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

WORLDWIDE OBSERVATORY OF
MALICIOUS BEHAVIORS AND ATTACK THREATS

D06 (D3.1) Infrastructure Design

Contract No. FP7-ICT-216026-WOMBAT

Workpackage	WP3 - Data Collection and Distribution
Author	-
Version	0.1
Date of delivery	M9
Actual Date of Delivery	M9
Dissemination level	Public
Responsible	FORTH
Data included from	POLIMI, NASK, FT, EURECOM, VU, HISPASEC, SYMANTEC

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. Hervé Debar
Rue des Coutures, 42
14066 Caen
France

e-mail: herve.debar@orange-ftgroup.com
Web: http://www.wombat-project.eu
Phone: +33 23 175 92 61
Fax: +33 23 137 83 43

Contents

Abstract

This document contains a description of the WOMBAT architecture and a high level design of the new sensors. The WOMBAT architecture is covered by a comprehensive review of all its components. Part of this architecture is also the data sources and especially the new ones that will be implemented as part of the WOMBAT project. Each of them will be described in the design level, focusing on the way that they will be integrated with the WOMBAT infrastructure.

1 Introduction

The purpose of this document is to present the architectural design choices that have been made in the context of the WOMBAT project in order to facilitate the collection as well as the analysis of data related to Internet threats.

The ultimate goal of the WOMBAT project is to offer a better understanding of the threats computer systems are facing. The approach followed by the project to reach this goal is to start from real, actual data. The line of actions is made of three distinct steps. In the first one, we build a framework to collect data, in the second step we extract metadata out of the obtained raw data, and in the third one we carry out various analyses on the metadata in order to produce meaningful results on the modus operandi and on the strategies of the malicious actors responsible for the current and future Internet threats.

This document presents the data collection framework required for the first step. It also addresses the issue of how this data collection can be used for the metadata extraction phase.

The scope of this document is limited to the design of the general architecture. It offers a high level description of the various functional elements that come into play in this architecture, their respective roles and interactions. It offers some usage scenarios to describe how we do plan to use this framework in the subsequent work packages. It also highlights how the framework is able to integrate new data feeds offered by external parties or, conversely, how third parties could use the data we collect in order to carry out some specific analysis on their own.

The details of the specific new sensors defined and developed within the WOMBAT project to populate this framework are to be found in a companion deliverable, namely *Design and prototypes of new Sensors - D3.2* .

The rest of this document is made of two distinct sections. In section 2 we offer a high level overview of the architecture itself (section 2.1) and briefly present its main components, namely: the data sources, the WAPI and the centralized DB (section 2.2). That section ends with a brief presentation of the foreseen usage of the presented architecture. The second main part of the document, section 3, offers a more detailed description of each class of components. Section 3.1 describes WAPI, the WOMBAT API, that aims at providing a common method to query the various data feeds composing the architecture. Section 3.2 describing the already existing data feeds that we have at our disposal.

For each of them, we describe their current integration in the architecture as well as the plans for an enhanced integration in the coming months. Section 3.2.5 covers the external data feeds, the owners of which have expressed interest in being integrated, in one way or another, to the WOMBAT architecture. Section 3.3 briefly introduces the new sensors that are to be developed in the context of WOMBAT, leaving a detailed presentation of these techniques for the other deliverable D3.2. Last but not least, section 3.4 presents the centralized database. Its current status is described as well as plans for future enhancements. Some simple usage scenarios highlights how complementary this system is to the various autonomous data feeds we have presented before.

2 Architectural Overview

2.1 General Design

The architecture we present in the following pages has been designed with a few key principles in mind, namely: openness, scalability, timeliness, ease of use and controllability. These principles stem from constraints and/or desires that had been expressed by the various members of the consortium and also by the numerous partners attending the first WOMBAT workshop held in Amsterdam in April 2008 and documented in deliverable *D2.1 : Workshop* [1]. We describe each of them hereafter to help the reader understand the rationales that motivate the infrastructure described afterwards.

- **Openness:** The WOMBAT consortium has no intend to remain a closed circle. On the contrary, we acknowledge the existence, outside the consortium, of important actors also involved in the collection and analysis of threats related data. Some of these actors were actually present at the first WOMBAT workshop organized in Amsterdam in April 2008. They do come from all over the world. It is our goal to implement a system capable of welcoming their contributions, i.e. their datasets or their analysis of our datasets. The cost for them to integrate into our system should be as small as possible.

- **Scalability:** We do foresee to collect a possibly very large amount of data coming from many different sources. The total amount of sources is likely to evolve as the project goes since, hopefully, our increased visibility will lead new external contributors to join forces with us. Therefore, we must propose a system that is able to grow dynamically. Also, the total amount of data collected by all participating collectors is likely to be huge. A naive solution that would consist in gathering every possible source of information into a single place is, therefore, not acceptable.

- **Timeliness:** The implementation of this infrastructure is not a goal on its own. Its sole raison d'etre is to enable metadata extraction and, hence, knowledge mining. We want to be able to start working on these tasks as early as possible even if

[1] The proceedings are available at `http://ieeexplore.ieee.org/xpl/tocresult.jsp?isnumber=4627301&isYear=2008`

the implementation of the infrastructure is not yet as polished as it should. In other terms, the proposed solution must be such that it can take advantages of data sources that would not, in a first phase, fully comply with the integration guidelines advocated by the architecture. In other words, the architecture must be flexible and able to deal with, if needed, "dirty hacks" to enable us to benefit from external sources as quickly as possible.

- **Ease of use:** We want to offer a simple way to interact with the various data sources at our disposal within the infrastructure. However, we have to expect these sources to be of many different kinds. Some may offer something as simple as data files whereas others will be complicated SQL Databases. We need to offer a simple mechanism to hide, as much as possible, the differences and complexities of these sources without forcing them to modify their internal structures. Clearly, the idea of offering guidelines for a generic (as much as possible) API is appealing for that purpose.

- **Controllability:** Experience shows that the sharing of some of these threats related data may sometimes be problematic, be it for privacy, confidentiality of liability reasons. There are valid concerns linked to legal, business or even political reasons that can justify why a data source may or may not be shared with some or all the partners. The infrastructure must offer a simple, convenient and flexible way to handle these access control problems.

These principles have guided the design of the architecture which is mostly made of three distinct components: (i) a diverse set of data sources, (ii) the WAPI and (iii) a centralized database. The data sources, as the name implies, are the sources of the data that we need in order to carry out the tasks defined in steps 2 and 3 of the WOMBAT project. These sources can be of different kinds, as we describe in Sections 3.2 and 3.3. What is important to understand is that these data sources are completely autonomous. The WOMBAT infrastructure does not impose anything neither on the way they are implemented nor on the data they are able to provide. The WOMBAT infrastructure does not consider, for instance, a complete copy or replication of the data they can generate into a centralized database. Such requirement would impose heavy constraint on the data owners and would violate several of the principles explained before, namely openness, scalability and controllability.

The WAPI (WOMBAT API) is a simple mechanism that we want to offer in order to facilitate the integration of new sensors while preserving the ease of use principle. The idea behind WAPI is to enable, for example, analysts to write programs that would use data from several data sources without having to worry about the specific querying

mechanisms offered by these data sources. Also, it enables the data owners to modify their internal data structure without having to notify the external users. However, in order to cope with the timeliness principle, we will, in a first stage, consider using the existing interfaces offered by some of these existing data sources. The new sources, though, will be WAPI compliant as soon as WAPI will be stable and mature. Also, at that stage, every new external source willing to join the WOMBAT infrastructure, will be required to provide a WAPI compliant interface. It will be the responsibility of the data owner to define what he is willing to share with whom, in which format (e.g. raw data or pseudonyms or anonymous Ids.). WAPI will contain the primitives to enable such access control.

Offering a purely distributed system, enriched with a generic API, may seem appealing and sufficient at first but experience shows that, for several reasons, such an architecture will not be able to fulfill efficiently the needs that will arise when dealing with metadata extraction and knowledge mining. Such system has to complemented by a centralized database that will, by no means, not contain a complete copy of all data sources but, instead, a certain amount of well identified aggregated information. The reasons for doing this are explained with some more details in Section 2.2.3. The centralized DB itself will be WAPI compliant and will use WAPI to talk to the various data sources it will collect information from.

So, from the previous discussion, it should be clear now that the WOMBAT infrastructure is a hybrid schema of autonomously implemented data sources owned either by WOMBAT partners or by external parties. A generic API, named WAPI, will offer a simple way to write analyser programs taking advantage, transparently, of a number of sources. However, in a first step, integration will be based on the existing interfaces provided by these data sources. A centralized database will also be available to provide efficiently aggregated information coming from several sources, acting as some sort of a proxy. In the next Section (2.2), we provide some more input on the various classes components and defer until Section 3 the actual description of instances of these components.

2.2 Components

2.2.1 Sources

There are different kinds of data sources that we are considering in the context of the WOMBAT architecture. First of all, some data sources simply provide information about events that they have observed and that are related to observable threats. We call these data sources the "data feeds". On the other hand, there are data sources that, when

presented with some sort of input related to an observable threat, are able to tell us what information they have about it. We call these sources the "analysers". Last but not least, there are data sources that combine both aspects, we call them the "hybrids". This informal classification simply aims at clarifying the classes of data sources. It should certainly not be seen as some sort of taxonomical attempt to define our data sources. We say a few more words about each of them in the following subsections.

Data Feeds

The data feeds offer simple facts about threat-related events that they have observed. Typical data sensors that fall within this category are the honeypots, darknets, and other honey farms that are deployed on the Internet. Also, several organizations do keep track of IP addresses of spammers, phishing sites and other botnet zombies.

It is worth noting that the amount and type of information provided by these sources varies greatly. For instance, on one extreme, the Leurré.com distributed system of honeypots implements a whole SQL database which is able to provide geographical information, reverse name lookup, Operating System fingerprinting, etc of observed attacking IPs. On the other extreme, some DNS Black List servers (DNSBL[7]) may simply provide a Boolean answer when asked if a given IP is known to be, or not, a spamming IP.

Also, the amount of time during which a given information remains available from a given data feed is highly variable. Here to, the Leurré.com [12] database is an extreme case where no information is ever removed, enabling everyone to query the system for data as old as 5 years. DNSBL, on the other hand, represent again another extreme since its content is regularly updated, for very good reasons.

From the previous examples, it comes that the integration of a given dataset into the architecture requires a good understanding of the data it offers as well as its durability. Since we aim, within WOMBAT, at carrying long term, strategic analysis of attack trends, it is worth trying to store into our centralized database, with a timestamp, the data that are likely to disappear quickly. This will ensure their availability for future use. This, of course, is only possible if the amount of information to be stored remains affordable.

Analyzers

The analysers are data sources that can be queried with an external observed event in order to obtain further information about it. VirusTotal [21], from Hispasec Systemas, is a very good example of such a system. Indeed, one can submit to this public web site, a suspicious binary file and it will be inspected by 36 distinct anti virus scanners. At

the end, the results of all these analysis are stored in a single report which is sent back to the submitter.

In this specific case, the analyzers provide important data that are worth being collected in order to study observed events. Similarly to the previous case with the DNSBL servers and a given IP address, the results provided by Virustotal for a given piece of malware may vary as time passes since the antivirus signatures are updated very frequently. Therefore, we do wish to study, e.g., over a long period of time how antivirus products are able to react to new threats, we need to submit regularly to Virustotal the malware we are interested in and we need to store, with a timestamp, the results provided for it.

We acknowledge the fact that the separation between analysers and datafeeds is rather artificial. Indeed, we could say that, e.g., a DNSBL is an analyzer since one can query it with an externally observed IP address to find out if others have also seen it sending spam. Nevertheless, we feel that, for the sake of clarity, stressing the existence of these two classes of data sensors help understanding the kind of data we are interested in.

Hybrids

The logical follow up to the previous point is the identification of data sources that, in some cases, are able to provide further information on an externally observed event and that, in other cases, are able to provide information on events they have observed by themselves.

One such example of hybrid analyzer is the private interface that VirusTotal offers to the antivirus companies. On one hand, the AV companies can use VirusTotal as an analyzer by submitting malware to it, as everyone can do it through the public interface. On the other hand, VirusTotal will alert the AV companies when it observed a malware, received through the public interface, which is not recognized by all AV products. In that case, VirusTotal plays the role of a datafeed.

2.2.2 WAPI

WAPI, the WOMBAT API, aims at improving the ease of use and of the various data sources that will be available in the WOMBAT infrastructure. As already eluded to, each data owner will be responsible for deciding what he is eager to share, in which format and to whom. WAPI will provide the primitives to easily specify some sort of access controls. The role of WAPI will be to make it as transparent as possible to programmers the specificities (naming schemes, querying methods, etc.) of each data source. Of course, differences will still exist since the sources offer neither the same

amount of information nor the same level of abstraction. However, WAPI-compliant interfaces will ensure that the same object will be named and presented the same way by all data sources that know about it. More details about WAPI is offered in Section 3.1

2.2.3 Database

There are, at least, three reasons to augment our distributed architecture with a centralized database.

- First of all, as we have already seen in the previous examples, some data sources do not store for a long period of time the data they offer at any point in time. Not storing them in a dedicated data structure for later use dramatically reduces the usefulness of the integration of this data source into the overall architecture.

- The second reason has to do with efficiency. One can foresee that there are certain classes of queries such as, e.g. the top-ten attacked ports, that are likely to be frequently asked. This is all the more true that we will start exposing some of our data through a public web site. Assuming we have a public web page that presents the histogram of the most frequently attack ports over time, it would be extremely inefficient to query each and every data source able to provide information on this topic whenever someone downloads the page. Instead, one would prefer to have the page built on the fly from some pre-computed tables containing aggregated information pulled from the various data sources.

- The third reason has to do with (the lack of) transparency. Indeed, for efficiency or privacy reasons, one may want to prefer to give access, through WAPI, to a proxy server that will be responsible for querying all available data sources rather than exposing all of them. Such proxy server should logically be implemented on the same machine as the centralized database so that he could, possibly, avoid any communication to the individual data sources if the DB already contains the aggregated information he is looking for.

The current status of the centralized database, descriptions of queries that can be run against it as well as a couple of usage scenarios are given in Section 3.4

2.3 Usage

2.3.1 Tactical vs. Strategic analysis

At this point, it is worth reminding the reader that the ultimate goal of the WOMBAT project is to better understand the modus operandi of the attackers and their strategies. We aim at studying trends that manifest themselves over long period of time in order to warn stakeholders as early as possible of new or increasing threats. We are pursuing a fundamentally different goal than the one classical response teams are after. Their responsibility is to address in near real time newly observed events by, e.g, creating new antivirus signatures to stop the spread of a new malware. This is what can be referred to as a tactical approach to the fight against Internet crimes. WOMBAT deals with the strategic battle instead. Both are, of course, complementary but involve different times scales. Tactical battles are won, or lost, in terms of minutes or hours whereas strategic approaches involve days, weeks or months. This is the reason why, so far, we have avoided mentioning any notion of automated alert mechanism or "push" protocol to convey information, preferring an infrastructure mostly relying on "pull" methods. As we acquire more experience with the metadata extraction and knowledge mining, we may have to revisit this design choice but, as of know, it appears to be a reasonable and efficient choice for the framework we are working on.

2.3.2 WAPI

As explained before, WAPI can be used either for programmers to write scripts that would process objects retrievable from several sources without having to worry about the specificities of each data structure and querying method. It will also be used by the maintainers of the centralized DB to update its content by querying other sources. Last but not least, it will also be a usable interface to query the centralized database.

WAPI is a key element to facilitate the integration of new sources. When WAPI will be stable and mature, we will make it a requirement to be WAPI compliant in order to integrate a new data source into the infrastructure.

2.3.3 Database

The centralized database, as explained before, will be used for two distinct purposes. On one hand, it will act as a proxy to hide, by choice or by necessity, the various data sources that can be queried to answer a given question. On the other hand, it will serve as a long-term repository for aggregated information that (i) are more efficiently stored

once than queried frequently to a number of sources or that (ii) are likely to disappear from the data sources that produced them (e.g. DNSBL).

3 Components

3.1 WAPI

Many of the sources involved in WOMBAT already provide or plan to provide solutions to allow data consumers to take advantage of their data sets (in case of data feeds) or of the results of their analysis (in the case of analyzers). We can identify two main disadvantages in the currently available solutions:

- Many of them are not easily scriptable. Many sources provide web-based interfaces that, while easily consultable by the human operators, are not convenient for automated analysis. The ultimate goal of WOMBAT is that of building automated analysis techniques to generate intelligence out of these datasets. The presence of easily scriptable interfaces is thus of prime importance.

- The interfaces are very diverse in structure and methodology. For instance, Leurré.com provides direct access to the underlying SQL database to trusted partners, while DNSBL provides a DNS-based technique to query about the presence of a given IP address in the blacklist. The analyst aiming at taking advantage of multiple data sources to perform correlations among the different dataset is thus forced to implement ad-hoc plugins and parsers for each data source. This process is not necessarily a simple task, and requires the analyst to deeply understand, for instance, the schema of the SQL database provided by the source.

We propose to define a common WOMBAT API (WAPI) to be shared among all the participants in order to address the above issues and simplify the task of the data consumer willing to take advantage of these datasets. The definition of this API is thus meant both for internal use, to ease the data sharing among project participants, and for the future external consumers of the output of the deliverables of the project.

Moreover, we intend to make publicly available the complete specification of the API as well as examples of implementation. We hope in fact that, with the increase in visibility of the project, other data sources will be willing to take advantage of our work.

3.1.1 Requirements

In the preliminar design of the WOMBAT API, we have identified a number of desired requirements and features.

- **Data control.** The WAPI architecture must allow each source to control the nature of the shared information. It is likely that some sources will be willing to share only a portion of the information stored in their datasets. Also, sources must be able to dynamically pre-process the information provided to data consumers, for instance applying anonymization algorithms.
- **Access control.** The confidentiality requirements of the various sources can lead to the definition of different trust levels for the data consumers. Data consumers belonging to different trust levels will be allowed to access information of different nature. While the practical implementation of these trust levels is not considered of prime importance in the short term, the WAPI architecture must be easily extensible to implement more sophisticated types of access control.
- **Common vocabulary.** The information shared within the WAPI must be based on a set of common definitions. We want to ensure that all the sources will share a common vocabulary and common definitions for all the shared concepts, such as IP addresses or malware hashes.
- **Extensibility.** Many sources in WOMBAT are still in an experimental phase and they are likely to evolve in the next years. This evolution may consist, for instance, in applying new analysis techniques to the dataset, enriching it with new types of information. Therefore we do not want to bind the WAPI specification to a specific set of primitives defined a priori for each type of source.
- **Client flexibility.** The technologies used for the practical implementation of the WAPI must not bind the data consumer to the usage of any specific programming language.

It is important to notice that the requirement of a common vocabulary conflicts with that of extensibility. In fact, in order to achieve a common vocabulary we would need to explicitly enumerate all the informations that can be provided by each source on each security phenomenon, generating an ontology. Because of the extensibility requirement and because of the complexity of a similar task, we do not want to follow this path. The WAPI must be a practical instrument to ease the integration of the different sources and the future analysis task within the project: its specification and its implementation must be kept as simple as possible.

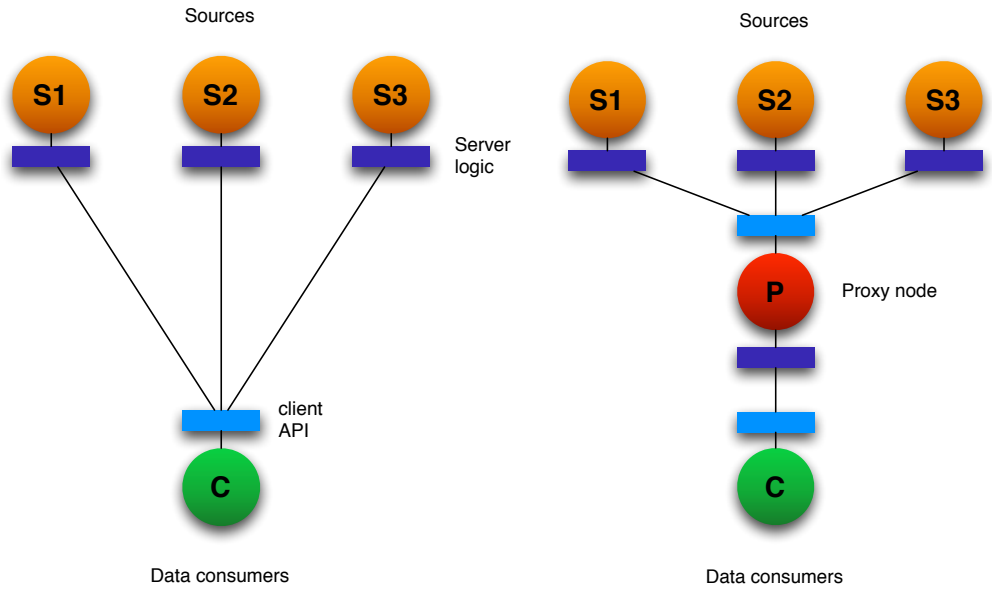


Figure 3.1: WAPI architecture.

We have thus decided to define within the WAPI specicifation only a set of high-level concepts shared by all the datasets. We do not instead specify in depth all the possible actions or informations associated to each of these objects. These high-level objects can be considered as a naming skeleton upon which every specific WAPI implementation will build. The specific actions and attributes of these high level objects implemented by each WAPI service can be dynamically defined and are not part of the WAPI specification. For instance, VirusTotal WAPI, when queried, will offer a method to retrieve information on the ability of AntiVirus software to recognize a certain malware sample. The Anubis [2] WAPI, instead, may reply to the same query with various methods to retrieve behavioral information on that sample. This will be possible taking advantage of a small set of primitives, described more in depth in Section 3.1.4.

3.1.2 Architecture

The WAPI aims at providing data consumers a remote access to the information collected by the sources. The WAPI is a remote API allowing consumers to retrieve remote information from sources according to a given protocol. Figure 3.1(a) shows the envisaged

architecture for the WAPI. Three components can be identified:

- **Server logic.** Every data provider must implement a server application converting the protocol interaction into a set of queries to their dataset. For instance, in the case of an SQL database, each protocol interaction will be converted to one or more SQL requests generating the information required by the client. The logic at server side is thus responsible for hiding the specificities of the underlying SQL dataset from the data consumer.
- **Client API.** On the client side, a programming interface is needed to generate the protocol interaction required to retrieve the information. The choices made in the implementation of the server-side logic and the communication protocol must allow the client implementation to be independent from any specific programming language.
- **Communication protocol.** The communication protocol will be based on a PULL communication pattern. The client will generate queries to the server following the primitives defined in Section 3.1.4 and the server will provide back to the client the corresponding answers. We have chosen to implement the communication protocol using the SOAP protocol. The reasons for this choice are twofold. Firstly, NASK already has experience in the usage of this protocol and is willing to offer its expertise in the implementation phase. Secondly, the SOAP protocol is very well supported by a high number of client libraries making it the perfect choice to satisfy the client flexibility requirement.

A desired feature for the communication protocol is the ability to support proxied setups such as that represented in Figure 3.1(b). Such a setup allows to offer to data consumers external to the WOMBAT an "aggregated service" transparently offering to the users the information resulting from the union of all the perspectives provided by all the sources that will support the WAPI. For instance, we want to offer to the data consumer a "super-source" P transparently providing all the information offered by $S1$, $S2$, and $S3$ on a certain security phenomenon.

3.1.3 WAPI concepts

The specification of the WAPI protocol is based on three concepts: objects, attributes and methods.

Among all the WOMBAT sources, we can identify a small set of high level concepts that are represented under different perspectives in all the datasets. These high level

concepts are what we call here WAPI objects. Every WAPI object can be associated to a set of attributes and methods.

Attributes are information items that are provided by a certain source upon instantiation of a WAPI object. For instance, the object modeling an attacker can be associated to an attribute "IP address".

Methods are instead additional queries associated to a given WAPI object to retrieve additional information about it. While the attributes are computed and provided to the WAPI client at every instantiation of a WAPI object, methods allow to retrieve on demand more costly information. For instance, a `geolocation()` method can be associated to an attacker object to retrieve information about the geographical location of the corresponding IP.

The WAPI objects introduced here have been derived from the analysis of the *input data and information* to the WOMBAT system proposed in Section 4.1.3 of deliverable *D2.3 : Requirements analysis*. Figure 3.2 represents the relationships among the different WAPI objects under the form of a UML class diagram. Some common attributes are specified for every object. The definition of most of the attributes and of the methods is left to the specific implementation of each WAPI source, since they heavily depend on the characteristics of the dataset.

Attacker and victim

Every security event observed by a source is normally characterized by an attacker and a victim. In the case of server-side attacks, an attacker corresponds to an Internet host actively generating unsolicited traffic towards its victim. In the case of client-side attacks such as those observed by the NASK honeyclients, the attacker is a server (e.g. a Web server) trying to run malicious code on the honeyclient once contacted.

These WAPI objects are thus associated to a set of common characteristics, such as an address (in most cases, an IP address), an optional TCP/UDP port, and a type that identifies their role in the conversation (client or server).

Event

This WAPI object models an instance of a security event observed between an attacker and a victim at a given point in time. An event is identified by the address of the attacker, of the victim, and by a timestamp.

The notion of event is very generic, and strongly depends on the type of sensor that collected the information. For this reason, the generic concept of event needs to be further specified.

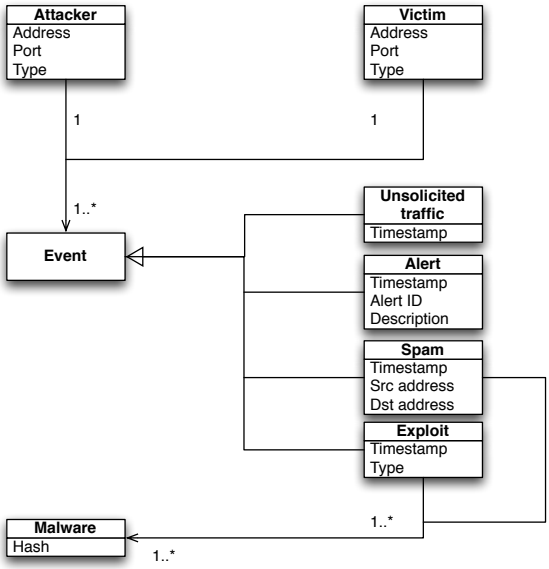


Figure 3.2: WAPI objects

- **Unsolicited traffic.** Firewall logs or low-interaction honeypots generate events corresponding to the detection of blocked traffic or unsolicited traffic in general. This information often cannot be easily mapped to a specific root cause, and is characterized by a timestamp and by additional information that is sensor dependent.

- **Alert.** Intrusion Detection Systems or Early Warning Systems generate alerts generally associated to a unique ID and to a description of the alert.

- **Spam.** Spamtraps and similar spam-monitoring techniques will generate an event for every observed spam mail. Further parsing of the mail content can lead to the identification and download of a malware sample.

- **Exploit.** More sophisticated honeypots (such as Argos and SGNET) and honeyclients generate this kind of events whenever an exploit is detected. The ability to understand a successful exploitation is normally coupled with the ability to download malware samples, and provide more detailed information on the nature and the type of exploit. Most of the available information heavily depends on the technology of the sensor that triggered the event.

Malware

This WAPI object models malicious executables. While some data sources such as honeypot deployments will be able to generate a link between a malware sample and the observed exploitation event, in many other cases (e.g. analyzers such as VirusTotal) this link will not be present. For analyzers such as VirusTotal and Anubis, the malware object will actually be the root of a hierarchy of information that is specific to each analyzer.

Custom objects

The list of WAPI objects provided until now is by far not exhaustive. We can identify in every data source other specific concepts that may be wrapped into WAPI objects. For instance, some datasets may take advantage of clustering techniques to define *classes* of exploits. All these source-specific concepts are not modelled here. As we will see in Section 3.1.4, the WAPI protocol primitives do not pose any restriction on the type of object that can be instantiated for a given source and full freedom is left to each data source to implement interfaces to more specific concepts.

Special objects

We plan to define to additional objects with special capabilities: the `dataset` object and the `timeiterator` object.

The `dataset` object is a special object providing aggregate information on the state of the dataset. This object can be used to implement methods providing statistics on the state of the observed events. For instance, a honeypot deployment such as Leurré.com can implement a dataset method to compute statistics on the top 10 attacking sources witnessed in a given timeframe.

The `timeiterator` object addresses instead the common need to have information on the evolution in time of the observed events. Figure 3.2 provides a schematic structure of different aspects of a *specific* attack instance. Timeiterators aim instead at iterating over time among different instances of an object, when the characteristics of the dataset make it possible.

3.1.4 Protocol primitives

The interaction among WAPI objects, attributes and methods is defined by a small set of primitives that need to be implemented by any WAPI client and server. We describe here their general structure and purpose.

list_objects()

This primitive requests to the WAPI server the list of all the WAPI objects currently implemented, together with the list of attributes required to instantiate each of them. For instance, the output of a WAPI server able to provide information on all the standard WAPI objects will be:

object_type	required_attributes
Attacker	address
Victim	address
Event	src_address,dst_address,timestamp
Malware	hash

list_methods(object_type)

This primitive allows to query a WAPI source for the list of supported methods for an object type. Once queried, the WAPI server replies with the list of methods currently supported and with the list of arguments required for the execution of that method, if any.

generate_object(object_type,required_attributes)

This primitive allows the generation of a WAPI object starting from a set of attributes identifying it. The object_type is the string representation of the object type as returned by `list_objects`. In order to produce an unambiguous instantiation, a value must be assigned to each required attribute specified by the output of `list_objects`.

If the object is known to the WAPI source, an identifier of the object will be returned, as well as the list of attributes associated by the source to that object.

run_method(object_identifier,method_identifier,arguments)

Given an object identifier and a method name, this primitive allows to request to the WAPI server the execution of such method.

3.1.5 Conclusion

The characteristics of the WAPI architecture reflect the requirements listed in Section 3.1.1.

The server logic is in charge of carrying on the SOAP interaction and of converting the objects instantiations and the method calls into queries for the internal dataset. Each data source has complete freedom in the choice of the methods to be provided to the data consumer, and can process the generated data with anonymization routines before sending it back to the requestor.

The dynamic generation of the list of available methods can easily allow in the future to implement access control based on trust levels. Once an authentication method is provided to determine the trust level of the user, each trust level can be mapped to a set of methods considered safe for that user class.

We have defined an infrastructure of common concepts, called WAPI objects, upon which each data source can build methods to be offered to the data consumer. While no dictionary is defined a priori for these methods and for their syntax, we consider the task of manually uniforming such syntax a small cost when compared with the increased flexibility of the approach.

The dynamic declaration of the methods for the WAPI objects, separated from the specification of the protocol, allows to easily add new features without impacting the implementation of the clients or of other sources implementing the WAPI.

Finally, the protocol API allows to easily generate application level proxies offering to data consumers a set of methods resulting from the union of all the methods offered by the proxied sources. If all the WOMBAT sources will implement such API, we will

be able to easily offer to external data consumers an aggregated vision over all the information collected within the WOMBAT project.

3.2 Existing Sources

3.2.1 Hispasec

Hispasec

Hispasec provides feeds of different nature. On the one hand, VirusTotal will be able to provide the system with a wide amount of malware samples that will help later to identify different types of malicious software. Together with the binary samples themselves, we'll provide metadata associated to the file sent that will help enrich the information about the sample. In the other hand, we'll use internal services from Hispasec, associated or not with VirusTotal, that will provide URLs associated to malware, fraud sites and exploits, and also malware identificators that are referencing associated URLs. That information is important to know vectors of the infections in one hand, and locations for components and drop sites for information stealing malware for instance.

VirusTotal

VirusTotal is a public online service that lets users check files with a list of antivirus engines. Initially made public in June 2004, it featured 11 antivirus engines. In this moment it uses more than three times that number (36 to be exact). Although it was initially thought as an easy service for regular users so they could check the possible maliciousness from a given file (typically an attached file received by email that his own resident antivirus dont recognize as malicious), with time it has been gaining respect in the security community and today it is used daily also by entities like CERTs, universities, response teams from a list of security companies, researchers and even antimalware companies.

VirusTotal Interfaces

Simplicity in the usage of the service is one of the key elements to make it a popular tool. The overall way of working is quite simple. Users are given a couple of interfaces to send files to the system: a minimalistic and tool-like web and an email address.

- Using the web interface is the preferred method by normal users: once the file is sent, the user is informed about the queue status (if the service is very loaded)

and about how long the scanning is estimated to start. Once the scanning begins, first the user is shown with metadata of different nature related with the file: part of that information is very basic (file size and type, different hashes, etc.) while other is more useful for security professionals able to read such specialized information (that is the case, for instance, of basic Portable Executable info). Once this information is shown, the file is scanned with all antivirus engines available at the moment, and results are shown in real time. One small tool were also developed by Hispasec to make users able to send files to the service using a Microsoft Windows context menu option. This is a very simple tool that basically opens the browser and sends the file directly to the service, so the user does not have to open the browser, select the file to scan, etc. We also know of the existence of third-party tools like browser bars or plugins for malware research tools - that makes use of both email and HTTP interfaces of VirusTotal, both for mass automations and for deep scanning procedures of single files.

- The other way to get scan results with VirusTotal is the email interface: users can send files to scan@virustotal.com, and using different email subjects, they can get their scan result in plain text (scan) or XML (scan+xml). This second version is especially useful for automations, and it is widely used by heavy users like CERTs or security companies that send thousands of files each day, and uses that reports for malware classification.

In both cases, this kind of professional users gives the feed received by VirusTotal a better level of quality, not only because theyre more prone to be real malware, but also for being fresh, something very useful for characterization of new threats.

Other factor that makes VirusTotal popular in different zones of the world is the availability of the web interface in several languages. That factor makes it more comfortable for non-expert users, and provides the service with samples from very different locations that are of interest in terms of malware sources (i.e. Russia, China, Brazil, etc.).

VirusTotal as malware feed source

The popularity of the service has made it process a quite respectable amount of files per month. In the first month of public life of VirusTotal, the service received more than 8.400 files detected as malicious by at least one of the engines used. A couple of years later (September 2006), the number of samples detected as malicious raised up to 120.000 per month: thats more than 14 times more than the original amount. In June of 2008, the number of samples of this kind raised up to more than 680.000: more than 5 times the number of samples seen in September 2006. The uniqueness ratio of files

received, using the MD5 hash as basis for that check and in the context of that month, is usually between 70 and 80%.

When a sample is detected by at least one antivirus engine as malicious, it is sent to the sample distribution subsystem. If this sample has been previously sent to any antivirus laboratory we check repeated files with their md5 hash and a database of previously sent items - it is discarded for distribution, as it would make no sense to send repeated files to their labs. Once a file is in this stage, a list of rules is applied for checking which vendor is interested in receiving it. The standard rule for sending that sample is to send the sample to those vendors that dont detect it if at least any other vendor does it. Besides that, samples detected with heuristic or generic signatures are also sent to that given company. The first rule is of obvious usage, as antivirus labs are interested in files that they dont detect but others do. The second rule is important also, as antivirus vendors are interested in knowing how their heuristic engines are behaving with new in-the-wild threats. This way it is easier for them to check if theyre having any problem with false positives, and they can also have a deeper look to the sample to include it in a given family instead of detecting it only with heuristic or generic means. Other vendors use different rules, like adding extra rules for skipping files only detected by certain other antivirus products (prone to false positives), or receiving only files detected by at least N engines (once again, trying to avoid receiving false positives).

All this process is done in real time, as for antivirus labs studying this samples as soon as possible is very important to be able to protect their customers properly. Even discarding files that are wrongly identified as malware, with cases like false positives and corrupt samples for instance, the freshness of and volume of the feed sent daily makes VirusTotal a very interesting source of new malware for antivirus vendors, and thats also one of the most interesting aspects of the binary feed sent to Wombat. One difference in that sense compared to antivirus labs is that Wombat system will also process samples undetected by any of the antivirus engines used, as our experience dealing with this matter shows us that there are some very new threats are not detected even with more than 30 different products.

Metadata associated with sample feed

Besides the binary itself provided by the distribution system, samples are sent with extra information that is useful for a first level of classification of its nature. Basically, the information generated is separated in three different categories:

- **Basic metadata:** original file name, file size, date of reception, packer identification, source (anonimized in the case of samples sent by users), country code of

origin, file type and common name. In the case of samples that has been extracted from a given location, the URL will be included in the source field. Common name is the most common core name given by the N antivirus that detected it. Basically it uses fuzzy string matching with family names given in each antivirus. Together with this basic properties of files, we include a list of different hashes: MD5 (128 bit), SHA1 (160 bit), SHA256 (256 bit) and SHA512 (512 bit). Rivers MD5 is the de facto standard for file integrity check. The problem is that there has been attacks against it (called collision attacks) so two binary different files can have the same MD5 signature. SHA1 is also a quite common hashing algorithm used for file integrity check, and until now no successful attacks has been achieved against it. In order to be prepared for possible future problems with the SHA1 strength against collision attacks, we are also including SHA256 and SHA512 hashes, that uses far more bits for the signature hash generation, and therefore harder to compromise. Other kind of hash that will be included is SSDEEP, also called context triggered piecewise hashing. This new hashing technique have one big advantage over classical methods like MD or SHA: this fuzzy hashes can be compared to check binary differences between files. The basic feature of identifying a file with a more or less short signature is achieved, although this kind of signatures doesnt have a fixed length. A single byte changed in a file makes MD and SHA hashing generate completely different signatures, while a fuzzy hash would only change slightly. That way, it is easy to identify two binary different files that has a certain level of similarity. This kind of hashing is usually applied in computer forensics procedures.

- **Portable Executable basic information:** in the case of the file being a Portable Executable one, some of their characteristics are included, like Entry Point, TimeStamp, Machine Type, DLL identification. Besides that fields, information about the executable sections is included, with section name, virtual address, virtual size, raw size, entropy level and md5 hash. Imports done by the file are also included, with the name of the library used and the resource imported. In case of the file exporting resources (like in a DLL), the names of the exported names are included.

- **Antivirus reports:** here we include the information given by the antivirus engines when the file was scanned. Here we include the detection ratio, and for every engine detecting the sample: its name, version, last update time, result reported, annotations, and core name. Annotations can include, depending on the engine, information about packing, behavior observed, or general data in the form of an URL to the antivirus vendors database. Core name is the result of dissecting the

name given by the specific engine, getting only the specific family or denomination. Typically, antivirus engines name malware in 3 parts: the first one (head) is usually related to the platform or nature of the threat. The second one (core) is the name of the family assigned to that malware. The third one (tail) is usually a postfix used to include things like variant number or some special characteristics (like being of mass distribution, etc). A typical example of this would be Win32/Bagle.B, with Win32 as head, Bagle as core and B as tail. This information is used to generate the common name in the basic metadata part.

- **Tools report:** some extra tools are used by VirusTotal that are not antivirus engines, but that can generate information that can be useful for classification of the file. One example of this is TrID, a tool that describes the possible nature of the file with certainty ratios, identifying different types of executable files, office documents, image files, etc. This kind of information can be used later for deciding which subsystem must process it in order to extract the possible malicious behavior (for instance, it would make no sense sending a JPG file to an subsystem dedicated to analyzing PE files behavior).

List of URLs pointing to malicious content (URL -> Payload)

Other information that Hispasec will provide to Wombat will be a list of URLs related to malicious content, like malware, phishing sites or exploits. In the malware case, as described in the Basic Metadata section, the source of the file, in the case that were extracted from an URL, will be included there. That is quite important information as it links the malware itself to a source, that way the enrichment part of Wombat will have extra information about patterns. This URLs are captured using different subsystems:

- **VirusTotal scan URL feature:** we are working to integrate in the short term a new feature at the service that will let users check URLs besides than the usual way only with binaries. Together with the normal analisys with 30+ antivirus of the file pointed by that URL, the location will be checked with a list of site-checking services that will provide information about if theyre labeled as containing phishing sites or exploits.

- **Spamtraps:** Hispasec owns a network of spamtraps that we use for detecting, among other things, phishing (fraud) attacks against certain companies. Some of this spam messages also contains URLs that, from time to time, points to malware. Nowadays is very typical for malware distribution to use this mixed style: instead of worms that self-propagate as attachment in messages, many malware creators

upload the file to certain locations and then proceed to do a spam campaign to entice users to download and execute it using one excuse or another.Spamtraps: Hispasec owns a network of spamtraps that we use for detecting, among other things, phishing (fraud) attacks against certain companies. Some of this spam messages also contains URLs that, from time to time, points to malware. Nowadays is very typical for malware distribution to use this mixed style: instead of worms that self-propagate as attachment in messages, many malware creators upload the file to certain locations and then proceed to do a spam campaign to entice users to download and execute it using one excuse or another.

- **Public repositories:** there are, open to public use, certain sites that contains raw spam messages and URLs that are theoretically related to malware and other fraudulent content. We process this information to extract anything related to malware, phishing and/or exploits.

- **Crawlers:** We use our own crawling technology with URLs related to phishing and malware, accessing that way to new URLs pointing to malware. It is basically a proactive evolution of the pure reactive URL fetching from Spamtraps.

List of Malware using URLs (Malware -> URL)

Hispasec will also provide with the relation of malware files that are using specific URLs. That is very typical in downloaders and modular malware that downloads different components using one logic or another: dynamic configuration files and payloads, etc. This list will include the identification of the file (typically hashes associated with the file as described in the Basic Metadata section) and the list of URLs that are associated with it. This kind of information is specially interesting, given the nature of many fraud-oriented malware, as configuration files can be fetched, and stolen information destinations can be identified.

Basically, we use several subsystems that extracts URLs from malware samples: some of them are based on virtualization of environments, where URLs are collected as theyre referenced through the network subsystem. We also have sandbox technology that let us execute that malware samples and intercept that same network activity. This redundance is important for handling cases of malware that is aware of virtual environments, not running correctly on them. In both cases, we also apply different techniques to locate strings in the executables that can be URLs accessed by it. For that goal, we use different kind of unpacking techniques, something specially important given the big amount of samples that uses this kind of technology for complicating this kind of automated processing.

3.2.2 Leurré.com

Launched in 2003 by EURECOM, the Leurré.com project is based on a worldwide distributed system of honeypots running in more than 30 different countries. The main objective of the project is to get a more realistic picture of certain classes of threats happening on the Internet, by collecting unbiased quantitative data in a long-term perspective.

In the first phase of the project, the data collection infrastructure relied solely on low-interaction sensors based on Honeyd [44] to collect unsolicited traffic on the Internet. Recently, a second phase of the project was started with the deployment of medium-interaction honeypots based on the ScriptGen [36] technology, in order to enrich the network conversations with the attackers.

All the information collected by the sensors distributed in different locations of the Internet is retrieved by a central entity on a daily basis. This information is parsed and stored in a centralized database at different levels of aggregation. Moreover, the collected traffic is enriched with different types of contextual information (e.g. geographical localization, reverse DNS lookups, PE information on collected malware, ...). All the generated contextual information is stored in the database, and is used to generate a complete dataset on the evolution of Internet attack threats.

The participation to the Leurré.com project is open to any institution interested in taking advantage of such a dataset. In order to participate, the interested institution is required to install one of the honeypot sensors and receives in exchange access to the whole dataset. Information about the identity of the partners and the observed attackers is protected by a Non-Disclosure Agreement signed by each entity participating to the project to minimize legal concerns.

Data collection: Leurré.com 1.0

EURECOM has started collecting attack traces on Internet threats by means of honeypot responders in 2003. The first platform consisted of three high interaction honeypots built on top of the VMware technology (the interested readers in the platform configuration are invited to read [26] for more information). As shown in [26, 25], these first experiments allowed to detect some locality in Internet attacks: activities seen in some networks were not observed in others. To validate this assumption, it was decided to deploy multiple honeypots in diverse locations. With diversity, we refer both to the geographical location and to the sensor environment (education, government, private sectors, etc). However, the VMware-based solution did not seem to be scalable. First, this solution had a high cost in terms of security maintenance. Second, it required significant hardware

resources. In fact, to avoid legal issues it is necessary to ensure that these systems could not be compromised and could not be exploited by attackers as stepping stones to attack other hosts. For those reasons, a low-interaction honeypot solution were preferred, honeyd [44]. This solution allowed to deploy low-cost platforms, easy to maintain and with low security risk, hosted by partners on a voluntary basis. The low-cost of the solution allowed to build a distributed honeynet consisting now of more than 50 sensors distributed all over the world, collecting data on network attacks and representing this information under the form of a relational database accessible to all the partners.

Data collection: SGNET

The choice of using low interaction honeypots in the first version of the Leurré.com deployment was primarily motivated by a tradeoff between the level of interaction, and thus the richness of the collected data, and the cost of the sensors. High interaction honeypots have resource requirements that are too high to allow their deployment on low cost hardware offered by volunteering partners. Also, the deployment of high interaction honeypots poses security problems that are not easily addressable without requiring high maintenance cost.

This tradeoff led to the initial decision of using low-interaction techniques for the data collection, but led also to the experimentation of alternate solutions better balancing the two ends of this tradeoff. This led to the development of ScriptGen [36, 35].

ScriptGen is an automated technique that tries to *learn* the behavior of a given network protocol when facing deterministic attack tools. ScriptGen aims at being protocol agnostic: no a-priori assumption is made on the characteristics of the protocols. Its characteristics are reconstructed by looking at the sample set and partially rebuild the protocol semantics taking advantage of bioinformatics algorithms [43]. These algorithms, conceived to detect similarities among different DNA sequences, are used in ScriptGen to detect structural similarities in the protocol stream and identify protocol *regions* likely to carry a different semantic value. ScriptGen takes advantage of these semantic abstractions to build a FSM representation of the protocol behavior during the interaction of a client with a server as shown in Figure 3.3 for an excerpt of the SMTP FSM. This representation can be used to emulate the protocol behavior, and thanks to the semantic abstraction it allows to handle future instances of similar activities.

The properties of the ScriptGen approach allow to perform a completely automated incremental learning of the activities as shown in [35]. ScriptGen-based honeypots are able to detect when a client request falls out of the current FSM knowledge (a 0-day attack or, more exactly, a yet unseen attack) by simply detecting the absence of a matching transition. In such case, the honeypot is unable to provide a valid answer to

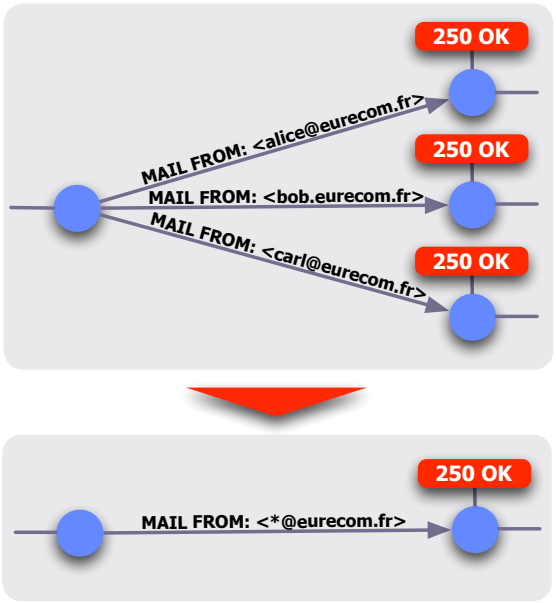


Figure 3.3: ScriptGen semantic abstraction

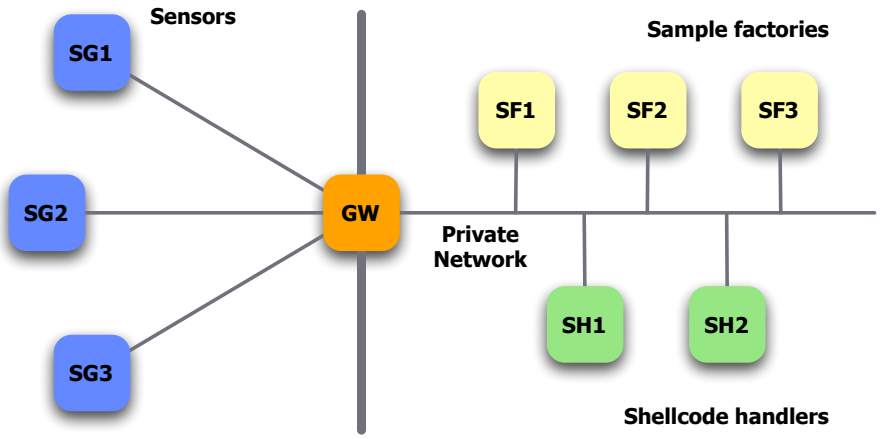


Figure 3.4: SGNET architecture

the attacker. We showed in [35] how the honeypot can react to this situation relying on a real host (an *oracle*) by acting as a proxy between the attacker and the real host. This allows the honeypot to continue the conversation with the attacker, and to collect a new sample of protocol interaction that can be used to refine the protocol knowledge.

ScriptGen is able to correctly learn and emulate the exploit phase for protocols as complex as NetBIOS [35]. ScriptGen allows to build highly interactive honeypots at low cost. The oracles needed to learn new activities can be hosted in a single virtualization farm and contacted by the honeypots through a tunneling system, in a structure similar to Spitzner's *honeyfarm* concept. Differently from classical honeyfarms, access to the real hosts is a rare event resulting from the occurrence of a new kind of attack. As a consequence, systems based on the ScriptGen honeypots potentially have a high degree of scalability.

The practical implementation of ScriptGen to a distributed honeypot deployment led to the generation of an evolution of the data collection system used by Leurré.com, called SGNET [34]. SGNET is a distributed honeypot framework that combines the strengths of the ScriptGen learning with other techniques developed by other WOMBAT participants to ultimately emulate the full attack trace of code injection attacks and download samples of self-propagating malware. Currently implemented under the form of an experimental prototype, SGNET will be extended and mature within the context of this project.

SGNET data collection framework combines together a very diverse set of tools and

techniques:

- **ScriptGen (EURECOM).** ScriptGen learning techniques are used within SGNET in order to increase the level of interaction of honeypot sensors without significantly impacting their cost.
- **Argos (VU Amsterdam).** Whenever a new activity is observed, the SGNET sensor relies on an Argos based high interaction host using the proxy algorithm introduced in [35]. Taking advantage of Argos, the generated sample provides information not only on the network interaction, but also on the presence of successful code injections and on their behavior. More specifically, Argos provides information on the position of the first byte of malicious code (shellcode) that the attacker is forcing the victim to execute.
- **Nepenthes.** Nepenthes is used in SGNET to emulate the behavior of the shellcode, and retrieve malware samples.

SGNET architecture is represented in Figure 3.4. Similarly to the first version of the Leurré.com deployment, SGNET consists of a number of small sensors deployed throughout the Internet. Taking advantage of the ScriptGen FSM knowledge, these sensors achieve at low cost a significantly higher level of interaction with the clients. While the "normal" interaction is completely handled through the FSM knowledge held by the sensor, two scenarios require the communication of the sensor with a central farm providing more sophisticated services.

1. **Unknown activity.** If the sensor encounters a network activity not falling into the existing FSM knowledge, it is unable to handle the activity autonomously. As previously explained, the sensor needs to act as a proxy towards a high interaction machine, based on Argos, acting as an oracle. This high interaction machine, called sample factory, is provided by the central farm.
2. **Successful code injection attack.** Once a successful code injection attack is detected, the information provided by the Argos-based sample factories is used to identify the shellcode pushed by the attacker to the victim's memory. SGNET relies on an external entity, the shellcode handler, based on Nepenthes, to understand the behavior of this shellcode and emulate it.

The integration of these different tools in the SGNET architecture allows to collect in-depth information on the different stages of a code injection attack. All the information generated by all the sensors participating to the deployment is collected daily and stored in a centralized database, described hereafter.

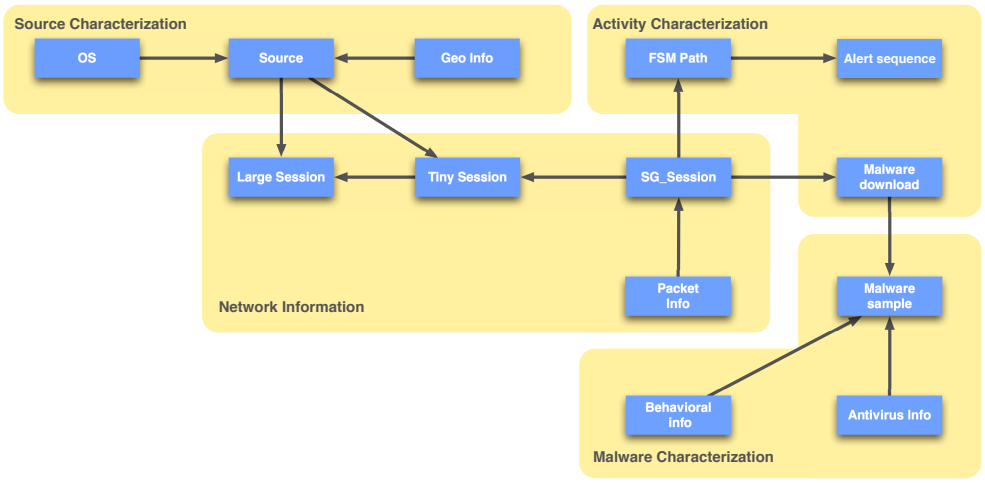


Figure 3.5: Leurré.com database structure

Collected data

The distributed sensors part of the two Leurré.com architectures collect a different amount of information on the observed activities. All the information is centrally collected in an SQL database, accessible to all the partners. As previously described, the increased level of interaction of the SGNET sensors allows to collect detailed information on the structure of code injection attacks that is unavailable to the first generation deployments. Consequently, the database schema associated to SGNET is an extension of that of the first generation deployment, adding new tables and concepts to those already available in the first generation one.

Figure 3.5 provides an overview over the main concepts and the corresponding tables in the Leurré.com schema, including those specific to SGNET data. The interested reader can find in [37] a more exhaustive tractaction of the Leurré.com dataset. While the diagram is not exhaustive, it provides a schematic view on the types of information provided in the dataset. We can identify the following classes of information:

- **Source characterization.** In order to take into consideration the artifacts caused by dynamic addressing, the activity of an attacking source is defined in Leurré.com as all the activity generated by a given IP address separated by packet inter-arrival times smaller than 25 hours. That is, if the same IP is observed in two timeframes

separated by more than 25 hours, it is associated to two separate attacking sources.

Additional tools are used to characterize each attacking source. More specifically, an IP geolocation database [38] is used to determine the ISP and the country of origin of each IP address. Also, passive OS fingerprinting techniques [48] provide hints on the version of the Operating System used by the attacker.

- **Network information.** The database provides statistics on the network activity generated by each source at different aggregation levels. The term *large session* is used to refer to the whole network activity generated by a given source towards a honeypot platform. The concept of large session is refined through the identification of multiple *tiny sessions*, grouping together the activity generated by the source towards *each* of the IPs of the platform. In SGNET, the concept is further refined by the *SG sessions*, corresponding to each TCP session or UDP request and answer handled by the ScriptGen FSMs.

  Each of these aggregation levels is associated to various network statistics and its nature is characterized in different ways. For instance, a tiny session is associated to a *ports sequence*, representing the ordered sequence of ports contacted by the attacker during the interaction. In SGNET, each SG session is associated to the identifier of the traversal of the FSM, that proved to be a very good indicator of the type of activity.

- **Activity characterization.** In SGNET we have recently tried to collect more information on the nature of the observed activities by taking into consideration the alerts generated by the Snort IDS for the packets associated to each FSM traversal. While this is currently ongoing work, we believe that the increased level of interaction can be exploited to collect meaningful information on the characteristics of the activity.

  Moreover, the information provided by the interaction of the ScriptGen learning with the Argos-based sample factory allows the identification of the SG sessions that lead to successful code injections. SGNET is able to collect detailed information on the characteristics of a successful code injection attack. The database provides detailed information on the binary shellcode, and on the interpretation of that shellcode according to the shellcode handler. This allows to classify the different download strategies used by the attackers to upload malware to the victims. For instance, it is possible to distinguish a malware that is pushed to the victim through a small downloader from malware that the victim is forced to download taking advantage, for instance, of the *tftp* system utility.

- **Malware characterization.** Every malware sample downloaded by SGNET is an extremely valuable source of information on the ultimate goal of the attacking source. For this reason, each malware sample is enriched by different types of information resulting from the integration with other WOMBAT sources, namely Anubis (TU Vienna) and VirusTotal (Hispasec). Each malware collected by infrastructure is automatically submitted to these services, and the results of the analysis are stored in the SGNET database taking advantage of a simple plugin-based framework.

  Different submission policies are applied for the two services. In the case of Anubis, each malware sample is submitted only once for analysis, and the corresponding analysis result is stored in the database in aggregate form. In the case of VirusTotal, the malware is submitted multiple times to observe the behavior of the different AV vendors in recognizing the sample. Every sample is submitted at least 30 days to VirusTotal; the submission policy then stops submission when the last 7 obtained results do not differ.

Sharing data

A time consuming task in the management of the Leurré.com datasets consists in providing support to the users of the dataset. Historically, the access to the Leurré.com dataset was provided to interested partners either through a web interface or through direct SQL access via an SSH account. While the latter was adequate to the task of building automated analysis scripts, a set of inconvenients were identified.

- The full Leurré.com SQL schema is rather complex. Also, it often rapidly evolves: new analysis methods are generated and new data feeds are continuously added. The impact on the external user of such complexity is noticeable, and often leads to misunderstanding of the semantics of the various concepts or of the content of the tables.
- Errors of a single user can have repercussions on the availability of the whole system. For instance, an erroneous SQL query can lead to an excessive cost on the system draining the DBMS resources.

For the above reasons, an alternate solution was investigated. This led to the generation of an API, called *horasis*, and based on python. The horasis API allows the interested data consumer to access most of the information stored in the database through a set of object instantiations and method calls. No knowledge is required on the underlying SQL schema: the library transparently converts all the python interaction into SQL

queries used to reply to the user. The horasis library provides a unified interface to both the Leurré.com first generation dataset and to the additional information generated by the experimental SGNET prototype.

The horasis library provides to the user three main concepts: DB objects, iterators and predicates.

The horasis DB objects are python objects wrapping the main concepts defined in the Leurré.com DB schema. Examples of these objects are, for instance, the `Malware` object, the `TinySession` object, and so on and so forth. Each of these objects is instantiated using an identifier, for instance the file MD5 hash for a Malware object or the internal identifier for a TinySession object. Upon instantiation, the library queries the database to retrieve information on the existance of the instance in the dataset, and retrieve a set of informations that is easily retrievable from the dataset. For instance, the `MALWARE` table in the Leurré.com schema contains the MD5 of each sample, its size in bytes and other similar information. When verifying the existance of the MD5 in the table, the library can retrieve all the content of the row without significantly impacting the performance. All this information thus appears as an *attribute* of the object instance. Each object also provides a set of methods that, once invoked, generate more expensive SQL queries and eventually link to other DB objects. For instance, a malware object provides a method to retrieve the list of all the `InjectionAttack` events that led to its download.

The horasis iterators are used to iterate in time over a collection of DB objects. For instance, the `CodeInjectionIterator` allows to iterate over all the code injection objects detected by the deployment over a certain period of time.

Finally, the horasis library provides a set of predicates used to query the database. Two different types of predicates are provided:

- Interrogative predicates. They are used to query the database about the available knowledge on a certain event. For instance, the predicate `whois_ip` queries the database about its knowledge on a certain IP address if any. These predicates return an activity identifier, an opaque identifier used within the Leurré.com database to identify a certain class of activities.
- Explicative predicates. They are used to retrieve additional information about an activity identifier generated by the interrogative predicates. For instance, the predicate `activity_srcnetblocks` provides the list of CIDR network prefixes containing at least an attacking source that performed a given activity class.

Figure 3.6 shows an example of the type of information that a data consumer can retrieve from the Leurré.com SGNET dataset taking advantage of the horasis library.

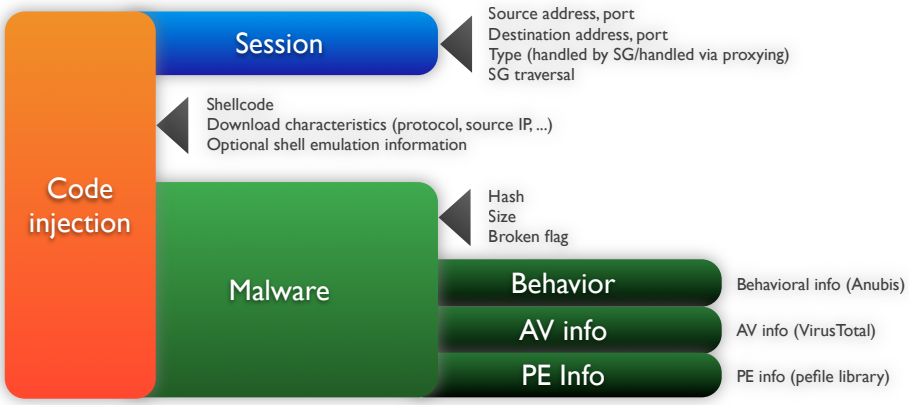


Figure 3.6: Horasis library

Taking advantage of an iterator, the data consumer can have access to all the code injections observed by the infrastructure for a given timeframe. Each of these events is associated to three main objects: the code injection itself, the SG session and the malware object. Through this information, the user can retrieve detailed knowledge on the various phases of a code injection attack.

Summarizing, the horasis library allows data consumers to easily take advantage of most of the Leurré.com dataset without an in-depth knowledge of the underlying SQL schema. The horasis API is currently used by FORTH for the integration of the Leurré.com datasets in the WOMBAT database. The experience and some of the underlying ideas have been integrated in the specification of the WOMBAT API.

3.2.3 Arakis

Arakis [42] is na Early Warning System designed and implemented by NASK and operated by CERT Polska. The system was first deployed in 2004, but development of new functionality was continued until 2008. The system consists of over 50 sensors deployed in many Polish institutions and a central server aggregating the data provided by sensors and performing offline analyzes of collected data, resulting in more advanced alerts.

Unlike many other data sources in WOMBAT, Arakis is not a general purpose system with data collection and analysis as the goal. The goal of the system is the protection of CERT Polska constituency. For this reason some of the collected data are considered

sensitive and cannot be provided to WOMBAT.

Arakis uses several types of sensors. Data for WOMBAT come mostly from honeypots and darknets, but other types of data are collected as well.

Data provided

The amount of data made available to WOMBAT depends on the source. Some information, including destination IPs of captured flows and pcap dumps, is only available from the CERT Polska sensor.

WOMBAT can access the Arakis database in three different ways: by observing Arakis alerts, rankings and through searches. Arakis made public five kinds of alerts and four kinds of rankings.

All alerts provide some basic information – the automatically generated message containing most important information about the alert in text format, timestamp, short title, alert type, etc. Additional information is provided if available. The alerts are as follows:

- NCLUS – signifies that a new cluster of traffic signatures was created. This is an effect of the automatic analysis, so it may mean both that a new form of behavior is becoming frequent (possibly a new threat), or that a known attack is variable enough that some incidents were not recognized as a part of an already existing cluster, but created a new one. This is resolved manually: a new cluster has no name, it is assigned after analysis of the cluster. Additionally a threat level is assigned, qualifying the cluster as representative of malicious or benign traffic, or as trash – meaningless, accidental similarity of several signatures, not indicative of malicious or benign behavior.

  The alert makes it possible to access all information associated with the cluster, including the creation date, supersignature (aggregation of signatures in the cluster, potentially useful as content of snort signatures, but only after manual verification), ports and protocols associated with the cluster (especially meaningful if only one pair is identified), as well as LCS signatures belonging to the cluster, including the "core" (the central signature, most representative of the cluster), the manually created descriptive labels for the signatures and flows with those signatures.

  All LCS signatures used in clustering are the product of analysis called HLCS (Horizontal Longest Common Subsequence) in which each flow is treated as a list of sequences - packets, and flows are compared packet-by-packet. There is also another analysis called VLCS (Vertical LCS), treating each flow as a single long sequence and comparing entire flows, used e.g. in the NWORM analysis. For this reason it is possible that a signature in the system is very similar to the core of an

existing cluster, but is not included in it - this happens if it only appears in VLCS analysis, but not in HLCS.

- NWORM – signifies that a new, unknown behavior was identified.This is a rather complex analysis, making this alert rare but valuable. The alert is generated only when during analysis of sensor events a signature (VLCS) with a single associated port-protocol pair is found such that:

  1. flows with that signature were identified as important events on several sensors,
  2. the signature is representative of those events (there is only one VLCS in these events),
  3. there were no matching snort rules for these events,
  4. the VLCS has no label (not yet analyzed manually),
  5. the VLCS is not identical to a HLCS already belonging to a named (manually analyzed) cluster,
  6. at least one of the reported events includes an HLCS which does not belong to a named cluster.

  This set of constraints should guarantee that the collected events are significant and that there is no information in the database suggesting that the events are a part of a known type of behavior.

  The port-protocol-VLCS set identified by the analysis is included in the alert's message, but is also available directly, so all associated data, like flows associated with the LCS, can be accessed. Verifying this information later is also useful, since manual analysis by CERT Polska may provide more information, either by labeling that LCS or naming associated clusters. The events used to generate the alert are also provided, making it possible to obtain a full list of all LCS signatures and flows included in these events.

- NSNORT – signifies that a new snort rule has been matched to observed traffic. The analysis uses a TTL, so the rule may not actually be new, just not matched for a long time. The flow matched by the rule is accessible, as well as the signature details.

- NPORT – signifies that a new port-protocol pair has been observed in an event identified by a sensor. The available information includes the port and protocol numbers and the event, making it possible to find all flows associated with the alert.

- SWEEP – signifies that a suspected portsweep attempt was identified. The alert is generated when a port-protocol pair is found such that the flows using that port-protocol pair detected during a specified time came from a small number of unique source IPs but were addressed to many unique destination IPs. This is probable if an obscure vulnerability in a rare program is being exploited. The port and protocol numbers and all flows contributing to the alert can be accessed.

The Arakis system collects a lot of information, allowing it to produce meaningful statistics about suspicious or malicious traffic in the monitored IP ranges. The statistics are available to WOMBAT as the following TOP N rankings:

- Top rising destination ports (honeypot) – this statistic is periodically computed and available directly in the database. The port-protocol pairs are scored depending on sensor weights and sensor-generated scores measuring the activity.
- Top clusters – this statistic, computed on demand, shows the most active clusters, ordered by the number of captured flows with signatures grouped in each cluster. Since inclusion in a cluster usually correctly identifies the threat (or normal behavior) of flows with given signatures, this statistic can be viewed as a ranking of most active threats, including benign traffic erroneously targeted at honeypots.
- Top snort rules – this statistic, computed on demand, lists the snort rules that have registered the most events in a given time range. Rule name, id, references (URLs) and list of events (allowing access to sample flows) are provided.
- Top rising source and destination ports (darknet) – these statistics are precomputed only on a per sensor basis, but WOMBAT is provided with global rankings, computed on demand from sensor rankings. Two variants of this ranking are available, with scores counting individual flows, or the number of unique source IPs. Due to the nature of darknet sensors, no more details are available – this ranking cannot be used as a starting point for browsing the Arakis database.

Additionally, the Arakis system provides search capability, making available information about selected IPs or IP ranges in given time windows. This information may include:

- Honeypot flows registered from that IP. Details such as source and destination ports and addresses, timestamp, signatures, etc. are available. Pcap dumps are collected as well and may in some cases be available to WOMBAT. However, some information (pcap, destination IP) can only be provided for flows captured by the CERT Polska sensor.

- (future) Darknet flows registered from that IP. Currently the system cannot provide actual flows with detailed information, but this kind of functionality is planned for future extension of Arakis and – if implemented – will be added to the search available to WOMBAT. The amount of data may range from a simple number of flows registered to detailed flow data including pcap dumps.
- The search results can be enriched using the rest of the database. After flows are found it is trivial to add information such as:
  - LCS signatures with labels (if available), identifying the type of activity,
  - clusters to which the signatures were assigned, giving a higher-level description of the IP's behavior,
  - events in which the flows were grouped, providing correlation between the flows,
  - history of alerts generated using the flows or events associated with the IP,
  - snort rules matched to the flows (with references if desired)

  Some of these details are available to WOMBAT by default, others can be made accessible if desired.

Searches for LCS sequences may also be made available if useful.

Integration with WOMBAT

The Arakis database contains more data than the portion made available to WOMBAT. To keep the rest of the database confidential, a layer of isolation was necessary. Direct SQL access to the database was therefore immediately ruled out. This layer of isolation is useful for many reasons:

Efficiency – developers of Arakis have a much better understanding of the database structure, its strengths and weaknesses, than can realistically be expected of anyone else, regardless of the amount of documentation made available. The isolation layer allows SQL query optimization by developers familiar with the database.

Security and confidentiality – by restricting database access to the isolation layer NASK keeps control of the type of data provided to WOMBAT, as expected by the constituency.

Anonymization and presentation – some of the data provided to WOMBAT are stored in the database in a format which makes sense in the original system, but is not very useful to WOMBAT users. Other data needs special processing to provide adequate anonymization – for example, destination IPs require anonymization to

keep secret the honeynet IP ranges (except one sensor, where the IPs are public), for the same reason cluster supersignatures require parsing to eliminate any IP addresses explicitly listed in the signature. The isolation layer provides a good place for this kind of operations.

(D)DoS protection – the Arakis system is an important countrywide system and while an occasional failure would not be critical, it is necessary to protect it from obvious threats. It is obviously most important to ensure that data from sensors reach the central database. Since the data for WOMBAT must be taken from the central database, direct access would make it possible to overload the database with queries, crippling the system's ability to collect data, process it and serve the user interface. The isolation layer makes it possible to limit the amount of queries coming from WOMBAT to an acceptable level. Note that this level is expected to exceed the actual needs of the WOMBAT system, the protection is necessary in case of WOMBAT malfunction (quite possible in the development phase), deliberate misuse of the WOMBAT system by malicious users or successful spoofing attacks targeting Arakis directly.

The isolation layer was implemented as a SOAP server, providing WOMBAT with API for Arakis access, called ArakisWAPI. Note that this design predates the design of WAPI as previously mentioned in this document. ArakisWAPI is currently not WAPI-compatible, although both have a lot in common. A new, WAPI-compatible version of ArakisWAPI will be developed as soon as integration using the current version will be sufficiently tested, so that the new version will suit the needs of the WOMBAT central system as well as possible.

The Web Services are provided over SSL and require a client certificate to access the system. This is basically the only requirement for the client – no implementation details are enforced. The server is implemented in PHP and a sample client in the same language was provided, but implementation in a different language is certainly possible. The API was designed by NASK, but it is open to changes. The ArakisWAPI will be modified to simplify integration with WOMBAT.

The ArakisWAPI provides a number of services, grouped as follows:

- Access services. These services are used as starting points, when the client has no initial information. The group includes the getNewAlerts service, providing information about alerts generated in a given time range and the group of getTop*Object* services providing the rankings.
- Search services. These services can be used when querying Arakis for any information about a given object. Currently this group only includes the getFlowIDs-

ForSIPRange service which finds all flows in a given time range originating from a given IP or IP range.

- Enrichment services. The other groups of services usually provide only basic information about objects. This information can be enriched using the services in this group, browsing the database by following the links between different objects. This is currently the largest group of services, including: getFlowIDsForAlert and getEventIDsForAlert, identifying the flows and events associated with a given alert; getClusterName and getClusterDetails, providing information about clusters of signatures; getLCSDetails, getFlowIDsForLCS, getFlowDetailsForLCS, providing information about LCS signatures of flows and flows with a given signature; getFlowDetails, returning the details of a given flow; getEventDetails, providing information about a given event, including the individual flows; and finally getSnortRulesForFlows, providing details about snort rules triggered by given flows.

The implementation of ArakisWAPI is basically finished, the only missing part is pcap dump access – this functionality must be implemented in a different way than the rest of the services. Pcap dumps for most flows in the database will not be available anyway, due to privacy constraints, so the service is not of key importance and has been scheduled as the last step in the implementation.

3.2.4 Anubis

Malware, which is a generic term to denote all kinds of unwanted software (e.g., viruses, worms, or Trojan horses), poses a major security threat to computer users. According to estimates, the financial loss caused by malware has been as high as 14.2 billion US dollars in the year 2005. Unfortunately, the problem of malicious code is likely to grow in the future as malware writing is quickly turning into a profitable business. Malware authors can sell their creations to miscreants, who use the malicious code to compromise large numbers of machines that can then be abused as platforms to launch denial-of-service attacks or as spam relays. Another indication of the significance of the problem is that even people without any special interest in computers are aware of worms such as Nimda or Sasser. This is because security incidents affect millions of users and regularly make the headlines of mainstream news sources.

The most important line of defense against malicious code are virus scanners. These scanners typically rely on a database of descriptions, or signatures, that characterize known malware instances. Whenever an unknown malware sample is found in the wild, it is usually necessary to update the signature database accordingly so that the novel malware piece can be detected by the scan engine. To this end, it is of paramount

importance to be able to quickly analyze an unknown malware sample and understand its behavior and effect on the system. In addition, the knowledge about the functionality of malware is important for removal. That is, to be able to cleanly remove a piece of malware from an infected machine, it is usually not enough to delete the binary itself. It is also necessary to remove the residues left behind by the malicious code (such as unwanted registry entries, services, or processes) and undo changes made to legitimate files. All these actions require a detailed understanding of the malicious code and its behavior.

The traditional approach to analyze the behavior of an unknown program is to execute the binary in a restricted environment and observe its actions. The restricted environment is often a debugger, used by a human analyst to step through the code in order to understand its functionality. Unfortunately, anti-virus companies receive up to *several thousand* new malware samples each day. Clearly, the analysis of these malware samples cannot be performed completely manually. Hence, automated solutions are necessary.

One way to automate the analysis process is to execute the binary in a virtual machine or a simulated operating system environment. While the program is running, its interaction with the operating system[1] (e.g., the native system calls or Windows API calls it invokes) can be recorded and later presented to an analyst. This approach reliefs a human analyst from the tedious task of having to manually go through each single malware sample that is received. Of course, it might still be the case that human analysis is desirable after the automatic process. However, the initial results at least provides details about the program's actions that then help to guide the analyst's search.

Current approaches for automatic analysis suffer from a number of shortcomings. One problem is that malicious code is often equipped with detection routines that check for the presence of a virtual machine or a simulated OS environment. When such an environment is detected, the malware modifies its behavior and the analysis delivers incorrect results. Malware also checks for software (and even hardware) breakpoints to detect if the program is run in a debugger. This requires that the analysis environment is *invisible* to the malicious code. Another problem is when the analysis environment does not monitor the complete interaction with the system. When this happens, the malicious code could evade analysis. This might be possible because there exist thousands of Windows API calls, often with arguments that are composed of complex data structures. Furthermore, the malicious code could also interact directly with the operating system via native system calls. Thus, the analysis environment has to be *comprehensive* and cover all aspects of the interaction of a program with its environment.

[1] Because the vast majority of malware is written for Microsoft Windows, the following discussion considers only this operating system.

Analysing Unknown Binaries (ANUBIS) that was developed in the Secure Systems Lab of the Technical University of Vienna. It is a tool that automates the process of analyzing malware to allow a human analyst to quickly get a basic understanding of the actions of an unknown executable. Running a binary under ANUBIS results in the generation of a report that contains information to give the human analyst a very good impression about the purpose and the functionality of the analyzed sample. This report includes detailed data about modifications made to the Windows registry and to the file system, information about interactions with the Windows Service Manager and other processes, as well as a complete log of all generated network traffic.
The following list summarizes the key features of ANUBIS:

- ANUBIS uses emulation to run the unknown binary together with a complete operating system in software. Thus, the malware is never executed directly on the processor. Unlike solutions that use virtual machines, debuggers, or API function hooking, the presence of ANUBIS is more difficult to detect for malicious code.
- The analysis is comprehensive because our system monitors calls to native kernel functions as well as calls to Windows API functions. It also provides support for the analysis of complex function call arguments that contain pointers to other objects.
- ANUBIS can perform function call injection. Function call injection allows us to alter the execution of the program under analysis and run our code in its context. This ability is required in certain cases to make the analysis more precise.

System Description

ANUBIS is a tool for analyzing Windows executables (more precisely, files conforming to the portable executable (PE) file format). To this end, the program under analysis is executed inside a PC emulation environment and relevant Windows API and native system calls are logged. In the following sections, we describe in more detail the design and implementation of key components of ANUBIS.

Emulation Environment. As mentioned previously, ANUBIS uses a PC emulator to execute unknown programs. When designing our system, we had to choose between different forms of emulation. In particular, we had to decide if the hardware of a complete PC should be emulated so that an actual off-the-shelf operating system could be installed, or if the processor should be emulated and our own implementation of (a subset of) the operating system interface should be provided. Virus scanners typically emulate the processor and provide a lightweight implementation of the operating system interface

(both native system calls and Windows API calls). This approach allows a very efficient analysis process. Unfortunately, it is not trivial to make the operating system stub behave exactly like the actual operating system, and the semantics between a real system and the simulated one differ in many cases. These differences could be detected by malware, or simply break the code. Thus, we decided to emulate an entire PC computer system, running an off-the-shelf Windows XP on top. While the analysis is significantly slower compared to a virus scanner, the accuracy of the emulation is excellent. Since our focus is on the analysis of the behavior of the binary, this trade-off is acceptable.

ANUBIS uses Qemu, an open-source PC emulator written by Fabrice Bellard, as its emulator component. Qemu is a fast PC emulator that properly handles self-modifying code. To achieve high execution speed, Qemu employs an emulation technique called *dynamic translation*. Dynamic translation works in terms of basic blocks, where a basic block is a sequence of one or more instructions that ends with a jump instruction or an instruction modifying the static CPU state in a way that cannot be deduced at translation time. The idea is to first translate a basic block, then execute it, and finally translate the next basic block (if a translation of this block is not already available). The reason is that it is more efficient to translate several instructions at once rather than only a single one.

Of course, Qemu could not be used in our system without modification. First, it had to be transformed from a stand-alone executable into a Windows shared library (DLL), whose exported functions can be used by ANUBIS. Second, Qemu's translation process was modified such that a callback routine into our analysis framework is invoked before every basic block that is executed on the virtual processor. This allows us to tightly monitor the process under analysis.

Before a dynamic analysis run is performed, the modified PC emulator boots from a virtual hard disk, which has Windows XP (with Service Pack 2) installed. The lengthy Windows boot-process is avoided by starting Qemu from a snapshot file, which represents the state of the PC system after the operating system has started.

Analysis Process. The analysis process is started by executing the (malware-)program in the emulated Windows environment and monitoring its actions. In particular, the analysis focuses on which operating system services are requested by the binary (i.e., which system calls are invoked). Every action that involves communication with the environment (e.g., accessing the file system, sending a packet over the network, or launching another program) requires a Windows user mode process to make use of an appropriate operating system service. There is no way for a process to directly interact with a physical device, which also includes physical memory. The reason for this stems from the design of modern operating systems, which prohibit direct hardware access so that multiple processes can run concurrently without interfering with each other. Thus,

it is reasonable to monitor the system services that a process requests in order to analyze its behavior.

On Microsoft Windows platforms, monitoring system service requests is not entirely straightforward. The reason is that the actual operating system call interface, called native API interface, is mostly undocumented and not meant to be used directly by applications. Instead, applications are supposed to call functions of the documented Windows API. [2] The Windows API is a large collection of user mode library routines, which in turn invoke native API functions when necessary. The idea is that the Windows API adds a layer of indirection to shield applications from changes and subtle complexities in the native API. In particular, the native API may change between different Windows versions and even between different service pack releases. On a Windows system, the native API is provided by the system file `ntdll.dll`. Parts of this interface are documented by Microsoft in the Windows DDK and the Windows IFS kit. Moreover, Gery Nebbett has written an unofficial documentation of the native API, which covers about 90% of the functions.

Malware authors sometimes use the native API directly to avoid DLL dependencies or to confuse virus scanner's operating system simulations. For this reason, ANUBIS monitors both the Windows API function calls of an application and also its native API function calls. The task of monitoring which operating system services are invoked by the program requires us to solve two problems:

1. We must be able to precisely track the execution of the malware process and distinguish between instructions executed on behalf of the malware process and those of other processes. This is essential because the virtual processor does not only run the malware process, but also instructions of the Windows operating system and of several Windows' user mode processes. Therefore, a mechanism is required that enables ANUBIS to determine for each processor instruction whether or not this instruction belongs to the malware process.

2. We need an unobtrusive way for monitoring the accessed operating system services. That is, we have to be able to determine that a native API call or a Windows API call is invoked *without* modifying the malware code. That is, we cannot hook API functions or set debug breakpoints.

We accomplish the precise tracking of the malware process with the help of the `CR3` processor register. The `CR3` register, which is also known as the page-directory base register (PDBR), contains the physical address of the base of the page directory for

[2] The Windows API is documented by Microsoft in the Platform SDK.

the current process. The processor uses the page directory when it translates virtual addresses to physical addresses. More precisely, to determine the location of the page directory when performing memory accesses, the processor makes use of the `CR3` register.

Windows assigns each process its own, unique page directory. This protects processes (in particular, their virtual memory address space) from each other by ensuring that each process has its own virtual memory space. The page directory address of the currently running process has to be stored in the `CR3` processor register. Consequently, Windows loads the `CR3` register on every context switch. Thus, we simply have to determine which page directory address has been assigned to the malware process by Windows. Then, we are able to efficiently determine whether or not the current instruction belongs to the test subject under analysis by comparing the current value of the `CR3` register to the page directory address of this test subject.

Determining the physical address of the page directory of the test subject is the responsibility of a probe component that is located *inside* the emulated Windows XP environment. This probe serves as a sensor in the emulated environment and consists of a kernel driver and a program that is run in user mode. The task of the kernel driver is to locate the page directory address that belongs to the test subject and report its findings back to the user mode process. The user mode component then informs ANUBIS. Note that ANUBIS is outside the emulated environment, thus, communication between the probe and ANUBIS has to take place over the virtual network that connects the emulated environment with its host system. To this end, an RPC server is used that runs inside the emulated PC.

The kernel driver is necessary because the page directory address is stored in a memory region that is only accessible to the Windows NT kernel and its device drivers. More precisely, the page directory address can be found as an attribute of that `EPROCESS` structure that corresponds to the test subject. The `EPROCESS` structure is a Windows-internal data object that plays a key role in the way Windows manages processes. For each process in the system, a corresponding `EPROCESS` structure exists. Thus, the device driver has to walk the list of system processes (which consists of `EPROCESS` members) until it finds the one corresponding to the process of the test subject. At this point, the appropriate page directory address can be read. Note that the page table address of the test subject's process has to be obtained *before* its first instruction is executed. To this end, the process is created in a suspended state. Only after successfully identifying the page directory address is the test subject allowed to run.

As mentioned previously, the second problem of our analysis is to monitor the invocation of operating system functions.[3] This task can be solved by comparing the current

[3]We use the term operating system function as a generic term for both Windows API and native API

value of the virtual processor's instruction pointer (or program counter) register to the start addresses of all operating system functions that are under surveillance. This comparison is performed in the callback routine of ANUBIS, which Qemu invokes at the start of each translation block. Note that the start address of a function always corresponds to the first instruction in a translation block. The reason is that a function call is a control transfer instruction, and whenever a control transfer instruction is encountered, Qemu starts a new translation block. At this point, ANUBIS is invoked and can check the current value of the program counter.

A Windows application typically accesses operating system functions by dynamically linking to system DLLs and calling their exported functions. Thus, we can extract the addresses of interesting functions simply from library export tables. For example, an application calls the Windows API function `CreateFile`, which is implemented in the shared library `Kernel32.dll` when it wants to create a file. In this case, determining the start address of `CreateFile` is easily possible by looking at corresponding entry in `Kernel32.dll`'s export table (and then adding the base address of `Kernel32.dll` to it, as DLLs may be loaded at a different base address).

Analysis Report. ANUBIS is a tool for analyzing malware. While, in principle, arbitrary functions can be monitored, we provide a number of callback routines that analyze and log security-relevant actions. After a run on a test sample, the recorded information is summarized in a concise report. This report contains the following information:

1. General Information - This section contains information about ANUBIS's invocation, the command line arguments, and some general information about the test subject (e.g., file size, exit code, time to perform analysis, ...).

2. File Activity - This section covers the file activity of the test subject (i.e., which files were created, modified, ...).

3. Registry Activity - In this section, all modifications made to the Windows registry and all registry values that have been read by the test subject are described.

4. Service Activity - This section documents all interaction between the test subject and the Windows Service Manager. If the test subject starts or stops a Windows service, for example, this information is listed here.

5. Process Activity - In this section, information about the creation or termination of processes (and threads) as well as interprocess communication can be found.

functions.

6. Network Activity - This section provides a link to a log that contains all network traffic sent or received by the test subject.

Collected Data. We have made ANUBIS available to the public through a public web interface. Using this interface, Internet users can submit samples and can obtain automatically generated analysis reports. We have collected thousands of binary samples since the introduction of the ANUBIS web site in 2007. In fact, the ANUBIS web service has become quite popular and is used regularly by organizations such as Shadowserver.org and Team Cymru that work on malware detection and analysis. Also, some CERTS such as the Aus-CERT (i.e., the Australian CERT) have made ANUBIS an important part of their daily business workflow).

Some simple analysis is currently performed on the collected samples. We try to identify samples that are malicious by applying simple rules such as the number of outgoing connections, and protocols that are used. For example, if a sample connects to an IRC server, this is usually a strong indication that we are dealing with a bot sample.

However, much more precise and effective techniques are still required to further analyse and cluster the samples that have been collected.

3.2.5 Other Sources

Apart from the data sources described in the previous sections, there are some other that have already been integrated to the WOMBAT infrastructure. These data sources are public and do not belong to any organization within the consortium. Currently the integrated ones are *Sans.org*, *OffensiveComputing.net* and *NEMU* but there are others that can be also integrated in the future.

Sans.org

The ISC [30] was created in 2001 following the successful detection, analysis, and widespread warning of the Li0n worm. Today, the ISC provides a free analysis and warning service to thousands of Internet users and organizations, and is actively working with Internet Service Providers to fight back against the most malicious attackers.

Each day the Internet Storm Center gathers millions of intrusion detection log entries, from sensors covering over 500,000 IP addresses in over 50 countries. It is rapidly expanding in a quest to do a better job of finding new storms faster, identifying the sites that are used for attacks, and providing authoritative data on the types of attacks that are being mounted against computers in various industries and regions around the globe. The Internet Storm Center is a free service to the Internet community. The work is supported by the SANS Institute from tuition paid by students attending SANS

security education programs. Volunteer incident handlers donate their valuable time to analyze detects and anomalies, and post a daily diary of their analysis and thoughts on the Storm Center web site.

The ISC relies on an all-volunteer effort to detect problems, analyze the threat, and disseminate both technical as well as procedural information to the general public. Thousands of sensors that work with most firewalls, intrusion detection systems, home broadband devices, and nearly all operating systems are constantly collecting information about unwanted traffic arriving from the Internet. These devices feed the DShield database where human volunteers as well as machines pour through the data looking for abnormal trends and behavior. The resulting analysis is posted to the ISC's main web page where it can be automatically retrieved by simple scripts or can be viewed in near real time by any Internet user.

Likewise, the Internet Storm Center uses small software tools to send intrusion detection and firewall logs (after removing identifying information) to the DShield distributed intrusion detection system. The ISC's volunteer incident handlers monitor the constantly changing database to provide early warnings to the community of major new security threats. The ISC also provides feedback to participating analysis centers comparing their attack profiles to those of other centers, and provides notices to ISPs of IP addresses that are being used in widespread attacks. The ISC maintains a very popular daily diary of incident handlers notes, and can generate custom global summary reports for any Internet user.

The value of the Internet Storm Center is maximized when the sensors are collecting data on attacks touching all corners of the Internet. Because of the vastness of cyberspace it is impossible to instrument the entire Internet. Instead, samples are taken in as many diverse places as possible to create an accurate representation of current Internet activity. Many ISC users send their log data directly to the ISC databases without going through an organizational or local analysis and coordination center. Several large organizations have expressed interest in mirroring the ISC's distributed intrusion detection system, placing sensors at the edges and within their networks to provide early detection of anomalous behavior.

Data provided. The Internet Storm Center provides the following reports:

- **Top Ports.** A report on the Top TCP ports attacked is published daily on the Intenet Storm Center. This report includes each port and the number of reported attacks that targeted it.
- **Top Sources.** Analogous to *Top Ports*, this report summarizes the Top daily attack sources. That is the IP address of each attack source and the number of reported attacks originated by it.

- **AS Reports.** These reports are based on ASes (Autonomous Systems[4]). Technically, these reports are an aggregation of the IP addresses of the attack sources based on the AS they belong to.
- **Country Reports.** Another aggregation based on the Country that an attack originated.
- **Survival Time.** The survival time is calculated as the average time between reports for an average target IP address. The average time between probes will vary widely from network to network.
- **Trends.** The "Trend" is an attempt to put a number to the increase in activity for a given port. This number is the comparison of the last 24 hours to the last 30 days. So if we see a rise in activity compared to the last 30 days, the trend is high.
- **Daily Data Volume.** The number of reports per day.

Currently, we considered integrating the first two data types, that is the *Top Ports* and *Top Sources*. In order to overcome the fact that these data are in the form of Top-X daily lists, we used the WOMBAT database. That way we are able to keep history over this reports.

Offensive Computing

Offensive Computing, LLC was formed by Valsmith and Danny Quist as a resource for the computer security community. The primary emphasis here is on malware collections and analysis for the purpose of improving people's abilities to defend their networks. There is a noticeable lack of public sources of malware and malware analysis available. Those that were available were either for sale or limited to a small number of users. They provide resources such as live copies of malicious software, MD5 checksums to search on and analysis of the malware to the general public. Offensive Computing currently has the largest publicly available malware collection on the Internet.

This way users can match malware they find on a system and they can quickly identify it and know the best defense. By removing barriers to information we believe this will make the Internet a safer place.

Samples are acquired in various ways:

[4]In the Internet, an Autonomous System (AS) is a connected collection of IP routing prefixes under the control of one or more network operators that presents a common, clearly defined routing policy to the Internet, cf. RFC 1930, Section 3.

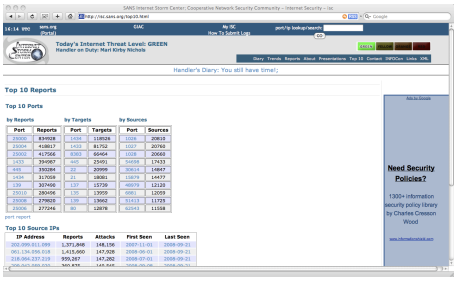


Figure 3.7: Internet Storm Center web-page.

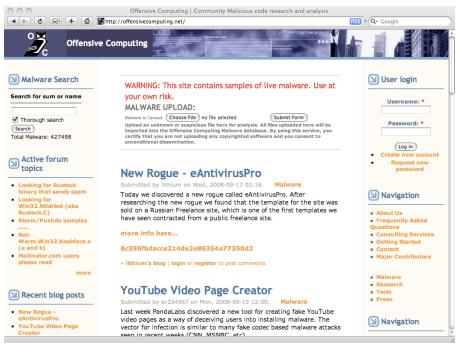


Figure 3.8: Offensive Computing web-page.

- User contributed.
- Captured via mwcollectors and other honeypots.
- Found via searches.
- Discovered on compromised systems.

This site does NOT encourage or condone the spreading or propagation of viruses or worms. Thats exactly what this site is designed to help defend against.

The intent of providing live copies of malware is so that the community can collaborate on identifying and analyzing them in order to develop snort signatures and other defenses.

NEMU

NEMU is a network intrusion detection system developed by FORTH. The way this system is able to detect an intrusion is by emulating a CPU on the network level, interpreting all streams of data as they were executable code. Whenever the length of the executed data of a stream exceeds a threshold an alert is generated and the stream is saved.

We integrated some of these PCAP network traffic trace files in the WOMBAT database mainly for debugging reasons.

3.3 New Sources

3.3.1 BlueBat

Motivations for development

Since mobile computing is such a pervasive technology, end users as well as organizations are extensively adopting mobile devices as a critical component of their IT environments. For this reason, it is becoming more and more important to understand the potential risks linked with all types of wireless devices and communication protocols. Bluetooth in particular is widely believed to be the dominant standard in short-range, "personal area" wireless communications, in particular for smartphones and PDA devices.

For this reason we are currently developing and field testing a viable Bluetooth honeypot sensor codenamed BlueBat. We build on a previous experience of site assessment and survey [24] to create usable honeypot sensors. We performed site surveys for sensor placement, and tested various hardware and software combinations for achieving an optimal collection capability with inexpensive sensors. We analyzed the results of the field tests, and demonstrated various design constraints.

State of the Art

Bluetooth is a short range radio communication protocol aimed to unify different wireless data transmission technologies among mobile and static electronic devices: commonly, PCs and cellular phones, but even Bluetooth enabled POS terminals, cars or household appliances are not uncommon. Basically it is an alternative to traditional infrared communication standards such as IrDA, and is based on a short-wave radio technology, which is reportedly able to transmit data across physical obstacles such as walls or other objects [39]. Bluetooth devices use the 2.4 GHz frequency range (the same range used by IEEE 802.11 standards for wireless Ethernet). An important improvement over IrDA is that Bluetooth device do not need a careful alignment, nor in fact a clear line of sight among devices, making connections easier over a slightly increased range w.r.t. IrDA. This is one of the key reasons why Bluetooth can conceivably be used as a transport for automatically spreading malware, or as a mean of attack, while this is not the case with IrDA, since the requirement of aligning transmitting and receiving devices avoids"casual" or unwanted interaction.

Bluetooth technology is characterized by a low power (from 1 to 100 mW) and a communication speed of around 1 Mbps in its original version; towards the end of 2004, a new (but backward compatible) implementation of the Bluetooth technology (version 2.0) was released, allowing for transfer speeds of up to 2 and 3 Mbps, as well as lower

energy consumption: it was further updated to 2.1 in 2007. With regards to transmission power (and thus effective range), Bluetooth devices can be grouped in classes:

- Class 1: 100 mW (20 dBm), 100 m range
- Class 2: 2.5 mW (4 dBm), 10 m range
- Class 3: 1 mW (0 dBm), 1 m range

Most common are class 2 and 3 devices: for instance notebooks and cellular phones are normally Class 2 peripherals.

The Bluetooth standard incorporates very robust security mechanisms [27] that can be used to create very secure architecture. A series of theoretical glitches and possible attacks were discovered in the core specifications of Bluetooth [31, 28]. The most serious of these (described and implemented in [45]) can lead to a compromise of the cryptographic algorithm protecting communication through sniffing, but this is less than practical since the attacker needs to be present to the pairing of devices, and to be able to sniff communications among them. This is more difficult than it would be for the 802.11 family protocols, since Bluetooth divides the 2.4 GHz spectrum range into 79 channels, through which devices hop with a pseudorandom hopping sequence which is different from PAN to PAN. This is done both to avoid interferences among different PANs and as a security enhancement. Common, off-the-shelf hardware is not able to perform sniffing on all channels, and dedicated (and costly) hardware is required. However, recently some researchers touted the possibility of manually modifying common hardware to perform sniffing [40].

Even if Bluetooth is theoretically quite robust, since late 2003, a number of security issues in various specific implementations of the standard stack surfaced. Such attacks are very well described on the website [20], 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, something that can be effectively used to transform a telephone into a spyphone [22]. To further stress that implementation glitches lurk below the surface, on June 2008 there was a very interesting security bulletin from Microsoft [13] which reported a vulnerability in the Bluetooth stack in Windows that could allow remote code execution, with system privileges.

These flaws demonstrate how, in many cases, it is possible to steal information from mobile devices, controlling them from a distance, making calls, sending messages, or even connecting to the Internet. This type of problems is traditionally handled, in computer systems, with the release and application of patches. However, this approach does not

extend to GSM handsets, since in most cases a firmware update can be performed only at service points and shops, not by the customers themselves: therefore many vulnerable phones and firmwares keep going around even long after a vulnerability is discovered.

Some of these attacks are implemented in "Bloover", a proof-of-concept application developed and released by Martin Herfurt, which runs on Symbian cellphones. This counters the idea that an attacker would need a laptop in order to execute these attacks, therefore making themselves visible. Some scripts and tools have also been ported to the Nokia 770 Internet Tablet. Most of these attacks can also be performed at a distance using long range antennas (similar to the ones we show in Section 3.3.1) and modified Bluetooth dongles: a Bluetooth Class 2 device was reportedly able to perform a BlueSnarf attack at an astounding distance of 1.08 miles.

Viruses for mobile devices propagating over Bluetooth reportedly exist. The propagation of a Bluetooth virus can take place in several different ways. The most common, until now, is through simple *social engineering*. The worm sends messages with copies of himself to any device which comes into range through an OBEX push connection. OBEX (OBject EXchange) is the protocol used for exchanging binary objects over Bluetooth. There are different profiles for this service, and "push" is the profile generally used for phone to phone occasional transfers without authentication (e.g. for exchanging electronic business cards). Much like in the case of e-mail worms and trojans, the receiver, finding an "attractive" message on the cellular phone with the invitation to download and install an unknown program, often has no clue that this can pose a danger. For instance, Cabir [4], one of the first cellular phone worms, and the first case of malware able to replicate itself only through Bluetooth, used this technique. Using some vulnerabilities [14], also seemingly innocent files such as images could be used as a viral propagation vectors.

MMS messages are another potential medium of propagation, e.g. the worm Commwarrior [6] propagated also through MMS (in fact, it spread from 8 A.M. to midnight using Bluetooth connections, and from midnight to 7 A.M. through MMS messages). Bluetooth attacks, such as the ones described above, could also be used: but since they are quite platform-specific, they are a difficult and unreliable mean of propagation when compared to the simplicity of social engineering. A final method of propagation, since most smartphones can use e-mail and potentially offer TCP/IP services, could be fairly similar to mail based or TCP based worms, such as the ones we usually witness on the Internet. This type of method has not been really used until now.

Cabir and Commwarrior are both targeted at SymbianOS devices, but worms for other platforms exist. In addition, if these two specimens are not very dangerous, other malware carries more dangerous payloads. For instance, CardBlock.A [5] locks the memory card with a random password. PBStealer.A [17] is able to steal the phonebook of the

device. An even worst scenario would certainly be a worm spreading over Bluetooth and acting like MultiDropper.H [15], which contains both Symbian and Windows-targeted malware, meant to infect also the user's computer during synchronization.

BlueBat design

In order to create a Bluetooth honeypot, a distributed approach seems attractive. So, at first we envisioned a distributed honeypot similar to the concept of Honey@home for PCs [9]. A distributed honeypot working on mobile phones, however, must be even easier to install and more reliable than one running on a computer, if regular phone owners are expected to deploy it. Since it also needed to be cross-platform and work on a variety of devices, developing a small J2ME application was the only possible option. Such an application is in principle very simple, being just an OBEX push server that accept any incoming files, obfuscating them through a simple XOR, in order to make them inoperative and avoid any possible risk to the user.

However, the inner workings of the Bluetooth stack on most cellphones, and their interaction with the J2ME framework, make this approach unpractical. Each service on a Bluetooth device must be registered in a Service Discovery DB (SDDB) on a certain UUID. This is roughly equivalent to the concept of "port" on a TCP/IP connection. When another device wants to connect, it runs a SDP (Service Discovery Protocol) scan and then communicates. 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.

Obviously, registering the software on another UUID works, but it is not an option for a honeypot. There is no documented way to "unregister" a service of the manufacturer from J2ME, as can be seen in the JSR-82 specifications [11]. It is only possible to register, unregister and modify an entry from the same application which created it. A dirty hack on the SDDB or the stack is also out of discussion, as it wouldn't be portable and quite unreliable. Shutting down the service is also not portable and not always cleanly feasible.

The only way to pursue this distributed approach would be to create a software for an open platform, such as OpenMoko [16] or Android [1]. However, this would not solve the problem of diffusion, as these platforms are not, as of now, widespread.

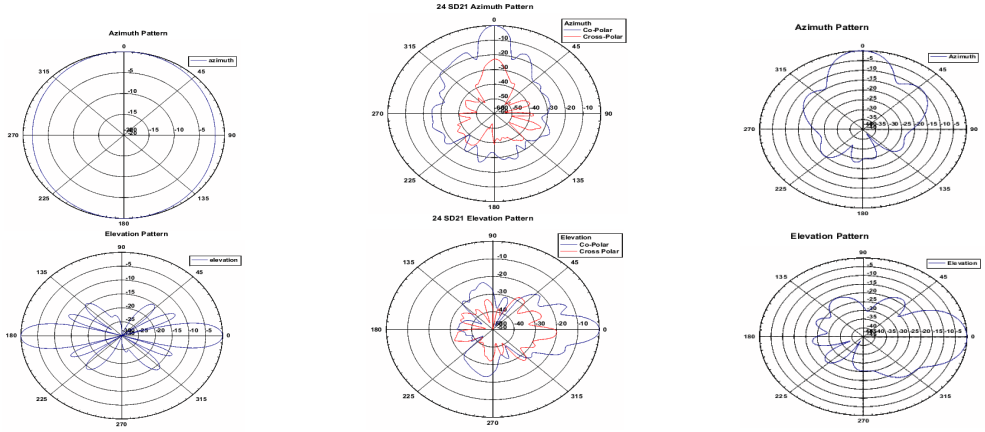
We resorted to a more traditional design, creating an ad hoc device based on the GNU/Linux OS to collect the samples. We made use of the official Bluetooth stack implementation named BlueZ [3]. Specific utilities allow to perform device configuration,

Figure 3.9: All the material used during the experiments.

scanning and information gathering. We created a python software, using the pybluez [18] package, to glue such utilities together, along with the gpsd [8] GPS daemon, and Colin Mulliner's secure OBEX server [19]. 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.

The hardware design is extremely complex, and a number of tradeoffs and choices were made (and are discussed in [33]). We bought and tested a wide array of devices, in different combinations and solutions. The complete set of devices tested is shown in Figure 3.9.

We chose the Asus EEE PC as being an optimal base platform: this device has a limited cost (250 €), very good autonomy (4–5 hours of honeypot operations when screen is disabled), as it uses a SSD instead of a classical hard disk, and is quite resistant to vibrations and movement. A common laptop would be similar, but with more limited autonomy and more sensitive to dust or vibrations as it uses a classical hard disk. Other



(a) Omnidirectional antenna (b) Parabolic antenna (c) Patch antenna

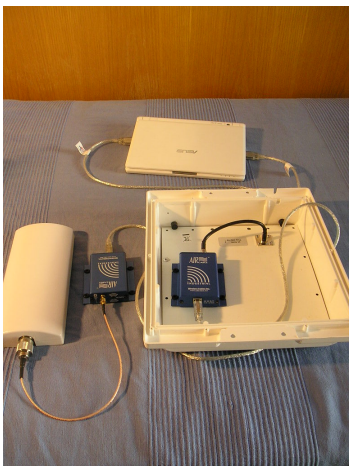
Figure 3.10: Specification diagrams of the different types of antennas used.

embedded devices (e.g. Soekris) were tested, but their limitations do not match the cost savings.

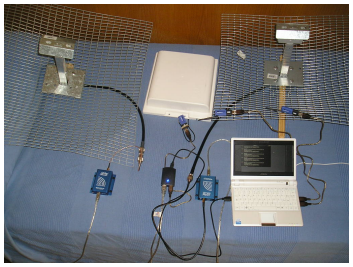
We tested several different types of antennas, both directional and omnidirectional:

- 12.5 dBi directional patch antenna
- 19 and 20.5 dBi directional parabolic antenna
- 3 dBi omnidirectional antenna
- 9 dBi omnidirectional antenna

Directional antennas offer obvious range advantages but only in a single direction. They are thus interesting for monitoring long, narrow, non-crowded zones, for instance monitoring a street from an apartment, or to be placed in an elevated position. Parabolic antennas (whose specifications are drawn in Figure 3.10(b)) are much more powerful than the patch one (specifications drawn in Figure 3.10(c)), but they are not easy to conceal whereas patch antennas are smaller and are easy to conceal in a bag. Omnidirectional antennas are discrete and easy to conceal in a bag as well. They are best to cover a large place, in particular if crowded and/or concealing them. Their specification is reported as Figure 3.10(a). All the specifications in Figure 3.10 are drawn from the hardware vendor's website [10].



(a) Midrange distance honeypot, based on patch antennas, easy to conceal.



(b) Long range honeypot, with 2 parabolic and 1 patch antennas.

(c) Concealed honeypot, based on two Aircable dongles and two 9 dBi omnidirectional antennas, hidden in a backpack.

Figure 3.11: Photos of three different honeypot sensors.

Range	Antenna Configuration	Dongle	Max Range	Cost	Conceal.	Mobile	Remarks
Long	2 parabolic + 1 patch	Normal	1.5 Km	620 €	No	Fixed	Long distance, wide angle (Duomo square)
Long	2 parabolic	Aircable	1.5 Km	490 €	No	Fixed	Long distance, wide angle (Duomo square)
Long	2 patch	Normal	600m	520 €	Yes	Yes	Mobile, can cover Duomo square
Long	1 parabolic	Normal	1.5 Km	400 €	No	Fixed	Long distance, narrow angle
Long	1 patch	Normal	600m	400 €	No	Yes	Long distance, narrow angle
Medium	2 omnidirectional	Aircable	120–200m	410 €	Yes	Best	For concealed monitoring
Short	just dongles	Normal	40–60m	310 €	Yes	Yes	Low cost reference

Table 3.1: Comparison of the different hardware honeypot solutions.

Normal class 1 dongles can be used for almost everything, but if an external antenna is needed, they must be modified (see [20] or [23]). Alternatively, an Aircable dongle can be used, as it has excellent coverage, works out of the box, and has a connector for external antennas (it is, of course, a bit more expensive). Most of time, we used 2 classical class 1 dongles and one Aircable dongle.

We tested different honeypot configuration, aimed for different uses :

Long range configurations aim to cover the longest possible distance. We use a powerful, directional antenna for receiving malware transfers, and different combinations of smaller ones for scanning (for a reason which will be explained in the following Section 3.3.1). The most powerful is shown in Figure 3.11(b).

Midrange configurations use just patch or omnidirectional antennas. Effective in small and not very crowded zones. Shown fitting in a compact box in Figure 3.11(a).

Short range configurations, for reference, create a honeypot based on simple class 1 dongles.

The comparative results of our tests are shown in Table 3.1. Maximum range is evaluated in open space, while effective range in crowded areas is difficult to define due to the effects that we describe in the following Section 3.3.1. Honeypots are defined as concealed if they can fit as "devices in a bag" that can be brought in public places (something akin to our experiment in [24]). An example of an honeypot with two Aircable dongles equipped with 9 dBi omnidirectional antennas is shown in Figure 3.11(c).

Preliminary Field Test Results

First of all, we tested the antennas for visibility of devices. We made various experiments in open spaces and in crowded spaces, with different types of devices (computer dongles and two different phones, Nokia e65 and 6680). We observed that:

- With two class 2 phones, in an open space, the range is approximately 20m
- With a Class 1 dongle (without an antenna) and a phone, in an open space, the range is approximately 57m
- With a Linksys dongle with external antenna and a phone, in an open space, we reached 90m
- With an Aircable dongle in an open space, we reached 110m (with a 3dBi omnidirectional antenna), 175m (9 dBi omnidirectional antenna), 400m (12.5 dBi patch antenna), 1.48Km (20.5 dbi parabolic antenna).

We were not able to replicate the long distance Bluesnarf experience which reportedly reached 1.78Km [20], even if our material was equal or better than the one used in that occasion.

After testing the software successfully with manual transmission of files from various devices, we field-tested the various combination of devices. A first test was made by placing for several days a long range honeypot on a street. 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.

Field tests revealed some unexpected issues: 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. 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 use 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, leads to a very difficult time for the scanner trying to enumerate devices, let alone for trying to receive a file. Therefore, a dense crowd will limit the effectiveness of long range solutions. 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.

3.3.2 VU's new sensors

Shelia

Shelia is a Windows-based intrusion detection system for the client side. The current version of Shelia is available from `http://www.cs.vu.nl/~herbertb/misc/shelia`. It consists of two main parts: a client emulator and an intrusion detection engine. The client emulator emulates the human user. It will read the email in a mail folder, follow the links in emails to visit potentially malicious websites, open attachments, etc. The intrusion detection engine will then determine whether the website, or the attachment is malicious.

Shelia's client emulator hooks into the API of Outlook Express, a standard mail client provided with Windows. It uses Outlook Express to move between messages, find the URLs and attachments, as well as their associated applications.

Use of Shelia is intended to be as intuitive as possible. For instance, in a command line version it is sufficient to specify which mailfolder should be scanned as follows:

```
shelia_client -sf "My Spam Inbox" [optional other parameters]
```

This means that shelia will read every email in this folder. If the email contains a URL, shelia will fire up the default browser and vistit the website. If the file contains an attachment, shelia will open it with the appropriate application. The optional other parameters allow users to specify things like time-outs and containment strategy.

The time-out determines how long Shelia will wait until the applications are killed. For instance, a time-out of one minute means that if Shelia finds a URL in an email, it will give the browser exactly one minute to get infected. After that, it will kill the browser and all potential helper applications that it has started. In other words, time-outs may help to deal with infections that do not occur immediately.

The containment strategy determines what should happen after an attack is detected. Should the attack be allowed to proceed or should it be contained? Containment means that when an exploit manages to take control of a client application (e.g., the browser or word processor), we try to contain what the payload that is then executed is allowed to do. A particularly powerful example of containment is that Shelia may allow the attack payload to download an executable (very typical behaviour for a remote attack), while preventing the execution of the malware. Indeed, it will collect the malware sample and store it in a log folder.

In the detection engine, Shelia is slightly more clever than many other client-side honeypots in that it tries to make sure that some suspect action (such as creating a file, or changing the registry) really is an attack, rather than a false positive.

Shelia monitors the processes and generates alerts when the process attempts to execute an 'dangerous' operation (i.e., execute a call to change the registry, create files, or attempt specific network operations) from a memory area that is not supposed to be executable code. As mentioned earlied, Shelia may even allow the attack to run until it downloads the malware, which it then tries to capture and store in a specific directory (not unlike the download of malware offered by projects like Nepenthes).

Multi-tier intrusion detection

There are many ways to detect and analyse intrusion. Some are heavy-weight but accurate, others fast but less accurate. Ideally, you would like to use different methods simultaneously. The problem with very accurate intrusion detection methods (such as taint and data flow analysis) is that they are heavy-weight slowing down processes

tremendously. But certainly applying all methods at the same time incurs unacceptable slowdowns.

Recently, we have developed a different approaches toward multi-level intrusion detection that tries to decouple the actual intrusion detection from the normal execution of the process. We are currently working towards harnessing these approaches in a usable sensors.

The approach works at the level of the VM. In summary, we record the execution of unmodified OSs runing on Qemu (in a first tier) in such a way that we are able to replay the execution exactly on another node/core. Just like in the ReVirt project we capture the sources of non-determinism and results of (certain) syscalls. Besides recording, the first tier performs no intrusion detection whatsoever and can therefore be fairly fast. In the 2nd tier, we can apply whatever intrusion detection method we fancy. In our case, we have experimented with different versions of Argos. Execution in the 2nd tier, while instrumented, tends to be faster than the first tier, because we do not need to perform many of the syscalls (and we certainly do not need to perform any network activity, as this was already done by the first tier). This allows us to do fairly expensive checking and still keep close to the original execution of the first tier. Since the execution trace is in a file, different checkers can look at the same execution in parallel. In fact the model trivially extends to more tier (for instance, where argos detects the intrusion and prospector determines exactly which network bytes where involved).

The paranoid Android

Related to the decoupling approach described in the previous section, we have turned our attention to mobile phones. Modern phones, like Apple's iPhone or Google's Android, are like general purpose computers in that they run increasing numbers of increasingly complex applications. There is no doubt that with the added complexity, the vulnerability to remoted attacks is also on the rise.

Unfortunately, smart phones are unlike general purpose computers in several other ways. For instance, they work on batteries and battery time is one of the most crucial factors in the phone's usability. Adding heavy-weight instrumentation to phones to detect reliably the occurrence of Internet attacks is simply not an option since the battery constraints dictate that every cycle is precious and heavy-weight intrusion detection method would severely limit the battery time.

Obviously one may take a less advanced intrusion detection methods (such as in-network anomaly detection) that is capable of finding some attacks. While OK for battery time, the lack of accuracy is not very satisfying.

We are currently inverstigating an alternative model, whereby the operational func-

tionality of the phone and the detection mechanisms are explicitly decoupled. Phrased differently, we do not attempt to perform any intrusion detection on the phone itself, but record enough of the execution state to allow replaying on a wired replica. We only apply heavy-weight instrumentation on the grid-powered replica.

For this project we have chosen to use Android phones, because of their open source nature, and the availability and support of the development environment. While real Android hardware has recently appeared, the work is still in the design phase. Nevertheless, if the project is successful, it means that we will be able to apply heavy-weight security instrumentation that would otherwise be wholly beyond the capabilities of modern phones.

3.3.3 NASK's HoneySpider Network (HSN) sensor

As part of NASK's contribution to WOMBAT, a new kind of sensor will be introduced as a feed to the WOMBAT database. The sensor is a type of honeyclient, and is being developed under a joint venture called the HoneySpider Network [32] project, together with GOVCERT.NL and SURFnet. The goal is to develop a complete client honeypot (or honeyclient) system - the HoneySpider Network (or HSN for short) based on existing state-of- the-art client honeypot solutions and a novel crawler application specially tailored for the bulk processing of URLs. The system focuses primarily on attacks against, or involving the use of, Web browsers. These potentially include the detection of drive-by downloads, malicious binaries and phishing attempts. Initially, the main area of exploration is drive-by downloads. Apart from identifying browser exploits (including 0day attacks), the system is expected to automatically obtain and analyze the attacking malware. Under WOMBAT, HoneySpider will be equipped with an API allowing the WOMBAT infrastructure to perform queries in a manner similar to the way the WOMBAT infrastructure can now query ARAKIS.

The introduction of a honeyclient-based sensor will allow the WOMBAT consortium access to information about malicious Web pages and malware that is spread through client side vulnerabilities. This is significant, as the existing WOMBAT sensors are based on traditional server side honeypots. They can therefore collect malware that spreads through vulnerabilities on the server side. It is often perceived however, that the current trend in attacks has shifted to the client side, rendering blind server side honeypot solutions. The reasons for this shift are seen as two-fold:

1. operating systems and network access (through firewalls for instance) have become better secured,

2. attacks have become more selective, targeting specific groups to make them more difficult to detect and to deploy efficient countermeasures.

Passing URLs in mail messages allows many security checks to be bypassed, while at the same time exploiting unpatched Web browsers and their plugins (or other client applications) on user desktops when users click on those URLs.

Low interaction and high interaction honeyclients

The HoneySpider Network leverages the benefits of both low interaction and high interaction solutions by utilizing both types of technologies and integrating them together through a management framework. Low interaction client honeypot solutions are faster, easier to maintain than high interaction solutions, but at the same time are usually unable to obtain malware that is being served by malicious web pages. The detection scope of both solutions is different as well:

- Low interaction client honeypots are able to detect suspect web sites in a generic manner (for instance, they can look for JavaScript code obfuscation, which is often used by attackers to hide attack vectors), but will have difficulties detecting a new exploit,
- High interaction client honeypots are more suited to the detection of new types of exploits thanks to the fact that they utilize real operation systems and configurations, but will not detect an attack against an application that they do not run.

Drive-by download detection

HSN focuses on the drive-by download problem. A drive-by download is the process whereby a user's operating system is compromised and malware is automatically installed with no user interaction other than the fact that the user pointed a browser to a URL. In a drive-by download (as well as in other, less sophisticated web based attacks) JavaScripts, VBScripts and iframes play an important role. Exploit code is not normally hosted directly on a malicious site visited by a user, but is served through a series of redirects from an exploit server. Scripting languages such as JavaScript are used to hide (obfuscate) code that is used for these redirects and then launch exploits that download and install malware.

Emulating browser behavior to handle the drive-by download process is a difficult task for low interaction client honeypots. The HSN crawler (based on heritrix [29] integrates

Rhino [41] and a DOM implementation to handle JavaScript and various heuristics to determine whether the scripts that are being executed are suspicious or malicious in nature. These heuristics include machine learning techniques, that are utilized in a manner similar to the way spam detection is performed. The main detection mechanism is based around the Na?ve Bayes classifier, implemented through Weka [47]. A large set of script samples are classified manually and labeled either benign, malicious or suspicious. N-grams are generated over each script sample, with the top most frequent n-grams (the exact number is configurable), selected as representative of a sample. These manually selected scripts and their n-gram representation are used for training the Naïve Bayes classifier. When the HSN crawler discovers JavaScript in a URL it is pointed to, it automatically computes the n-gram vector representation of the script and applies the classifier to determine its benign, suspicious, or malicious nature. At the same time it attempts to execute the script - any scripts obtained as a result of execution are also sent through the classifier.

The high-interaction component of HSN is Capture-HPC [46] based. Capture-HPC detects attacks against clients by modifying system calls and intercepting and registering their use. The low-interaction component is intended as filter for this component. HSN features a workflow process, whereby URLs are processed based on priorities and confidence levels assigned to their sources. These sources can be of various types, including spam URLs, URLs returned as a result of search engine queries, lists of URLs pulled through HTTP, lists of URLs published in files, as well as manually entered URLs.

An architecture overview of the HoneySpider Network is shown in Fig. 3.12.

Data contribution to WOMBAT

When the HoneySpider Network will be completed and operational, the new HSN sensor will contribute information about existing and emerging Web browser threats, and will provide to WOMBAT, through an API:

- lists of URLs suspected or verified as spreading malware,
- the method used to detect the URL, such as, for example, through a machine learning heuristic, or antivirus engine detection or through a high interaction honeypot,
- the type of source through which the URL was obtained, such as spam, search engine query etc,
- exploit information, if possible along with information as to whether the latest patched versions of applications were affected,

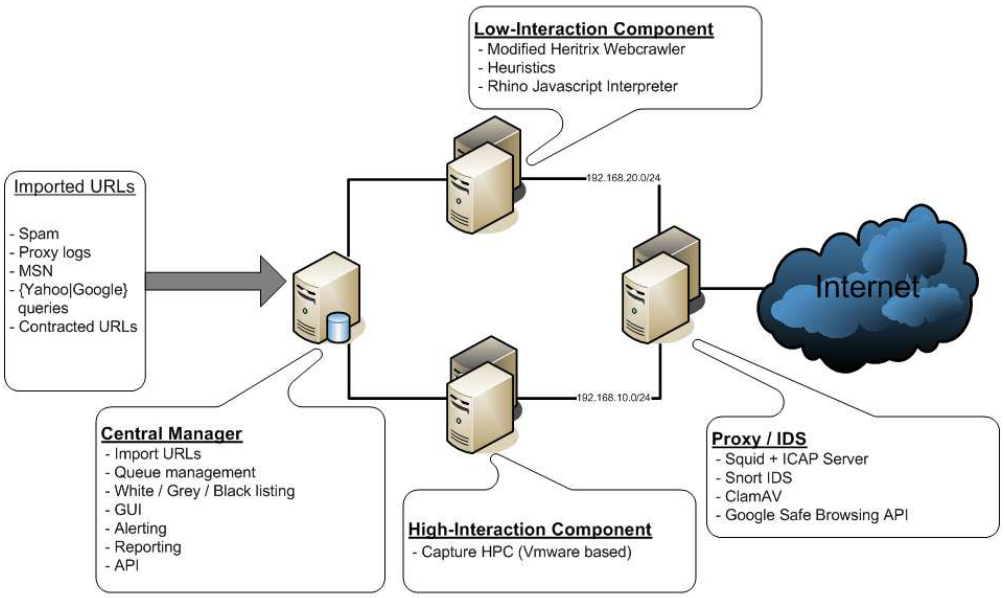


Figure 3.12: HoneySpider Network architecture overview.

- malware obtained, if any, along with associated meta-data, such as timestamps, MD5/SHA1 hashes of the data, output from external analysis engines,
- search capabilities, with search parameters including timestamps, URLs, malware hashes returning suspicious or malicious URLs and the associated malware.

The API will be compliant with the WAPI introduced in this document.

3.4 Existing Database

As mentioned in the previous sections, one of the main components that combine the WOMBAT infrastructure is the centralized database. This section will cover the current status of this component. At this point, it is worth briefly reminding the reader that the purpose of this database is threefold: i) provide a long term storage for data sources that do not provide it, ii) precompute and store the aggregated results of the most frequently asked queries and iii) act as a proxy to hide the various data sources that can be queried to answer a given question.

Currently,the database provides long term storage and also is able to hide the identities of the data sources. That is, it already implements purposes number (i) and (iii), but it does not yet provides storage for aggregated results. This will be implemented later for two reasons. Firstly, the WAPI should get more mature in order to be able to define all the data types/structures. Secondly, as more and more data sources will be integrated in the WOMBAT infrastructure and users will start to actually using it we will be able to tell which queries are "hot". These queries will be the ones that will be periodically precomputed and stored in the database for efficiency.

In the next section we give a detailed description of the current database's structure. In Section 3.4.3 we summarise the classes of queries that are supported by the database and finally Section 3.4.4 provides some examples of usage.

3.4.1 Structure

Figure 3.13 shows the current schema of the database. The core of the structure is the *elements* table which is used to distinguish all the elements stored. This table is "connected" to some other ones that contain information which is more specific to the type of each element. These are what we call the *Data type specific tables*. As shown in the schema of the database, these tables share some connections with the *files* table, which falls in the category *General use tables* because it can be used by any *Data type specific table*. Finally, there are three other tables on the right side of the schema, namely *Higher level tables*, that are somehow broken away from the previous tables. These tables are used to store more higher level information that is not used for the storage of the raw data but to store some more user friendly names and also some relationships.

In the following subsections we will go through each of the four categories that where previously mentioned. These are: *Core table, Data type specific tables, General use tables,* and *Higher level tables*.

Core table

The abstraction of the data unit in the WOMBAT database is the element. Each element has some fundamental attributes that are stored in the *elements* table. These attributes include a human readable name, a unique ID used to differentiate every element, a data source origin, a data type and the date that it was added to the database. All these attributes can be also seen from the figure of the schema (fig. 3.13).

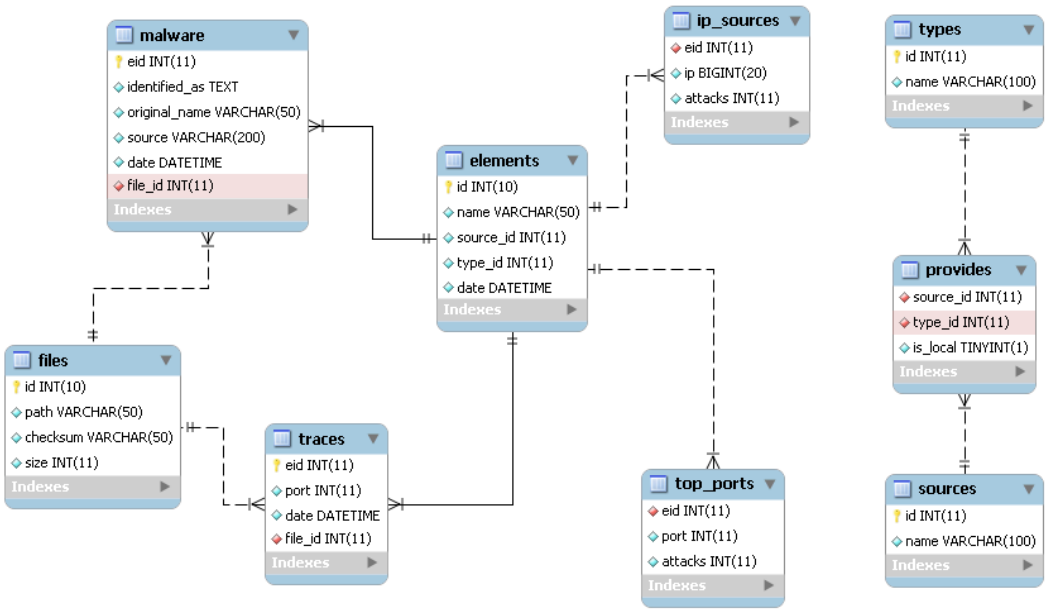


Figure 3.13: Database Schema

Data type specific tables

The category next to the *Core table* is the *Data type specific tables*. These tables are used to store some attributes that are related to each element's data type. For example, a malware sample from the Virustotal system has different metadata than a summary of the *Top 100 attacked TCP ports*.

Currently, the database supports the storage of four data types, which means that it contains four tables of this category. These are:

- **malware.** As its name implies, this table is used to store malware metadata. These include the original name of the binary, the names given to that malware by a set of Anti-Virus vendors etc.
- **traces.** Similarly, the *traces* table is use to store PCAP network traffic traces metadata. Currently, the metadata that can be stored are just the destination TCP port, but this table can be easily extended to include more as needed (e.g. the duration of the trace file, the protocol of each layer etc).
- **ip_sources.** One of the goals of the database is to provide long term storage

for data sources that do not support it. This table fulfils that need for the IP addresses of attack sources. Using this table, the WOMBAT database enables saving the history over data sets that are available in the form of daily TOP-X lists of IP addresses of attack source. Technically speaking, this is done efficiently be storing only the different parts of that list daily.

- **top_ports.** This tables is much more like the previous one - *ip_sources* - but for a different type of data, that is the *attacked TCP ports*.

General use tables

The common characteristic among the tables falling to this category is the supply of a common place to store attributes shared by more than one data types. Currently this category contains just one table, the *files* table, which provides the storage of files to any data type that needs it. For example, both malware samples and PCAP network traffic traces have to store files, so they both use the *files* table which is more efficient than duplicating the effort for file storage.

Higher level tables

This category consists of tables that are storing somewhat more high level information about the system. None of these tables is used for raw data storage. Thats why these tables are not connected to any of the previous described ones.

There are two types of *Higher level tables*: mappings and relationships. Currently, there are two tables that provide mappings and one that represents a relationship. All of them are described in more details next.

- **sources.** This table provides a simple **mapping** between IDs and the human readable names of the data sources.

- **types.** Similar to the previous but the **mapping** provided is between IDs and the types of data. For example, 'malware samples', 'IP addresses of attack sources', 'PCAP network traffic traces' etc.

- **provides.** As opposed to the previous two tables, this one provides a **relationship**, namely *provides*. This relationship is between the data sources and the data types and states what types of data is provided by each data source and whether this is stored locally to the database of should be fetched by a remote repository. In other words, this table stores the who-provides-what-and-how relationship.

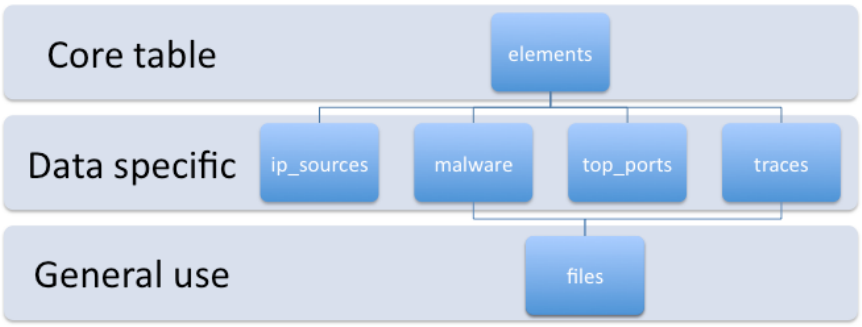


Figure 3.14: Hierarchical Overview

3.4.2 Extensibility

A clear target that is always in our mind during the designing of the database is the extensibility. The design of the database should be easily extensible to include more data types and data sources without altering its previous functionality. In addition, it should also leave space for improving its performance by making more efficient storage decisions.

We believe that both of this goals are achieved by the current preliminary design of the database. Figure 3.14 shows a hierarchical view of the tables used for data storage. In the top layer is the *Core table*, the *elements*. Right bellow is the *Data type specific tables*, e.g. *malware*, *ip_sources* etc. Finally, on the bottom layer are the *General use tables*, which currently consists of just the *files* table. As far as the first goal is concerned, that is extending the data types and data sources, our database design deals with it in the following way. If we want to add a new data type from a data source that does not provide long term storage then what we have to do is just add an appropriate table on the middle layer in the hierarchy. In addition, the current design supports the addition of new data sources for already integrated data types for free. Technically, we can reuse the tables for storing data type specific attributes for more that one data sources. Also, as far as the second goal is concerned, we can improve the overall robustness of the database by adding more *General use tables* in the bottom layer as needed.

3.4.3 Queries

In this section we will go through the *queries* that are currently supported by the WOMBAT database. The number and the nature of these queries depends on the current data sources and also on the current data types integrated to the database. Information on both these quantities is stored in the *Higher level table provides*. Recall that this table keeps track of who (data source) provides what (data types) and how (local or remote).

Table 3.2 shows the contents of the *provides* table as of now. These are also somehow represent the progress of the integration of other systems to the WOMBAT infrastructure. Each cell of the table may have three possible values on for each pair of "data source - data type". These are: *blank*, local or remote. Blank means that this combination is not integrated to the WOMBAT infrastructure. This may happen either simply due to the fact that the specified source does not provide the specified data type or because the integration is still in progress. "local" means that the current pair is already integrated and also the actual data are stored locally to the WOMBAT database. The reason for the local storage can be because a data source may not provide long term storage to its data (e.g. Sans.org). Finally,"remote" also denotes that the pair is already integrated and the actual data are stored to a remote repository - not in the WOMBAT database.

Currently, we consider the data query as a three step procedure. On each step the user in narrowing his search criteria in order to express the actual data subset he wants. The steps are the following:

1. **Data type selection.** The first step is to select the data type the user is interesting for. Currently, as shown in table 3.2, these are *malware, PCAP trace files, Top sources* and *Top Ports*.

2. **Data source(s) selection.** After the data type is selected then the user should select the data source or sources he wants to query. The set of the data sources that provide the actual data can be determined by querying the *provides* table.

3. **Search parameters.** Finally, after selecting the data type and data source(s), the user should now enter the actual search parameters. For example, if the user wants to query the database for a set of malware samples then some possible search parameters could be the original name of the binary, the name that this sample was identified by other AntiVirus vendors, the date that this sample was inserted to the database etc.

After the successful completion of the previous procedure, a subset of the database's data will have been selected. This subset can then be used for further processing.

Source/Type		Malware	PCAP Trace	Top Sources	Top Ports
VirusTotal		local			
NEMU			local		
Leurré.com				remote	remote
Sans.org				local	local
Offensive Computing		remote			
SGNET		remote			

Table 3.2: *provides* table contents.

After a subset of the database's data has been selected using this procedure, further processing can incur. In this preliminary stage of the database, the processing we have considered is simple operations like intersection or union between the resultsets of different data sources. Although, this procedure is sufficient for this type of operations, as the database evolves and new data types and processing operations are added we may need to revise it. For instance, under some circumstances we may want to correlate different data types, for example the destination port of the PCAP network traffic trace files with the Top Ports.

3.4.4 Sample Usage

In this section we provide a couple of examples for the better understating of the current design of the WOMBAT database. The first example has to do with data insertion whereas the second one with data retrieval.

Example 1: Malware sample insertion

In this example we suppose that a new malware sample arrives from a data source that does not provide long term storage. In that case, the malware sample should be saved among its metadata. This is done in the following three steps:

1. **Element insertion.** The first step is to insert a new record in the *elements* table containing a newly generated unique element ID, a name, source id with the value of the actual source, type id with the value of malware and the current date.

2. **File insertion.** Next, a new record is inserted to the general use *file* table. This record contains again a newly generated unique file ID, the path of the file on the filesystem, its checksum and its size.

3. **Malware insertion.** Finally, a new record is inserted in the *malware* table, recall that it is one of the *Data type specific tables*. This record is connected to the previously inserted element and file by storing the previously generated IDs. It also contains the actual malware metadata, like the original binary name etc.

After the successful completion of the previous steps, the new malware sample will have been added to the WOMBAT database.

Example 1: Data query

The scenario in this example of usage is the following: we want to query the database in order to get the *Top TCP ports attacked* as seen by all the data sources. This includes the following steps:

1. **Find candidate data sources.** The first step is to enumerate the data sources that provide *Top TCP ports attacked* by querying the *provides* table. Recall that this table stores the who-provides-what-and-how relationship.

2. **Retrieve the results.** Generally the data from the sources that provide *Top TCP ports attacked* can be either stored locally, if the specific data source does not support long term storage, or remotely. In the first case our database will have to be queried whereas in the second one, the results must be fetched from a remote repository.

3. **Process.** After the results from each data source, either locally or remotely stored, are gathered, further process can incur - like correlation.

4. **Present the results.** Finally, the results are presented to the user.

Bibliography

[1] Android. http://code.google.com/android/.

[2] Anubis. http://anubis.iseclab.org/.

[3] Bluez website. http://www.bluez.org/.

[4] Cabir. Analysis available online at http://www.symantec.com/security_response/writeup.jsp?docid=2004-061419-4412-99.

[5] Cardblock. Analysis available online at http://www.symantec.com/security_response/writeup.jsp?docid=2005-100315-4714-99.

[6] Commwarrior. Analysis available online at http://www.symantec.com/security_response/writeup.jsp?docid=2005-030721-2716-99.

[7] DNSBL definition on wikipedia.org. http://en.wikipedia.org/wiki/DNSBL.

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

[9] Home page of "Network of Affined Honeypots (NOAH)". Available online at http://www.fp6-noah.org.

[10] Home page of the Stella Doradus group. Available online at http://www.stelladoradus.com.

[11] *JSR-000082 Java(TM) APIs for Bluetooth 1.1 Maintenance Release*. Chapter 6. Available online at http://www.jcp.org/en/jsr/detail?id=82.

[12] Leurre.com. http://www.leurrecom.org/.

[13] Microsoft security bulletin nr. 30, 2008. Available online at http://www.microsoft.com/technet/security/Bulletin/MS08-030.mspx.

[14] Motorola RAZR JPG Processing Stack Overflow Vulnerability. Available online at http://www.zerodayinitiative.com/advisories/ZDI-08-033/.

[15] Multidrop. Analysis available online at `http://research.sunbelt-software.com/threatdisplay.aspx?name=Trojan.MultiDrop.IC\&threatid=198898`.

[16] Openmoko. `http://www.openmoko.com`.

[17] Pbstealer.a. Analysis available online at `http://www.symantec.com/security_response/writeup.jsp?docid=2005-112216-0519-99`.

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

[19] Secur obex server. Available online at `http://www.mulliner.org/bluetooth/sobexsrv.php`.

[20] Trifinite.org website. `http://www.trifinite.org`.

[21] VirusTotal. `http://www.virustotal.com/`.

[22] P. Betouin. Dossier sécurité bluetooth - partie 5 - scénarios d'attaques & synthése. Available online at `http://www.secuobs.com/news/05022006-bluetooth5.shtml`.

[23] L. Carettoni. "moddare" un dongle bluetooth con 14€. Available online at `http://www.ikkisoft.com/stuff/moddongle.pdf`, 2005.

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

[25] M. Dacier, F. Pouget, and H. Debar. Attack processes found on the internet. In *NATO Symposium IST-041/RSY-013*, Toulouse, France, April 2004.

[26] M. Dacier, F. Pouget, and H. Debar. Honeypots, a practical mean to validate malicious fault assumptions. In *Proceedings of the 10th Pacific Ream Dependable Computing Conference (PRDC04)*, Tahiti, February 2004.

[27] C. Gehrmann, J. Persson, and B. Smeets. *Bluetooth Security*. Artech House, Inc., Norwood, MA, USA, 2004.

[28] S. Hager, C.T.; Midkiff. Demonstrating vulnerabilities in bluetooth security. In *Global Telecommunications Conference GLOBECOM '03*, volume 3, pages 1420 – 1424, December 2003.

[29] Internet Archive. Heritrix - home page. Available online at `http://crawler.archive.org/`.

[30] Internet Storm Center. Home page of "Internet Storm Center". Available online at `http://isc.sans.org`.

[31] M. Jakobsson and S. Wetzel. Security weaknesses in bluetooth. In *CT-RSA 2001: Proceedings of the 2001 Conference on Topics in Cryptology*, pages 176–191, London, UK, 2001. Springer-Verlag.

[32] P. Kijewski, C. Overes, and R. Spoor. The HoneySpider Network: Fighting client-side threats. *20th Annual FIRST Conference on Computer Security Incident Handling*, June 2008.

[33] A. Kokos, A. Galante, and S. Zanero. Bluebat: Towards practical bluetooth honeypots. Submitted for publication to the EC2ND conference, 2008.

[34] C. Leita and M. Dacier. SGNET: a worldwide deployable framework to support the analysis of malware threat models. In *7th European Dependable Computing Conference (EDCC 2008)*, May 2008.

[35] 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.

[36] C. Leita, K. Mermoud, and M. Dacier. Scriptgen: an automated script generation tool for honeyd. In *21st Annual Computer Security Applications Conference*, December 2005.

[37] C. Leita, V. H. Pham, O. Thonnard, E. Ramirez-Silva, F. Pouget, E. Kirda, and M. Dacier. The Leurre.com Project: Collecting Internet Threats Information using a Worldwide Distributed Honeynet. In *1st Wombat Workshop*, 2008.

[38] Maxmind. Ip geolocation and online fraud prevention, `www.maxmind.com`.

[39] R. Morrow. *Bluetooth Implementation and Use*. McGraw-Hill Professional, 2002.

[40] M. Moser. Busting the bluetooth®myth – getting raw access. Available online at `http://www.remote-exploit.org/research/busting_bluetooth_myth.pdf`, 2007.

[41] mozilla.org. Rhino: Javascript for java. Available online at `http://www.mozilla.org/rhino/`.

[42] NASK/CERT Polska. Public home page of project arakis. Available online at `http://www.arakis.pl`.

[43] S. Needleman and C. Wunsch. *A general method applicable to the search for similarities in the amino acid sequence of two proteins.* J Mol Biol. 48(3):443-53, 1970.

[44] N. Provos. A virtual honeypot framework. In *12th USENIX Security Symposium*, pages 1–14, August 2004.

[45] Y. Shaked and A. Wool. Cracking the bluetooth pin. In *MobiSys '05: Proceedings of the 3rd international conference on Mobile systems, applications, and services*, pages 39–50, New York, NY, USA, 2005. ACM Press.

[46] The Honeynet Project. Capture-HPC Client Honeypot/Honeyclient. Available online at `https://projects.honeynet.org/capture-hpc`.

[47] The University of Waikato. Weka 3 - data mining with open source machine learning software in java. Available online at `http://www.cs.waikato.ac.nz/ml/weka/`.

[48] M. Zalewski and W. Stearns. Passive OS Fingerprinting Tool.