If so, it raises an alert.

The other design goal is that it should be easy to manage. For instance, it should be as trivial as sending email to have Shelia check certain links or attachments.

Since Shelia has changed considerably since the previous deliverables, and since the changes are mostly due to deployability, we will use this deliverable to explain how the current version of Shelia improves over previous ones.

¹A false positive is another way of saying mistake. As applied to the field of anti-virus programs, a false positive occurs when the program mistakenly flags an innocent file as being infected.

²A false negative is the opposite of false positive. This happens when a malicious file is flagged as benign.

4.2 Deployment and experiences

As of the summer of 2009, Shelia is in active (production) use at the Vrije Universiteit of Amsterdam. Initially, it was used to process a large amount of older spam. The problem with older spam is that many of the malicious sites that the emails link to no longer exist. Next, we had Shelia process thousands of URLs obtained from Symantec. Currently, Shelia processes a regular stream of spam from the Vrije Universiteit Amsterdam: all rejected email of the computer science research network. Since then, Shelia has checked hundreds of thousands of URLs and attachments, uncovered hundreds of malicious websites and downloaded many different malware samples.

More precisely, we finished checking a large number of suspect URLs in the beginning of November. At that point Shelia had 250 malicious sites in these 45759 (older) URLs, 61 of which downloaded malware by means of a drive-by-download, 14 of which were unique. After that time, we initiated Shelia's new phase where we used it to process the department spam.

Shelia is available via the WAPI to all partners in the consortium and also to external parties. Finally, Shelia is currently being evaluated by GovCERT in the Netherlands.

4.3 Current and future work

In recent months, the Shelia architecture has witnessed a major overhaul. As a result, the system is more stable, more flexible and less tied to particular software or mode of operation. For instance, the previous version of Shelia (described in deliverable D3.1), required OutlookExpress and POP to access email. Neither are necessary in the current version, illustrated in Figure 4.1. In this section we summarise the changes:

Improved input handling The old version of Shelia depended on OutlookExpress to obtain spam messages from a POP server. Nowadays, the Shelia inputs are retrieved from a data base. Any technique that can fill the database is compatible with Shelia. For instance, we have an IMAP mail client that accesses an account, strips the URLs and attachments and enters them in the data base. But we also have a method that simply takes a list of URLs to check in a file and uploads these in the database.

The database entries have a priority. The highest priority entries will be checked first. If no explicit priority is specified, a static priority is used, whereby attachments are checked first. This effectively allows us to push urgent objects into Shelia without waiting for a long back log.

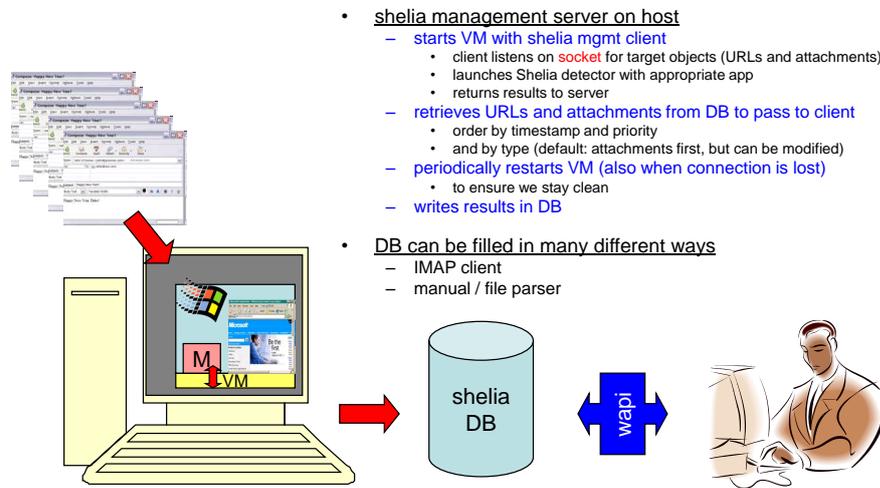


Figure 4.1: Shelia architecture

Better control and safety Shelia runs in a virtual machine like Qemu and is restarted every n checks to prevent infections not detected by Shelia from causing harm. Moreover every individual check is aborted after t seconds (by default: $t = 40$).

Hardened access We have attempted to redesign the interaction of the guest OS and the host OS in such a way that malfunctioning or malicious guests cannot hang the system.

Alert database and WAPI Whenever Shelia detects an exploit, it performs extensive analysis (which API calls does the code make, what is the payload of the attack, etc.) and stores everything in a structured fashion in a database (in the old version everything was simply dumped in a log file). Moreover, we have unlocked the Shelia database by making it accessible via the WAPI for all clients with the right credentials.

Future plans for Shelia include merging it with Argos to increase the scope of detection (some of the attacks detected by Argos are not covered by Shelia). To do so, we will have to rework the analysis of Shelia in the Argos detector. Doing so is a substantial effort, as Argos currently does not execute even a single instruction of the attacker's code. This will have to change if we want to be able to download the malware in the same way as is currently done by Shelia.

Several parties have expressed interest in Shelia, either to run a copy of Shelia themselves, or to contribute data for Shelia to process. While the Shelia code is online, proper documentation and even a detailed installation guide are still missing. As man power is limited, we will focus, for the near future at least, on helping knowledgeable users (e.g, GovCERT, and NASK who have expressed interest) to install Shelia, while others are encouraged to submit suspicious emails and links.

5 Paranoid Android and Multi-level intrusion detection

5.1 Introduction

Paranoid Android is our project that aims at protecting new smart phones. It uses what we called multi-level intrusion detection in previous deliverables. For this reason, we will discuss them together. The Paranoid Android architecture is depicted in Figure 5.1.

In the Paranoid Android project, we propose to *outsource smartphone security checks to the cloud* for providing high-grade security assurances without greatly impacting the device’s performance and battery life. Off-loading security checks provides more and cheaper processing cycles, allowing us to apply even very costly techniques such as dynamic taint-analysis¹. Furthermore, *multiple security checks can be applied in parallel* to achieve a broader detection scope (e.g., a combination of anti-virus scanning, taint analysis, system call monitoring, etc).

To achieve our goal, we replicate the state of running mobile devices to security servers in the cloud. We do so by starting with an exact same image and conveying all sources of non-determinism on the phone to the server in the cloud. In earlier deliverables, we have referred to such architectures as ‘multi-level intrusion detection’. At the moment, we have several subprojects that aim for multi-level intrusion detection, but since Paranoid Android is by far the most mature, we will limit our discussion to this example.

Execution traces can be long and typically contain a lot of data (the system calls and their arguments and results, signals, scheduling, etc.). Clearly, there is a risk that transmitting lengthy execution traces in itself places an unacceptable burden on the phone’s CPU and battery. By aggressively pruning the trace and a host of optimisations, we manage to reduce the performance, size and power impact of Paranoid Android to less than 2 KiB/s² and approximately 7% battery lifetime reduction for even the most heavy-weight tasks on the phone.

¹Dynamic taint analysis is a very computationally expensive detection technique that cannot normally be applied to production systems due to its overhead of one or several orders of magnitude. So far, taint analysis is only applied in honeypots.

²States ratio (KiB/s = 1024 Bytes / second)

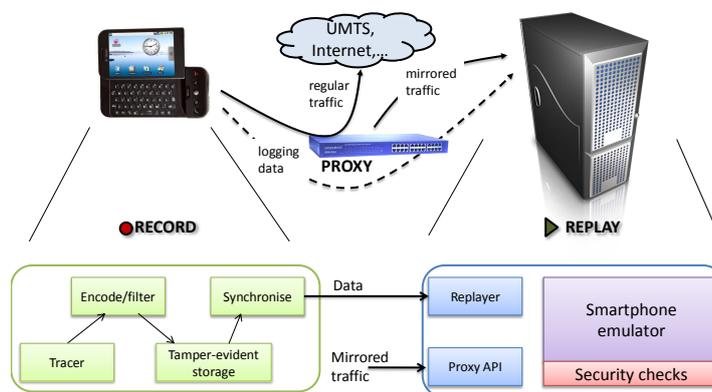


Figure 5.1: Paranoid Android architecture overview

5.2 Deployment and experiences

We implemented the Paranoid Android architecture, depicted in Figure 5.1, for HTC’s Android G1 phones. As the status of our system is still mostly research, deployment focuses on the phones of the researchers, although we have started experiments with protecting phones beyond normal lab conditions. For instance, researchers have applied Paranoid Android’s tracer on the phone that they use in practice. However, we still need stability and scalability tests before the system can be deployed at scale. Some initial results are shown in Figures 5.2 – 5.4.

Our traces with actual users show, not surprisingly perhaps, that mobile devices are mostly idle, or used for voice communications. This is good for the architecture, as it means that a single security server may support many clients (in our simulation experiments, we were easily able to support a hundred phones on a single server). A typical plot of the amount of data that is generated over time is shown in Figure 5.2. Meanwhile, Figure 5.3 shows that the data generated when performing such tasks is negligible, with an average of 64B/s for idle operation, and 121B/s when performing a call. Even when performing more intensive tasks, such as browsing or listening to music, the tracer generates less than 2KiB/s. For instance, 5 hours of audio playback would generate about 22.5MB of trace data. This shows that the trace is small enough to be stored locally on smartphones, which already offer relatively large amounts of storage (the iPhone 3GS comes with 32GB of storage).

Paranoid Android imposes two types of overhead on the phone. First, it consumes additional CPU cycles and thus incurs a performance overhead. Second, it consumes more

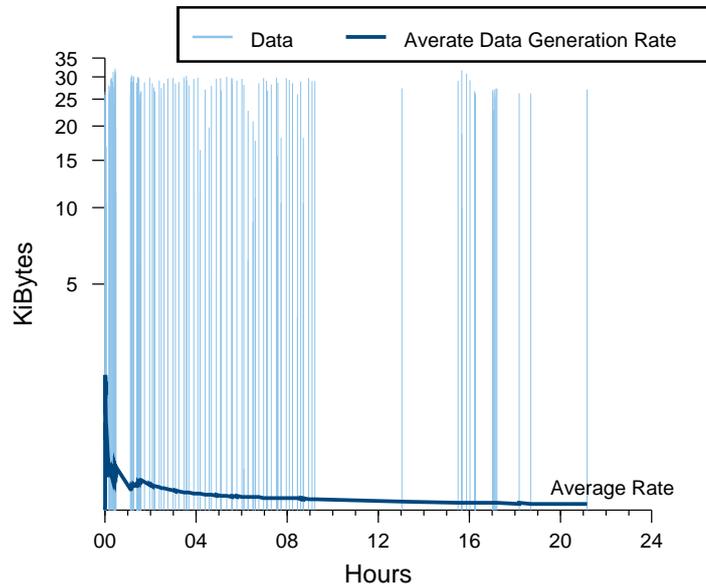


Figure 5.2: Data generated by Paranoid Android running for a day: the light coloured lines represent blocks of data generated by the tracer. Their bursty nature is due to the compression library, which buffers data to increase the compression ratio. The darker line near the x-axis represents the average data generation rate in time.

power because of the increased CPU usage and the synchronisation with the server. To quantify these costs, we monitored the device’s CPU load average, and battery capacity, while randomly browsing a set of URLs. We performed this task natively as well as under Paranoid Android, and show the results in Figure 5.4.

5.3 Current and future work

Current work focuses on making Paranoid Android robust, and implementing a low-threshold release. In our initial release, we will focus on individuals who want to protect their own phone, but who are willing to accept a delay of up to a day in detecting intrusions. In that case, we do not need an infrastructure that involves carriers. Instead, we can keep everything local to the phone.

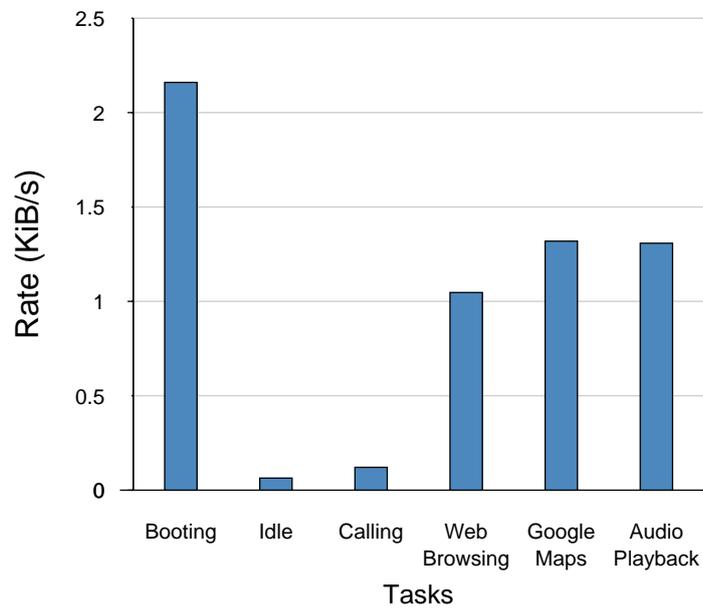


Figure 5.3: Average data generation rate, when performing various tasks

When the prototype proves to be mature, we will make it available for early testing by external parties.

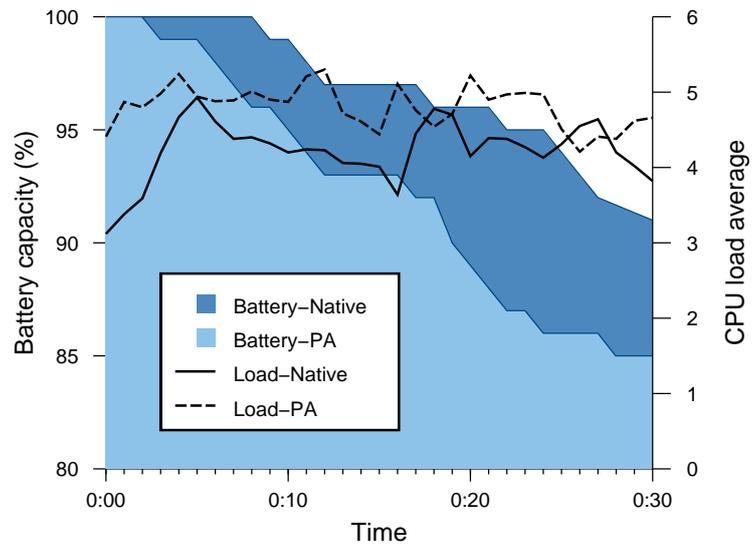


Figure 5.4: Battery consumption and CPU load average when browsing on the HTC G1 phone. We compare native execution, with execution under Paranoid Android. Note that the y-axis on the left starts at 80%

6 HoneySpider Network

6.1 Introduction

This section describes the experiences acquired during the deployment of the HoneySpider Network system at NASK. The HoneySpider Network is a client honeypot system that aims to discover malicious Web sites, in particular those that perform drive-by download attacks. It consists both of low interaction crawlers and high interaction honeypots, that run real browsers inside VM images. The system is integrated with WOMBAT through the WOMBAT API (WAPI). In this document, we describe our experience with the HSN WAPI and our experiences and lessons learnt in deploying HSN. We currently have multiple instances of HoneySpider running on the NASK network.

6.2 Deployment and experiences

Our experiences using the HoneySpider Network system can be broadly split into 4 categories that are determined by:

- the architecture of the system
- the detection mechanisms used
- the behaviour of malicious sites
- HSN WAPI

6.2.1 Experiences relating to the architecture of the system

The architecture of the HoneySpider Network (HSN) system is designed to be modular and scalable. This means that the system can consist of a central manager which manages multiple low interaction crawlers and multiple high interaction crawlers, spread across many physical or virtual machines. Data is stored in a distributed fashion, which while improving the scalability of the system, as not all information is stored in a centralized database, does make its retrieval later on more complex.

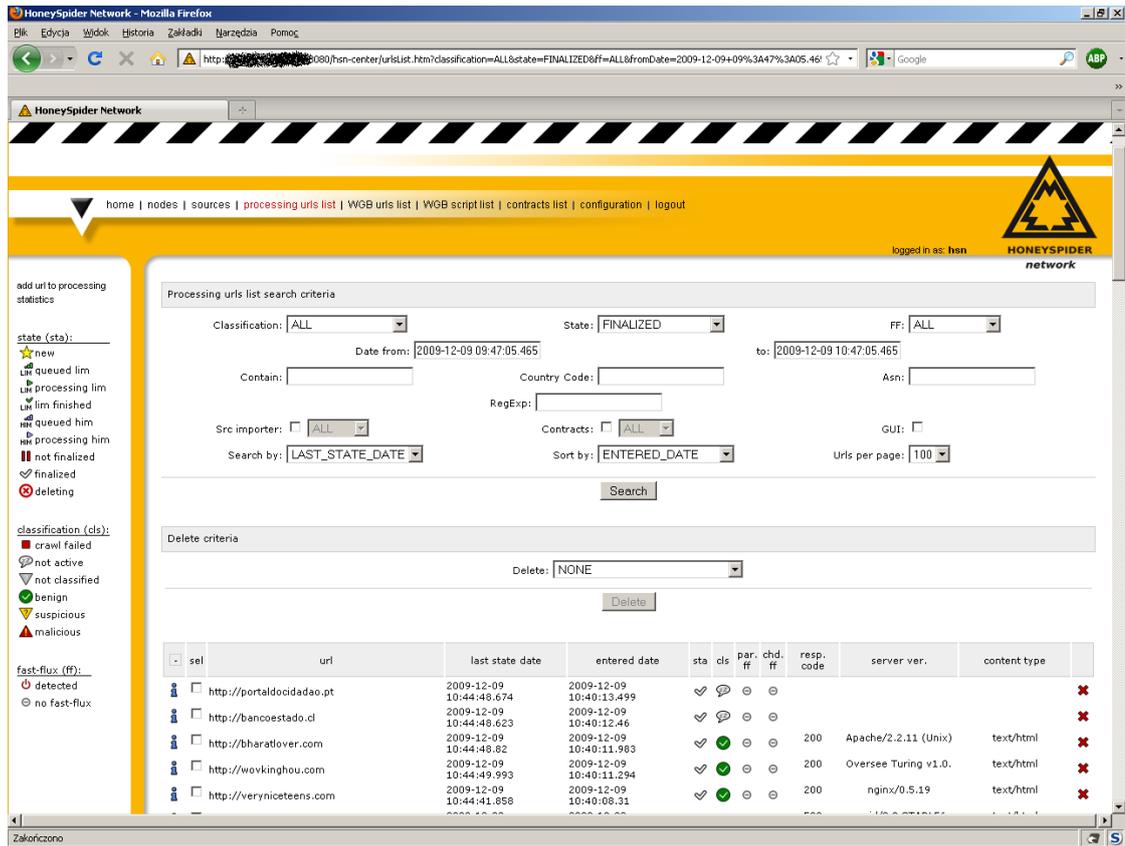


Figure 6.1: Screenshot of the HoneySpider Network GUI

The HSN system is designed not just as a service to others, as is the case in most of the other WOMBAT sensors, but as complete software that can be deployed by itself, enabling not just one instance of HSN but many, each consisting of as many low and high interaction crawlers as the installing party likes. The main target group of this software are CERTs that do not operate for profit. These CERTs are able to receive a copy of the HSN system for free if the three main parties behind the project, NASK, GOVCERT.NL and SURFnet unanimously agree. Sharing the code with others increases demands in terms of general user friendliness, not just of the system itself, but installation-wise as well.

During deployment of the HSN, we discovered that in general the system is easy to install. The main problematic area is the high interaction component of the system, based around the Capture-HPC solution. Capture-HPC uses VirtualBox as a guest system to check for malicious changes in Windows during visitation of the suspect web site. As each installation carried out by a new organization requires a setup of a Windows image, this can be quite time consuming. Also, the system parameters (memory, processor power) affect the way Capture-HPC behaves, meaning that tweaking of some Capture-HPC parameters is often necessary. Furthermore, setting up virtualization software is not trivial and requires some experience. We found that users installing the software had problems in this area, despite being provided with detailed instructions.

6.2.2 Experiences relating to the detection methods used

HoneySpider uses both low and high interaction client honeypot solutions. Its primary goal is to examine a large dataset of URLs and scan each one, to detect whether a web site is malicious or not, focusing in particular on drive by download mechanisms. A secondary issue is the ability to download the malware provided.

We found that the two most useful detection heuristics used by the low interaction system are the a) Naïve Bayesian algorithm applied to ngrams generated from JavaScript and b) the heuristic that is responsible for detecting constructs of JavaScript that are interpreted differently in different browsers. In a), we discovered that the training data set needs updating every couple of months in order to be effective at detecting malicious web sites. We also discovered the need to provide the capability of black and whitelisting JavaScripts through their MD5 hashes to improve detection effectiveness and lower false positives. In particular, the increasing prevalence of usage of JavaScript libraries, such as jQuery, makes it more difficult to detect code that is obfuscated for malicious purposes (our solution classifies obfuscated code as suspicious). We also found that obfuscated JavaScript has become more complex in that many different HTML elements of a page need to be parsed for successful deobfuscation, meaning that the DOM support in the

crawlers needs extensions. In b), the heuristics check for the use of such functions as `argument.callee.toString()` or different `try {} catch () {}` constructs that are known to be interpreted differently in IE and Firefox. These are used to determine what browser to target and to avoid deobfuscation, as most analysts analyzing sites tend to use the Firefox engine to handle JavaScript. We found this method to be very effective at identifying suspect sites, delivering no false positives.

Different problems were faced by our high interaction system, which is based around a modified version of Capture-HPC. Modifications were primarily made to improve stability of the software, its logging mechanisms, and to facilitate a move away from VMWare to VirtualBox. This allowed for easier, more reliable integration with the HSN framework. However, the key problem area is the design of the Capture-HPC detection model. System calls invoked during visitation of a web site are captured, allowing for the monitoring of file, process and registry changes. These changes are then compared against exclusion lists (lists that specify what changes are good or bad) to identify potential malicious web sites. In practice, exclusion lists have to be adjusted individually to every Windows image, depending on its version and the application versions installed. Exclusion lists are cryptic and very long making them difficult to create and maintain. Whenever any file, process or registry change does not match an exclusion list entry, Capture-HPC flags a site as malicious. This binary decision system is insufficient as from our experience Windows tends to behave in a very non-deterministic manner, making false positives a serious issue. Apart from making sure that exclusion lists undergo long periods of testing, we correct some of the verdicts, by flagging them as malicious only log changes that are either above a certain threshold or contain at least one file system modification. While imperfect, this does lower the false positives, which are in our opinion the biggest obstacle in performing large scale reliable assessment of Web sites. We are however, working on developing more sophisticated algorithms that will be able to make a better assessment of whether a certain Capture-HPC log file consists of bad modifications or not. Finally, we found it to be problematic at times to extract malware from sites, as quite often older versions of Windows XP tended to crash before we were able to transfer files outside a guest. We are attempting to solve this issue by introducing a shared folder which is mounted from the host, that stores all system modifications in real time, eliminating the need to transfer files from guest.

6.2.3 Experiences relating to the behaviour of malicious web sites

In terms of the behaviour of malicious web sites, as explained already, we are observing more sophisticated uses of different elements of an HTML page to decode JavaScript. Some of the methods employed appear to need manual analysis. Another issue is the

use of fast flux networks in URLs embedded in web pages, which sometimes results in different content being served at different times, making a malicious-benign verdict unreliable. We partly solve this issue by adding a fast-flux assessment of each encountered URL. This is done in real time. URLs that are found to be fast-flux are then flagged appropriately, which gives a systems operator indication that a benign verdict may not necessarily be true. Blacklisting of our crawler IPs is a problem observed as well. It happens quite often in the case of fast-flux urls that are to be analyzed. As these often point to a central location, it is quite easy for the miscreant maintaining the attack infrastructure to observe sequential queries to a fast-flux URL, characteristic of web crawlers. To deal with this case, we make sure that every instance of our web crawler uses a different IP exit address, especially when a URL is checked first by a low interaction crawler, and then by a high interaction machine. However, this does not prevent us from getting blacklisted in the long run. A possible solution is the creation of a large set of proxies by the CERT community that can be used by the crawlers at random or using TOR. The first case requires organizational work within the CERT community, but is not infeasible. In the TOR case, many exit nodes are known anyway and the general slowness of the network also makes it less useful. Another hindrance to the analysis of web sites that we observed is the requirement to use proper referrer headers in order to get served a malicious page. For instance, viewing a web page after clicking on the results of a Google search query may deliver an exploit, but accessing the web page directly may not. There appears to be no easy solution to such a problem, other than attempt to retry sites with different referrer headers.

6.2.4 Experiences using HSN WAPI

The first application of HSN WAPI was the demo during the second WOMBAT workshop. The participants were asked to follow a realistic scenario, investigating an infection. The results were fully satisfactory – the part of the scenario using the HSN dataset was not a problem for the participants. The structure of the HSN WAPI was clear enough and provided the necessary information. From the technical point of view there were no problems either. The test proved that HSN WAPI is free from serious bugs, but more importantly it confirmed the value of having a unified interface to different datasets – getting the information requested in the scenario without WAPI would be a very difficult and time consuming task.

After the workshop the HSN WAPI was left available to workshop participants and is still open. The stability of the HSN WAPI can therefore be considered sufficient, although it should be stressed that during this time it never reached a high load. The most important aspect of this test is the fact, that it was performed while the HoneySpider

Network was being actively developed. All the installations were in fact testbeds, often reinstalled or moved. Providing a service from a single, constant URL was possible by using the WRAPI – a Python-based WAPI proxy implemented using the generic server developed in WOMBAT.

Apart from these tests, the HSN WAPI has already been used in practice. When there appeared a need to allow interaction between the Arakis early warning system and the HoneySpider Network, the natural approach was to use a remote API. Since WAPI was already stable, we decided to use it. The conclusions from the implementation of the interface between the two systems are mostly positive. Using the SOAP protocol makes it possible to connect systems using completely different technology – in this case WAPI implemented in Java was accessed from PHP scripts. Some small incompatibilities were caused by an old version of the PHP SOAP library and have been resolved by a simple extension of the WAPI. We also found that we were able to get all the required public information using just the available WAPI services.

Since the internal data exchange between the two systems involved some data that we did not consider public, while the WAPI was designed as a public interface, we had to extend it. The changes were implemented as an additional method of the standard object Dataset and a new kind of object. We also used this occasion to implement some of the operations possible using existing WAPI services in a new, specialized, but also faster way. The changes were easy to implement, proving the extendability of our WAPI. At the same time this was another proof of the usefulness of the WRAPI. There is no need to support two separate versions of the WAPI. The Arakis interface uses the extensions implemented in the Java-based HSN WAPI, but they are not available through WRAPI, so the external users only have access to public data. The advantage of this approach is that the restricted extensions are hidden by omission, not active filtering – the new services are simply not registered in the WRAPI. This makes accidental publication of restricted information improbable.

In summary, the HSN WAPI is easy to use, fully functional and effective both when used interactively and as an API for programmers. It is also stable, safe and easily extendable.

6.3 Current and future work

We are currently working on improving issues relating to the stability of the system and in improving the performance of our crawlers. Future work includes redesigning of our low interaction architecture in order to be able to better handle and detect attacks that involve Flash objects and PDF files. We also plan extensions to our DOM implementa-

tion, so that we are more often able to get access to exploits and malware through low interaction machines. We intend to experiment with Shelia as a possible replacement for Capture-HPC, as Shelia's design is less likely to introduce false positives when assessing Web sites. Enhancements to the GUI are also being considered.

7 BlueBat

7.1 Introduction

BlueBat is an experimental Bluetooth honeypot sensor. As discussed in the previous deliverable D3.1 (“Infrastructure design”) [3], Bluetooth exhibits a number of security issues in various specific implementations of the stack. Such attacks are very well described on the website [9], and they allow different degrees of data access (from the agenda to any file on a vulnerable device), 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.

We designed BlueBat, in its first working prototype (BlueBat v.1.0), as an ad hoc device based on the GNU/Linux OS to collect the samples. We made use of the official Bluetooth stack implementation named BlueZ [2]. Specific utilities allow to perform device configuration, scanning and information gathering. We created a Python software, using the `pybluez` [7] package, to glue such utilities together, along with the `gpsd` [5] GPS daemon, and Colin Mulliner’s secure OBEX server [8]. We used the latter because of his security option (chroot, privilege separation), and of the possibility to control its behavior via a Python script. Basically, BlueBat. Honeypot opens an OBEX server modified to accept any incoming file transfer. In parallel, we perform a continuous scanning for devices, and we fingerprint the ones we find, using `pybluez`. We also use `gpsd` to log the position data for each activity, to support mobility. The script gathers the data and pushes it in a MySQL Database, correlating the results.

We used Asus EEE PCs as the basic platform, plus a combination of antennas as described in D3.2[4].

7.2 Deployment and experiences

Besides testing the antennas combination as described in D3.3, we deployed the devices in several locations in Milano. A first test was made by placing a long range honeypot on a street for several days . We also tested, for several hours each, various locations in our own University, in the underground and the Duomo square, using appropriately

concealed and unobtrusive hardware. During all these tests, no files were transmitted to the honeypots (except the test ones). We are currently discussing a semi-permanent placement of some of the honeypots in several high-affluence positions in Milan. As far as the number of observed “discoverable” devices, in the Duomo square location we reached over 500 unique devices per hour on average, from a single observation point equipped with a 6 dBi omnidirectional antenna.

In parallel to these tests, we used two cellphones as portable honeypots for a continuous time of 6 months. These devices were handed out to voluntary students, traveling in Europe in the context of Erasmus programme, which brought them along on several train travels, and also in various airports, in various nations (Italy, Austria, Switzerland, Spain), on board various trains and in airports. A total of 3 files were received. One, named `sarah.jpeg`, contains a photograph of a girl, while the second turned out to be a video promoting a leading brand of sportswear and footwear. It must be noted, though, that in order to actually receive the latter, the bearer of the honeypot phone had to willingly stop near the marketing totem and wait for the download to end, further stressing the unreliability of this communication channel. A third one is a `.sis` file (`485zp6x6.sis`, an executable for the Symbian platform), which could be malicious but was incompletely received and thus is difficult to identify.

While a complete explanation of the reasons for this low number of received files is beyond the scope of this report and will be the subject of more extensive research, some data are already clear. First, the honeypots (the fixed ones as well as the portable ones) observed a large number of devices in conditions which were in many cases optimal for transfers (i.e., users in crowded places and not moving). Mobility can lead to aborted file transfer (as was the case with one of the files we received), but those are logged. The only explanation which seems reasonable is that the actual prevalence of Bluetooth self-propagating worms using OBEX is extremely low (but not zero, as shown by the `.sis` file which is, with every probability, a malware specimen).

Field tests also revealed some unexpected issues in our original design: correlating scanning data and data obtained by the honeypot is a good idea in theory but difficult to realize in practice, as device scanning is very slow, consuming up to 5 minutes for a single pass of scanning using only a standard Class 1 dongle. Using an Aircable dongle with a 9 dBi omnidirectional antenna such a scan may take up to 15 minutes, trying to lock on almost out of range devices. Unluckily, this cannot be solved by sampling, but some modifications could be introduced in the low-level drivers to shorten timeouts for scanning operations (trading off completeness for effectiveness). During this timeouts, the scan process doesn't see other devices which may have entered and exited the study zone. This makes scanning substantially useless in crowded zones when a powerful antenna is in use. So we resorted to using the most powerful antennas just for running

the OBEX server, and less powerful ones for additional scanning and tracking.

Another unexpected result was that, actually, the human body (even the body of the device owner can be enough) is able to shield Bluetooth signals. This, in crowded areas, makes trying to enumerate devices difficult, let alone trying to receive a file. Therefore, a dense crowd will always limit the effectiveness of long range honeypot solutions, making the placement of a higher number of shorter-range sensors much more efficient.

Placement of the sensors turns out also to be of paramount importance. Density of devices varies wildly, and population movement is also important: while any touristic place such as the Duomo, train stations or airports have a crowd of people passing by, some places such as metro stations have the additional advantage that people move around slowly, or do not move at all: this also limits the issues with the “human shield” effect. Besides metro stations, entry/exit of attractions or exhibits are other good places.

7.3 Current and future work

The experiences with the early deployments gave us insights for two developments of our research.

Firstly, since the prevalence of OBEX transfers seems so low, we decided to expand our honeypot to also handle other Bluetooth interactions. A new version of Bluebat (v 2.0) has been developed and is currently being deployed among the partners.

Bluebat v2.0 has been completely redesigned from scratch. It is written in C, and it listens to over 30 RFCOMM channels, as well as the first 254 L2CAP channels, which are the most used by attackers while exploiting a device. The OBEX transfer honeypot itself has been reimplemented from scratch, in order not to depend on python and to be more flexible in handling data.

Bluebat 2.0 separates a client (sensor) and a server (database) component. The sensor runs in a single device (at the moment a Linux PC sensor), and it collects data, as stated, regarding all RFCOMM connections, L2CAP channels, and OBEX transfers. All data are stored in XML format. Binary data are encoded in base64 for storage. At regular intervals, if the sensor is connected to a wired or wireless Internet link, it pushes data to a server over TCP port 5168. An alternate mechanism dumps data over a USB key.

The server, besides collecting data over TCP or from a USB key, stores it in a MySQL database and offers WAPI interrogation interfaces via XML SOAP over TCP port 8080.

The Bluebat client has been designed to be lightweight and have few dependencies, in order to be executed in low power netbooks and smartphones. The new design also makes it easier to scale the honeypot as required by the global approach of WOMBAT. A self-installing Bluebat client CD ROM will help foster distribution of the sensors.

In the second place, since our Bluetooth honeypots on mobile phones seemed to produce interesting results, we are currently testing the port of the Bluebat client on Android [1] devices. Producing a reliable honeypot for mobile phones running closed stacks proved impractical, as we reported in D06 [3], because of the J2ME framework implementation. Each service on a Bluetooth device must be registered in a Service Discovery DB (SDDB) on a certain UUID. There are some standard numbers, equivalent to the “well-known ports”: for instance, OBEX push is commonly associated to UUID 1105. Therefore, our software must be registered in the SDDB under that same UUID. But the phone manufacturer OBEX service is already registered with this UUID, and it has priority: if a request reaches the device, it is the manufacturer server which answers it. Our honeypots on mobile phones used hacks that made them not portable and not suitable for distribution to a wide population of users. Android, on the other hand, makes it possible to develop a version that can be deployed in a distributed fashion.

8 NoAH

8.1 Introduction

NoAH is focused on honeypots that listen to unused IP address space and analyze and/or interact with malicious traffic. The architecture of NoAH presents a flexible design for deployment and collaboration of honeypots. NoAH is not restricted to a single type of honeypot but tries to combine the good characteristics of both types. Its modular architecture permits the construction of a network of honeypots with minimal overhead and affordable administrative overhead.

Honeypots in NoAH are deployed inside the “NoAH core”, the center of decisions. Apart from honeypot deployment, services like automated signature generation for zero-day attacks run inside the core. The NoAH core is not a centralized farm of honeypots. On the contrary, it is a distributed set of honeyfarms that can collaborate. Inside the core, both low-interaction (LI) and high-interaction (HI) honeypots are deployed. Low-interaction honeypots are used as a traffic filter. Therefore, activities like port-scanning can be effectively detected by LI honeypots and stop there. Traffic that cannot be handled by LI honeypots is handed over to HI honeypots. In this case, LI honeypots are used as proxies whereas HI honeypots offer the optimal level of realism. To prevent the HI honeypots from infections, a containment environment is used. Any containment environment can be used, like VMware, Xen or Qemu virtual machines. Another proposed environment is Argos.

The address space covered by NoAH core can be further extended. Institutes, campuses or organizations can collaborate with NoAH by deploying a “plug” to NoAH core. This “plug” is actually a tunnel to NoAH core. All traffic going to dark space of a collaborative party is tunneled to honeypots in the NoAH core for processing. Replies from honeypots are tunneled back and injected to party’s network. Using tunneling, honeypot deployment is not needed and thus the administrative overhead is minimal.

Apart from organizations and Institutes, simple home users can help NoAH. Home users or small size enterprises can share their black address (or port) space in a similar way as the participating organizations described before. To do so, they install honey@home [6].

Overall, NoAH glues together various network and host components to form a flexi-



Figure 8.1: The NoAH sensor deployment map

ble network of honeypots. Although the NoAH core is the main component of NoAH architecture, NoAH is more than a set of honeyfarms. Approaches like tunneling and honey@home extend NoAH far beyond its core and have the potential for wide address space coverage with minimal overhead.

8.2 Deployment and experiences

The NoAH infrastructure includes ten static sensors that monitor more than nine thousand unused IP addresses so far. The static sensors are geographically distributed and monitor unused addresses from diverse environments; such as from Universities and Institutions to ISPs. Figure 8.1 displays the location of the deployed NoAH sensors printed on a Google map. In average, the high-interaction honeypots process around half a million packets per day. Such a number is impossible to be inspected manually by hand, thus automatic mechanisms that will display statistics and trends about received traffic

Source IP Addresses			Destination TCP/UDP Ports		
Source IP	Packet Count	Country	Destination Port	Packet Count	Trend
91.41.222.13 (?)	2269		445 (?)	48794	▲
174.37.228.48 (?)	1017		135 (?)	2272	▲
116.53.68.74 (?)	653		1434 (?)	1092	▲
70.44.1.34 (?)	606		3072 (?)	824	▼
93.190.137.241 (?)	569		1024 (?)	788	▼
89.248.172.196 (?)	412		2967 (?)	653	▲
88.80.5.157 (?)	235		23 (?)	606	▲
64.18.128.86 (?)	156		1720 (?)	569	▲
121.12.109.36 (?)	122		5900 (?)	148	▼
111.92.244.76 (?)	116	?	137 (?)	107	▲

Figure 8.2: The top 10 source IP addresses and destination ports as monitored by a NoAH sensor for one day

is needed. We present what types of statistics are gathered by each sensor and how they are visualized.

Each sensor runs three software components:

- *Daemon.* The first one is a minimal daemon based on the pcap library that listens to an interface and captures packets going to a given unused IP address space.
- *Database.* Specific pieces of information for the captured packets are stored to a local Postgres database. This information includes the packet protocol, source and destination IP addresses, source and destination ports, flags in the case of a TCP packet and finally the timestamp of when the packet was captured
- *PHP files.* The last component is a set of PHP files that retrieve and render various statistics for the traffic received. These statistics are:
 - *Top source IP addresses.* By default the top 10 source IP addresses that sent most packets for the last 2 hours is displayed. For each IP address the number of packets it sent and its geographic location is also displayed. The

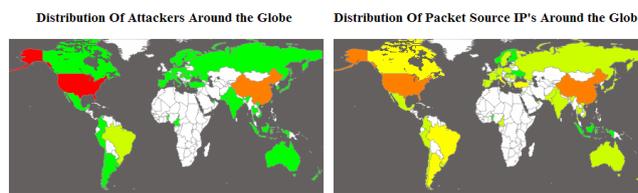


Figure 8.3: Distribution of Attackers Around the Globe

geographic location is retrieved by a local MaxMind database. Additionally, each IP address is clickable. By clicking it, the user is redirected to a webpage that displays all packets sent by that IP address for a configurable time period. The user is able to select that time period which varies from two hours up to the last month. Figure 8.2(a) shows how the top 10 source IP addresses look like.

- *Top destination ports.* The top destination ports targeted by attackers is displayed. For each port the number of packets and a trend indication is also shown. The trend indication represents whether the sensor received more or less packets to that port in comparison with the previous time period. Again, the user can configure the time period up to the last month. By clicking a port, a webpage containing all traffic sent to that port is sent. A screenshot for the top 10 source IP addresses and destination ports as observed by a NoAH for one day can be seen in Figure 8.2(b).
- *Attack maps.* This page includes two earth maps (Figure 8.3). The first map displays 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.
- *Attack graphs.* This page includes three graphs (Figure 8.4). The first one is a breakdown of the TCP ports while the second one is a breakdown of UDP ports for the last two hours. The third one is a breakdown of traffic type in terms of how much TCP, UDP and ICMP traffic was received during the last day.
- *Backscatter traffic.* Each sensor receives unsolicited traffic, that is traffic that comes in response to spoofed attacks. It is trivial to identify such traffic by

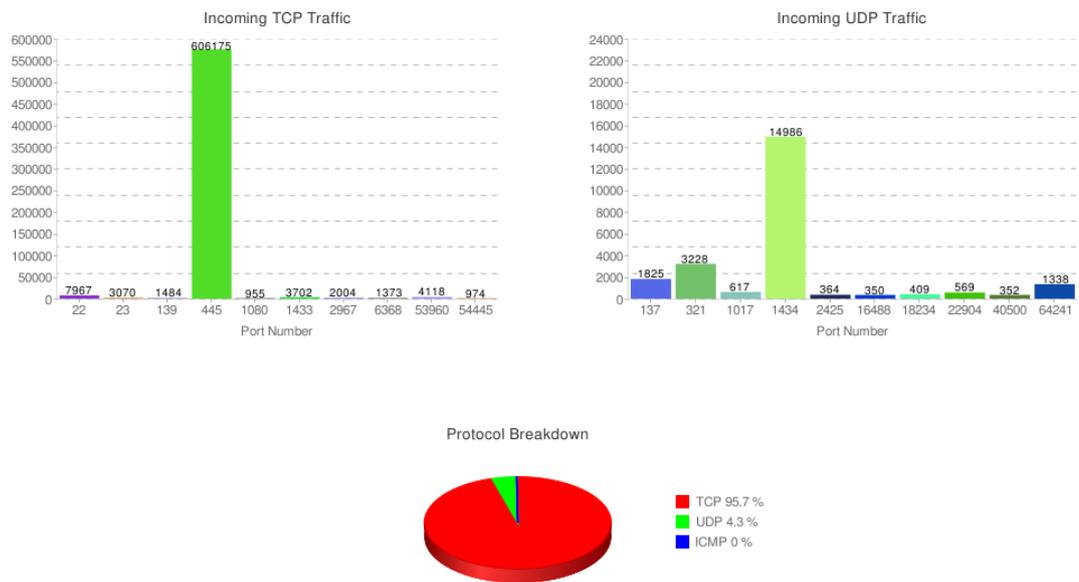


Figure 8.4: Attack graphs

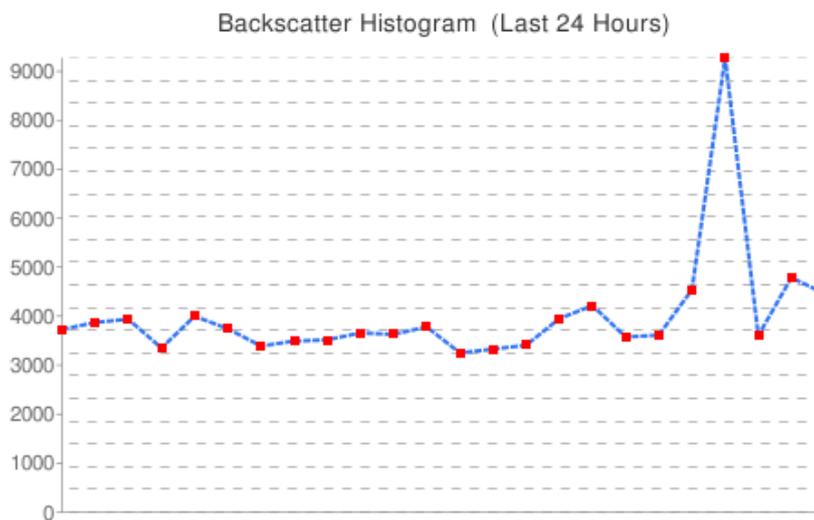


Figure 8.5: Backscatter traffic of 17th of September

inspecting the TCP flags of each packet. A plot for the number of backscatter packets received over the last 24 hours is displayed on Figure 8.5.

Furthermore, the user has several options to customize the displayed information. First of all, she can select traffic coming from or going to a specific IP address. Second, the time frame can be changed. The user can view traffic from the last 2 hours up to the last 1 month. Additionally, she can define an arbitrary date interval. Third, the number of top source IP addresses and top destination ports can be altered. By default, only 10 of the most active source IP addresses and targeted destination ports are displayed. These lists can be expanded up to 100 entries. The final filtering option is to display traffic going to specific destination ports. All of the above options can be combined. Thus, user can query our statistics database for more complicated questions, like "what traffic have you received that originates from IP address X, targets IP address Y to destination port Z from last Monday to last Thursday?" and many more.

All individual sensors send a daily list of the top 100 source IP addresses and top 100 destination ports they observed to the main `stats.fp6-noah.org` site. The lists are then aggregated and displayed in a similar template to the one of individual sensors. The role of aggregation is twofold. First, it permits us to view at a quick glance what traffic is sent to the honeypot sensors and see if trends have changed over the last

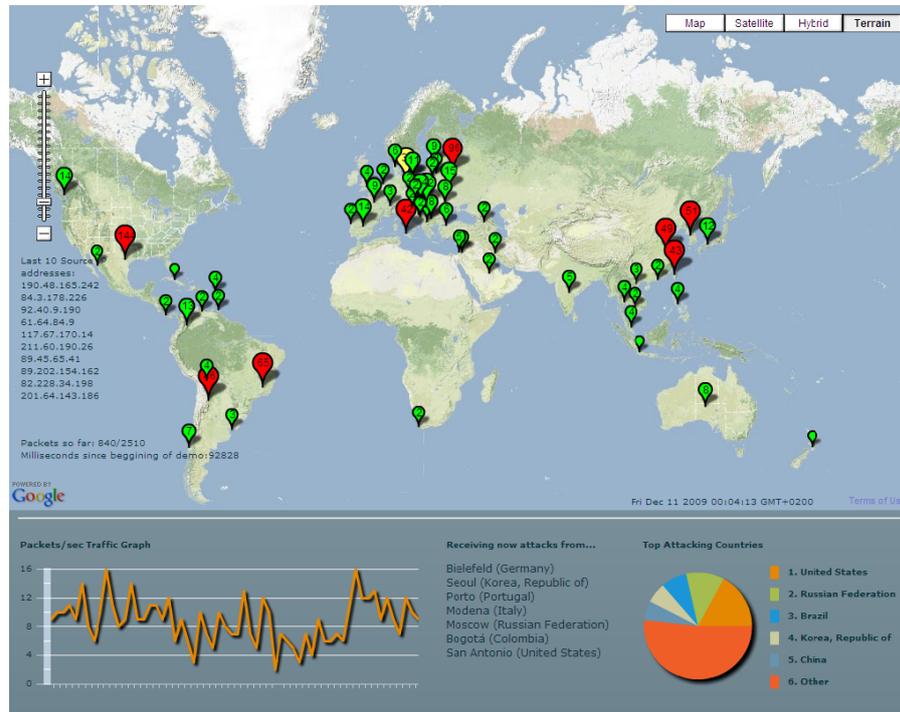


Figure 8.6: TrGeo visualizes the attackers around the world. Each balloon represents how much traffic the location sent in terms of packets.

monitoring periods. Second, it allows us to correlate traffic among various sensors. For example, we can check if two sensors receive traffic on a specific port and if their most attacked ports are the same. As all aggregated data are public, the source IP addresses are anonymized after the aggregation process. However, the geolocation of each source IP address is calculated before the anonymization process. The anonymization function applied to source IP address is the replacement of the address with random numbers. We have also considered other anonymization functions, such as prefix-preserving, but they present a high risk of revealing the honeypot topology.

In our effort to present an overview of what traffic our honeypots capture daily, we have implemented TrGeo. TrGeo is a platform for geographic visualization of packets captured by the NoAH infrastructure. The basic concept behind TrGeo is to track locations of the attackers and display the traffic volume they send on the earth map. For the purposes of this work, we have implemented TrGeo as an Adobe Flash application that renders

Country	# Conversations
USA	56,361
Russia	22,823
Taiwan	18,858
AR	9,327
Japan	6,662
MY	4,193
MD	3,474
PT	3,061
BG	2,663
China	2,646

Table 8.1: Top 10 source countries of attackers that targeted the NoAH sensors

IP Address	# Conversations	Country	Days
xx.xx.215.83	14,499	USA	1
xx.xx.92.207	7,862	USA	2
xx.xx.247.14	7,279	Russia	1
xx.xx.109.189	7,260	USA	2
xx.xx.101.25	5,327	Taiwan	2
xx.xx.98.156	4,853	Taiwan	1
xx.xx.56.118	3,798	Argentina	1
xx.xx.101.211	3,743	USA	3
xx.xx.42.29	3,628	Argentina	1
xx.xx.171.197	3,580	USA	2

Table 8.2: Top 10 attackers that targeted the NoAH sensors.

desired data on top of Google maps. On each source location a balloon is drawn that represents how much traffic the location sent in terms of packets. As time passes, the height of these bars changes according to the traffic they sent. In fact, TrGeo implements the visualization of a sliding time window. For example, if it has not observed packets from a location for a long time period, the balloon for that location will have its counter and size decreased. The information about packet volume and geographical origin is extracted from queries done to the stats.fp6-noah.org site. The aggregation is done at the level of countries, this means multiple attackers from different cities of a country are mapped to a single random location within the country. An instance of TrGeo could be found at Figure 8.6.

During a 2-month-long deployment period, from the end of July to September 2009, the infrastructure handled a total of 153,082 conversations with attackers that targeted the NoAH sensors. The maximum number of conversations handled in one day was 22,268 which occurred on the 17th of September.

Table 8.1 presents the top 10 source countries of attackers that initiated conversations with the NoAH sensors. The aggregated conversations from these 10 countries amount to 84.8% of the total conversations handled by our honeypots. Results show that the United States are the country that initiated the largest number of conversations with the NoAH sensors. In Table 8.2 we can see the statistics of the top attackers for the whole duration of the two month deployment period. It is interesting to note that all top attackers were active for a few days, and in all cases they were consecutive.

8.3 Current and future work

The area of honeypots is an active research division. However, new attack vectors should be taken under consideration. Malware propagation takes place in other communication channels, such as Instant Messaging networks and social networking sites. Malware authors take advantage of the fact that users trust the content sent by other users in their friend list. By infecting instant messenger clients or compromising IM accounts, they spread malicious URLs and executables by spamming the friend lists. We have taken some first steps for detecting these attack vectors and have performed a preliminary analysis of IM-enabled phishing and malware propagation.

Lately, the number of devices that access the Internet has increased dramatically. Users now crawl the Web by handheld devices, such as mobile phones and PDA's. The nature and volume of mobile networks makes them an attractive platform to launch attacks against. Honeypot technologies could be applied to that domain too, for example decoy services running on mobile phones. Platforms like Java and Android [1] allow the development and deployment of mobile honeypots.

9 WAPI and WOMBAT Workshop Scenarios

9.1 Introduction

As discussed in deliverable D3.1 (“Infrastructure design”) [3], we proposed to define a common WOMBAT API (WAPI) to be shared among all the participants in order to simplify the task of the data consumer willing to take advantage of these datasets. The WAPI is actually a remote API allowing consumers to retrieve remote information from sources according to a given communication protocol. The communication protocol we choose to use is the SOAP protocol. A strong argument for this choice is that SOAP is very well supported by a high number of client libraries making it the perfect choice for satisfying the client flexibility requirement.

The WAPI architecture is based in two components the WAPI server and client. Each WAPI server is created by each partner who is offering data. The partner is responsible for the type and the amount of data that the server offer to others. Moreover, he is responsible for the clients he is willing to share his data.

WAPI takes advantage of SSL protocol to provide confidentiality of the transmitted results and, most importantly, to implement access control. In order to be able to talk with a WAPI server, a client must provide a valid SSL certificate signed by the Certification Authority maintained by each partner providing WAPI services.

The implementation of WAPI server and client have been made using the Python Language. Python was chosen due its ability to be flexible, object-oriented and has wide range of libraries. These libraries offer a lot of WAPI characteristics such as security (SSL) and flexibility (SOAP). But, using SOAP protocol as the communication protocol offers programming language freedom. This means that anybody can create a WAPI server and client in any language, as far as the data been sending over the SOAP protocol meet the WSDL standards.

The following Sections also appears in the Deliverable 6.4 (“Second Open Workshop Proceedings”) devoted to the WOMBAT workshop organized in Saint Malo in September 2009. We have decided to include it into this deliverable as well in order to provide to the reader a document that would as self contained as possible to get a good understanding of the usefulness of these various sensors and how they can interact together.

9.2 Preliminaries

In this section we show how one can use the WAPI to query different WOMBAT datasets, namely Anubis (Figure 11.1), SGNET (Figure 11.2), HARMUR (Figure 11.3), Shelia (Figure 11.4), HSN (Figure 11.5), WEPAWET (Figure 11.6), VirusTotal (Figure 11.7) and FORTH (Figure 11.8).

The provided client takes advantage of the IPython interactive shell (<http://ipython.scipy.org/moin/>) as opposed to the original Python shell, providing better support for autocompletion and results visualization. The client starts by typing the following in the command line:

```
python wapi_client.py -c conf
```

`conf` is the necessary configuration file that the wapi client needs in order to connect with all the datasets. Below it presents a common configuration file.

```
# WAPI client configuration file. Specifies
# the list of WOMBAT datasets as well as the
# connection options to connect to each of them.

# general format:
# [<dataset_name>]
# url=<scheme>://<hostname or ip>/path/to/wapi/server/
# cert_ca=relative/path/to/ca/certificate.pem
# cert_client=relative/path/to/client/certificate.pem
# namespace=<namespace>

#NOTE: cert_ca, cert_client and namespace are optional.

[sgnet]
url=https://193.55.112.70/sgnet/
cert_ca=cert/sgnet/cacert.pem
cert_client=cert/sgnet/client.pem
namespace=sgnet.wapi.wombat-project.eu

[wepawet]
url=https://193.55.112.70/wepawet/
cert_ca=cert/wepawet/cacert.pem
cert_client=cert/wepawet/client.pem
namespace=wepawet.wapi.wombat-project.eu

[harmur]
url=https://193.55.112.70/HARMUR/
cert_ca=cert/harmur/cacert.pem
cert_client=cert/harmur/client.pem
```

```
namespace=HARMUR.wapi.wombat-project.eu

[anubis]
url=wapi://isis.iseclab.org:8080/anubis/
cert_ca=cert/anubis/cacert.pem
cert_client=cert/anubis/client.pem
namespace=anubis.wapi.wombat-project.eu

[shelia]
url=https://centaur.few.vu.nl:8080/shelia/
cert_ca=cert/shelia/cacert.pem
cert_client=cert/shelia/client.pem
namespace=shelia.wapi.wombat-project.eu

[forth]
url=wapi://139.91.90.201/forth/
cert_ca=cert/forth/cacert.pem
cert_client=cert/forth/client.pem
namespace=forth.wapi.wombat-project.eu

[virustotal]
url=wapi://62.15.230.161/virustotal/
cert_ca=cert/virustotal/cacert.pem
cert_client=cert/virustotal/client.pem

[hsn]
url=https://gror.nask.waw.pl:8888/hsn/
cert_ca=cert/hsn/hsn_ca.pem
cert_client=cert/hsn/hsn_user.pem
namespace=hsn.wapi.wombat-project.eu

[utils]
anubis_url=https://anubis.iseclab.org/index.php?action=result&task_id=%s&format=html
```

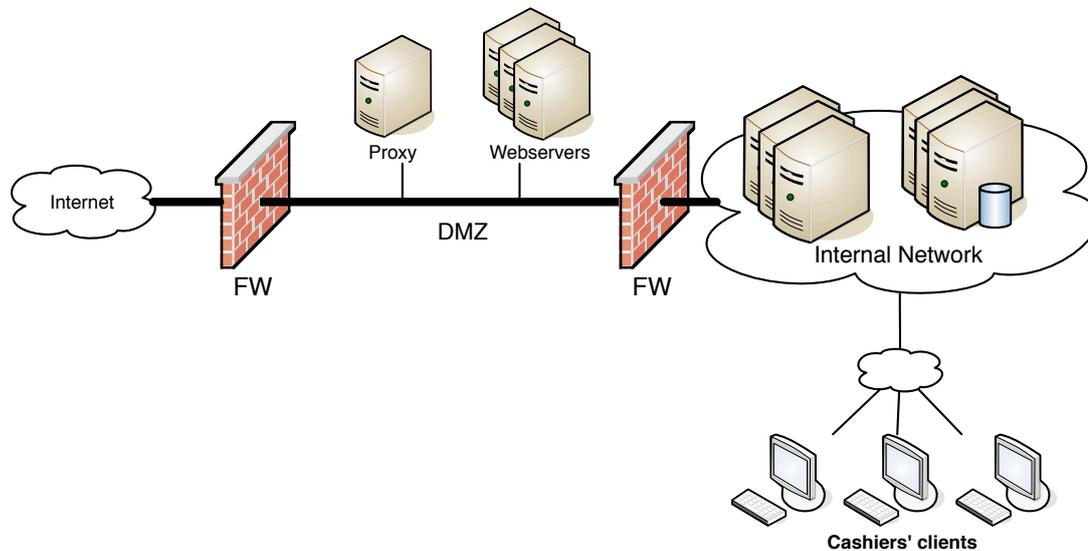


Figure 9.1: Bank Network

9.3 Investigation of a Banking Fraud

In this scenario, we take on the role of CERT responders from a Bank (See figure 9.1). The Bank needs to conduct a (forensics) investigation of the machine of a client that has reported a fraud case via electronic banking. The Bank up to now has excluded that the fraud was related to phishing or any other physical swindle.

A brief analysis of the infected client does not show any clear evidence of infection, no suspicious BHO is detected and no suspicious registry entries are found in the system. The client affected by the fraud is connected to the Internet through an HTTP proxy, and has agreed to give you the list of the HTTP activity of the infected machine in the last week. After a brief look at such activity, you notice a large amount of HTTP requests towards a suspicious domains. Such requests are performed every 20 minutes approximately, during working hours but also during night and weekends. All the queried URLs are similar to the following one: `http://ijmkyjves.net/iE=eQBHE8cNe8DRM`

9.3.1 Malware identification

Can we take advantage of the WAPI to link such suspicious behavior to a specific malware sample?

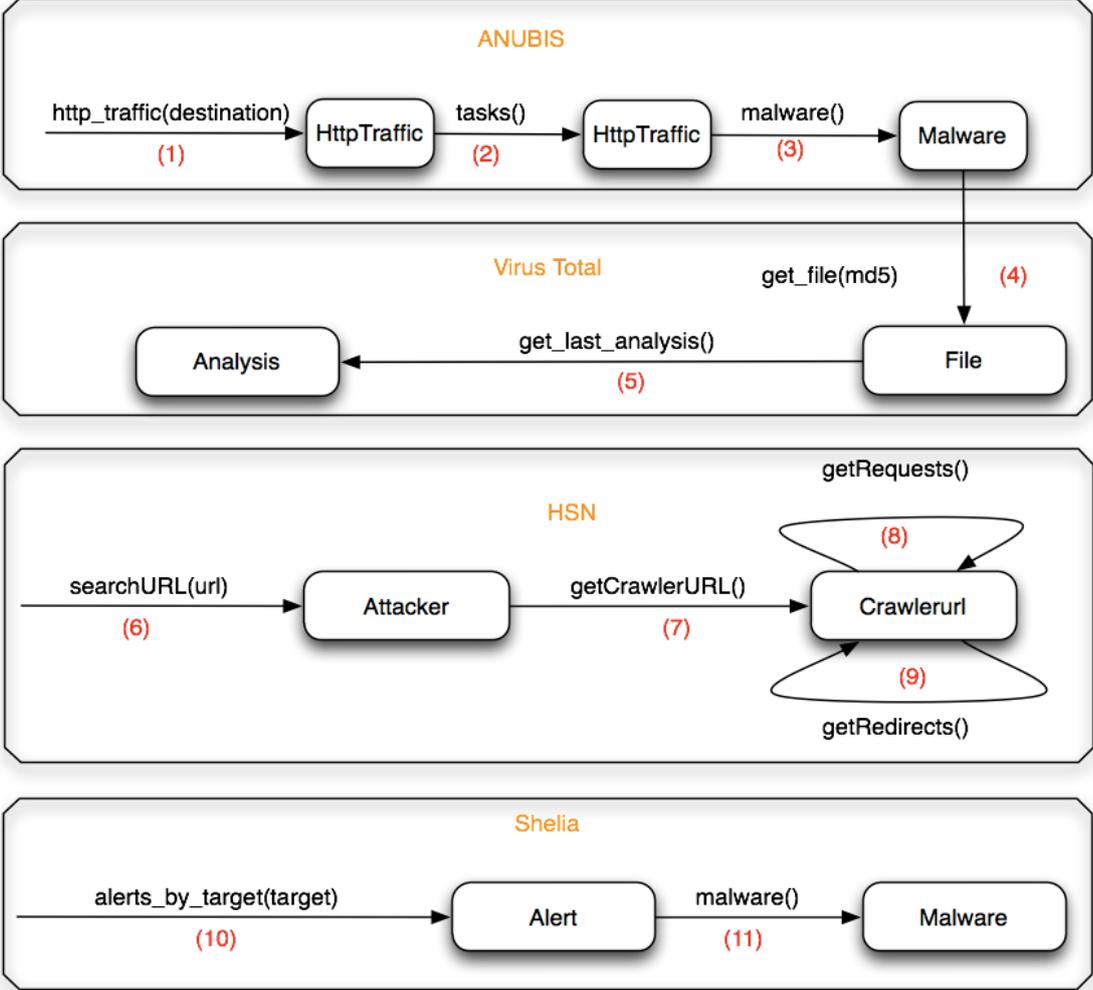


Figure 9.2: Investigation of a Banking Fraud (Part A)

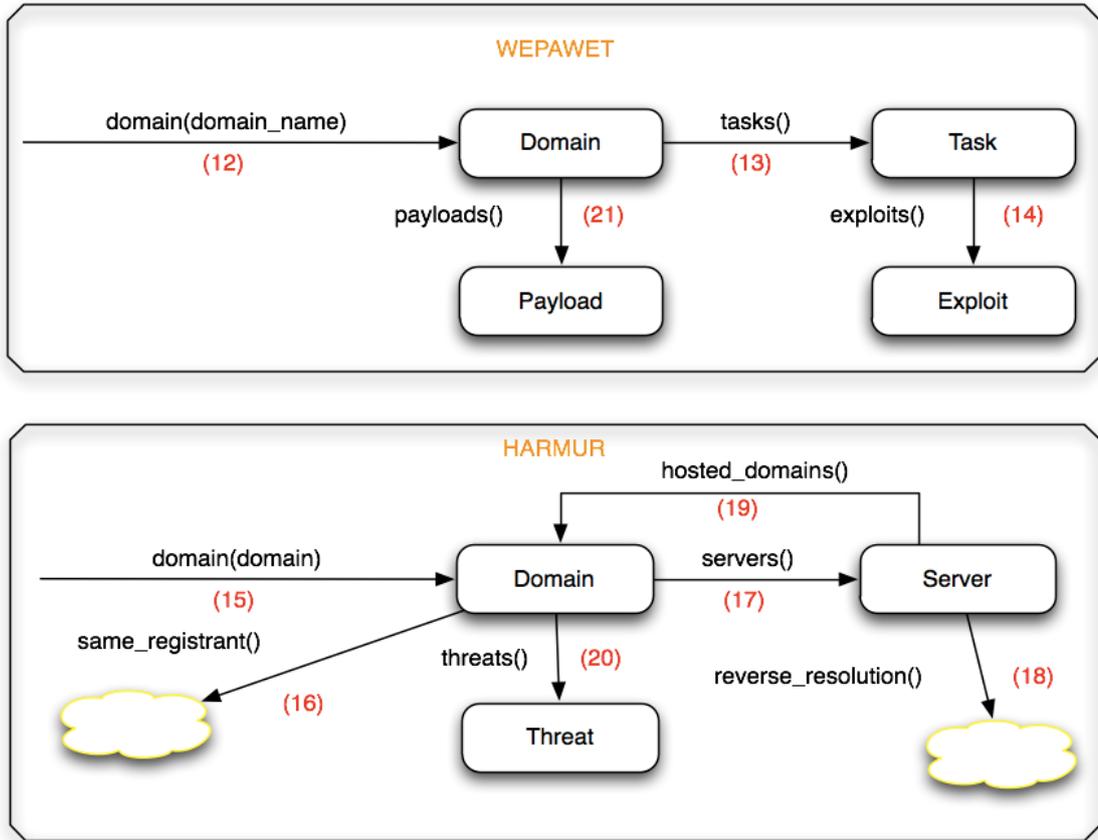


Figure 9.3: Investigation of a Banking Fraud (Part B)

Q Anubis stores behavioral information for thousands of malware samples during an execution time of two minutes approximately. Is there any malware sample analyzed by Anubis that exposed a similar behavior during its analysis?

A An example solution follows:

```
#let's search for any HTTP conversation targeting the identified domain
http = anubis.http_traffic(destination="ijmkkyjves.net")

#let's retrieve the WAPI malware objects associated to this behavior
malware = [h.tasks()[0].malware()[0] for h in http]
```

Figure 9.2 steps (1), (2) and (3).

Q Look more in depth at the MD5 hash of these samples and at their binary size. Is there anything striking about these characteristics?

A Let's extract these simple characteristics from the Anubis malware objects:

```
#what are the basic characteristics of these samples?
stats = set([(m.md5,m.file_size,m.mime_type) for m in malware])
```

Q Try to take advantage of the VirusTotal API to give a name to these samples.

A An example solution follows:

```
md5_hashes = set([m.md5 for m in malware])

for md5 in md5_hashes:
    print "=== %s"%md5
    print virustotal.get_file(md5=md5)[0].get_last_analysis()[0].
        av_positives_report
```

Figure 9.2 steps (4) and (5).

9.3.2 Infection analysis

In the previous step, we have been able to link the suspicious network behavior observed on the infected host with a specific malware sample, called Mebroot. Mebroot received a certain amount of press coverage (see, for instance, <http://www.symantec.com/connect/blogs/bootroot-trojanmebroot-rootkit-your-mbr>). While the low antivirus detection rate might explain why such malware was not detected by the antivirus software installed on the infected machine, we still do not know how the machine got infected in the first place.

We go back to the logs, and we are able to identify the moment in which the anomalous connection attempts started. Assuming that the infection happened approximately at that time, we extract a list of URLs visited by the infected machine in the hour preceding the beginning of the anomaly. The traffic was concerning the following domains:

```
domains = ["google.com",
"facebook.com",
"baidu.cn",
"adobe.com",
"bandwidthplace.com",
"azadars.com"]
```

We therefore hypothesize that one of these domains is the potential cause of the infection. Let's use the WAPI to verify this hypothesis.

Q Check if any of the domains visited by the client are known to the HoneySpider network.

A An example solution follows:

```
for d in domains:
    hsnatts = hsn.searchURL(url=d)
    if len(hsnatts) == 0:
        print "No results for %s"%d
    else:
        for hsnatt in hsnatts:
            print "Result for %s:"%d
            print "URL: %s, classified as %s"%(hsnatt.normalizedURL,
            hsnatt.classification)
```

Q `azadars.com` is seen by the HoneySpider Network as suspicious. Take advantage of the WAPI to understand what type of threat is associated to it.

A An example solution follows:

```
hsnazadars = hsn.searchURL(url="azadars.com")[1]
hsnazadars.dump();
# Was it scanned with the high interaction component?
print hsnazadars.highInteractionScanIds()

# Ok, and what was the result of low interaction scan?
hsnazalim = hsnazadars.getCrawlerURL()[0]
hsnazalim.dump()

#So, low interaction heuristics show, that this is just suspicious. Why?
```

```

#Let's have a look at the requests:"
hsnreqs = hsnazalim.getRequests()
for req in hsnreqs:
print "%s request %s"%(req.classification,req.request)

#Why is the suspicious one actually suspicious?
for req in hsnreqs:
if req.classification == "SUSPICIOUS":
req.dump()

#So, it was obfuscated. Were any redirections extracted from the scripts?
for redir in hsnazalim.getRedirects():
print "%s redirection to %s"%(redir.classification,redir.request)

#There was one, but heuristics found nothing interesting there."

```

Figure 9.2 steps (6), (7), (8) and (9).

Q Try to perform the same analysis taking advantage of the shelia dataset.

A An example solution follows:

```

sheliaazadars = shelia.alerts_by_target(target="azadars.com")[0]

#is there any malware downloaded as a consequence of the analysis?
sheliamalware = sheliaazadars.malware()[0]

#notice the MD5 and the file length and compare them with
#the malware identified in Anubis. Is there anything interesting?

```

Figure 9.2 steps (10) and (11).

Q Do you think that the temporal evolution might be the reason for which Shelia and HSN provide discording reports? Try to validate this hypothesis taking advantage of the different WAPI datasets.

A First of all, we should look at the timestamp of the analysis tasks for the domain `azadars.com`. Did Shelia and HSN look at the site at the same moment in time?

```

#when did HSN first look at the domain?
print hsnazadars.creationDate

#when did Shelia analyze the same domain?
print sheliaazadars.timestamp

```

If we look at the wepawet information, we can have a more clear proof that `azadars.com` changed over time.

```
wepawetazadars = wepawet.domain(domain_name="azadars.com")[0]

#let's look at the different analysis tasks
for t in wepawetazadars.tasks():
    print "On %s, there were %d exploits"%(t.analyzed_at,len(t.exploits()))
```

Figure 9.3 steps (12) and (13).

Q Which vulnerabilities have been exploited?

A The answer can be easily retrieved by looking in more detail the wepawet analysis.

```
#let's pick the oldest analysis task
wepa_ana = wepawetazadars.tasks()[0]

#which exploits were detected?
for expl in wepa_ana.exploits():
    expl.dump()
```

Figure 9.3 step (14).

9.3.3 The real culprit

In the previous Section, we have been able to identify a domain, `azadars.com`, that was visited by our victim just before the infection. We have been able to show that, before September, this domain was indeed capable of exploiting clients to install malware and compromise our victim machine. We don't know yet much about `azadars.com`: is it a malicious site set up on purpose to compromise victims redirected to it through phishing campaigns? Let's try to use the WAPI to know more about it.

Q Take advantage of the HARMUR dataset to know more about the `azadars.com` site.
Is the site registrant associated to other malicious domains?

A An example solution follows:

```
azadars = harmur.domain(domain="azadars.com")[0]
azadars.dump()

print azadars.same_registrant()
```

Figure 9.3 steps (15) and (16).

Q On what physical server is the site hosted?

A An example solution follows:

```
azadars_srv = azadars.servers()[0]

#let's print all the available information
azadars_srv.dump()
```

Figure 9.3 step (17).

We can query HARMUR to know more about other sites hosted on the same physical server.

```
azadars.servers()[0].reverse_resolution()
same_server = [(d.name,d.color) for d in azadars.servers()[0].hosted_domains()
               ]
```

Figure 9.3 steps (18) and (19).

Q From WHOIS/DNS information, `azadars.com` looks like a legitimate site hosted by a web hosting service that has been compromised by an exploit toolkit. Look at the exploit information provided by HARMUR and `wepawet` and try to confirm this hypothesis.

A HARMUR aggregates different threat information feeds to decide whether a certain domain is malicious or not. What can we say about this domain?

```
for threat in azadars.threats():
    threat.dump()
```

Figure 9.3 step (20).

`Wepawet` allows us to analyze more in detail the effect of the exploits detected in the previous phase.

```
for payload in wepa_ana.payloads():
    print "download %s from %s"%(payload.md5,payload.url)
    #compare with Anubis samples
    if payload.md5 in md5_hashes:
        print "the downloaded sample performs connections to ijmkyjves.net once
              executed!"
```

Figure 9.3 step (21).

The site azadars.com seems to be infected by an exploit toolkit that forces the victim to download from the domain ijmkkjves.com a malware sample whose MD5 was analyzed by Anubis and that is known to be performing connections to ijmkkjves.net.

Q “ijmkkjves” looks like a randomly generated string. Are these two domains part of a bigger picture?

A We can query the HARMUR dataset to find an answer to this question:

```
dl_site = harmur.domain(domain="ijmkkjves.com")[0]
#all the information available on the server:
dl_site.servers()[0].dump()
#which other domains are hosted on the same server?
dl_site.servers()[0].reverse_resolution()
```

9.3.4 Conclusions

During this demonstration we have explored the potential of the WAPI to build better pictures on the threats “ecosystem”. We have started from a real case scenario, and we have combined information retrieved by Anubis with the analysis performed by different client honeypot technologies. We have seen how each of these datasets is often able to show us only one facet of the truth, and the value of aggregating together these different facets to get a broader view on Internet threats.

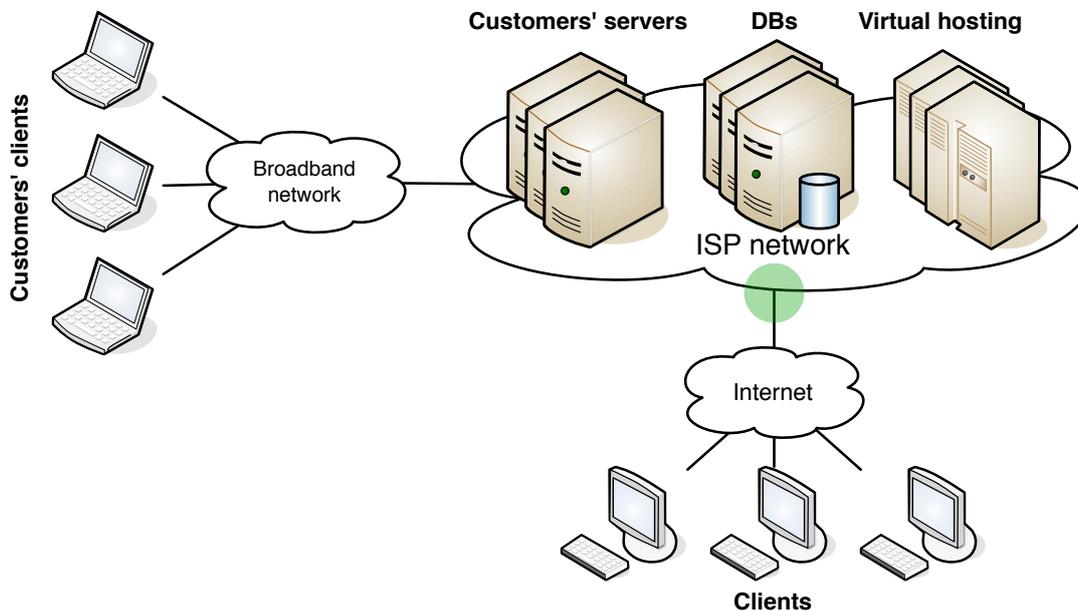


Figure 9.4: ISP Network

9.4 Monitoring of our Own Networks

In this scenario, we are in the security staff of an ISP 9.4 or an enterprise network (or even a CERT for a given country), and we are interested in querying WOMBAT datasets to get information about infected machines in our own network.

211.108.242.0/24

The idea is to offer to network administrators useful information that might help them in understanding what type of threat is affecting the different clients in order to clean them or notify them.

9.4.1 Searching for infections

Q We can start by querying the SGNET dataset to know if any honeypot has observed malicious activity generated by hosts of our own network.

- What exploit events have we found, if any?

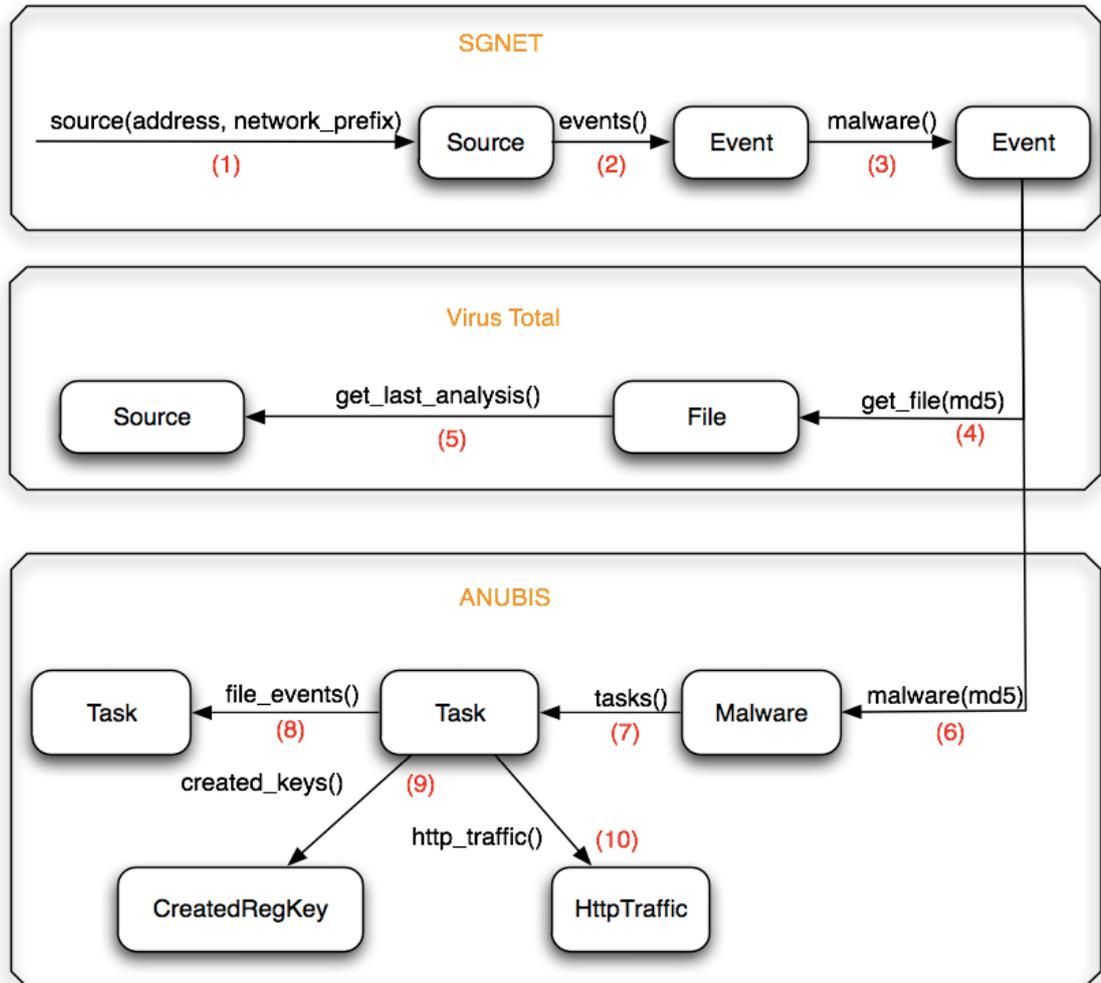


Figure 9.5: Monitoring of Own Networks (Part A)

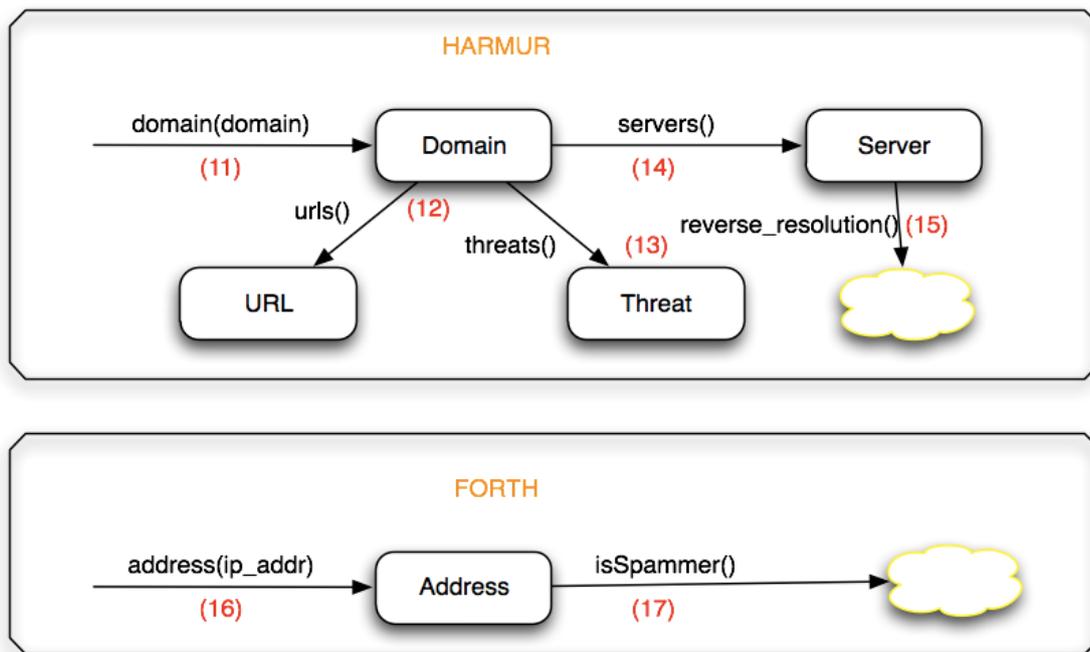


Figure 9.6: Monitoring of Own Networks (Part B)

- Can we associate to these exploit events the activity of a specific malware type?
- If yes, how many?

A We first retrieve all the known sources and, for each sources, we save the list of events associated with it.

```
ip = "211.108.242.0"
print "We are interested in our network %s/24" % ip

#What does sgnnet know about it
sources = sgnnet.source(address=ip,network_prefix=24)
print "We have %d sources" % len(sources)
```

Figure 9.5 step (1).

Q One host of our network has performed a successful code injection attack against one of the SGNET honeypots, and is therefore likely to be infected. Take advantage of the Anubis and VirusTotal dataset to know more about the nature of this malware. Try to give a “name” to such sample.

A An example solution follows:

```
source = sources[0]
print "Address is %s" % source.address

event = source.events()[0]
print "Here is the exploit event"

event.dump()
malware = event.malware()[0]
event_md5 = malware.md5
print "Uploaded malware with md5 %s\n" % event_md5

#What does virustotal say about the md5?
print virustotal.get_file(
    md5=event_md5)[0].get_last_analysis()[0].av_positives_report

event_tasks = anubis.malware(md5=event_md5)[0].tasks()
print "\nAnubis has %d tasks for this md5\n" % len(event_tasks)

#Here is one
event_tasks[0].dump()
```

Figure 9.5 steps (2), (3), (4), (5), (6) and (7).

Q We want to take a deeper look at this task to see, say, what type of activity it performed (e.g., files created, remote HTTP connections).

A An example solution follows:

```
#Let's look at this task in more detail
t = event_tasks[0]

#File events
print [x.name for x in t.file_events()]

#Registry keys created
print [x.key_name for x in t.created_keys()]

#HTTP traffic
print t.http_traffic()
print "No real files, and nothing very meaningful in the registry either\n"
```

Figure 9.5 steps (8), (9) and (10).

Q Recall that the web interface of Anubis contain many details about capture samples. Can we take a deeper look at it? We can rely on the `open_anubis_report` shortcut method provided by the `Utils` module.

A This can be done by invoking the appropriate method:

```
event_uuid = event_tasks[0].uuid
print "\nAnubis has this as task uuid %s\n" % event_uuid

#Look at the report
Utils.open_anubis_report(event_uuid)

#It has an exception, does not do much
```

9.4.2 Looking for similar malware samples

From the previous point, we have seen that the specific exploitation behavior used by this malware sample is used by lots of different malware groups. While such exploitation behavior is mainly famous for being associated to the Rahack/Allapple worm, it seems to be used also by other, less known, malware families.

Q Let's look more in depth at this malware cluster and let's take advantage of Anubis to identify any less visible, but interesting, behavior. To reduce querying time, we

can safely cap our requests to the first 30 samples. Also, remember that the `Utils` module provides the handy `flatten_list`, which just does what it says.

A An example solution follows:

```
#Are both samples in same cluster?
print set(Utils.flatten_list([t.cluster() for t in event_tasks]))

cluster = t.cluster()[0]
cluster.dump()

#It's a big cluster. Let's look at a few of the tasks in it
#Let's take the first 30, and see if they do http

cluster_tasks = cluster.tasks()[0:30]
http1 = [t.http_traffic() for t in cluster_tasks]
cluster_http = Utils.flatten_list(http1)
print set([x.dest_name for x in cluster_http])

#..or interesting file stuff

file1 = [t.file_events() for t in cluster_tasks]
cluster_files = Utils.flatten_list(file1)
print set([x.name for x in cluster_files])
```

Q It looks like the whole cluster is not doing much. Maybe we can retrieve other similar samples by leveraging the SGNNet EPM clustering and rely on Anubis to figure out what type of HTTP traffic these malware are generating. In particular, we are interested in finding malware that attempt to connect to a domain we retain suspicious: `zief.pl`.

A An example solution follows:

```
#Let's use sgnet EPM clustering, to see if similar activities
#have malware with more interesting behavior

activity = event.activity()[0]
print "Here is the activity:\n"
activity.dump()

#Looking up MD5s for this activity
epm = activity
md5s_epm = [m.md5 for m in epm.malware()[0:30]]
print "\nGot %d MD5s\n" % len(md5s_epm)
```

```

#Let's look for anubis tasks
malwares = [anubis.malware(md5=m) for m in md5s_epm]
malwares = filter(lambda m:len(m),malwares)
malwares = [m[0] for m in malwares]
tasksl = [m.tasks() for m in malwares]
tasks = Utils.flatten_list(tasksl)
print "Got %d tasks" % len(tasks)

#Looking up all of their HTTP traffic
httpl = [t.http_traffic() for t in tasks]
http = Utils.flatten_list(httpl)

#Here are all accessed domains
print list(set([x.dest_name for x in http]))

#We are interested in zief.pl
http_zief = filter(lambda h:h.dest_name.endswith("zief.pl"),http)

```

9.4.3 Looking more in depth at zief.pl

We have started from a single infection of a polymorphic worm and we have been able, taking advantage of SNET EPM clustering, to identify a larger set of malware samples and infections that are linked to the same malware group. While the specific sample from which we started our analysis did not expose any other behavior than simple worm-like propagation, we have been able to identify in other samples belonging to the same group some suspicious HTTP behavior related to a specific domain, `zief.pl`.

From now on, we want to dig a little bit deeper on the `zief.pl` domain. What is its role in the infection?

Q Take advantage of the HARMUR dataset to get information about the server(s) hosting this domain. Are they located in Poland or somewhere else?

A An example solution follows:

```

#What does harmur say of threats from zief.pl?
domain = harmur.domain(domain="zief.pl")[0]

#Here are urls harmur has on this domain
print [u.url for u in domain.urls()]

#Here is a summary of the threats on this domain
print [(t.type,t.id) for t in domain.threats()]

```

```
#Here is one of the bloodhound threat in more detail
domain.threats()[1].dump()

#Let's look up one of the threats on the symantec website
threat_urls = domain.threats()[1].help.split()
print threat_urls

#Let's look up the servers' geo location information
servers = domain.servers()

#only one of the three servers is located in China!
for srv in zief_servers:
    srv.dump()
    print srv.reverse_resolution()
```

Figure 9.6 steps (11), (12), (13), (14) and (15).

Q Looks like only one of the three servers is actually located in China. Maybe the FORTH dataset can provide further information about `zief.pl`. Let's query it.

A An example solution follows:

```
#Where is zief.pl located?
zief_ips = set([h.dest_ip for h in http_zief])
print "Here are the ips:\n%s\n" % str(zief_ips)
zief_ip = list(zief_ips)[0]

#Let's look up the whois data
forth.address(ip_addr=zief_ip)[0].dump()

#Does this address send spam?
print forth.address(ip_addr=zief_ip)[0].isSpammer()
```

Figure 9.6 steps (16) and (17).

Q `zief.pl` seems to be a download site for additional components retrieved by the malware once executed. Is there any additional threat on this domain? We can take advantage of the Anubis web interface to retrieve in depth information, on its servers and on any threat known to be associated to it.

A An example solution follows:

```
#Let's see if anubis tasks that contact zief.pl are more interesting
zief_http = filter(lambda x:x.dest_name.endswith("zief.pl"),http)
zief_tasks1 = [h.tasks() for h in zief_http]
```

```

zief_tasks = Utils.flatten_list(zief_tasks1)
map(lambda t:zief_tasks.extend(t),zief_tasks1)
print "We have %d tasks that contact www.zief.pl\n" % len(zief_tasks)

#How many anubis clusters are they in?
print set(Utils.flatten_list([t.cluster() for t in zief_tasks]))

#Here is one such anubis task
zief_tasks[0].dump()
print "MD5 is %s" % zief_tasks[0].malware()[0].md5

#Look at the report. A lot more going on...
Utils.open_anubis_report(zief_tasks[0].uuid)

```

Q Interestingly, we have found a match into Anubis. It's worth looking up VirusTotal analyses about the malware sample we just found in Anubis. Optionally, the WEPAWET dataset may be useful to find any client-side threat (e.g., JavaScript, PDF, Flash) related to `zief.pl`.

A An example solution follows:

```

#What does VirusTotal say about this malware?
print virustotal.get_file(
md5 = zief_tasks[0].malware()[0].md5)[0].get_last_analysis()[0].
av_positives_report

```

Figure 9.5 steps (4) and (5).

In addition, the WEPAWET report provides direct links to FIRE reports which are worth to inspect as they show the different activities (e.g., C&C, phishing) of the malicious IPs over time.

9.4.4 Conclusions

In this scenario, we have shown how it is possible to take advantage of the WAPI to investigate the security status of a certain network. We have shown how to take advantage of the different features offered by the datasets to investigate malware infections and get a better understanding of the underlying processes.

10 Conclusions

In this deliverable we described the deployment and experiences of SGNET, HARMUR, Shelia, Paranoid Android, HoneySpider Network, Bluebat and NoAH sensors. These sensors capture various types of data such as malware, IP sources, malicious URLs and exploits.

The early experiences show that the WOMBAT Project [10] is fulfilling our preliminary expectations about having powerful tools for collecting data. These data are useful for categorizing attackers and malware behaviors. Moreover our experiments reveal that the sensors can cooperate with each other, enriching in this way the information offered for analysis.

The diversity of the data sources led to the creation of the WAPI, an API for exchanging data, among partners, over the SOAP protocol. WAPI takes advantage of SSL protocol to provide confidentiality of the transmitted results and, most importantly, to implement access control. Thanks to WAPI, we are now able to trace attacks, find the methodology the attackers use and create new tools for faster reaction to any suspicious behavior. This happened because all data are now consolidated in a global infrastructure. The WAPI has already been tested and used. One of the biggest test it faced, was at the second WOMBAT Workshop where the attendees achieved to investigate two case scenarios with the help of sensors data, using the WAPI client.

Bibliography

- [1] Android. <http://code.google.com/android/>.
- [2] Bluez website. <http://www.bluez.org/>.
- [3] D06 (d3.1) infrastructure design. http://wombat-project.eu/WP3/FP7-ICT-216026-Wombat-WP3-D06_V02_Infrastructure_design.pdf.
- [4] D07 (d3.2) design and prototypes of new sensors.
- [5] Gpsd website. <http://gpsd.berlios.de/>.
- [6] Network of affined honeypots. <http://www.fp6-noah.org>.
- [7] Pybluez website. <http://org.csail.mit.edu/pybluez/>.
- [8] Secureobex server. <http://www.mulliner.org/bluetooth/sobexsrv.php>.
- [9] Trifinite.org website. <http://www.trifinite.org>.
- [10] Wombat project. <http://www.wombat-project.eu/>.
- [11] P. Baecher, M. Koetter, T. Holz, M. Dornseif, and F. Freiling. The Nepenthes Platform: An Efficient Approach to Collect Malware. In *9th International Symposium on Recent Advances in Intrusion Detection (RAID)*, September 2006.
- [12] U. Bayer, C. Kruegel, and E. Kirda. TTAalyze: A Tool for Analyzing Malware. In *15th European Institute for Computer Antivirus Research (EICAR 2006) Annual Conference*, April 2006.
- [13] C. Leita, M. Dacier, and F. Massicotte. Automatic handling of protocol dependencies and reaction to 0-day attacks with ScriptGen based honeypots. In *9th International Symposium on Recent Advances in Intrusion Detection (RAID)*, Sep 2006.
- [14] C. Leita, K. Mermoud, and M. Dacier. Scriptgen: an automated script generation tool for honeyd. In *21st Annual Computer Security Applications Conference*, December 2005.

- [15] G. Portokalidis, A. Slowinska, and H. Bos. Argos: an emulator for fingerprinting zero-day attacks. In *ACM Sigops EuroSys*, 2006.
- [16] D. Smith. Allapple worm (ISC diary), <http://isc.sans.org/diary.html?storyid=2451>.
- [17] D. Turner, M. Fossi, E. Johnson, T. Mack, J. Blackbird, S. Entwisle, M. K. Low, D. McKinney, and C. Wueest. Symantec global internet security threat report. Technical Report XIII, Symantec, 2008.
- [18] VirusTotal. www.virustotal.com.

11 APPENDIX

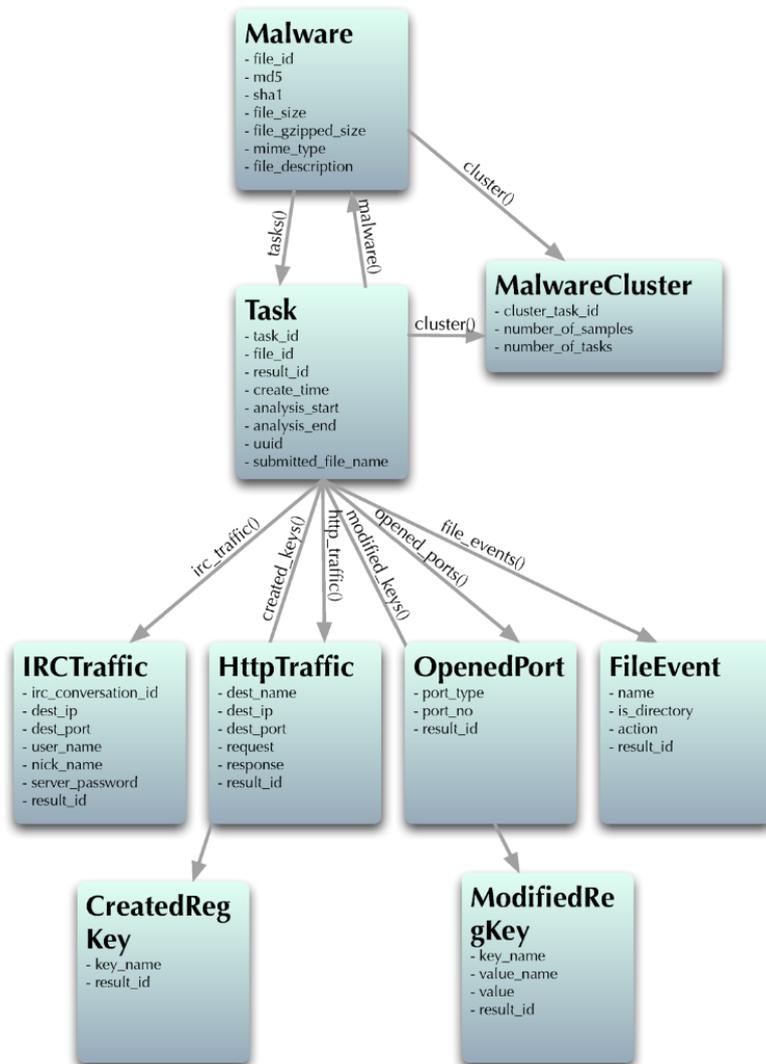


Figure 11.1: Anubis Dataset. The dataset contains nine Objects (Malware, Task, MalwareCluster, IRCTraffic, HttpTraffic, OpenedPort, FileEvent, CreatedRegKey and ModifiedRegKey) that are presented as list-boxes. The elements of each list-box are the Object identifiers. Arrows depicts the references among Objects.

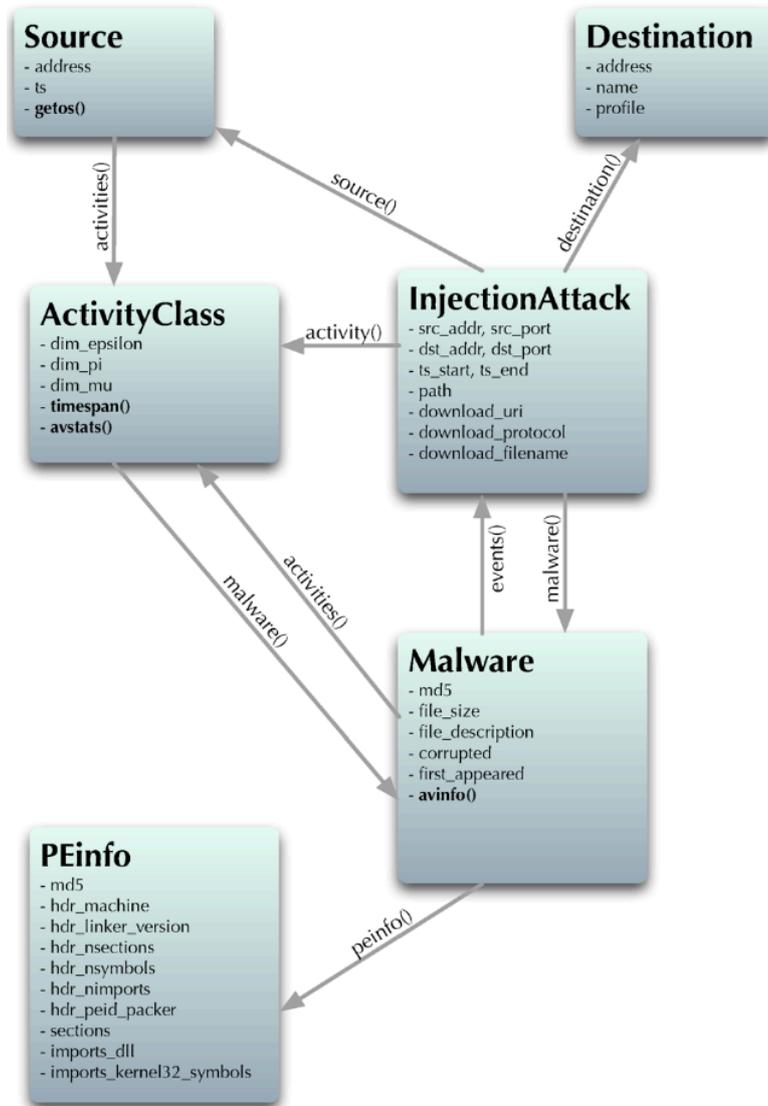


Figure 11.2: SGNET Dataset. The dataset contains six Objects (Source, Destination, ActivityClass, InjectionAttack, Malware and PEinfo) that are presented as list-boxes. The elements of each list-box are the Object identifiers. Methods are shown in bold. Arrows depicts the references among Objects.

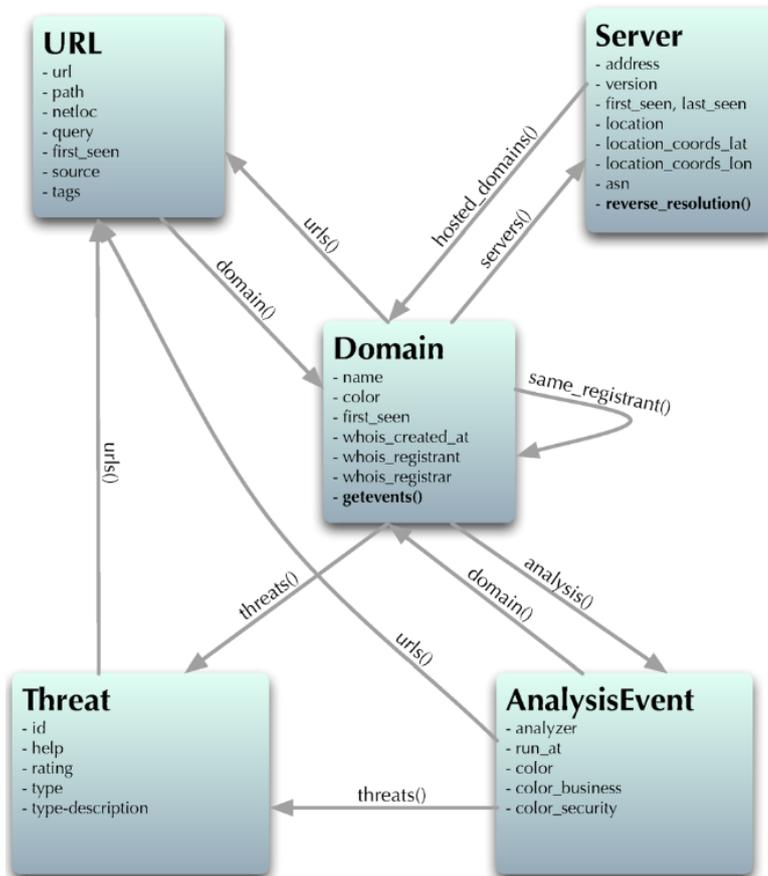


Figure 11.3: HARMUR Dataset. The dataset contains five Objects (**URL**, **Server**, **Domain**, **Threat** and **AnalysisEvent**) that are presented as list-boxes. The elements of each list-box are the Object identifiers. Methods are shown in bold. Arrows depicts the references among Objects.

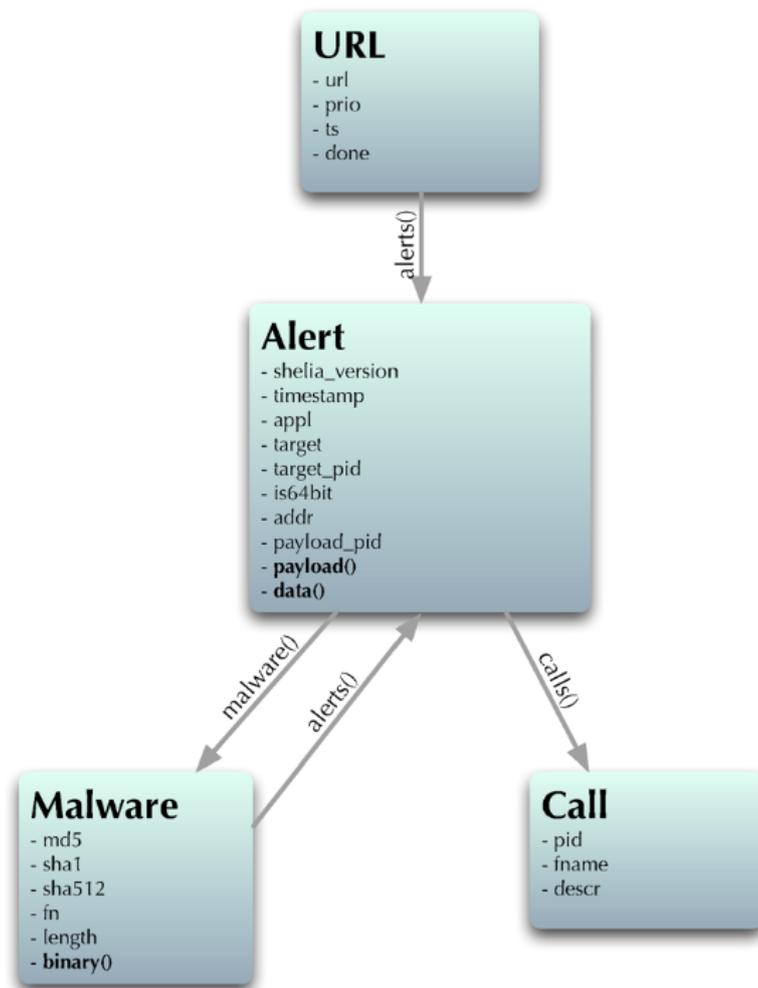


Figure 11.4: Shelia Dataset. The dataset contains four Objects (URL, Alert, Malware and Call) that are presented as list-boxes. The elements of each list-box are the Object identifiers. Methods are shown in bold. Arrows depicts the references among Objects.

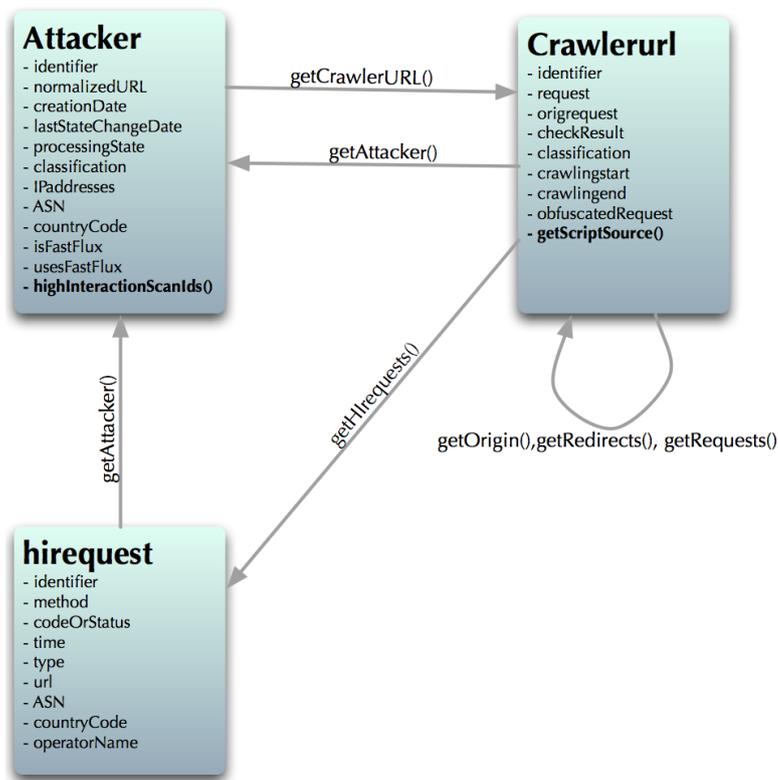


Figure 11.5: HSN Dataset. The dataset contains three Objects (**Attacker**, **Crawlerurl** and **hirequest**) that are presented as list-boxes. The elements of each list-box are the Object identifiers. Methods are shown in bold. Arrows depicts the references among Objects.

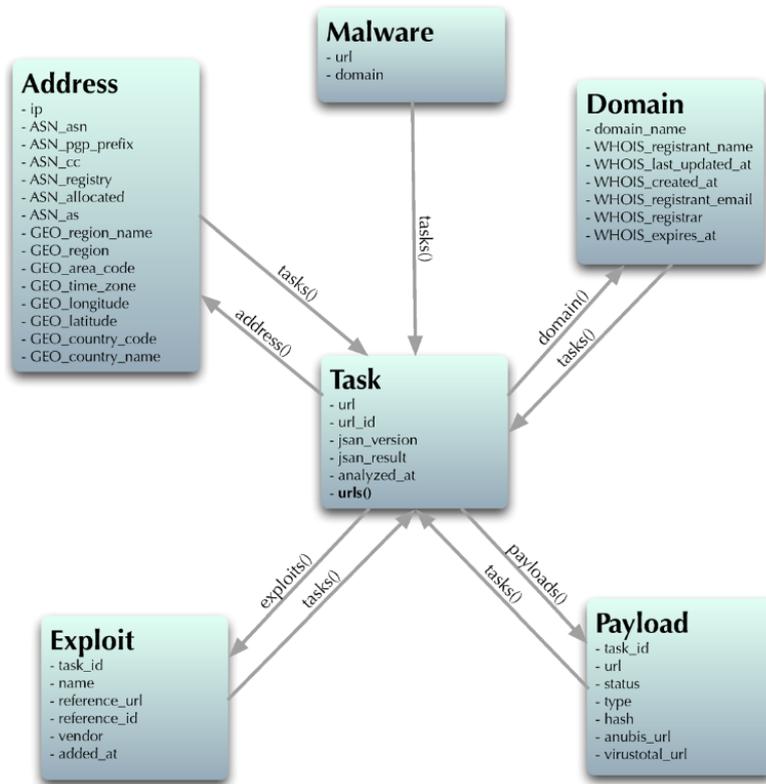


Figure 11.6: WEPAWET Dataset. The dataset contains six objects (Address, Malware, Domain, Exploit, Task and Payload) that are presented as list-boxes. The elements of each list-box are the Object identifiers. Methods are shown in bold. Arrows depicts the references among Objects.

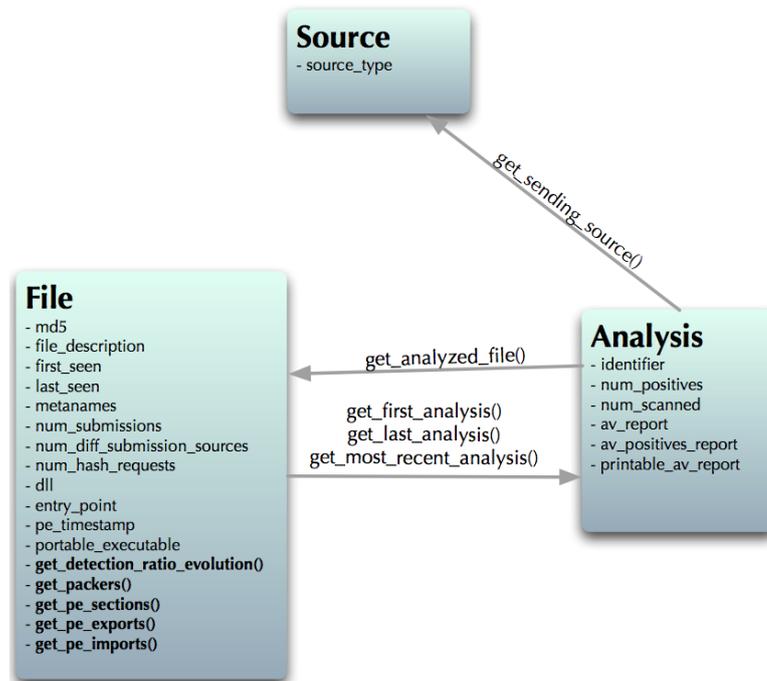


Figure 11.7: Virus Total Dataset. The dataset contains three Objects (Source, File and Analysis) that are presented as list-boxes. The elements of each list-box are the Object identifiers. Methods are shown in bold. Arrows depicts the references among Objects.

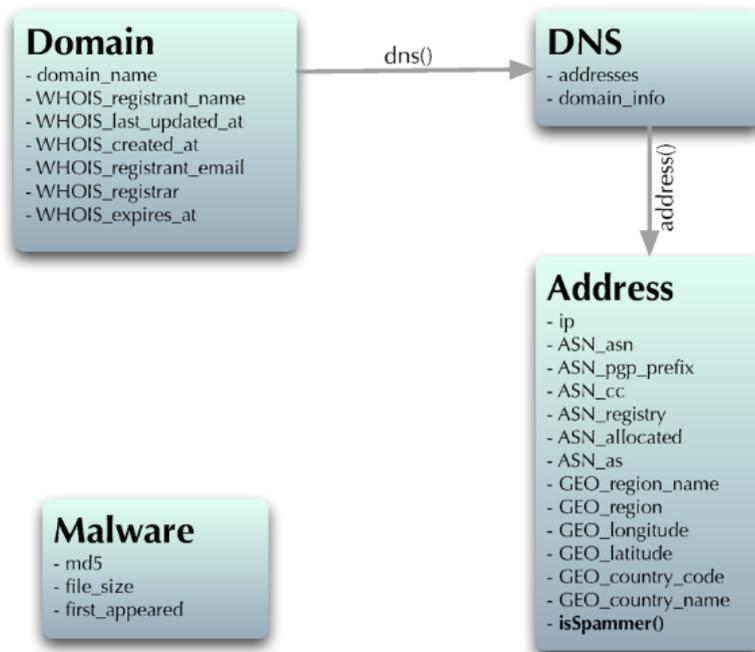


Figure 11.8: FORTH Dataset. The dataset contains four Objects (Domain, DNS, Address and Malware) that are presented as list-boxes. The elements of each list-box are the Object identifiers. Methods are shown in bold. Arrows depicts the references among Objects.