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

WORLDWIDE OBSERVATORY OF
MALICIOUS BEHAVIORS AND ATTACK THREATS

D23 (D5.3) Early Warning System: Experimental report

Contract No. FP7-ICT-216026-WOMBAT

Workpackage	WP5 - Threats Intelligence
Author	Adam Kozakiewicz
Version	1.0
Date of delivery	M40
Actual Date of Delivery	M40
Dissemination level	Public
Responsible	NASK
Data included from	EURECOM, FORTH, HISPASEC, NASK, TUV

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

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

The WOMBAT Consortium consists of:

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

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

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

Contents

Abstract

A large part of Workpackage 5 concerns the Early Warning System functionality. This deliverable offers a report of the experiments carried out as part of the effort to create the Early Warning System. Several specialized alerting systems are presented, including FIRE, Exposure, BANOMAD and HoneyBuddy myIMhoneypot

1 Introduction

Each year, the Internet becomes a more important part of the average person's life. Its abilities are no longer limited to presenting information. We use it to purchase things (and pay for them), plan evenings, stay in touch with friends, share documents, photos, chat and more. As a large part of the society becomes dependent on the Internet in everyday life, a rich hunting ground opens up for different kinds of malicious activity.

The role of security specialists is to investigate such threats, learn about them and counter them. A large part of the WOMBAT project focused on supporting them in these actions by enabling the sharing of data using the WOMBAT API, creating new data sources, developing new analysis methods and data enrichment techniques. An equally important task is to publish important security alerts in a timely manner. This is the goal of the Early Warning System.

The initial plan of this deliverable included the specification and creation of a single Early Warning System, which would have access to all the datasets and analysis methods, extract the important information and provide it to the users who need it.

During the work in WP5 it quickly became clear that this approach is actually counterproductive. The collected ideas for elements of the Early Warning System differ in too many aspects to smoothly integrate with each other. They work in different timescales, with different goals, monitor different parts of the Internet technology landscape and even have different target groups. The synergy is dwarfed by the differences. For this reason the consortium decided to develop parts of the Early Warning System as separate services, limiting the integration to places with sufficient synergy (e.g. sharing datasets).

The following chapters describe the services built as elements of the Early Warning System. The first two chapters describe the FIRE system (FInding Rogue nEtworks, TUV) – Chapter 2 describes the advancements in the system since its description presented in D12, while Chapter 3 concentrates on its integration with NASK's HoneySpider Network. Chapter 4 describes Exposure – the Eurecom's system warning about malicious domain. Chapter 5 describes the Hispasec's BANOMAD system, designed for the banking industry and warning against banking trojans and targetted attacks. Finally, in Chapter 6 FORTH describes the HoneyBuddy myIMhoneypot, warning instant messaging users about threats using this technology. The last chapter presents a brief summary of the document.

2 FIRE: FInding Rogue nEtworks

In this section, we will present results obtained from the long-term deployment of FIRE. FIRE is a system for identifying networks that persistently host malicious behavior such as botnet command and control servers or malicious web sites. FIRE thus helps locate networks that host malicious infrastructure and, by malice or disinterest, do not take timely action to take it down. The top offenders are listed in a public web site[1]. FIRE was developed as part of WOMBAT, and the system was presented in a research paper [13] and included in WOMBAT deliverable "D12 (D5.1) Root Causes Analysis". In this deliverable, we will present additional results, anecdotes and insight derived from the long-term operation of FIRE.

2.1 Improved Data

Since the initial deployment of FIRE described in D12 and in [13], we have made a number of improvements to the system that have increased the coverage of the malicious activities that we are able to track. FIRE tracks the following three classes of malicious infrastructure:

- **Command and Control (C&C)**: Servers used by botnet operators to dynamically control the behavior of infected computers.
- **Drive-by-Download**: A drive-by-download is an attack delivered by a malicious web sites to visitors with vulnerable web browser configurations, that allows an executable to be automatically installed on the victim machine without user interaction. Since the malicious web sites themselves are frequently hosted by benign, compromised hosts, they are not tracked by FIRE. Instead, we track the backend servers that deliver the malicious executables.
- **Phishing**: Servers that host web sites designed to trick users into revealing sensitive information, such as credit card numbers or credentials for other web sites.

[1]http://maliciousnetworks.org

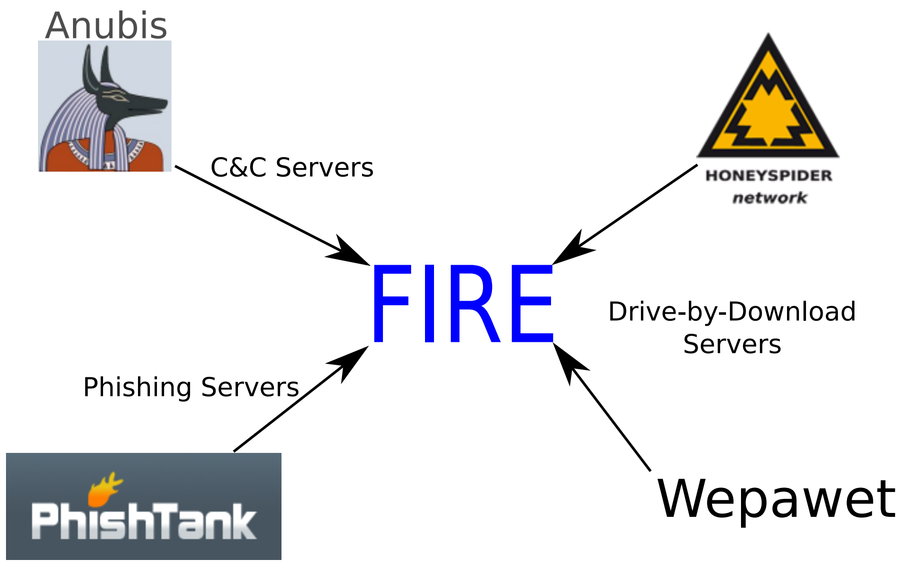


Figure 2.1: Sources of information used by FIRE

Figure 2.1 provides an overview of the sources of information on malicious infrastructure that are currently used by FIRE. To identify phishing servers, we rely on PhishTank[2], an external source of manually verified phishing URLs. C&C servers are detected based on the network behavior of samples observed within the Anubis malware analysis sandbox [6]. For drive-by-download attacks, we include servers identified by Wepawet [8] and HoneySpiderNetwork [11]. Compared to the initial version of FIRE, we have extended and improved the sources of data that the system relies on.

- **HoneySpider Network**: FIRE intially used two sources of information for drive-by-download servers. The first is Wepawet, a client honeypot that uses an instrumented browser to detect malicious javascript code. Wepawet was developed at the University of California at Santa Barbara, and has been integrated into the WOMBAT project through the WOMBAT API. In addition to Wepawet, the original FIRE deployment made use of a high interaction client-honeypot based on Capture-HPC[3]. This solution had been developed at the Technical University Vienna, but suffered from limited scalability and high maintenance. Therefore, we have replaced this source of information with the HoneySpiderNetwork (HSN) that

[2]http://www.phishtank.com
[3]https://projects.honeynet.org/capture-hpc

was developed by NASK as part of WOMBAT. HSN also includes a high interaction honeypot component based on Capture-HPC, but it is a more robust and scalable system. Thanks to a large-scale deployment and a filtering layer that uses machine-learning techniques to select suspicious web sites for analysis, HSN is able to detect a significantly larger amount of malicious web sites. An in-depth description of HSN was provided in WOMBAT deliverable "D07 (D3.2) Design and prototypes of new sensors".

- **Improved C&C detection**: To identify C&C servers, we rely on the behavior of samples observed within the Anubis malware analysis sandbox. However, for a variety of reasons, not all the network traffic generated by analyzed samples is related to C&C activity. FIRE therefore makes use of signatures and heuristics to detect C&C communication within Anubis. Initially, FIRE was able to detect IRC-based C&C based on a set of heuristics, and HTTP-based C&C based on a small set of network signatures. Over time, we have made several improvements to C&C detection in FIRE:
  - We have greatly extended the set of signatures used to detect HTTP-based C&C traffic. We have also verified that these signatures detect C&C communication in a significant fraction of analyzed samples and that the detected C&C servers cover almost all of the most prevalent malware families currently in the wild.
  - We have extended C&C detection to TCP protocols other than IRC and HTTP, with a new set of signatures that reveals a significant number of previously "invisible" C&C servers.
  - C&C detection in FIRE also indirectly benefits from many improvements that have been made to Anubis itself, and from the increase in the number of samples that Anubis is able to analyze each day, which is now over 30,000.

2.2 Data Analysis

The internet threat landscape is extremely dynamic. The long-term deployment of a large-scale monitoring infrastructure such as FIRE provides us rich opportunities to observe trends in internet threats. However, when analyzing these trends, we have to take into account that our coverage can never be perfect. Because of the adversarial nature of the monitoring task, we may have fluctuations in our coverage of internet threats that reflect the current state of the arms race between our analysis and miscreants'

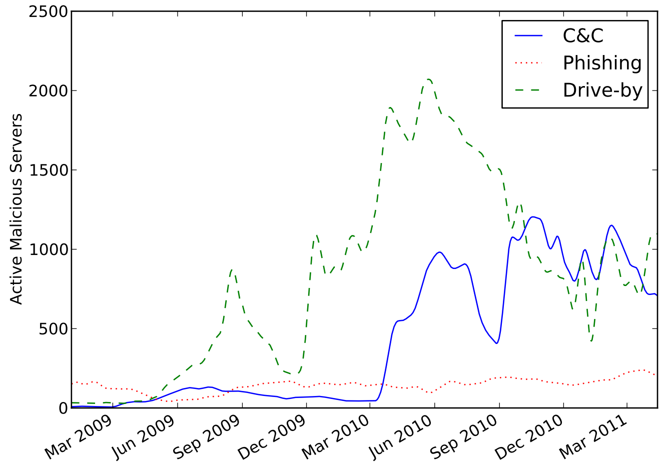


Figure 2.2: Active malicious servers by class of threat (smoothed).

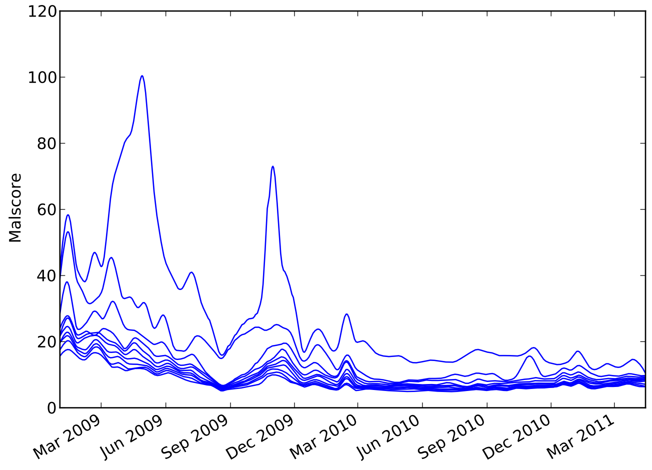


Figure 2.3: Malscores of top 10 malicious Autonomous Systems (smoothed).

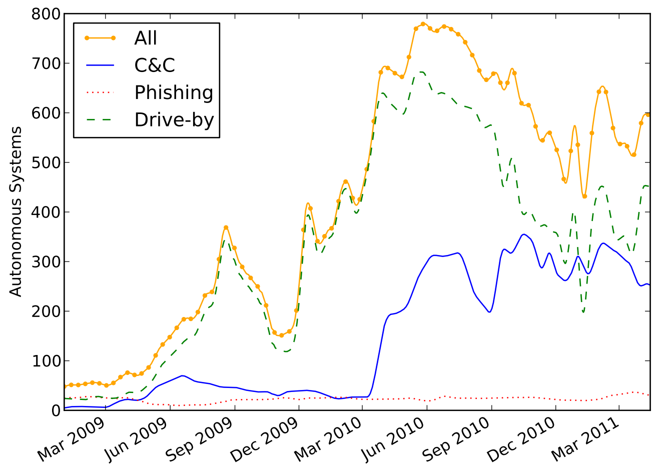


Figure 2.4: Autonomous Systems hosting at least one malicous server (smoothed).

attempts to escape detection. When discussing observed trends, we will therefore take into account the effect of changes in our own monitoring and detection techniques, and attempt to differentiate it from actual changes in the threat landscape.

Figure 2.2 shows the total number of long-lived malicious servers monitored by FIRE that were active on each day, over a period of over two years starting on January 1st, 2010. Note that the graph is smoothed using a 30-day sliding window to improve readability. The amount of active phishing servers is roughly constant over the entire period. For C&C and Drive-by-download servers, however, the picture is extremely different, with sharp increases compared to the early months of FIRE operation, and signficant fluctuations. Consider that the top malicious ASNs presented in [13] were based on data from June 1st, 2009. Since then, the amount of malicious servers detected has increased by an order of magnitude. For C&C servers, we can attribute a large part of this variation to improvements in our detection of C&C communication over time. Specifically, in March 2010 we deployed a large set of new signatures for HTTP-based C&C, and in October 2010 we introduced the first signatures for TCP protocols other than HTTP or IRC. Both events correspond to steep increases in the amount of C&C servers tracked. For drive-by malware download servers, the reasons for the variations are less clear. These variations largely correspond to fluctuations in the amount of malicious pages detected by Wepawet over time. These in turn may be related to the Wepawet's total analysis throughput (number of web pages analyzed per day) and to the effetiveness of its detection techniques. Wepawet is continuously updated by its developers to be able to recognize novel infection vectors and to remain ahead of evasion techniques employed by malicious web pages. Unfortunately, the effect of the integration of HSN on our coverage of drive-by-download servers is not yet visible in this data, because the integration was completed at the very end of the period considered. Since our detection techniques have such a significant effect on the amount of malicious servers we observe, we are unable to draw any conclusions on variations in the amount of malicious servers on the internet.

2.2.1 Autonomous Systems

The primary goal of FIRE is to identify the networks that provide reliable hosting to internet criminals. For this, we consider the Autonomous System (AS) to which each malicious IP belongs. We assign a *Malscore* to each AS that is based on the amount of malicious activity it hosts and on the size of the network. This is to avoid assigning excessively high Malscores to large networks that host a small amount of malicious servers, in proportion to the total amount of hosts. Figure 2.3 shows the evolution over time of the Malscores of the ten autonomous systems with the highest Malscores. Note that the top ten networks change over time: For this graph, we simply select the top ten Malscores on each day.

In this case, we can observe a clear trend. The top malscores have decreased over time: No network has had a Malscore above twenty in the last year, and the top ten networks have become tightly clustered around a Malscore of ten. This would seem to be in contrast with the results discussed in the previous section: The Malscores of the networks hosting the most malicious activity have decreased, despite the fact that the total amount of malicious activity that we are able to monitor has increased significantly, as reflected in Figure 2.2. In fact, there is no contradiction; The reason is that malicious servers are now spread across a much larger number of Autonomous Systems. Figure 2.4 shows that the total number of ASs tracked by FIRE has increased from under two hundred in June 2009, to a peak of almost eight hundred. Comparing Figure 2.4 with Figure 2.2 shows that this increase is roughly aligned with the variations in malicious servers discussed in the previous section.

The decrease in the top Malscores shown in Figure 2.3 seems to reflect an actual change in the threat landscape. The reasons for this change are twofold. On the one hand, we have a direct effect of FIRE's deployment, and of the public availability of the list of top twenty Autonomous Systems by Malscore that is provided by the maliciousnetworks website. This service is well known in the security community, and has attracted some media attention [9]. Several hosting providers who found themselves in the list of the top twenty malicious networks have taken an interest in addressing the problem and have contacted us. In each case, we provided additional information on the malicious servers in their network. This typically led to sharp decreases in Malscore over the following months. However, there is a second, probably more important reason for the decrease in the top Malscores. This is the fact that, after the de-peering of malicious networks such as Atrivio and McColo in 2008, and 3FN in 2009 [10], internet criminals have changed modus operandi to avoid "putting all their eggs in one basket". Instead of relying on a few complicit or negligent network operators, they now prefer to distribute their infrastructure across multiple providers. As a consequence, for example, taking down all the C&C servers of a botnet now typically involves interacting with several ISPs.

2.2.2 Threat Lifetime

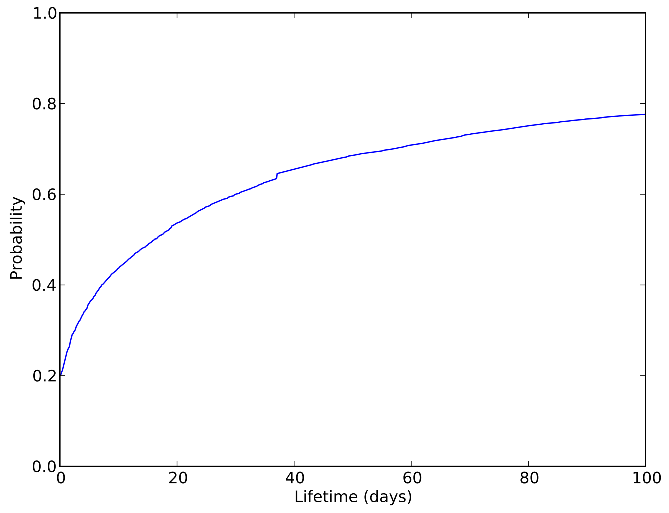


Figure 2.5: Cumulative distribution function of lifetime of drive-by-download server IPs

Figures 2.5-2.9 show the Cumulative Distribution Function of the lifetime of tracked malicious servers in the various classes. Results for C&C servers have been broken down to HTTP, IRC, and other TCP protocols. Results are consistent across most of these

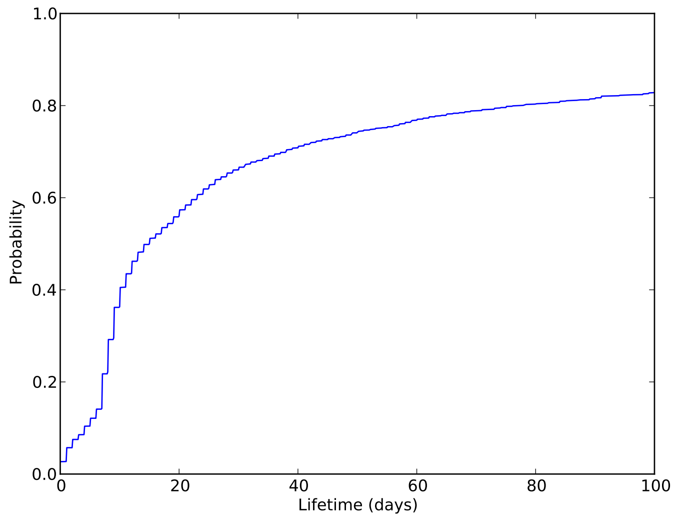


Figure 2.6: Cumulative distribution function of lifetime of phishing server IPs

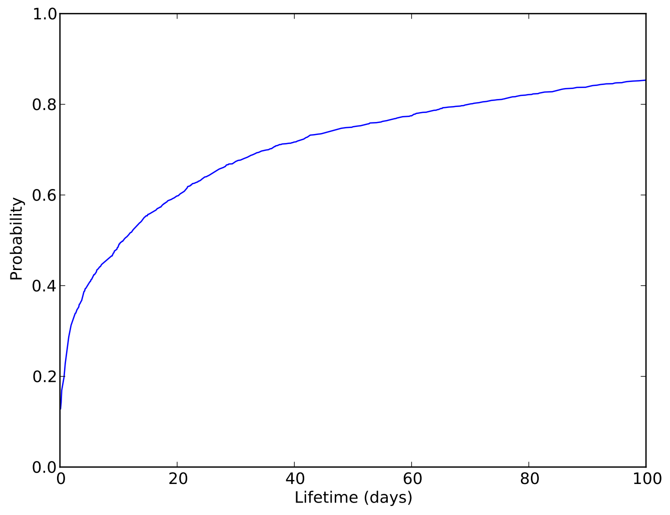


Figure 2.7: Cumulative distribution function of lifetime of IRC C&C server IPs

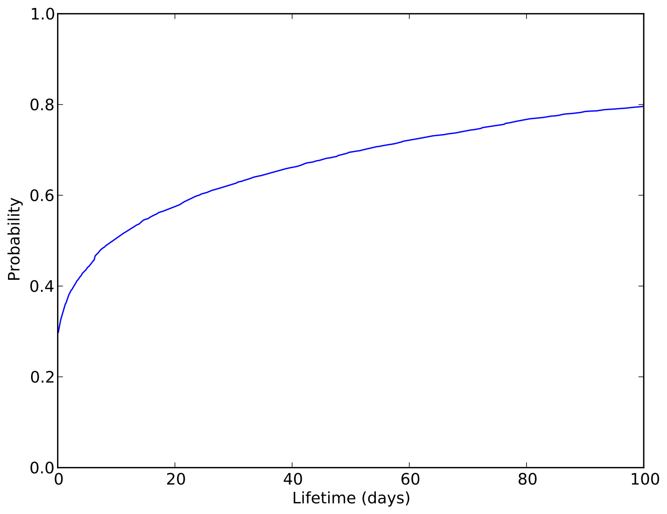


Figure 2.8: Cumulative distribution function of lifetime of HTTP C&C server IPs

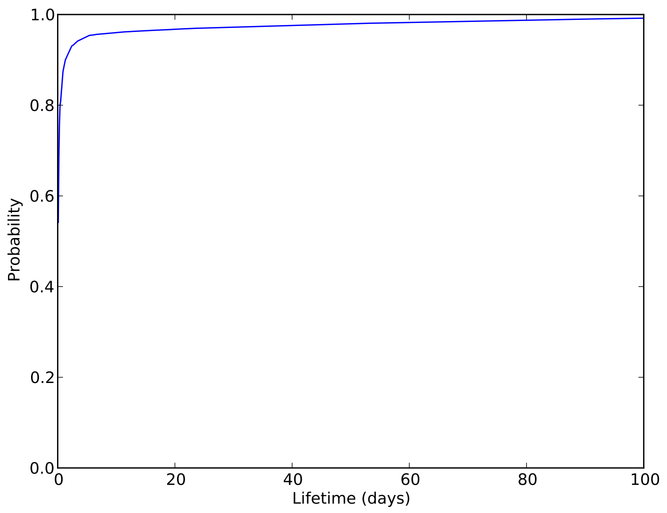


Figure 2.9: Cumulative distribution function of lifetime of other TCP C&C server IPs

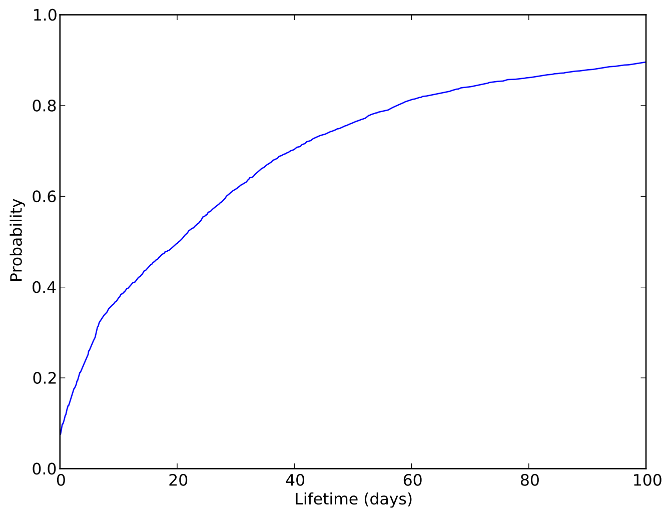


Figure 2.10: Cumulative distribution function of lifetime of other TCP C&C server domains

Table 2.1: Total malicious IPs and domain names observed.

	IPs	**Domains**
Drive-by	8357	8495
IRC C&C	987	237
HTTP C&C	10599	7568
TCP C&C	73873	3943

figures, with close to 20% of threats having a lifetime of less than one day, and another 20% approximately that have a lifetime longer than a hundred days. The obvious outlier is Figure 2.9, that shows the lifetimes for C&C servers that use other TCP protocols. In this case, about 80% of servers have a lifetime below one day. In fact, over 50% have a lifetime of exactly zero, meaning that the IP was successfully contacted only once by FIRE's monitoring infrastructure. The reason for these results is that most of these IPs are part of a fast-flux infrastructure. To contact these C&C servers, bots first resolve a domain name. In fast-flux hosting, botnet operators frequently change the IP addresses to which this domain resolves. These IP addresses may correspond to infected bots that are temporarily used to host the botnets' C&C servers. This is confirmed by Figure 2.10, which shows the lifetime of domain names of malicious TCP C&C servers. This graph is much more similar to Figures 2.5-2.8. Table 2.1 provides further evidence that fast-flux behavior is much more wide spread for C&C infrastructure that does not rely on the HTTP or IRC protocols. FIRE has observed over 70,000 IP addresses of TCP C&C servers, but these correspond to under 4,000 domain names. In contrast, IRC-based C&C frequently relies on hardcoded IP addresses, without making use of domain name resolution. The HTTP protocol, on the other hand, has the peculiarity that it allows the co-existance of multiple web sites on a single IP, distinguished only by the domain name. This allows malicious web sites deployed on shared hosting providers to share the hosting server and IP address with benign web sites, complicating takedown.

3 HoneySpider Network – FIRE integration

As mentioned in the previous chapter, the HoneySpider Network is one of the sources of information for the FIRE system. However, the integration of these systems was not a trivial task. In this chapter we will shortly present the encountered difficulties and the ways to overcome them.

3.1 Primary mode of use – clash of philosophies

The HoneySpider Network is a high-throughput client honeypot focusing on detection of drive-by-downloads. The detection methods are twofold:

- The low interaction component (LIM) performs mostly static analysis of the content of suspicious websites. The content is downloaded using browser emulation, and then processed using several heuristic techniques – mostly static analysis, although e.g. JavaScript code is actually executed in a simulated browser environment to aid deobfuscation. Obviously this mode of operation limits the ability of this module to reach the actual malicious payload.
- The high interaction component (HIM) performs dynamic anomaly detection. The webpage is opened in an actual browser running under control of a typical operating system inside a virtual machine. Actions performed in the system are recorded and analysed to detect malicious behavior. Payload can actually be executed, unless the exploited vulnerability is not present in the monitored system.

The FIRE system is also concerned with drive-by-downloads (although not as the only focus – C&C servers and phishing activity are also monitored). The goal in this case is to verify whether detected malicious behavior is promptly dealt with, thus identifying networks which do not respond adequately to identified threats, providing safe harbour for malicious activity.

Since both systems deal with drive-by-downloads, cooperation would seem not only natural, but relatively easy. Unfortunately, the goals of both systems are very different, resulting in incompatible data collection needs.

FIRE is focused on monitoring the backend servers, the only part of the infection chain which is directly malicious. The earlier stages in the tree of requests leading to a drive-by-download can be simply infected pages redirecting to the malware. Without a working backend server they become benign, even if the injected redirections are not removed.

FIRE is capable of monitoring of the availability of the malicious files, but to do that it first needs to identify the backend servers which serve them. This is the intended goal of integration with the HoneySpider Network.

The primary mode of use of the HoneySpider Network is, however, completely different. It was built to automatically classify large amounts of URLs from different sources, singling out the malicious ones. The main function of the system is to find out whether a given URL leads to an infection and – if so – provide sufficient data for effective investigation of the threat. In other words, the focus is on the original URL submitted to HoneySpider Network (or obtained automatically from some source). The system collects all available information, but there is no need to identify precisely the actual backend server. This information is therefore not present in the system.

Both systems look at the problem of drive-by-downloads in completely different ways. To provide any useful data to FIRE the HoneySpider Network would either have to be redesigned, or some way must be found to extract the necessary information from the available data.

3.2 Problem of false positives

Visiting many of the URLs classified by the HoneySpider Network as malicious or suspicious does not in fact result in infection of the user's machine. Such cases may or may not be false positives from the HoneySpider's point of view. The classification may be a result of successful detection of an injected script or other redirection mechanism (the main function of LIM). The backend server may already be down, but the result is still important, as it allows us to identify a compromised web page, which may easily start infecting again – the administrators of that system should be notified. Still, from the point of view of FIRE, such URLs are definitely false positives. In the best case they increase the amount of processing needed to identify the backend servers. In the worst case, one of the requests may be selected, even though it is not really malicious.

This type of URLs can easily be eliminated using results from HIM. If the drive-by-download does not occur, HIM classification should be benign – redirections do not directly cause any suspicious activity inside the virtual machine. Unfortunately, HIM also creates many false positives.

The above conclusion made it clear, that the rate of false positives from HIM must be reduced. This resulted in the Capture False Positives Reduction project, described in detail in Deliverable D21.

Even after this modification, false positives do occur quite frequently. The generation of data for FIRE must therefore be performed in a rather conservative way – unless there are convincing reasons to believe that the backend server was identified correctly, the result should not be passed to FIRE.

3.3 Identification of backend servers

The identification of backend servers must rely only on the information available in the HoneySpider Network. The most useful data are:

- The final classification provided by LIM and HIM.
- List of requests generated by the LIM while scanning the URL and the individual classification of each request.
- List of requests generated by the HIM while scanning the URL.
- List of suspicious actions recorded by HIM.

Manual analysis of multiple logs provided some rules which may identify a request to a backend server.

- The URL must be classified as malicious by HIM and at least as suspicious by LIM. Among the suspicious actions recorded by HIM there must be at least one case of writing to an executable file. These two conditions must be met before any of the later rules can be applied. Creation of a process from the written executable file is a good confirmation, but it is not strictly required (Capture may miss some of the actions).
- The lists of requests generated by LIM and HIM are compared. The requests appearing only in HIM are especially interesting.
- All requests which only appear in HIM and which ask for an executable file should be reported – they almost surely refer to a backend server. This is especially true if there is only one such request and/or there is only one non-benign request from LIM and it is to the same host.

- If there is only one non-benign request from LIM, then it can be a candidate, especially if it also appears as a request from HIM and if it is for and executable file.

As can be seen, the weight of the above rules may differ - a single request for an executable appearing only among requests from HIM is much more likely to really reference a backend server than a request appearing both in HIM and LIM requests. Finding the proper threshold is an experimental task.

3.4 Summary

Since the HoneySpider Network is nowadays complete, modifying it is not a very good idea. Instead, the entire backend server identification logic was implemented in HSN WAPI, as an additional method of the object Dataset. The implementation correlates data from two or more (depending on Honeyspider configuration) databases and provides a cache avoiding renewed processing of the same candidate requests. This proves the versatility of the WAPI – the offered services may be quite complex, not just simple database access methods.

The data used to identify backend servers was already available in the WAPI, so the selection could alternatively be implemented on the FIRE side, using only existing WAPI services. Implementation of a new service was chosen for performance reasons. There are many requests in each scanned URL and each one is presented in WAPI as a separate object. This is very useful for manual use and for automated investigation of a suspicious URL, but not for automatic processing of many URLs, where all requests must be analysed, resulting in hundreds of long distance SOAP calls for each URL. Replacing them with one call to a complex method, which however can work directly on the database not only reduces the delays by orders of magnitude, but also in fact reduces the load on the server.

Using the HoneySpider Network for backend server identification proved difficult but entirely possible. Depending on further observations we plan to add more rules, identifying the servers correctly in more cases.

4 EXPOSURE (Eurecom)

In this section, we will present EXPOSURE and give some results obtained from its real-time deployment. Exposure is a system that employs large-scale, passive DNS analysis techniques to detect domains that are involved in malicious activity. The malicious domains identified by Exposure are reported on a daily basis in a public web site[1]. Exposure was developed as part of WOMBAT and employs a number of systems that are parts of WOMBAT such as Anubis malware analysis sandbox [6] and Wepawet [8]. It was presented in a research paper [7]. In this deliverable, we will provide technical details about Exposure and results of its real-time deployment.

4.1 Finding Malicious Domains Using Passive DNS Analysis

The Domain Name System (DNS) is a hierarchical naming system for computers, services, or any resource connected to the Internet. Clearly, as it helps Internet users locate resources such as web servers, mailing hosts, and other online services, DNS is one of the core and most important components of the Internet. Unfortunately, besides being used for obvious benign purposes, domain names are also popular for malicious use. For example, domain names are increasingly playing a role for the management of botnet command and control servers, download sites where malicious code is hosted, and phishing pages that aim to steal sensitive information from unsuspecting victims.

In a typical Internet attack scenario, whenever an attacker manages to compromise and infect the computer of an end-user, this machine is silently transformed into a bot that listens and reacts to remote commands that are issued by the so-called botmaster. Such collections of compromised, remotely-controlled hosts are common on the Internet, and are often used to launch DoS attacks, steal sensitive user information, and send large numbers of spam messages with the aim of making a financial profit.

In another typical Internet attack scenario, attackers set up a phishing website and lure unsuspecting users into entering sensitive information such as online banking credentials and credit card numbers. The phishing website often has the look and feel of the targeted legitimate website (e.g., an online banking service) and a domain name that sounds similar.

[1]http://exposure.iseclab.org

One of the technical problems that attackers face when designing their malicious infrastructures is the question of how to implement a reliable and flexible server infrastructure, and command and control mechanism. Ironically, the attackers are faced with the same engineering challenges that global enterprises face that need to maintain a large, distributed and reliable service infrastructure for their customers. For example, in the case of botnets, that are arguably one of the most serious threats on the Internet today, the attackers need to efficiently manage remote hosts that may easily consists of thousands of compromised end-user machines. Obviously, if the IP address of the command and control server is hard-coded into the bot binary, there exists a single point of failure for the botnet. That is, from the point of view of the attacker, whenever this address is identified and is taken down, the botnet would be lost.

Analogously, in other common Internet attacks that target a large number of users, sophisticated hosting infrastructures are typically required that allow the attackers to conduct activities such as collecting the stolen information, distributing their malware, launching social engineering attempts, and hosting other malicious services such as phishing pages.

In order to better deal with the complexity of a large, distributed infrastructure, attackers have been increasingly making use of domain names. By using DNS, they acquire the flexibility to change the IP address of the malicious servers that they manage. Furthermore, they can hide their critical servers behind proxy services (e.g., using Fast-Flux [14]) so that their malicious server is more difficult to identify and take down.

Using domain names gives attackers the flexibility of migrating their malicious servers with ease. That is, the malicious "services" that the attackers offer become more "fault-tolerant" with respect to the IP addresses where they are hosted.

The key insight of Exposure is that as malicious services are often as dependent on DNS services as benign services, being able to identify malicious domains as soon as they appear would significantly help mitigate many Internet threats that stem from botnets, phishing sites, malware hosting services, and the like. Also, the premise is that when looking at large volumes of data, DNS requests for benign and malicious domains should exhibit enough differences in behavior that they can automatically be distinguished.

Exposure introduces a passive DNS analysis approach and a detection system to effectively and efficiently detect domain names that are involved in malicious activity. It uses 15 features (9 of which are novel and have not been proposed before) that make it possible to characterize different properties of DNS names and the ways that they are used (i.e., queried). The techniques applied to accomplish this do not rely on prior knowledge about the kind of service the malicious domain provides (e.g., phishing, Fast-Flux services, spamming, botnets that use a domain generation algorithm, etc.). This is significantly different from existing techniques that only target Fast-Flux domains used in botnet operations. Furthermore, the approach requires less training time, and less training data than previous approaches [5], and does not have some of their limitations.

4.2 The Approach of Exposure

The goal of EXPOSURE is to detect malicious domains that are used as part of malicious operations on the Internet. To this end, a passive analysis of the DNS traffic is performed. Since the traffic monitored is generated by real users, the assumption is that some of these users are infected with malicious content, and that some malware components will be running on their systems. These components are likely to contact the domains that are found to be malicious by various sources such as public malware domain lists and spam blacklists. Hence, by studying the DNS behavior of known malicious and benign domains, the goal was to identify distinguishable generic features that are able to define the maliciousness of a given domain.

4.2.1 Extracting DNS Features for Detection

Clearly, to be able to identify DNS features that allow to distinguish between benign and malicious domains, and that allow a classifier to work well in practice, large amounts of training data are required. As the offline dataset, the recursive DNS (i.e., RDNS) traffic from Security Information Exchange (SIE) [4]) is recorded. An offline analysis on this data is performed to determine DNS features that can be used to distinguish malicious DNS features from benign ones. The part of the RDNS traffic we used as initial input to our system consisted of the DNS answers returned from the authoritative DNS servers to the RDNS servers. An RDNS answer consists of the name of the domain queried, the time the query is issued, the duration the answer is required to be cached (i.e., TTL) and the list of IP addresses that are associated with the queried domain. Note that the RDNS servers do not share the information of the DNS query source (i.e. the IP address of the user that issues the query) due to privacy concerns.

By studying large amounts of DNS data, 15 different features that are used in the detection of malicious domains are defined. In the following sections, we describe these features and explain why they might be indicative of malicious behavior.

Time-Based Features

The first component of a DNS record that is analyzed is the time at which the request is made. Clearly, the time of an individual request is not very useful by itself. However, when many requests to a particular domain over time are analyzed, patterns indicative of malicious behavior may emerge. In particular, the changes of the volume (i.e., number) of requests for a domain are examined.

The malicious domains often show a sudden increase followed by a sudden decrease in the number of requests. This is because malicious services often use a technique called *domain flux* [12] to make their infrastructures more robust and flexible against take downs. Each bot may use a domain generation algorithm (DGA) to compute a list of domains to be used as the command and control server or the dropzone. Obviously, all domains that are generated by a DGA have a short life span since they are used only for a limited duration. Examples of malware that make use of such DGAs are Kraken/Bobax, the Srizbi bots and the Conficker worm. Similarly, malicious domains that have recently been registered and are involved in scam campaigns show an abrupt increase in the number of requests as more and more victims access the site in a short period of time.

To analyze the changes in the number of requests for a domain during a given period of time, the period is divided into fixed length intervals. Then, for each interval, the number of DNS queries that are issued for the domain are counted. In other words, the collection of DNS queries that target the domain under analysis can be converted into time series (i.e., chronologically ordered sequences of data values).

The time series analysis is performed on two different scopes: First, the time series is analyzed globally. That is, the start and end times of the time series are chosen to be the same as the start and the end times of the entire monitoring period. Second, local scope time series analysis where the start times and end times are the first and last time the domain is queried during the analysis interval is applied. While the global scope analysis is used for detecting domains that either have a short life or have changed their behavior for a short duration, the local scope analysis focuses on how domains behave during their life time.

A domain is defined to be a *short-lived domain* (i.e., Feature 1) if it is queried only between time t_0 and t_1, and if this duration is comparably short (e.g., less than several days). A domain that suddenly appears in the global scope time series and disappears after a short period of activity has a fairly abnormal behavior for being classified as a benign domain. Normally, if a domain is benign, even if it is not very popular, the number of queries it receives should exceed the threshold at least several times during the monitoring period. Therefore, its time series analysis will not result in an abrupt increase followed by a decrease as the time series produced by a short-lived domain does.

The main idea behind performing local scope analysis is to zoom into the life time of a domain and study its behavioral characteristics. From the local scope analysis three features are extracted. These features may distinguish malicious and benign behavior either by themselves or when used in conjunction with other features. All the features involve finding similar patterns in the time series of a domain. First feature checks if there are domains that show daily similarities in their request count change over time (i.e., an increase or decrease of the request count at the same intervals everyday). The second feature aims to detect regularly repeating patterns. Finally, last one checks whether the domain is generally in an "idle" state (i.e., the domain is not queried) or is accessed continuously (i.e., a popular domain).

The problem of detecting both short-lived domains and domains that have regularly repeating patterns can be treated as a change point detection (CPD) problem. CPD algorithms operate on time series and their goal is to find those points in time at which the data values change abruptly. The CPD algorithm that is implemented outputs the points in time the change is detected and the average behavior for each duration.

DNS Answer-Based Features

The DNS answer that is returned by the server for a domain generally consists of several DNS A records (i.e., mappings from the host to IP addresses). Of course, a domain name can map to multiple IP addresses for load balancing.

Malicious domains typically resolve to compromised computers that reside in different Autonomous Systems (ASNs), countries, and regions. The attackers are opportunistic, and do not usually target specific countries or IP ranges. Whenever a computer is compromised, it is added as an asset to the collection. Also, attackers typically use domains that map to multiple IP addresses, and IPs might be shared across different domains.

With this insight, four features are extracted from the DNS answer. The first feature is the number of different IP addresses that are resolved for a given domain during the experiment window. The second feature is the number of different countries that these IP addresses are located in. The third feature is the reverse DNS query results of the returned IP addresses. The fourth feature is the number of distinct domains that share the IP addresses that resolve to the given domain.

TTL Value-Based Features

Every DNS record has a *Time To Live* (TTL) that specifies how long the corresponding response for a domain should be cached. Systems that aim for high availability often set the TTL values of host names to lower values and use Round-Robin DNS. That is, even if one of the IP addresses is not reachable at a given point in time, since the TTL value expires quickly, another IP address can be provided. A representative example for such systems are Content Delivery Networks (CDNs).

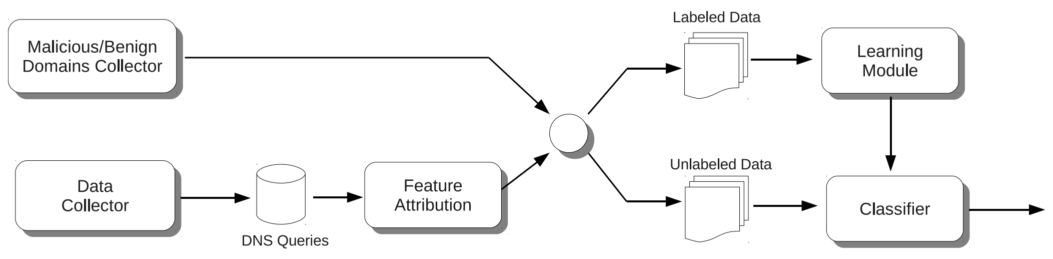


Figure 4.1: Overview of EXPOSURE

Unfortunately, setting lower TTL values and using Round-Robin DNS is useful for the attackers as well. Using this approach, malicious systems achieve higher availability and become more resistant against DNS blacklisting (DNSBL) and take downs. For example, Fast-Flux Service Networks (FFSN) [14] are malicious systems that abuse Round-Robin DNS.

From the TTL value, five features are extracted: Average value for TTL, The standard deviation of TTL, Number of distinct TTL values used for each domain, Number of TTL changes observed over time and the percentage usage of specific TTL ranges such as $[0, 1)$, $[1, 10)$, $[10, 100)$, $[100, 300)$, $[300, 900)$, $[900, inf)$.

Domain Name-Based Features

Benign services usually try to choose domain names that can be easily remembered by users. For example, a bank called "The Iceland Bank" might have a domain name such as "www.icelandbank.com". In contrast, attackers are not concerned that their domain names are easy to remember. This is particularly true for domain names that are generated by a DGA.

The main purpose of DNS is to provide human-readable names to users as they often cannot memorize IP addresses of servers. Therefore, benign Internet services tend to choose easy-to-remember domain names. In contrast, having an easy-to-remember domain name is not a concern for people who perform malicious activity. This is particularly true in cases where the domain names are generated by a DGA. To detect such domains, two features were extracted from the domain name itself: First, the ratio of the numerical characters to the length of the domain name, and second, the ratio of the length of the longest meaningful substring (i.e., a word in a dictionary) to the length of the domain name.

4.2.2 Architecture of EXPOSURE

Figure 4.1 gives an overview of the system architecture of the EXPOSURE. The system consists of five main components:

The first component, the Data Collector, records the DNS traffic produced by the network that is being monitored.

The second component is the Feature Attribution component. This component is responsible for attributing the domains that are recorded to the database with the features that are searched in the DNS traffic.

The third component, the Malicious and Benign Domains Collector, works independent of, and in parallel to the Data Collector Module. It collects domains that are known to be benign or malicious from various sources. Our benign domains sets are composed of information acquired from Alexa and a number of servers that provide detailed WHOIS data. In contrast, the malicious domain set is constructed from domains that have been reported to have been involved in malicious activities such as phishing, spamming, and botnet infections by external sources such as malwaredomains.com, Phishtank[2], and malware analyzers such as Anubis [6]) and Wepawet [8]. Note that these lists are constantly updated, and become even more comprehensive over time. The output of the Malicious and Benign Domains Collector is used to label the output of the Feature Attribution component.

Once the data is labeled, the labeled set is fed into the fourth component: The Learning Module. This module trains the labeled set to build malicious domain detection models. Consequently, these models, and the unlabeled domains, become an input to the fifth component: The Classifier.

The Classifier component takes decisions according to the detection models produced by the Learning component so that the unlabeled domains are grouped into two classes: domains that are malicious, and those that are benign.

4.2.3 Real-Time Deployment

The deployment phase of EXPOSURE consists of two steps. In the first step, the features that we are interested in are monitored and the classifier is trained based on a set of domains that are known to be benign or malicious. In a second step, after the classifier has been trained, the detection starts and domains that are determined to be suspicious are reported. Note that after an initial period of seven days of training[3], the

[2]http://www.phishtank.com
[3]We have experimentally determined the optimal training period to be seven days)

classifier is retrained every day. Hence, EXPOSURE can constantly keep up with the behavior of new malware.

4.3 Evaluation of Exposure

4.3.1 DNS Data Collection for Offline Experiments

Our sensors for the SIE DNS feeds receive a large volume of traffic (1 million queries per minute on average). Therefore, during our offline experimental period of two and a half months, we monitored approximately 100 billion DNS queries. Unfortunately, tracking, recording and post-processing this volume of traffic without applying any filtering was not feasible in practice. Hence, we reduced the volume of traffic that we wished to analyze to a more manageable size by using two filtering policies. The goal of these policies was to eliminate as many queries as possible that were not relevant for us. However, we also had to make sure that we did not miss relevant, malicious domains.

The first policy we used whitelisted popular, well-known domains that were very unlikely to be malicious. To create this whitelist, we used the Alexa Top 1000 Global Sites [4] list. By applying this first filtering policy, we were able to reduce 20% of the traffic we were observing.

The second filtering policy targeted domains that were older than one year. The reasoning behind this policy was that many malicious domains are disclosed after a short period of activity, and are blacklisted. As a result, some miscreants have resorted to using domain generation algorithms (DGA) to make it more difficult for the authorities to blacklist their domains. Typically, the domains that are generated by DGAs and registered by the attackers are new domains that are at most several months old. In our data set, we found 45.000 domains that were older than one year. These domains received 40 billion queries. Hence, the second filtering policy reduced 50% of the remaining traffic, and made it manageable in practice.

Clearly, filtering out domains that do not satisfy our age requirements could mean that we may miss malicious domains for the training that are older than one year. However, our premise is that if a domain is older than one year and has not been detected by any malware analysis tool, it is not likely that the domain serves malicious activity. To verify the correctness of our assumption, we checked if we had filtered out any domains that were suspected to be malicious by malware analysis tools such as Anubis and Wepawet. Furthermore, we also queried reports produced by Alexa, McAfee Site Advisor, Google Safe Browsing and Norton Safe Web. 40 out of the $45,000$ filtered out domains (i.e., only

[4]http://www.alexa.com

0.09%) were reported by these external sources to be "risky" or "shady". We therefore believe that our filtering policy did not miss a significant number of malicious domains because of the pre-filtering we performed during the offline experiments.

4.3.2 Experiments with the Offline Data Set

During the two and a half month offline experimental period, we recorded and then analyzed 4.8 million distinct domain names that were queried by real Internet users. Note that a domain that only receives a few requests cannot produce a time series that can then be used for the time-based features we are analyzing. This is because a time series analysis produces accurate results only when the sampling count is high enough.

In order to find the threshold for the minimum number of queries required for each domain, we trained our known malicious and benign domain list with differing threshold values. Based on these empirical results, we set the threshold to 20 queries, and excluded the 4.5 million domains from our experiments that received less than 20 requests in the two and a half months duration of our monitoring.

For further experiments, we then focused on the remaining 300,000 domains that were queried more than 20 times. EXPOSURE decided that 17,686 out of the 300,000 domains were malicious (5.9%).

4.3.3 Real-World, Real-Time Detection with Exposure

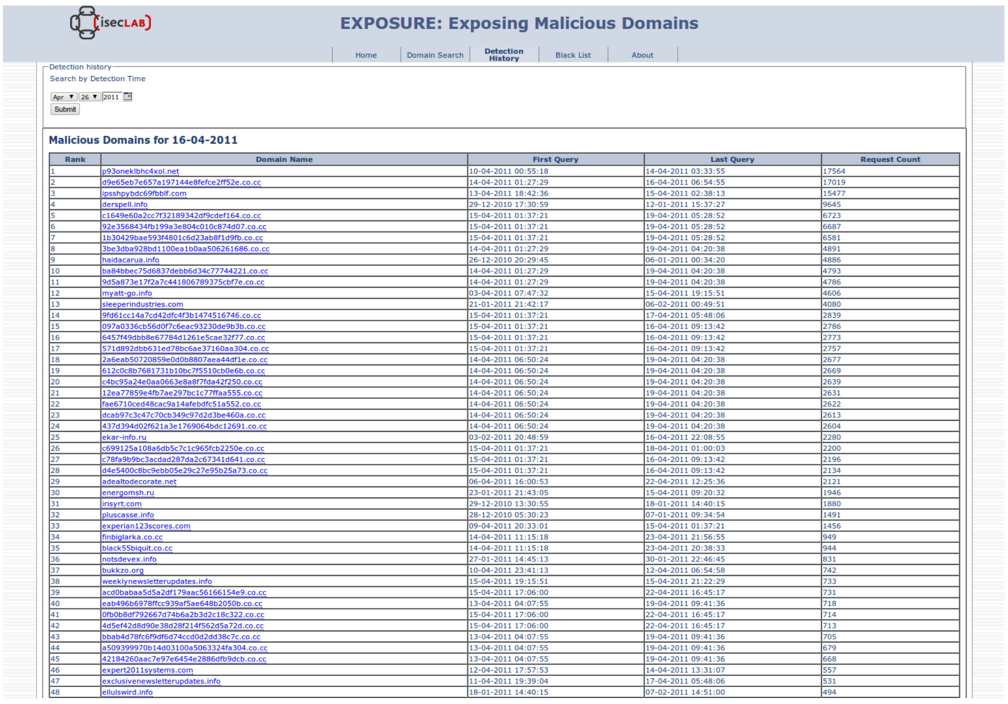
To test the feasibility and scalability of EXPOSURE as a malicious domain detector in real-life, we deployed it in the network of an ISP that provided us complete access to its DNS servers for two weeks. These servers receive DNS queries from a network that supports approximately 30,000 clients.

During the two-week experimental period, EXPOSURE analyzed and classified 100 million DNS queries. No pre-filtering was applied. At the end of two weeks, EXPOSURE detected 3117 new malicious domains that were previously not known to the system and had not been used in the training. 2821 of these domains fall into the category of domains that are generated by a DGA and all belong to the same malicious entity. 5 out of the remaining 396 domains were reported as being malicious domains by security companies such as Anvira, one month after we had detected them.

We cross-checked the rest of the remaining domains we had detected. All detected domains were classified as being risky by McAfee Site Advisor [5].

After the experiments, we provided the ISP with the list of clients that were potentially infected, or had been victims of scams.

[5]http://www.siteadvisor.com



Malicious Domains for 16-04-2011

Rank	Domain Name	First Query	Last Query	Request Count
1	p93oneklbhc4xol.net	10-04-2011 00:55:18	14-04-2011 03:33:55	17564
2	d9e65eb7e657a197144e8fefce2ff52e.co.cc	14-04-2011 01:27:29	16-04-2011 06:54:55	17019
3	ipsshpybdc69fbblf.com	13-04-2011 18:42:36	15-04-2011 02:38:13	15477
4	derspell.info	29-12-2010 17:30:59	12-01-2011 15:37:27	9645
5	c1649e60a2cc7f32189342df9cdef164.co.cc	15-04-2011 01:37:21	19-04-2011 05:28:52	6723
6	92e3568434fb199a3e804c010c874d07.co.cc	15-04-2011 01:37:21	19-04-2011 05:28:52	6687
7	1b30429bae593f4801c6d23ab8f1d9fb.co.cc	15-04-2011 01:37:21	19-04-2011 05:28:52	6581
8	3be3dba928bd1100ea1b0aa506261686.co.cc	14-04-2011 01:27:29	19-04-2011 04:20:38	4891
9	haidacarua.info	26-12-2010 20:29:45	06-01-2011 00:34:20	4886
10	ba84bbec75d6837debb6d34c77744221.co.cc	14-04-2011 01:27:29	19-04-2011 04:20:38	4793
11	9d5a873e17f2a7c441806789375cbf7e.co.cc	14-04-2011 01:27:29	19-04-2011 04:20:38	4786
12	myatt-go.info	03-04-2011 07:47:32	15-04-2011 19:15:51	4606
13	sleeperindustries.com	21-01-2011 21:42:17	06-02-2011 00:49:51	4080
14	9fd61cc14a7cd42dfc4f3b1474516746.co.cc	15-04-2011 01:37:21	17-04-2011 05:48:06	2839
15	097a0336cb56d0f7c6eac93230de9b3b.co.cc	15-04-2011 01:37:21	16-04-2011 09:13:42	2786
16	6457f49dbb8e67784d1261e5cae32f77.co.cc	15-04-2011 01:37:21	16-04-2011 09:13:42	2773
17	571d892dbb631ed78bc6ae37160aa304.co.cc	15-04-2011 01:37:21	16-04-2011 09:13:42	2757
18	2a6eab50720859e0d0b8807aea44df1e.co.cc	14-04-2011 06:50:24	19-04-2011 04:20:38	2677
19	612c0c8b7681731b10bc7f5510cb0e6b.co.cc	14-04-2011 06:50:24	19-04-2011 04:20:38	2669
20	c4bc95a24e0aa0663e8a8f7fda42f250.co.cc	14-04-2011 06:50:24	19-04-2011 04:20:38	2639
21	12ea77859e4fb7ae297bc1c77ffaa555.co.cc	14-04-2011 06:50:24	19-04-2011 04:20:38	2631
22	fae6710ced48cac9a14afebdfc51a552.co.cc	14-04-2011 06:50:24	19-04-2011 04:20:38	2622
23	dcab97c3c47c70cb349c97d2d3be460a.co.cc	14-04-2011 06:50:24	19-04-2011 04:20:38	2613
24	437d394d02f621a3e1769064bdc12691.co.cc	14-04-2011 06:50:24	19-04-2011 04:20:38	2604
25	ekar-info.ru	03-02-2011 20:48:59	16-04-2011 22:08:55	2280
26	c699125a108a6db5c7c1c965fcb2250e.co.cc	15-04-2011 01:37:21	18-04-2011 01:00:03	2200
27	c78fa9b9bc3acdad287da2c67341d641.co.cc	15-04-2011 01:37:21	16-04-2011 09:13:42	2196
28	d4e5400c8bc9ebb05e29c27e95b25a73.co.cc	15-04-2011 01:37:21	16-04-2011 09:13:42	2134
29	adealtodecorate.net	06-04-2011 16:00:53	22-04-2011 12:25:36	2121
30	energomsh.ru	23-01-2011 21:43:05	15-04-2011 09:20:32	1946
31	irisyrt.com	29-12-2010 13:30:55	18-01-2011 14:40:15	1880
32	pluscasse.info	28-12-2010 05:30:23	07-01-2011 09:34:54	1491
33	experian123scores.com	09-04-2011 20:33:01	15-04-2011 01:37:21	1456
34	finbiglarka.co.cc	14-04-2011 11:15:18	23-04-2011 21:56:55	949
35	black55biquit.co.cc	14-04-2011 11:15:18	23-04-2011 20:38:33	944
36	notsdevex.info	27-01-2011 14:45:13	30-01-2011 22:46:45	831
37	bukkzo.org	10-04-2011 23:41:13	12-04-2011 06:54:58	742
38	weeklynewsletterupdates.info	15-04-2011 19:15:51	15-04-2011 21:22:29	733
39	acd0babaa5d5a2df179aac56166154e9.co.cc	15-04-2011 17:06:00	22-04-2011 16:45:17	731
40	eab496b6978ffcc939af5ae648b2050b.co.cc	13-04-2011 04:07:55	19-04-2011 09:41:36	718
41	0fb0b8df792667d74b6a2b3d2c18c322.co.cc	15-04-2011 17:06:00	22-04-2011 16:45:17	714
42	4d5ef42d8d90e38d28f214f562d5a72d.co.cc	15-04-2011 17:06:00	22-04-2011 16:45:17	713
43	bbab4d78fc6f9df6d74ccd0d2dd38c7c.co.cc	13-04-2011 04:07:55	19-04-2011 09:41:36	705
44	a509399970b14d03100a5063324fa304.co.cc	13-04-2011 04:07:55	19-04-2011 09:41:36	679
45	42184260aac7e97e6454e2886dfb9dcb.co.cc	13-04-2011 04:07:55	19-04-2011 09:41:36	668
46	expert2011systems.com	12-04-2011 17:57:53	14-04-2011 13:31:07	557
47	exclusivenewsletterupdates.info	11-04-2011 19:39:04	17-04-2011 05:48:06	531
48	ellulswird.info	18-01-2011 14:40:15	07-02-2011 14:51:00	494

Figure 4.2: A screenshot of daily malicious domains list reported by Exposure.

4.4 Real-Time Deployment of Exposure

The real-time deployment of Exposure has been running and reporting potentially malicious domains it detects at http://exposure.iseclab.org since the 22nd of December, 2010. Exposure prepares its reports on a daily basis as it can be seen from Figure 4.2. Exposure has been detected 17924 distinct malicious domains in last four months and Figure 4.3 shows number of distinct domains detected during this period.

We believe that Exposure is a useful system that can help security experts and organizations in their fight against cyber-crime. For each malicious domain reported on the website, we sketch its time-series graphics and produce the list of IP addresses that are mapped to it (Figure 4.4. Moreover, we link the IP address list table with FIRE [6] to see whether the AS of the IP is already known for hosting/sourcing malicious activities.

[6]http://maliciousnetworks.org

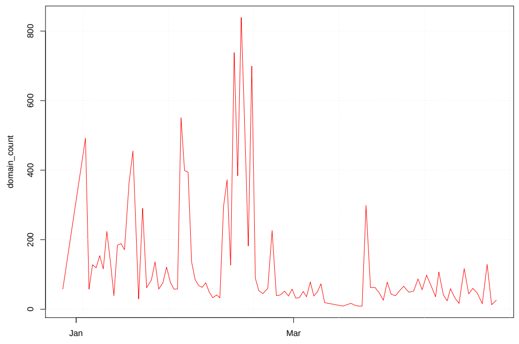
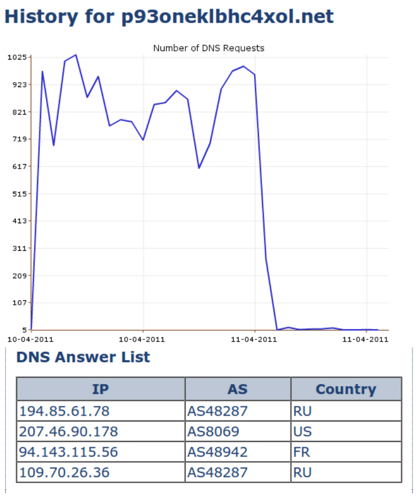


Figure 4.3: Number of malicious domains detected over time.



History for p93oneklbhc4xol.net

DNS Answer List

IP	AS	Country
194.85.61.78	AS48287	RU
207.46.90.178	AS8069	US
94.143.115.56	AS48942	FR
109.70.26.36	AS48287	RU

Figure 4.4: A screenshot of an example malicious domain's time-series table and the IP address list mapped to it.

5 BANOMAD: BANking Oriented Malware Analysis Droid

5.1 What is a banking trojan?

Over the last few years there has been a dramatic change in the goals and modes of operation of malicious hackers. As hackers realized the potential monetary gains associated with Internet fraud, there has been a shift from hacking for fun (or bragging rights and celebrity within and outside the hacker community) to hacking for profit. This shift has been leveraged and supported by more traditional crime organizations, which eventually realized the potential of the Internet for their endeavors.

The integration of sophisticated cyber attacks with well-established fraud mechanisms devised by organized crime has resulted in an underground economy that trades compromised hosts, personal information, and services in a way similar to legitimate economies. This expanding underground economy makes it possible to significantly increase the scale of the frauds carried out on the Internet and allows criminals to reach millions of potential victims. One means that attackers are exploiting in order to achieve these lucrative goals are banking trojans.

The term banking trojan (a.k.a bankers) is used to designate a piece of malware that targets funds from an online bank account. Certain other financial services such as online stock brokerage services and online payment systems (e.g. PayPal) are also considered online banks throughout this report.

Banking trojans belong to a broader category of malware called crimeware. This malware bucket comprises clickers, spam proxies, ransomware and any other malicious software that bring some sort of financial gains to their developers or distributors.

5.2 How do banking trojans work?

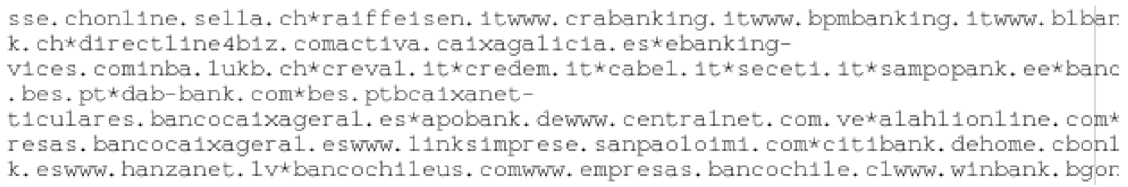
5.2.1 Data filtering

Trojans specifically crafted to intercept banking credentials were first spotted in 2004. Fraudsters had used malware before that, however it was mainly spam proxy backdoors rather than software to harvest banking passwords. In 2004 some of their malware specimens started to filter out keylogging data that was not related to the banking sessions of interest. Traditional keylogging produces a lot of data and mining it requires a lot of effort, moreover, some banks use virtual keypads that cannot be intercepted through simple keylogging. By filtering out as much as possible and implementing techniques to circumvent theft protection mechanisms the cybercrooks have a higher success rate. Typically this filtering is done based on the URLs accessed by the user, specific data capturing procedures are applied based on whether there has been a regular expression match or search string hit on the URL visited by the user.

So as to focus on specific banking sites the trojan typically contains, or downloads from a mothership server, a list of filtering strings/regular expressions that it should seek in the victim visited URLs. These filters use certain properties of a site in order to discriminate whether it is of any interest (i.e. data should be harvested for it):

- URL: regular expression matches or simple string searches (e.g. *www.banesto.com* or */cs/Satellite?appID=banesto.internet.WCBanestoes*) may be performed on the visited URLs.
- Site title: the site discrimination procedures are sometimes applied to the title of the page being visited, this title is fixed by the <title>tag in its HTML (e.g. *Banesto Particulares*).
- Site text: some trojans just read the whole HTML code of the banking site and strip its tags, looking then for certain text patterns. For example, the Spyforms trojan was seen seeking the following text stream in order to flag a visit to Rabobanks site: *Mensen in staat stellen om hun dromen en ambities waar te maken. Dat is waar Rabobank Private Banking voor staat. Daarom bieden we de beste financile diensten. En tonen we een grote betrokkenheid bij onze klanten en hun omgeving.*
- HTML tag attributes related to the bank's authentication form: In this case the trojan will traverse the DOM of the site being visited and try to find nodes related to authentication forms, once found, it will try to match certain attributes of the form fields. For example, Banesto's authentication form has a password field with the attribute *name* equal to *DatosCliente.passAux*, this is specific enough to serve as site discriminator.

We can think of other ways of deciding when to perform data filtering: check the IP of the visited web server, inspect the SSL certificate returned by the it, hash or even do some image processing on pictures found on the visited website, etc. However, none of these suggested techniques have been seen in the wild.



```
sse.chonline.sella.ch*raiffeisen.itwww.crabanking.itwww.bpmbanking.itwww.b1bar
k.ch*directline4biz.comactiva.caixagalicia.es*ebanking-
vices.cominba.lukb.ch*creval.it*credem.it*cabel.it*seceti.it*sampopank.ee*banc
.bes.pt*dab-bank.com*bes.ptbcaixanet-
ticulares.bancocaixageral.es*apobank.dewww.centralnet.com.ve*alahlionline.com*
resas.bancocaixageral.eswww.linksimprese.sanpaoloimi.com*citibank.dehome.cbonl
k.eswww.hanzanet.lv*bancochileus.comwww.empresas.bancochile.clwww.winbank.bgor
```

Figure 5.1: Portion of Sinowal/Mebroot decrypted entity monitoring configuration file

5.2.2 Entity monitoring

At this point we know that banking trojans only capture interesting data from banking activity. This means that the trojan has to know when the user is banking online. In other words, the trojan needs a means to be able to access the URL or URL properties previously described, a means to monitor what the browser is doing and where it is going.

The following techniques have been used by attackers so as to determine where the user is surfing:

- Hooking. Hooking of a function means the interception of any call to it. When a hooked function should be executed, control is delegated to a different location, where customized code resides: the hook or hook function. The hook can then perform its own operations and later transfer control back to the original function or prevent its execution completely. By hooking Windows API functions such as *InternetOpenUrl* in *WinINet.dll* a trojan can, in addition to other operations, inspect its parameters, among which we find *__in LPCTSTR lpszUrl*, which is a pointer to a null-terminated string variable that specifies the URL to begin reading. In other words, the trojan is acquainted with the URL being visited by the user.

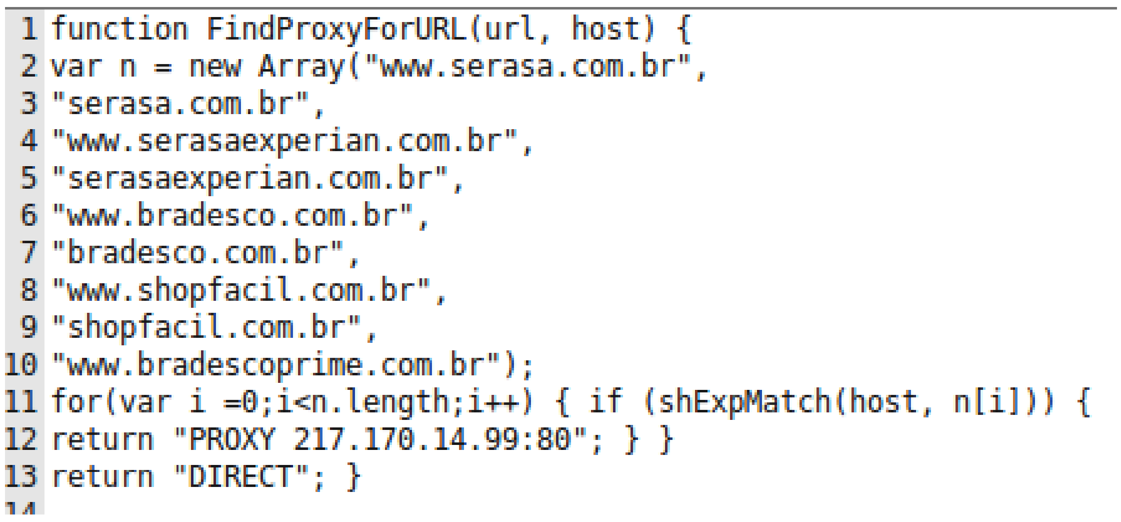
- BHO (Browser Helper Object). This is a DLL module designed as a plugin for Microsoft's Internet Explorer web browser to provide added functionality (e.g. PDF rendering within the browser). The BHO API exposes hooks that allow the plugin to access the Document Object Model (DOM) of the current page and to control navigation. Because BHOs have unrestricted access to the Internet Explorer event model, some forms of malware have also been created as BHOs. For example, by using the *DWebBrowserEvents2::BeforeNavigate2* event a BHO can learn the URL that the user is heading to.

- Window Title Enumeration. As stated in the Data Filtering subsection, some trojans apply patterns to the title of a given site rather than its URL. The title of a site can be easily accessed by BHOs or by reading the content of the server response through API hooking, however, there is yet another trivial way of doing so. Site titles are used to fix the title of the active browser window (e.g. *Banesto Particulares — Windows Internet Explorer*), and the titles of open windows can be easily searched through with the *FindWindow()* function from *user32.dll* or other similar legitimate Windows APIs. Of course, trojans rarely call Windows API functions directly, instead they make use of higher level language libraries that perform these calls.

- DDE. Dynamic Data Exchange is a technology for communication between multiple applications under Microsoft Windows or OS/2. The primary function of DDE is to allow Windows applications to share data. For example, a cell in Microsoft Excel could be linked to a value in another application and when the value changes, it would be automatically updated in the Excel spreadsheet. In the same manner a trojan can trigger certain actions based on Internet Explorer's data sharing via DDE . For example, through the *WWW_RegisterURLecho* interface an application can be informed about the URL visited by the user.

- COM (Component Object Model)/OLE (Obect Linking and Embedding) interfaces. This is a Microsoft centric interface standard for software componentry. It is used to enable interprocess communication and dynamic object creation in any programming language that supports the technology. The term COM is often used in the software development industry as an umbrella term that encompasses the OLE, OLE Automation, ActiveX, COM+ and DCOM technologies. Using these interfaces a third party process can, for example, query Internet Explorer and ask it what page it is displaying. As an example, the *IWebBrowser2* interface exposes methods that are implemented by the *WebBrowser* control (Microsoft ActiveX control) or implemented by an instance of the Internet Explorer application (OLE Automation). Among its properties we find the *IwebBrowser2::LocationURL*, that allows the querying entity to know the URL being visited.

- Firefox/Chrome/Opera/etc. extensions. This would be the equivalent of a Browser Helper Object for the rest of web browsers. The difference lies in that other web browsers base their extension model in traditional web technologies: HTML, CSS, JavaScript, XML, etc. As with BHOs, these extensions have unrestricted access to the DOM of the visited sites and may register JavaScript routines to attend browser events. Hence, it is trivial to build an *onPageLoad* callback that queries the *docu-*



```
function FindProxyForURL(url, host) {
var n = new Array("www.serasa.com.br",
"serasa.com.br",
"www.serasaexperian.com.br",
"serasaexperian.com.br",
"www.bradesco.com.br",
"bradesco.com.br",
"www.shopfacil.com.br",
"shopfacil.com.br",
"www.bradescoprime.com.br");
for(var i =0;i<n.length;i++) { if (shExpMatch(host, n[i])) {
return "PROXY 217.170.14.99:80"; } }
return "DIRECT"; }
```

Figure 5.2: Malicious proxy installer trojan

ment.location.href property (or more specifically *event.originalTarget.location.href*) in order to discover the URL being visited by the victim.

- LSP (Layered Service Provider). Layered Service Provider (LSP) is a feature of the Microsoft Windows Winsock 2 Service Provider Interface (SPI). A Layered Service Provider is a DLL that uses Winsock APIs to insert itself into the TCP/IP stack. Once in the stack, a Layered Service Provider can intercept and modify inbound and outbound Internet traffic. It allows processing all the TCP/IP traffic taking place between the Internet and the applications that are accessing the Internet (such as a web browser, the email client, etc).

- Inspection at intermediate network node. Some trojans simply configure an HTTP(S) proxy in the victim's machine so as to redirect their web activity to a malicious node. This proxy can easily access the host HTTP header field in order to decide whether the visited site is of any interest.

Once again, other techniques can be thought off, for example, network traffic inspection through libpcap (or any similar library) in order to access the visited URL by looking at the HTTP requests (of course, this requires sites not using SSL or some other means to forcing this to happen).

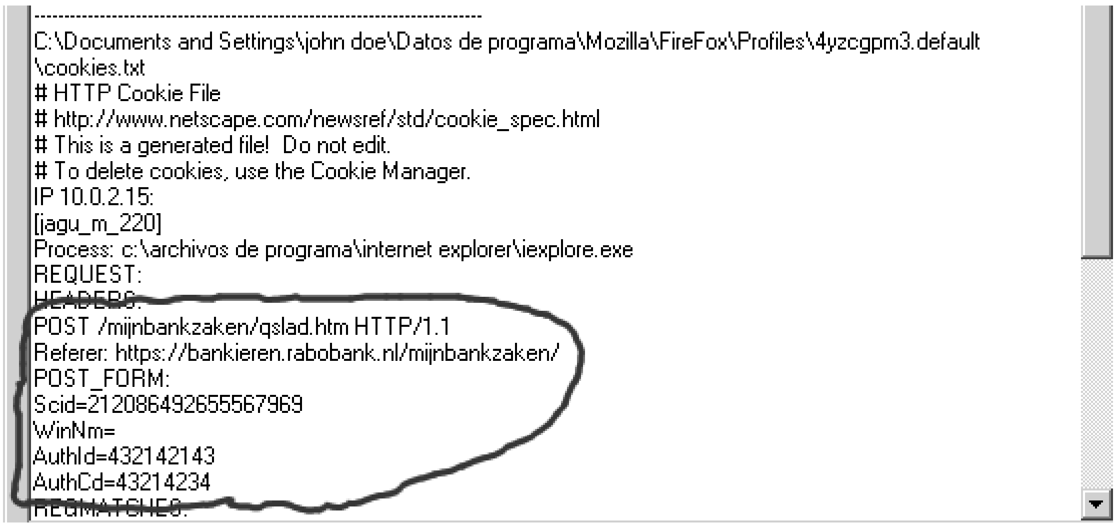
Note that all of the previous techniques not only allow the trojan to be acquainted with browsing activity of the user, but they also allow it to read and modify the HTML being rendered by the browser. This feature proofs itself very useful for some methods described in the following subsection.

5.2.3 Data Harvesting

Once the trojan knows that the user is accessing a banking site, it tries to capture the user's credentials or his authenticated banking session. Manual reverse engineering of a large amount of banking trojans received at VirusTotal has revealed the use of the following techniques to achieve such purpose:

- Keylogging. This is a functionality shown by the well known traditional keyloggers, the goal of these specimens is stealing passwords, instant messaging conversations, electronic mails, etc. Banking trojans have combined this technique with methods for entity monitoring in order to filter out non-interesting information and capture all keys pressed while visiting certain financial organizations. It is not a very efficient method since most banks have certain credentials that must be introduced via alternative methods to the keyboard (e.g. virtual keypad). Additionally, if a user makes a mistake when inserting his data and rewrites the password, the trojan will capture the whole brute data typed and credential identification will be difficult. Having said this, it is normally used in combination with other stealing procedures, hence, the overall outcome may be considered a high risk. Traditional keylogging can be implemented using something as simple as the Windows *GetAsyncState* API function.

- Form grabbing. Generic keylogging is not a good idea when capturing banking credentials. If a specimen logs every single key pressed without applying any filtering, the attacker will find himself with a set of senseless data. This is why banking malware mostly use form grabbing. After all, data of sensitive nature is eventually introduced into a form. The advantage of this method is that it filters and structures the data captured while the credential theft is still prior to the encryption that is usually performed when sending the login information to the legitimate server, hence, passwords can be captured as plain text. Form Grabbing is trivial using browser extensions for example, unrestricted access to the DOM of a given site allows the attacker to query the *value* attribute of form *input* fields, this is where credentials are typed in. Figure 5.3 shows an example of a credential theft by a specimen making use of this technique.



```
--------------------------------------------------------------------------
C:\Documents and Settings\john doe\Datos de programa\Mozilla\FireFox\Profiles\4yzcgpm3.default
\cookies.txt
# HTTP Cookie File
# http://www.netscape.com/newsref/std/cookie_spec.html
# This is a generated file!  Do not edit.
# To delete cookies, use the Cookie Manager.
IP 10.0.2.15:
[jagu_m_220]
Process: c:\archivos de programa\internet explorer\iexplore.exe
REQUEST:
HEADERS:
POST /mijnbankzaken/qslad.htm HTTP/1.1
Referer: https://bankieren.rabobank.nl/mijnbankzaken/
POST_FORM:
Scid=212086492655567969
WinNm=
AuthId=432142143
AuthCd=43214234
REGMATCHES:
```

Figure 5.3: Credential theft by form grabber (Mebroot/Sinowal)

- Screenshots and video capture. As already mentioned, many electronic banking login pages include virtual keyboards with the aim of avoiding credential keylogging and form grabbing (if virtual keyboard data is pertinently obfuscated before fixing a given form input field). In order to circumvent this security layer malware authors have adopted techniques related to screen capturing. What these samples normally do is produce screenshots each time a mouse click is detected when visiting a specific page. Since sending full screenshots to the attacker could generate an important network load, usually these specimens only capture a limited size area around the click position. Some banks are aware of this menace and have decided to implement countermeasures such as swapping digits for asterisks whenever a mouse click is performed. In order to combat this, malware developers have adopted other methods of credential theft such as recording a video of the authentication process. Figure 5.4 shows an example of a credential theft by a specimen making use of delimited area screenshots combined with keylogging.

- Pharming. This technique consists in faking the DNS resolution of certain domains in order to redirect victims to fraudulent pages which are identical to those of their bank. In these cases the SSL certificate (if HTTPS is indeed used) of the page is

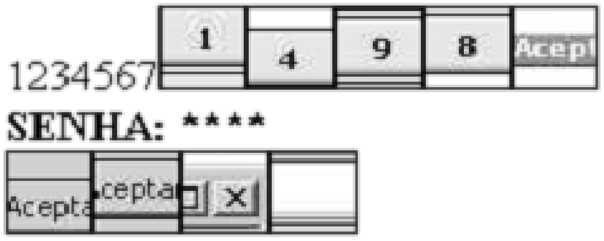


Figure 5.4: Combination of keylogging and delimeted area screenshots (MDCoco)

not the one of the legitimate entity. This can be summarized as follows. Whenever a user writes an address in his web browser it must be converted to an IP address. This process is called name resolution and DNS servers are responsible for it. These servers store tables linking domain names to IP addresses. The idea behind pharming is forcing the resolution of the domain name of a banking site to a non-legitimate IP address. The server responding at the non-legitimate IP will display a fake page identical to that of the entity and any data inserted into it will be captured. Every computer has a file that stores a small table of domain names and IP addresses, this is done to avoid DNS queries for particular domain names. The least sophisticated Trojans simply change this file so as to perform pharming, nonetheless, there are other more advanced methods to do so such as exploiting the victim's router or DNS cache poisoning.

- Man-in-the-middle (MITM) attacks. The 17th August 2007 Trend Micro spotted a trojan making use of ARP poisoning in order to launch MITM attacks in a local network[1] so as to target non-compromised computers in such network. This technique is very powerful since compromising one unique host inside a corporate network will mean compromising the electronic transactions of all hosts inside the organization. The attack scenario requires a host compromised by a trojan making use of some network packet manipulation library such as WinPcap in order to craft the pertinent poisoning packets.

- Session cookie theft. A cookie, also known as a web cookie, browser cookie, and HTTP cookie, is a piece of text stored on a user's computer by their web browser. A cookie can be used for authentication, storing site preferences, shopping cart

[1]Man-In-The-Middle Attack Used by Trojan. http://blog.trendmicro.com/man-in-the-middle-attack-used-by-trojan/

contents, the identifier for a server-based session, or anything else that can be accomplished through storing text data. Most web sites use cookies as the only identifiers for user sessions, because other methods of identifying web users have limitations and vulnerabilities. If a web site uses cookies as session identifiers, attackers can impersonate users' requests by stealing a full set of victims' cookies. From the web server's point of view, a request from an attacker has the same authentication as the victim's requests; thus the request is performed on behalf of the victim's session. Cookie theft by trojans can be as trivial as reading a given file from disk. However, even though this task can be accomplished very easily, it is of little use since it does not allow interception of advanced banking credentials required to authorize money transfers.

- Social engineering to infect secondary factor authentication device. Due to all of the previous attacks, many banks have moved towards a two factor authentication strategy where a transaction dependent SMS is sent to the user's mobile phone whenever he is going to perform a transfer. This approach drastically undermines the effectiveness of the previous techniques since the SMS contains not only a transaction dependent code, but also the target account, hence, the victim must deliberately ignore this SMS field so as to be deceived by the fraudster. With the rise of smartphones, in order to overcome this security mechanism attackers have made use of social engineering combined with HTML injections. Whenever a user logs into his account he is informed that there is a new protection measure that requires him to install a security certificate (or any other similar excuse) in his phone. The installed component is simply a phone trojan that will automatically forward banking SMSs to the attacker, this allows him to order transactions on behalf of the victim and then retrieve the transaction code required to validate the operation.

5.2.4 Data forwarding

Sooner or later the stolen data must be sent to the attacker so that he can perform the robbery (if this is not directly managed by the trojan itself). Analysis of samples received at VirusTotal has revealed the following data sending mechanisms:

- HTTP(S) POST/GET requests to a mothership server.
- FTP upload to a credential storage.
- Use of SMTP to send emails with the stolen credentials.

- Direct database (MySQL) connections executing data insertion statements (*INSERT/UPDATE*).
- IRC connections depositing thefts as conversations in particular channels whose activity is being logged by the attacker.

In making use of these techniques the stolen data very often travels in a cyphered/obfuscated fashion, probably to make dynamic analysis of the samples more difficult and to prevent system administrators from easily noticing anomalous behaviour.

The vast majority of the studied specimens use HTTP(S) requests, the reason being that there are far less chances of this type of communication being blocked by firewalls.

Many other techniques are available to attackers, for example, the attacker could use simple DNS resolutions where the theft travels encoded as a base64 string built as a subdomain of a malicious domain. The authoritative name server for this domain would then log those requests and in doing so it would be receiving the stolen credentials.

Stealthier (from the takedown point of view) and more resilient techniques that do not make use of a centralized data gathering point have been much discussed. One such example are P2P botnets. While P2P has indeed been used by malware in the past, the most famous case being the Waledac botnet, it has not gained momentum among banking trojans, probably because of the fear of firewalls hindering the thefts.

Having said this, very often the trivial solution is not sending the stolen credentials at all, simply storing them locally and using them to perform transaction orders using the victim's machine. This approach has several advantages for the attacker, the main ones being that it is more difficult to track him down and that the takedown of the C&C server may not end the unauthorized transfers (provided the trojan already has a list of target accounts to which the transfers should be made).

Additionally, many banks perform server-side data and metadata correlation in order to try to determine whether a given transfer is potentially fraudulent (by looking at the IP requesting the transfer for example). Trojans that use the victim machine to make the orders may easily circumvent some of these correlation mechanisms.

5.2.5 Money theft

Many of the credential interception mechanisms discussed above send the stolen data to a server managed by the attacker. The attacker can then log into the online bank and place a transaction to send money to an account belonging to himself or more likely to a hired money mule.

Other stealthier and less easily tracked means of performing robberies have also been seen in the wild. Certain e-crime groups using trojans that perform fraudulent form field injections will also ask for the credit card and CVV of the victim. After the data robbery, rather than logging into the victim's account and ordering a transaction to a money mule account, the gangs will log into an online casino of their property and bet money using the victims credit card. The gangs will bet with the aim of losing always, this allows the money to be directly retrieved from the online casinos' profit without any need of money mules.

Hal Cash[2] , Western Union or other similar services further improve the ease of thefts. Hal Cash is a service that allows one to send money to be withdrawn from an ATM without the need of a credit or debit card, just with a transaction code that can be sent via SMS. Many banks have started to introduce Hal Cash or similar services and ecrime gangs have found out that it is an ideal means for performing fraud since the money mule is once again not needed. The unique disadvantage lies in the fact that the amount of money that can be transferred via this method is limited.

What are the consequences for the customer of an online bank if money is stolen from his account? He might get his lost money back from the bank, this is the unspoken convention in Spain for example. However, even in this best case scenario for the user, he can lose a lot of personal data as well and that loss cannot be undone. Therefore, this kind of online crime is a very concrete threat to all computer users.

Nowadays online banking is the norm. This is good for the banks because it reduces costs. However, this also means that not all online banking users are early adopters any more, in other words, many online banking users are not very computer savvy. The people who are at the greatest risk of getting infected by a banking trojan are also the people who will have the biggest problems learning how to use multi-factor authentication and will not be suspicious of many of the discussed fraud schemes.

Banks are aware of this panorama and have been demanding antimalware and security companies new services to track these banking trojans, take down their infrastructure and provide detailed reports on their inner workings so as to gather intelligence on the enemy.

5.3 BANOMAD: VirusTotal based early warning system for banking trojan and targeted attacks

Years of manual reverse engineering of banking trojans by Hispasec's antifraud team has allowed us to learn and describe how these specimens get to fulfill their purpose. Manual analysis of malware is rather difficult, not only because it requires advanced

[2]Hal-Cash. http://www.halcash.us/

reverse engineering notions and very often malware uses code-obfuscation techniques like compression, encryption of self-modification but also because it does not scale and it is slow.

How can we fight this threat then? Fraud prevention requires automation, BANOMAD (BANking Oriented Malware Analysis Droid) is multi-modular setup that acts as an early warning system by automating the analysis process of known (previously studied) banker families received at VirusTotal. The idea is that we can run collections of malware in a test system that will discriminate banking trojans and automatically produce reports with the information of interest demanded by banks in order to combat this threat.

5.3.1 Keystone hypothesis

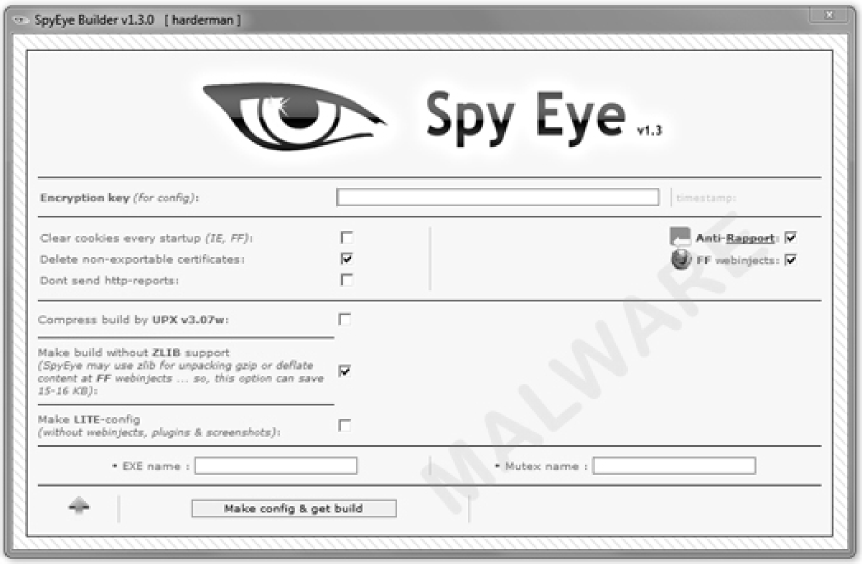
Banking trojans are generated with creation kits for dummies (builders) or as recompilations of some original source code changing exclusively monitoring strings and dropzone information, this is the keystone around which BANOMAD has been built. Figure 5.5 shows the builder for the SpyEye banker family.

Just as in the field of economy, return on investment (ROI) governs many of the decisions and approaches taken by attackers. When a malware coder develops a banking trojan he is investing time and knowledge into a project and he is interested in making the most out of it. This behaviour has given rise to the concept of banker families, hundreds/thousands of variants of a same source code where only certain parameters like the attacked banks or the data gathering infrastructure changes.

This working hypothesis - that attackers will reuse their code in many different attacks - allows us to adopt a template based approach when studying bankers. The hypothesis suggests that rather than having thousands of completely different specimens perpetrating the attacks, there is a limited amount of behavioural patterns giving rise to thousands of unique binaries. The idea is that these differentiated behavioural patterns can be easily spotted and clustered so as to perform in depth automated study of binaries with family analysis templates.

5.3.2 Functional blocks

BANOMAD processes all files received at *VirusTotal*. Let us remind the reader that VirusTotal submissions are performed by the average PC end user as well as academic institutions, CERTs, security companies, honeypot setups, antivirus companies, etc. Hence, VirusTotal itself is the engine driving the early warning system – since relatively fresh malware is sent to the online scanner, the quick response requirement can be accomplished.



SpyEye Builder v1.3.0 [harderman]

Spy Eye v1.3

Encryption key *(for config)*: timestamp:

Clear cookies every startup *(IE, FF)*:	☐
Delete non-exportable certificates:	✓
Dont send http-reports:	☐
Anti-Rapport:	✓
FF webinjects:	✓
Compress build by **UPX v3.07w**:	☐
Make build without **ZLIB** support *(SpyEye may use zlib for unpacking gzip or deflate content at* ***FF*** *webinjects ... so, this option can save 15-16 KB)*:	✓
Make **LITE**-config *(without webinjects, plugins & screenshots)*:	☐

• EXE name : • Mutex name :

Figure 5.5: SpyEye banking trojan builder

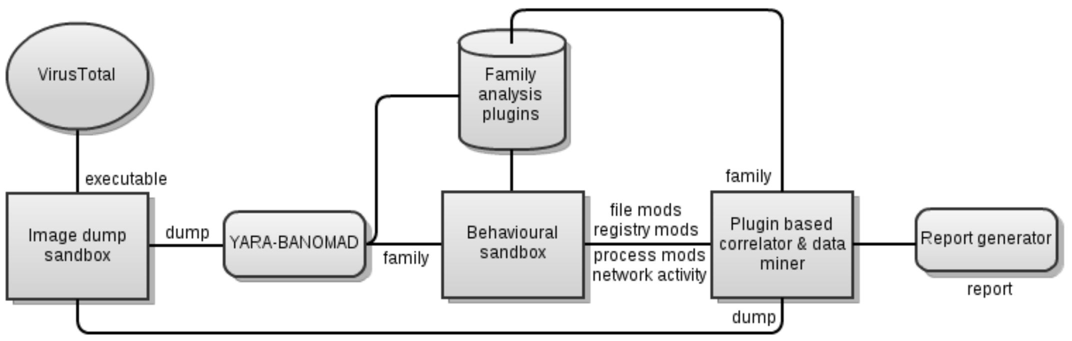


Figure 5.6: BANOMAD functional block diagram

VirusTotal's flux is then filtered extracting exclusively executable samples. These executable samples are fed into what we call the *image dump sandbox*, a setup that will try to obtain an unpacked version of the executable.

The generated dump is then fed into a malware family classifier (*YARA-BANOMAD*) based on binary signatures that spot certain bankers of interest. The bankers of interest must have been previously identified through other means (which will be briefly discussed later). This stage produces a feed of interesting banking trojans that should be further analysed.

The bankers are then executed in a *behavioural analysis sandbox*, since the family to which they belong has been previously identified by YARA-BANOMAD, *family specific analysis plugins* can be automatically loaded by the sandbox to extract data that may then help in deciphering configuration files, identifying malicious network communication points, etc. Note that the data produced by the behavioural analysis sandbox allows us to reject most of the false positives that may have been generated by the YARA-BANOMAD family labeling.

At this point we already have quite a lot of information on the sample: its family, an executable image dump, the file system modifications it produces, registry modifications, process modifications (processes launched, killed, remote thread injections, etc.) and the network activity it generates.

All this information is now fed into a *plugin based correlator and dataminer* that can also have family specific plugins. This new block extracts the banks targeted by the malware, their dropzones, configuration download points and other infrastructure that should be taken down in order to neutralize the data theft performed by the trojan.

Now that all pieces of the puzzle are available, the information is inputted into a *report generator* that produces a normalized (XML) writeup that can be easily parsed and styled by the entities receiving the information in order to take any actions they consider necessary. Note that the template based approach allows this report to be far more verbose than other similar setups since the generator can load in depth standard descriptions produced through prior manual reverse engineering of one family variant.

5.3.3 Image dump sandbox

Overview. This sandbox consists of several VirtualBox instances running an English Windows XP SP2 installation restricted through the use of a driver. The main purpose of this driver is to allow a malware process to be executed in a native environment for unpacking/image dump purposes without the risk of the malware process doing anything malicious, i.e. interacting with files, registry, processes (including spawning new processes), the system (like rebooting or logging of a user), security settings, driver loading, and similar. The driver creates an image dump from kernel mode at process exit.

The main characteristic of this BANOMAD block is speed, samples are just executed during 3 seconds, if after this time the sample did not unpack itself into memory, the image dump will be of little use provided the executable was indeed packed. Additionally, the sandboxing restrictions mean that no virtual machine reverts are necessary, further increasing execution throughput. Both of these features allow the whole flux of PE samples received at VirusTotal to be executed regardless of whether they were already seen in the past.

System description. The image dump sandbox driver works by registering a *CreateProcess* notify routine for process spawn monitoring purposes, replacing both the Kernel and GUI syscall tables, and creating wrapper hooks around certain syscall functions.

The replacement of the Kernel and GUI syscall tables is done by duplicating the original tables and switching the table pointers both in the exported *KeServiceDescriptorTable*, as well as in the not exported *KeServiceDescriptorTableShadow* (we will refer to it as the shadow table). Thanks to this, enabling/disabling hooks requires only switching the pointers of the tables (3 pointers) instead of switching many entries in the syscall tables themselves.

Since the shadow table is not exported, it has to be found using other methods. The sandbox driver uses a method proposed by Alexander Volynkin[3] , which works by seeking a pointer to the shadow table in the *KeAddSystemServiceTable* function.

Some entries in the duplicated syscall tables are replaced by wrapper functions that perform additional security checks whenever the caller is a monitored malware process. These functions range from very small checks that just deny access to a certain functionality, to larger functions that take into account more input parameters and perform additional validation.

Since the syscall ID numbers change between major Windows versions, the sandbox driver also consists of a syscall name to ID mapping table.

Architecture. The image dump sandbox consists of several architectural blocks, the ones implementing the dumping and filtering deserve a greater description.

User-to-kernel mode interaction. The before mentioned driver creates a device called Sandbox that awaits connections from a user-land master process. The master process is responsible for spawning malware processes. By default, all the master's children are monitored by the driver, and if any of them exits (this is monitored by a *Create/DestroyProcess* notify routine registered by the driver), a kernel level PE image dump is performed. This image dump is transfered to the master process using a simple packet-based protocol. This protocol is implemented by the user-land sandbox library.

Kernel syscall filter. The goal of this block is to provide the malware specimen with enough working space for unpacking but not enough to harm the system in any way that would make a virtual machine revert a must. So as to achieve this aim the kernel syscall functions detailed Table 5.1 in are filtered.

Anyone acquainted with the Windows Kernel will rapidly see the purpose of these filters, a thousand feet view of this would be:

- Malware launched processes are tracked.
- Generally, read-only access to the registry is allowed while write access is denied.
- Generally, read-only access to files is allowed while write access is denied.

CSRSS packet filter. The CSRSS is the Windows Subsystem support process. A Windows process communicates with the CSRSS processes using the ApiPort port and a set of port syscalls. The packets that are sent to the Windows Subsystem support

[3] Obtaining KeServiceDescriptorTableShadow address in Windows XP Kernel mode. http://hi.baidu.com/netelife/blog/item/8058363bf55200e914cecbcb.html

Table 5.1: Kernel syscall functions filtered in the image dump sandbox

Category	**Functions**
Process-related	*NtCreateProcess, NtCreateProcessEx, NtOpenProcess, NtDebugActiveProcess, NtDebugActiveProcess, NtCreateDebugObject, NtSuspendProcess.*
Registry-related	*NtOpenKey, NtCreateKey, NtSaveKey, NtSaveKeyEx, NtDeleteValueKey, NtDeleteKey, NtSetInformationKey, NtUnloadKey, NtUnloadKeyEx, NtSetValueKey.*
File-related	*NtOpenFile, NtCreateFile, NtLockFile, NtCreatePagingFile, NtSetInformationFile, NtDeleteFile, NtUnlockFile, NtSetEaFile, NtSetVolumeInformationFile.*
Thread-related	*NtSuspendThread, NtOpenThread.*
Other	*NtOpenEvent, NtCreateEventPair, NtOpenEventPair, NtCreateKeyedEvent, NtReleaseKeyedEvent, NtCreateNamedPipeFile, NtCreateMailslotFile, NtShutdownSystem, NtRequestWaitReplyPort (this function is used for CSRSS filtering), NtRaiseHardError, NtAdjustGroupsToken, NtAdjustPrivilegesToken, NtCreateDirectoryObject, NtCreatePort, NtCreateWaitablePort, NtCreateProfile, NtCreateSymbolicLinkObject, NtCreateToken, NtDeleteObjectAuditAlarm, NtOpenDirectoryObject, NtOpenObjectAuditAlarm, NtOpenSymbolicLinkObject, NtPrivilegeObjectAuditAlarm, NtPrivilegedServiceAuditAlarm, NtSetDebugFilterState, NtSetDefaultHardErrorPort, NtSetDefaultLocale, NtSetDefaultUILanguage, NtSetEvent, NtSetHighEventPair, NtSetHighWaitLowEventPair, NtSetInformationDebugObject, NtSetInformationJobObject, NtSetInformationObject, NtSetInformationToken, NtSetIntervalProfile, NtSetLdtEntries, NtSetLowEventPair, NtSetLowWaitHighEventPair, NtSetQuotaInformationFile, NtSetSecurityObject, NtSetSystemEnvironmentValue, NtSetSystemInformation, NtSetSystemPowerState, NtSetSystemTime, NtSetUuidSeed, NtStartProfile, NtStopProfile, NtSystemDebugControl*

process contain different function requests such as console related functionality or session related functionality. The image dump sandbox filters the *Exit/Logoff (0x30400)* CSRSS packets.

GUI syscall filter. Following the same filtering principle described for the Kernel filters, this block filters the following calls: *NtUserDestroyWindow, NtUserFindWindowEx, NtUserPostMessage, NtUserQuerySendMessage, NtUserChangeDisplaySettings, NtUserCreateWindowEx.* The main goal of these actions is to prevent the malware process from killing sandbox or crucial system processes.

Summary. In its current form, the image dump sandbox allows the malware process to have enough working space to perform initialization and unpacking, but not enough to cause harm to other processes or system, nor to escape the sandbox. Thanks to the methods used, the sandbox implementation is rather short, simple, and extensible, easily allowing for modifications to be made, both in the wrapper function list, as well as in the wrapper functionality.

5.3.4 BANOMAD-YARA: banker family identifier

Overview. YARA[4] is a tool aimed at helping malware researchers to identify and classify malware families. With YARA you can create descriptions of malware families based on textual or binary information embedded in those families' samples. These descriptions, referred to as rules, consist of a set of strings and a Boolean expression that determines the rule logic. Rules can be applied to files or running processes in order to determine if it belongs to the described malware family.

An example will help in clarifying YARA's functionality. Suppose that we have a malware family consisting of two variants, one of them downloads a malicious file from *http://foo.com/badfile1.exe*, the other downloads a file from *http://bar.com/badfile2.exe*, the URLs are hardcoded in the binary. Both variants drop the download into a file named *win.exe*, which also appears hardcoded in the samples. For this hypothetical family we can create a rule like this:

```
rule BadBoy
{
        strings:
                $a = "win.exe"
                $b = "http://foo.com/badfile1.exe"
                $c = "http://bar.com/badfile2.exe"
```

[4] YARA-Project. http://code.google.com/p/yara-project/

```
        condition:
                $a and ($b or $c)
}
```

This rule instructs YARA to report as BadBoy those files or processes containing the string win.exe and any of the two URLs.

This is just a simple example, but more complex and powerful rules can be created by using binary strings with wild-cards, case-insensitive text strings, regular expressions, and many other features provided by YARA.

System description. YARA can be invoked from Python scripts. The yara-python extension is provided in order to make YARA functionality available to Python users. Using this extension a Python infrastructure for automatic processing of the dumps generated by the image dump sandbox has been developed (BANOMAD-YARA).

The output of the sandbox is checked against a set of banker family identification rules. The following is an example of a rule created to identify the infamous Zeus/Zbot family:

```
rule zbot : banker
{
     strings:
        $a = "__SYSTEM__" wide
        $b = "*tanentry*"
        $c = "*<option"
        $d = "*<select"
        $e = "*<input"

     condition:
        ($a and $b) or ($c and $d and $e)
}
```

For the time being, generating these rules requires manual intervention on behalf of a reverse engineer. This is not a problem since one of the working hypotheses of BANOMAD as a whole is that there is a limited amount of banking trojan families and these are generated with creation kits for dummies. Hence, once a rule has been generated for a particular family the reverse engineer is not required to continue studying other samples belonging to that banker category, he would only need to do so whenever there is a version upgrade of the family that evades the rule.

As to how new banking trojan families are detected for a first manual reverse engineering study, this is out of the scope of this document but some ideas are briefly pointed out:

- Making use of the behavioural analysis sandbox explained in the following section the network traffic generated by the samples received at VirusTotal is searched through looking for URLs accepting POST/GET parameters. These parameters might be being used for information theft. Sometimes thefts will travel in a cyphered format towards the mothership server, in these cases the entropy of the POST/GET parameter payload can be automatically studied to produce candidates for reverse engineering.

- The behavioural analysis sandbox can also be used to identify system activity that may reveal fraud-related mechanisms: modification of the local hosts file, configuration of an HTTP(S) proxy, etc.

- Certain Portable Executable imports may reveal banker activity. For example, samples stealing bank-related user certificates often import Windows CryptoAPI functions. Using libraries such as pefile[5] it is trivial to implement a reverse engineering candidate generator.

- Another YARA ruleset with rules for identifying banking entity names is in place flagging samples whose dumps contain potential financial organization monitoring strings. Very often the banker strings are also cyphered/obfuscated, in order to prevent trivial algorithms from hindering the detection, dumps are processed by a set of decyphering/decoding routines for standard algorithms (base64-ASCII, hex-ASCII, XOR, etc.) and the result is also subjected to the banking entities YARA ruleset.

- Threatscope (antivirus, malware research companies, etc.) blogs are followed since they sometimes publish information on manually detected banking trojans.

- Blackhat forums and communities are inspected with the aim of getting access to banking trojan creation kits being sold in the underground economy.

These are only some of the approaches being used to develop the banker ruleset. Work is in progress in order to try to automatize rule creation, not only for banker families but for malware families as a whole.

Summary. This functional block is a keystone element of BANOMAD's infrastructure since it filters VirusTotal's reception flux creating a feed of interesting samples for further

[5]Pefile a Python module to read and work with Portable Executable (PE) files. http://code.google.com/p/pefile/

automatic study. Moreover, BANOMAD-YARA labels samples with their particular family, this allows other proecessing stages (Python analysis framework) to apply family dependent analysis criteria on the binary.

5.3.5 Behavioural analysis sandbox

Overview. This sandbox executes the malware received at VirusTotal in a controlled environment and observes what the malware is doing. Based on these observations an analysis report is generated. The sandbox analysis can be extended through a Python plugin system that allows specific actions to be performed based on the nature of the executed sample.

This tool fulfills three design criteria: automation, effectiveness, and correctness for the Win32 family of operating systems.

Automation is achieved by performing dynamic analysis of the malware and through the Python plugin architecture later described. Malware is analysed by executing it within a simulated environment, which works for any type of malware in almost all circumstances.

Effectiveness is achieved by using the technique of API hooking. Following the same working principle that many trojans use to monitor targeted bank entities, calls to the Windows Application Programmers' Interface (API) are rerouted to the monitoring software before the actual API code is called, thereby creating insight into the sequence of system operations performed by the malware sample. API hooking ensures that all those aspects of the malware behaviour for which the API calls are hooked are monitored. API hooking therefore guarantees that system level behaviour (which at some point in time must use an API call) is not overlooked unless the corresponding API call is not hooked.

Finally, the correctness goal is fulfilled through the technique of DLL code injection. Roughly speaking, DLL code injection allows API hooking to be implemented in a modular and reusable way.

System description. Sandboxing is provided through KVM[6] virtualization. KVM (for Kernel-based Virtual Machine) is a full virtualization solution for Linux on x86 hardware containing virtualization extensions (Intel VT or AMD-V). It consists of a loadable kernel module, kvm.ko, that provides the core virtualization infrastructure and a processor specific module, kvm-intel.ko or kvm-amd.ko.

[6] Kernel Based Virtual Machine. `http://www.linux-kvm.org/page/Main_Page`

Using KVM, one can run multiple virtual machines running unmodified Linux or Windows images. Each virtual machine has private virtualized hardware: a network card, disk, graphics adapter, etc. These characteristics are shared by other alternatives such as VMWare or VirtualBox, however, KVM was chosen because:

- It is open source. Although so is VirtualBox and some versions of VMWare for example.
- Implementing sandbox detection is more complex than doing so for VMWare, VirtualBox or QEMU.
- Scripting KVM is easier than doing so for its alternatives.

With respect to API hooking, this is implemented with the EasyHook[7] library. This library was chosen over other alternatives such as Microsoft Detours due to the following main reasons:

- It is by far the more stable API hooking library.
- It includes more functionality than its alternatives: deadlock protection in multithread/multicpu environments, 32 and 64 bit support, hooks may be written in managed code (C#, IronPython, VB.Net, etc.), etc.
- It is open source (LGPL).

As to the hooks themselves, they are implemented in C# in a DLL called *vtinject*. Specific information about the hooked functions is provided in the Architecture subsection.

The sandbox does not execute all samples received at VirusTotal since many of these are not Portable Executables, are corrupt or are unpacked executable versions whose import and relocation tables have been damaged or destroyed. So as to perform the appropriate filtering a pyew[8]-centric module is used to check whether the analysis candidate is indeed a Portable Executable and its format is, at a first glance, suitable for the Windows loader.

[7]EasyHook The reinvention of Windows API hooking. http://easyhook.codeplex.com/

[8]Pyew — A Python tool like radare or *iew for malware analysis. http://code.google.com/p/pyew/

Architecture. The sandbox itself consists of two applications: *vtinject.exe* and *vtinjectlib.dll*. The sandbox creates a suspended process of the malware application and injects the DLL into it. At the initialization of this DLL, API hooks for all critical API functions are installed. In addition to injecting the hooking library into the malware process, injections are performed on the legitimate explorer.exe and svchost.exe system processes. After this initialization phase, the malware process is resumed and executed for a given amount of (configurable) time, usually 3 minutes. During the malware's execution, all hooked API calls are rerouted to the referring hook functions in the DLL. These hook functions inspect the call parameters, and inform *vtinject.exe* about the API call in the form of message queue notification objects following this format:

```
process_name;pid;time;sandbox;API;arg1;arg2;...;argn;return_value
```

vtinject.exe then outputs these messages to the standard output, where they can be processed by other infrastructure units. The hook function may sometimes modify the the API call result before returning to the calling malware application in order to hide the presence of the sandbox.

Besides the monitoring, the DLL also has to ensure that whenever the malware starts a new process or injects code into a running process, the sandbox is informed about that. The sandbox then injects a new instance of the DLL into that newly created or already existing process so all API calls from this process are also captured.

IPC Between Sandbox and the DLL. There is a lot of communication between the executable and all the loaded instances of the monitoring DLL. Since the communication endpoints reside in different processes, this communication is called interprocess communication (IPC). To implement this IPC Windows Message Queues (MSMQ) are used. MSMQ is essentially a messaging protocol that allows applications running on separate servers/processes to communicate in a failsafe manner. A queue is a temporary storage location from which messages can be sent and received reliably, as and when conditions permit. This enables communication across heterogeneous networks and between computers which may not always be connected. By contrast, sockets and other network protocols assume that direct connections always exist.

Implementation of *vtinject.exe*. The work of the sandbox can be summarized into three phases:

1. Initialization phase.
2. Execution phase.
3. Analysis phase.

Table 5.2: Behavioural sandbox hooks

Library	**Functions**
KERNEL32.DLL	*CreateFile, DeleteFile, CreateDirectory, RemoveDirectory, CreateProcessInternal, ReplaceFile, MoveFileWithProgress, SetFileAttributes.*
NTDLL.DLL	*NTCreateThread, NTCreateMutant, NTWriteFile.*
ADVAPI32.DLL	*RegConnectRegistry, RegCreateKey, RegDeleteKey, RegSet-ValueEx, RegDeleteValue.*
USER32.DLL	*SetWindowsHookEx, MessageBoxTimeout.*
URLMON.DLL	*URLDownloadToFile.*
WININET.DLL	*InternetOpenURL, InternetOpen, InternetConnect, HTTPOpenRequest.*

In the first phase, the sandbox initializes and sets up the malware process. It then injects the DLL and exchanges some initial information and settings. If everything worked well, the process of the malware is resumed, and the second phase is started. Otherwise, the sandbox kills the newly created malware process and also terminates. The second phase lasts as long as the malware executes, but it can be ended prematurely by the sandbox. This happens if a timeout occurs or some critical conditions require an instant termination of the malware. During this phase, there is a heavy communication between *vtinjectlib.dll* instances in the pertinent running processes and *vtinject.exe*. In the third phase, all the collected data is analyzed, and a TXT analysis report is generated from that. The Setup output section shows an XML containing a DOM space built using the TXT output produced by the sandbox.

What API functions are hooked? There often are multiple API functions that can be used for the same purpose. Just as often there are layered API functions that can call each other recursively. So it is necessary to find a minimal subset of those functions that cover all possible execution chains. The details of the hooking choices are summarized in Table 5.2.

Most of these functions are self-explanatory. The aim of the hooks is to trace and monitor all relevant system calls and generate an automated, machine readable report that describes the following:

- Which files the malware sample has created or modified.
- Which changes the malware sample pefromed on the Windows registry.

- Which dynamic link libraries (DLL) were loaded before executing.
- Which virtual memory areas were accessed.
- Which processes were created.
- Which network connections were opened and what information was sent over such connections.

This last point is fulfilled in combination with network traffic capturing outside the guest operating system (performed by KVM itself).

Rootkit functionality. Since the malware sample should not be aware of the fact that it is executed inside of a controlled environment, *vtinject.dll* implements some rootkit functionality. All system objects that belong to the sandbox are hidden from the malware binary. These are processes, modules, files, registry entries, mutexes events, and handles in general. This at least makes it much harder for the malware sample to detect the presence of the sandbox.

Summary. With the help of the behavioural analysis sandbox we are able to automatically generate a report of the behaiour of a given malware binary. We can observe how a malware process interferes with the rest of the system. Compared with manual code analysis reports, the sandbox reports all the important actions, but some small details and behaviour variants (e.g. creating certain event objects) are not detected. This is because the corresponding API calls are not hooked in the current implementation. Having said this, there is no need to extend the hooks to detect these differences since the present setup gives all the information that is of interest for bank CERTs. Note that the whole of BANOMAD has been built taking into account the demands of several financial organizations which are clients of Hispasec.

5.3.6 Python analysis framework

Overview. For the sake of brevity, the previous description of the behavioural sandbox has omitted a key feature of its architecture. The family dependent analysis plugins. The sandbox has three plugin injectable execution points:

1. Pre-execution: executes a piece of code before the execution of the sample under analysis.
2. In-execution: executes a piece of code while the malware is under execution.

3. Post-execution: executes a piece of code before the sandbox virtual machine is reverted to a clean state, just after the analysis has ended or timed out.

In addition to these plugin injection points there is an independent analysis framework that is in charge of grouping the information provided by the rest of the BANOMAD blocks so as to gather the pertinent intelligence and extract the data that will build the analysis reports.

System description. The final goal of this architectural piece is to identify the entities targeted by the trojan, its data dropzone(s) and any additional malicious communication points. Some of this information will be easily accessible through direct parsing of the dumps generated by the image dump sandbox, however, the more advanced trojans will implement rootkit techniques, cyphering, C&C communication and other mechanisms that make the dump of little use.

In such cases there is a need to access the sample execution information and even interact with the sample and its actions if required. Moreover, sometimes the fraud-related activity of the malware must be triggered. For example, the FlashFaker banker family displays a Windows error message just after execution, unless the OK button is clicked by the user, the malware does not continue with its normal execution. Hence, there is a need to interact via Windows messages with the sample in order to emulate the end user actions (clicking in the OK button), this is done via an in-execution python plugin.

Architecture. As already described, this framework is divided into the behavioural sandbox plugins and the analysis scripts that are fed by the data generated by the rest of the BANOMAD subsystems.

The sandbox plugins are simply python modules or module bundles that implement a given set of (pre-execution, in-execution, post-execution) functions expected by the behavioural sandbox's main executable. Not all families will require routines for each of these code injectable points, some families don't even require plugins at all.

Pre-execution routines. These routines set up the environment required by the malware sample for a correct execution. For example, some families will require some specific non-standard OCX or DLL files that can be copied into the machine by pre-execution routines. Other families may require the sample to be named in a specific way, renaming can also be performed by the pre-execution routines.

In-execution routines. We have already mentioned one of the uses of these routines, clicking on dialogs displayed by the malware that prevent further execution of the sample. Other responsibilities of these routines may be simulating browsing activity using

Python's PAMIE library[9] . For example, executables installing BHO's will require Internet Explorer to be launched after the installation so as to record the communication with the trojan's mothership server since the malicious actions are performed by the BHO DLL executing in the context of Internet Explorer.

Post-execution routines. Certain banker families use rootkit techniques or multi-modular approaches that require some sort of interaction prior to the virtual machine revert in order to retrieve all the pieces of information required to extract the necessary data for building the final sample reports. For example, Zbot executables downlaod a cyphered configuration file that contains bank monitoring strings, this file is hidden using rootkit techniques. Additionally, the encryption of this configuration file is infection dependent, hence, before reverting the virtual machine the file should be decyphered by a post-execution routine and the targets should be reported to BANOMAD.

As to the other key element of the susbsystem, the plugin based correlator & data miner. It is just a set of family focused python scripts that act on all the data produced by BANOMAD in order to extract the information that builds the final banker reports. These scripts may range from something as simple as applying regular expressions to the sample's image dump up to applying complex logic to the network traffic generated by the executable or deciphering registry key values stored by it. Additionally, the scripts use the Wombat APIs (WAPI) of the other Wombat partners in order to include additional information about artifacts mentioned in the report. For example, all spotted HTTP communication points are queried against Honeyspider's dataset in order to see if those sites have also been seen as distribution points (via drive-by-downloads or browser exploits) for the reported family or any other malware file.

Summary. The block hereby described groups the elements labeled as family analysis plugins and plugin based correlator & data miner in the functional block diagram at the beginning of this section. The setup interacts with the behavioural analysis sandbox and receives the information collected at all prior BANOMAD stages in order to extract information of interest so as to further understand the specific family variant and identify any infrastructure that should be taken down in order to neutralize or mitigate the data thefts produced by the trojan.

[9]Python Automation Module for I.E. http://pamie.sourceforge.net/

5.3.7 Report generator

The report generator simply formats all of the information previously extracted into an easily parseable XML report. The XMLs are generated with webpy templetor[10] templates. The only detail worth mentioning of this setup is that the templates contain family specific descriptions that extend the information produced by the rest of the BANOMAD setup. These descriptions have been written after manual reverse engineering of some variants of each family and after tracking the cybercrooks behind the samples in underground forums, irc channels etc. The descriptions are thus common to all trojans belonging to a given family.

5.4 Setup output

As already stated, the final BANOMAD output is an XML report that may be used by the service receiver to set up a fraud tracking web portal, develop a dropzone take down system, implement a server side malicious BHO CLSID inspector to deny access to electronic banking private spaces, etc.

The following is a summarized version of a report generated for a sample belonging to the infamous Zeus/Zbot banker family.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE TARGETED_ATTACK_REPORT SYSTEM "fullfeed-1.5.dtd">

<TARGETED_ATTACK_REPORT>

<REPORT_METADATA>
  <REPORT_ID><![CDATA[EX-20110310-124]]></REPORT_ID>
  <REPORT_DATE><![CDATA[10/03/2011]]></REPORT_DATE>
  <FOCUS_ENTITY><![CDATA[Wellsfargo]]></FOCUS_ENTITY>
  <ENTITY_CODE><![CDATA[0090]]></ENTITY_CODE>
</REPORT_METADATA>

<SAMPLE_IDENTIFICATION>
  <MD5><![CDATA[bc4ee6e6d1348a5da94f8af23009eb27]]></MD5>
  <SHA-1><![CDATA[1f87bbcc5c3ef21a93951961ce9d3077f2636110]]></SHA-1>
  <SHA-256><![CDATA[4115287b886040991fb6b7949233f937ac0874d3b575b65f8b90ea0de1924350]]></SHA-256>
</SAMPLE_IDENTIFICATION>

<PRELIMINARY_SAMPLE_CHARACTERIZATION>
  <DETECTION_DATE><![CDATA[10/03/2011]]></DETECTION_DATE>
  <SIZE><![CDATA[89088]]></SIZE>
  <FILE_TYPE><![CDATA[PE32 executable for MS Windows (GUI) Intel 80386 32-bit]]></FILE_TYPE>
  <PACKER><![CDATA[None/Unknown]]></PACKER>
  <LAB_FAMILY_NAME><![CDATA[Zeus/Zbot]]></LAB_FAMILY_NAME>
  <ENTITY_MONITORING_TECHNIQUES>
    <TECHNIQUE><![CDATA[API Hooking to compare requested URLs with a list of interesting entities]]></(...)
          TECHNIQUE>
  </ENTITY_MONITORING_TECHNIQUES>
  <DATA_CAPTURING_TECHNIQUES>
    <TECHNIQUE><![CDATA[Form grabbing]]></TECHNIQUE>
    <TECHNIQUE><![CDATA[Injection of fraudulent form fields]]></TECHNIQUE>
    <TECHNIQUE><![CDATA[Restricted area screenshots]]></TECHNIQUE>
    <TECHNIQUE><![CDATA[Man-in-the-middle fake sites]]></TECHNIQUE>
```

[10]Web.py templating system (codename: templetor). http://webpy.org/templetor

```
    <TECHNIQUE><![CDATA[Protected Storage content harvesting]]></TECHNIQUE>
  </DATA_CAPTURING_TECHNIQUES>
  <AFFECTED_BROWSERS>
    <BROWSER><![CDATA[Internet Explorer]]></BROWSER>
    <BROWSER><![CDATA[Mozilla Firefox]]></BROWSER>
    <BROWSER><![CDATA[Google Chrome]]></BROWSER>
    <BROWSER><![CDATA[Opera]]></BROWSER>
  </AFFECTED_BROWSERS>
  <IMPORTED_SYMBOLS>
    <LIBRARY library_name="advapi32.dll">
      <SYMBOL><![CDATA[AddAccessDeniedAce]]></SYMBOL>
      <SYMBOL><![CDATA[AddAuditAccessAce]]></SYMBOL>
      <SYMBOL><![CDATA[AdjustTokenGroups]]></SYMBOL>

       [... more imports ...]

      <SYMBOL><![CDATA[TrusteeAccessToObjectW]]></SYMBOL>
    </LIBRARY>

     [... more libraries ...]

  </IMPORTED_SYMBOLS>
</PRELIMINARY_SAMPLE_CHARACTERIZATION>

<ANTIVIRUS_ANALYSIS>
  <AV_RESULT>
    <AV_ENGINE><![CDATA[AhnLab-V3]]></AV_ENGINE>
    <SIGNATURE><![CDATA[Win-Trojan/Zbot.88576]]></SIGNATURE>
  </AV_RESULT>
  <AV_RESULT>
    <AV_ENGINE><![CDATA[AntiVir]]></AV_ENGINE>
    <SIGNATURE><![CDATA[TR/Crypt.ZPACK.Gen]]></SIGNATURE>
  </AV_RESULT>
  <AV_RESULT>
    <AV_ENGINE><![CDATA[Avast]]></AV_ENGINE>
    <SIGNATURE><![CDATA[Win32:Zbot-MYU]]></SIGNATURE>
  </AV_RESULT>

   [... more antivirus results ...]

</ANTIVIRUS_ANALYSIS>

<TARGETED_ENTITIES>
  <TARGET>
    <ENTITY><![CDATA[Banco Santander]]></ENTITY>
    <MONITORING_CHAIN><![CDATA[https://www.gruposantander.es/*]]></MONITORING_CHAIN>
  </TARGET>
  <TARGET>
    <ENTITY><![CDATA[Gad eG]]></ENTITY>
    <MONITORING_CHAIN><![CDATA[https://internetbanking.gad.de/banking/*]]></MONITORING_CHAIN>
  </TARGET>
  <TARGET>
    <ENTITY><![CDATA[Citibank]]></ENTITY>
    <MONITORING_CHAIN><![CDATA[https://www.citibank.de/*/jba/mp#/SubmitRecap.do]]></MONITORING_CHAIN>
  </TARGET>

   [... more targets ...]

</TARGETED_ENTITIES>

<COMMUNICATION_POINT CATEGORY="OTHER" STATUS="FUNCTIONAL">
  <DESCRIPTION><![CDATA[<p>This request downloads the trojan's configuration file. The file is encrypted.</p>(...)
        ]]></DESCRIPTION>
  <NETWORK_TRAFFIC>
    <HTTP METHOD="GET">
      <PACKET>
        <DESTINATION_IP><![CDATA[208.101.9.140]]></DESTINATION_IP>
        <DESTINATION_PORT><![CDATA[80]]></DESTINATION_PORT>
      </PACKET>
      <DESTINATION_HOST><![CDATA[televisionfree.co.tv]]></DESTINATION_HOST>
      <PATH><![CDATA[/maknyus/cfg.bin]]></PATH>
    </HTTP>
  </NETWORK_TRAFFIC>
  <SITE_CHARACTER><![CDATA[Specifically created, wholly bad]]></SITE_CHARACTER>
</COMMUNICATION_POINT>

<COMMUNICATION_POINT CATEGORY=”DROPSITE” STATUS=”FUNCTIONAL”>
  <DESCRIPTION><![CDATA[<p>This request sends the stolen data in a cyphered fashion.</p>]]></DESCRIPTION>
  <NETWORK_TRAFFIC>
    <HTTP METHOD=”POST”>
          <PACKET>
            <DESTINATION_IP><![CDATA[208.101.9.140]]></DESTINATION_IP>
        <DESTINATION_PORT><![CDATA[80]]></DESTINATION_PORT>
          </PACKET>
          <DESTINATION_HOST><![CDATA[televisionfree.co.tv]]></DESTINATION_HOST>
          <PATH><![CDATA[/maknyus/gate.php]]></PATH>
    </HTTP>
  </NETWORK_TRAFFIC>
  <SITE_CHARACTER><![CDATA[Specifically created, wholly bad]]></SITE_CHARACTER>
</COMMUNICATION_POINT>

<SYSTEM_MODIFICATIONS>
  <EVENT MONITOR=”PROCESS” TYPE=”CREATED”>
    <CAUSATOR><![CDATA[C:\WINDOWS\explorer.exe]]></CAUSATOR>
    <FULLY_QUALIFYING_NAME><![CDATA[C:\Documents and Settings\john doe\Escritorio\sample.exe]]></(...)
         FULLY_QUALIFYING_NAME>
  </EVENT>
  <EVENT MONITOR=”REGISTRY” TYPE=”SETVALUEKEY”>
    <CAUSATOR><![CDATA[C:\Documents and Settings\john doe\Escritorio\sample.exe]]></CAUSATOR>
    <FULLY_QUALIFYING_NAME><![CDATA[HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\userinit]]></(...)
         FULLY_QUALIFYING_NAME>
  </EVENT>
  <EVENT MONITOR=”FILE” TYPE=”WRITE”>
    <CAUSATOR><![CDATA[C:\Documents and Settings\john doe\Escritorio\sample.exe]]></CAUSATOR>
    <FULLY_QUALIFYING_NAME><![CDATA[C:\WINDOWS\system32\sdra64.exe]]></FULLY_QUALIFYING_NAME>
  </EVENT>
  <EVENT MONITOR=”FILE” TYPE=”WRITE”>
    <CAUSATOR><![CDATA[C:\WINDOWS\system32\services.exe]]></CAUSATOR>
    <FULLY_QUALIFYING_NAME><![CDATA[C:\WINDOWS\system32\lowsec\user.ds]]></FULLY_QUALIFYING_NAME>
  </EVENT>
  <EVENT MONITOR=”FILE” TYPE=”WRITE”>
    <CAUSATOR><![CDATA[C:\WINDOWS\system32\svchost.exe]]></CAUSATOR>
    <FULLY_QUALIFYING_NAME><![CDATA[C:\WINDOWS\system32\lowsec\local.ds]]></FULLY_QUALIFYING_NAME>
  </EVENT>
</SYSTEM_MODIFICATIONS>

<DETAIL>
  <DESCRIPTION><![CDATA[Detailed family description template product of manual reverse engineering.]]></(...)
       DESCRIPTION>
  <STOLEN_INFORMATION>
    <DESCRIPTION><![CDATA[Description about any data theft that follows]]></DESCRIPTION>
    <INTERESTING_DATA_PACKET>
      <DESCRIPTION><![CDATA[Interception of initial authentication form]]></DESCRIPTION>
      <DATA TYPE=”DECRYPTED”><![CDATA[Template data theft example for family and focus entity]]></DATA>
    </INTERESTING_DATA_PACKET>
  </STOLEN_INFORMATION>
</DETAIL>

</TARGETED_ATTACK_REPORT>
```

Note that this report provides enough information so as to build other antimalware related setups. For example, a feed of reports of this nature can proof itself extremely useful in fraud related forensics investigations.

Very often bank CSIRT teams need to carry out an examination of the machines of customers that have been victims of fraud. If these teams were to build a database of registry keys, launched processes, created files, etc. reported in these XMLs they would have a means of automatically identifying the robbery culprit. Hispasec has been

Table 5.3: Volume of unique binary variants analysed for BANOMAD family

Family	**Number of unique samples**
Tanatos/Bugbear	49554
Delephant	34341
Zeus/Zbot	12059
SpyEye	5600
Sinowal/Torpig	4351
Goldun	2567
Multibanker	1020
Ambler/Limbo/Nethell	445
MDCOCO	264
FakeFlasher	96

following this approach as an early stage step in forensic investigations and it has had a very high success rate.

5.5 Experimental results

BANOMAD has been running in a stable fashion since September 2010, acting on VirusTotal's malware feed. At present plugins for automated analysis of 10 banker families have been written and more are on the way. The setup has been giving service both to Spanish and Latin American banks and important USA and other international security/telco companies that resell the reports to some of the largest world wide financial entities.

This section includes a summary of the findings worth mentioning after 6 months of nonstop execution.

Family sizes. Regarding the 10 banker families that have been surveyed during these six months, Table 5.3 shows the volume of unique binary variants (different hash) analysed.

A simple glance at these statistics may lead the reader to conclude that the Tanatos/Bugbear family is the most intense, this is not really true. This family is a very old pseudo-banker (2003) that autoreplicates via email and shares, each replication producing a new polymorphic variant of exactly the same piece of original code (dropzones included). The high numbers are due to this polymorphic engine and propagation technique, since

VirusTotal is used by many honeypot infrastructures the volumes are biased by their submissions.

One would expect the infamous Zeus/Zbot family to be the most predominant since its builder is by far the most sold one and the most distributed in underground forums. Moreover, the Zeus/Zbot family implements a server side metamorphic/polymorphic engine that further accentuates its numbers. Indeed, the cybercrooks behind Zbot have implemented automated scanning of their specimens, whenever a variant starts to have a high detection rate among the antivirus industry, the polymorphic/metamorphic server side engine generates a new fully undetectable strain and an update order is sent to the the already infected machines. This new strain is also used in the malware dissemination infrastructure in place (browser exploits, drive-by-downloads, spam emails, etc.).

While all of these reasons do acquaint for the Zeus/Zbot family being the third largest collection, it is still behind the Delephants. Delephants belong to the Brazilian community of malware developers. They are malicious executables developed in Delphi that use Delphi form overlapping to retrieve banking credentials (including two factor authentication mechanisms). Our investigations have led us to conclude that rather than having one unique developer or group of developers behind this set, they are developed and distributed by many different individuals making use of similar techniques. This is probably the reason for it being the second largest set, the long queue economic theory proofs itself to be true and less variants by a far larger developer community accounts for a higher overall family size.

Having made this brief analysis, we must emphasize the fact that the interaction of the security community with these specimens and their infrastructures is strongly biasing the sizes. For example, Zeus/Zbot analyses are easy to automate and many companies are providing takedowns of their dropzones as a service to financial entities. When the trojans' infrastructures are closed attackers are forced to release new variants making use of other dropzones, hence, the number of variants of the family seen in the wild is noticeably increased.

Targeted entities. If we focus on the 5 largest collections, we see that attackers are mainly interested in attacking as many entities as possible with each unique binary.

This makes sense, whenever an end-user gets infected they do not really know what their bank is and they must increase the chances of intercepting their credentials by targeting the largest possible number of entities, with the hope that the victim's bank will lie among them.

The exception to this rule are Delephants, each binary targets, on average, only 7 financial institutions. The reason behind this lies in the distribution vector being used

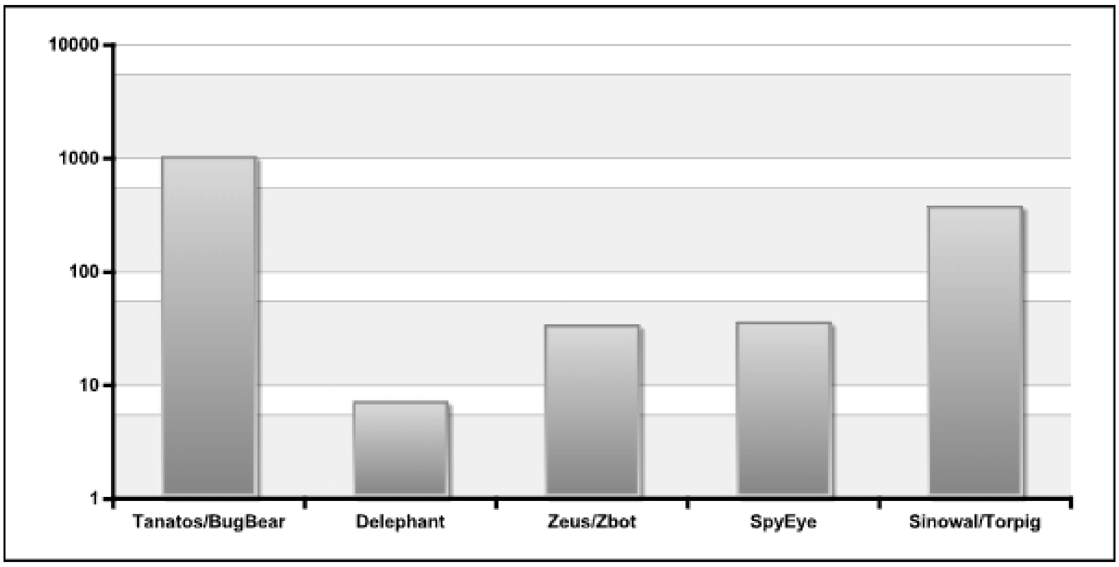


Figure 5.7: Average number of entities targeted by each binary belonging to the TOP5 families studied

by attackers. Let us recall that Delephants are produced mainly by brazilian attackers and the targeted entities are nearly always brazilian banks. Delephants get distributed via spam emails with attachments, these messages use brazilian bank-related baits (new security token executable, etc.) written in portuguese, hence, attackers are already filtering out their potential victim base, making it relatively useless to introduce other targets in their creations.

Figure 5.7 shows the average number of entities targeted by each binary belonging to the TOP5 families studied.

Data forwarding. During the period of time under consideration 23107 unique data dropzones have been recorded. These malicious communication points range from compromised legitimate web sites and email accounts to genuinely registered hosts and domain names to accomplish ecrime.

Given the volume of different binaries observed, this number reveals that (as expected) several samples are making use of the same communication points, the overall ratio being nearly 5 different samples per drop zone.

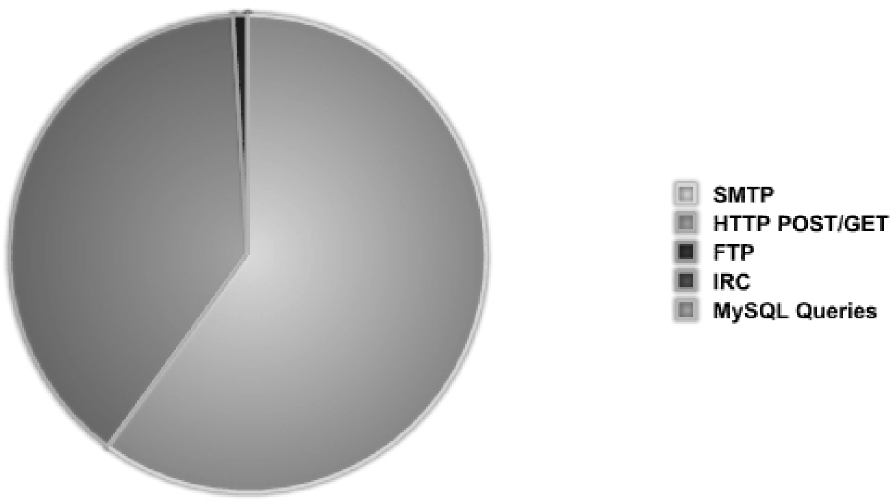


Figure 5.8: Popularity of the data forwarding techniques used by the tracked banker families

The pie chart illustrates that SMTP is the predominant data forwarding technique, with almost 60% of the drop zones being email accounts. This number is biased by the fact that Delephant samples mostly use email accounts as their data gathering points and Delephant is the largest family after BugBear. BugBear samples also use SMTP, yet since they are just polymorphic variants of the same source code, they always use the same set of reception emails and do not account for the high number of email dropzones observed.

After email, HTTP remains by far the method mostly used. 39% of the drop zones are web scripts receiving the data via HTTP GET/POST requests. Among those samples using the HTTP protocol to drop the data, only 0.5% of them used HTTP over SSL (HTTPS), this reveals that malware authors find far more interesting encrypting the data with custom algorithms that will increase study time rather than making use of HTTPS.

There has been a lot of fuzz regarding P2P covert channels, while malware making use of P2P has been seen, none of it used it to forward the robbed credentials.

We expect the HTTP protocol to continue increasing and eventually become the most popular technique. The reason for this prediction is that the mean life time of HTTP infrastructures is lower than email ones since they are easier to take down (ethical hacking can be used and it is easier to provide proof to abuse teams). The takedowns and the lower life time mean that attackers must set up new data gathering points, thus increasing the number of observed drop zones under this category. Additionally, only 10 families are being tracked, the work of Hispasec's antifraud team reveals that most of the families use HTTP communication, hence, when new family analysis modules will be in place we expect this data forwarding technique to be the most popular one.

Figure 5.8 shows the relative popularities of the data forwarding techniques used by the tracked banker families.

5.6 Current research

At the 2007 VirusBulletin in Vienna, F-Secure publicly presented a tool for analysing banking trojans called Mstrings. The tool was described as follows:

> We studied banking trojans and ended up with the following facts:
>
> 1. Trojans must use filter strings in order to reduce the amount of data they collect.
> 2. Filter strings are banking strings, typically bank URLs.
> 3. Malware that is not interested in banks does not include banking strings.

4. Banking trojans typically encrypt or obfuscate their filter strings within the file image, but decrypt them into memory.

This led to the idea that we could run collections of malware, running samples in a test system and searching the memory of the system for banking strings. If the memory contains banking strings we would collect them in a database and analyse them for trends and other statistical purposes. The same technique could also be used for locating banking trojans from incoming samples in lab automation.

We created an analysis tool called 'Mstrings' for this purpose. Mstrings is an F-secure internal research tool that currently has the following set of features:

- Has a database of search strings (currently contains 1,400 search strings)
- Can search through user-mode and kernel-mode memory.
- Can bypass basic forms of string encryption automatically.
- Has a whitelist of false positive strings.
- Is rather fast. With the current database, it goes through all required areas in memory in 10-30 seconds.

While the Mstrings approach may have been very successful in 2007, most of today's bankers use some sort of entity monitoring strings encryption. Very often these strings will not be decrypted into memory, instead, the visited URLs will be cyphered and then compared with the cyphered versions of the filter strings. Hence, the Mstrings keystone hypothesis, that the banking trojan filter strings will be found as plain text at some point is defeated, being unable to identify the most modern crimeware samples.

Additionally, Mstrings just identifies potential banking trojans, it does not provide any further information about the execution of the sample or its network communication points. Having said this, Mstrings does allow the identification of new banking trojan families since it is not template based. This is one of the reasons that led us to integrate their approach in BANOMAD in order to identify interesting families for which templates should be developed.

Yet another disadvantage of F-Secure's approach in order to automatically produce menace reports is that VirusTotal very often receives submissions of legitimate banking related software, phishing lists, etc. that give rise to many false positives when searching for bank strings.

A better idea for proactively identifying and reporting banking trojans was proposed by Florent Marceau (LEXSI) at SSTIC 2009[11] (Symposium sur la Scurit des Technologies de l'Information et des Communications). LEXSI's setup uses data tainting in order to track the use of sensitive banking related information by the system processes. Electronic banking behaviour is emulated in a web browser and the data inputed is tainted in order to see if the malware sample executed makes any use of it and what exactly it is doing. This system is far more interesting since it allows report generation for families that have never been seen in the past. However, it also presents important limitations:

- Very often malicious behaviour will only be triggered by activity on very specific bank sites, hence the user activity emulation should include interaction in thousands of world wide banks, this can be noticeably slow.
- Since no family specific analysis plugins are provided, non-malicious network communication could be erroneously reported and could lead to takedowns of innocuous sites. E.g. many bankers perform HTTP requests to legitimate sites just to see if they have Internet connection.

Nonetheless, once again this approach is really interesting for extending BANOMAD in order to identify new banker families for which templates should be developed.

5.7 Limitations

The before mentioned setups are not the only ones to present limitations, these can also be spotted at different BANOMAD architectural levels.

Malware could easily fool the image dump sandbox by delaying its unpacking routine so as to exceed the 3 second execution threshold. Empirical experience reveals this is extremely rare. Having said this, should this become common use, the execution threshold can always be adjusted, sleep calls could be fooled, or time delayed packers could be identified with YARA prior to execution and treated in a different manner (using static unpacking for example).

At the end of the day YARA is just a signature based approach for labeling malware samples. Hence, it is subjected to the same limitations that signature-based detection of virus, mainly false positives and false negatives. This is particularly critical in this setup because every so often attackers will release new versions of their trojan builders,

[11]Utilisation du Data Tainting pour l'analyse de logiciels malveillants. http://actes.sstic.org/SSTIC09/Utilisation_du_data_tainting_pour_lanalyse_de_logiciels_malveillants/SSTIC09-slides-F-Marceau-Utilisation_du_data_tainting_pour_lanalyse_de_logiciels_malveillants.ppt

the binaries produced by these new versions may evade the existing YARA rules for that specific family. So as to mitigate this problem statistics on the numbers of samples received at VirusTotal belonging to each of the families in BANOMAD are produced. Whenever a reception valley is spotted an investigation is carried out in order to identify the reason and try to locate and study new versions of a given family if that was the valley culprit.

With respect to the behavioural analysis sandbox, the main drawback is that is that it only analyzes a single execution of the malware. Additionally, API hooking can be bypassed by programs that directly call kernel code to avoid using the Windows API. However, this is rather uncommon in malware, as the malware author needs to know the target operating system, its service pack level and some other information in advance. Execution of VirusTotal's malware feed reveals that most banking trojans are designed to attack a large user base and thus commonly use the Windows API.

Problems related to malware behaviour depending on mothership server orders. Very often different entity monitoring configuration files will delivered based on the country of origin of the victim, other times the mothership server may only order the trojan to exhibit fraudulent behaviour exclusively under certain circumstances. Thus, the external infrastructure dependency may strongly bias the activity seen by BANOMAD.

Since the family analysis plugins depend on the label produced by the YARA subsystem, they are also subjected to its same limitations. In other words, each time there is a new family version release a new plugin may have to coded.

Having identified all of these problems, the overall main limitation is that BANOMAD follows a reactive approach. Banker families must have been previously identified and manually reversed engineered in order to produce automatic analysis plugins and report templates. While this is a characteristic that allows for further investigation and automation, it is still a much better approach than static analysis of all malicious files reported by bank customers to banks themselves or identified in fraud case forensics investigations.

5.8 Conclusion

Throughout the past years efforts have been made to mitigate the online banking fraud threat, these efforts have mainly focused on the server side (web server hardening, periodic pentesting, etc.) and authentication factors (two factor authentication). Little has been done in actually trying to neutralize data theft infrastructure, i.e. performing takedowns with the aim of mitigating the impact of banking trojans in the wild. The lack of efforts in this sense may have been due to the fact that in-depth malware analysis has always been a slow, manual task.

The truth is that the changes in the tools and tactics used by banking malware authors and distributors have allowed traditional strong two factor authentication mechanisms (SMS transaction dependent code) to be circumvented and clearly call for further research in order to provide new approaches to eradicate online theft.

Due to the ever-increasing presence of banking related botnets on the Internet, there is a need for automated systems aiding researchers, CERTs, and security companies/teams in their work. Although limited, some work has indeed been done in this sense. In 2007 F-Secure presented a novel setup for automatically detecting bankers based on automated execution of malware collections and the inspection of the infected machine's memory looking for bank-related strings. Later, in 2009, LEXSI approached this problem through data tainting of credentials introduced in the browser via user emulation in order to see if the malicious process made any use of it.

In the context of WOMBAT Hispasec has developed BANOMAD (BANking Oriented Malware Analysis Droid), a novel combination of non-novel techniques (signature based detection, sandboxing, templating, image dumping, etc.) that builds a banking trojan early warning system on top of VirusTotal. The early component is provided through automation and thanks to the fact that the whole of the online community acts as an early threat detector submitting fresh samples to VirusTotal. Thanks to the approach of family-based analysis templates, more in-depth and accurate alerts can be produced than with other current approaches like LEXSI's or F-Secure's setups, reducing the impact of false positives and providing a very interesting feed of data theft infrastructures to be taken down.

The setup has been running non-stop with the 10 family templates mentioned in this document for over 6 months, at present it is giving service to something more than 20 world wide banks, some of them being the largest financial entities of their respective headquarter countries. Rather than for intelligence and threat landscape purposes, banks have shown a special interest in this early warning system so as to then hire shutdown services to neutralize the trojan network infrastructures reported in BANOMAD's alerts.

6 HoneyBuddy

The popularity of instant messaging (IM) services has recently attracted the interest of attackers that try to send malicious URLs or files to the contact lists of compromised instant messaging accounts or clients. HoneyBuddy, a honeypot-like infrastructure for detecting malicious activities in IM networks, offers a systematic characterization of IM threats based on the collected information. The above infrastructure finds and adds contacts to its honeypot messengers by querying popular search engines for IM contacts or by advertising its accounts on contact finder sites. Our deployment has shown that with over six thousand contacts we can gather between 50 and 110 malicious URLs per day as well as executables. Our experiments show that 21% of our collected executable samples were not gathered by other malware collection infrastructures, while 93% of the identified IM phishing domains were not recorded by popular blacklist mechanisms. Furthermore, our findings show that the malicious domains are hosted by a limited number of hosts that remain practically unchanged throughout time.

6.1 Attacks on Instant Messaging networks

The high population of IM networks makes them an attractive target for attackers that try to exploit them for malicious purposes, such as spreading malware and scamming. We identify four different scenarios of attacks on IM networks.

Malware infection. Recent malware instances [3] can attach to a victim's instant messaging client and start sending URLs that point to malicious websites, or spread themselves by sending executables. In the most common case the malware instance logs in to the IM network, randomly selects users from the victim's contact list, sends the malicious URLs or files and then immediately logs out. In order to be more appealing to potential victims, the URLs point to domains whose name contains the username of the recipient, for example `http://contact_username.party-pics.com` . The vast majority of the attack campaigns we have detected send messages in English. However, we believe that attackers will soon shift towards localized messages, as is the case with one localized phishing site that we have detected.

Compromised accounts. Attackers can also use compromised credentials to log in as several different users and flood the victims' contact lists. Many services, like MSN, use unified credentials for e-mail and instant messaging, making life easier for attackers. Attackers can harvest IM accounts by setting up phishing sites for the service, by planting key-loggers or through social engineering. A relatively known attack campaign is that of websites advertising a service that can reveal to users if someone has blocked them. If the user enters her IM credentials in the website, she is redirected to a page from another domain where nothing happens. Later on, the phishing site owner logs in as the user and sends messages to the victim's contact list.

Exploiting weak privacy settings. Even in the absence of malware infection or stolen credentials, some messengers provide the option to allow incoming messages from people who are not in the user's contact list. We tested the latest client versions of the most popular IM services: MSN live messenger (version 14.0.8089), Skype (version 4.1), Yahoo (version 10) and AIM (version 7.1). MSN live messenger is the only IM client we tested that has a privacy setting enabled by default that blocks messages from accounts not contained in the contact list. Skype, Yahoo and AIM by default allow anyone to send instant messages to our account, but this setting can be opted-out. Attackers exploit these settings to send unsolicited messages to IM users.

Exploiting client software. IM client software suffers from the problem of monocultures. Once an exploit is discovered, then automatically millions of clients can be infected immediately [2]. While in the case of malware infection exploits take advantage of the IM client to spread, this case involves the attack where the IM client is used to infect the rest of the machine.

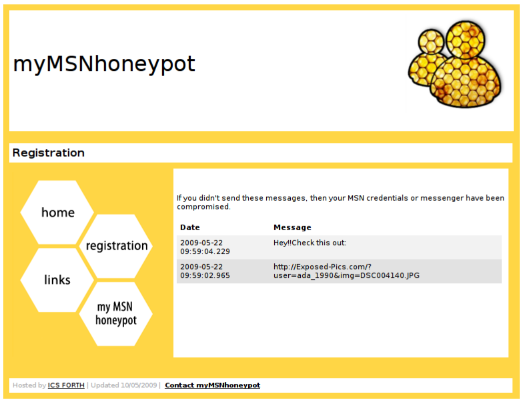
6.2 MyIMhoneypot, a detection service

In this section we present an overview of existing defense measures, and propose a service for the early detection of attacks targeting instant messaging networks. The existing defense mechanisms deployed by instant messaging service providers and other vendors, are insufficient for protecting users from the threats presented in Section 6.1. Anti-virus products that scan files received from instant messaging file-transfers fail to identify all malware used by IM attackers, as shown by our findings. Anti-virus vendors could provide more up-to-date signatures for IM malware by deploying HoneyBuddy for the early collection of such malware. Furthermore, anti-virus products designed to protect users from phishing attacks fail to detect 87% of the malicious URLs collected by our infrastructure. Pop up messages from IM client software that alert users of phishing, that are triggered by all messages that contain a URL even if it is benign, are ineffective since users tend to ignore warnings that are presented even for well-known benign URLs. We propose that IM clients should correlate received URLs with blacklists and alert users only when they belong to malicious domains. We present our client-side mechanism that is orthogonal to existing defense mechanisms; myIMhoneypot, an early detection service that can inform users if their accounts or IM clients have been compromised. IM attacks try to spread through the victim's contact list by sending either URLs or files to the victim's friends. Any user that wants to check if her account is compromised registers with the myIMhoneypot service. Upon registration, the service creates a unique IM honeypot account (for example, a new MSN account that will be used as a decoy account) and informs the user to add that honeypot account to her contact list. As the user will never start a conversation with the honeypot account but an IM attacker will (with great probability), the user can check if something is wrong by visiting the website of the service and checking the conversation logs with her unique honeypot account. If there are entries in the conversation log of her decoy account like the example in Figure 6.1, then there is a strong indication that her IM client or credentials have been compromised.

The reason that a unique IM account must be created per user is twofold. First, if the service has only one or a few honeypot accounts then they can be easily blacklisted (recall that anyone can subscribe to the service, including attackers). The attacker should not be able to distinguish whether a contact is a decoy account or not. The service creates accounts with human-like nicknames. Second, the attacker can try to hack into the service's accounts once she knows the user is a subscriber. Using a unique honeypot per user makes the attacker's life a lot harder. The attacker cannot correlate common friends across accounts and has to try to compromise all the accounts in the user's contact list. Even if she does that, most IM services (at least MSN and AIM) do not keep conversation logs at the server side so she cannot find her spam messages in the logs of decoy accounts.

The attacker could guess the decoy accounts by checking the locally stored conversation logs. Normally, a user will have conversations with all members of her contact list except the honeypot account. Therefore, the attacker could avoid sending messages to accounts for which no conversation logs were found. This attack can be easily circumvented by planting a fake conversation log on the user's side.

The myIMhoneypot service has a limitation. For each registered user, a new IM account must be created in order to be used as a decoy. This process involves the solution of CAPTCHAs [1] which prevents us from making it completely automatic. Although we could claim that MyIMhoneypot is a legal case for laundering CAPTCHAs, we did not implement it for obvious reasons. For the time being, we have to manually create decoy accounts. However, we propose that this service should be implemented by each IM provider as a means of protection for its users. We implemented a prototype of



If you didn't send these messages, then your MSN credentials or messenger have been compromised.

Date	Message
2009-05-22 09:59:04.229	Hey!!Check this out:
2009-05-22 09:59:02.965	http://Exposed-Pics.com/?user=ada_1990&img=DSC004140.JPG

Hosted by ICS FORTH | Updated 10/05/2009 | Contact myMSNhoneypot

Figure 6.1: A screenshot of the log presented to a user whose IM account has been compromised.

myIMhoneypot for the MSN platform. We call it myMSNhoneypot and it can be found at `www.honeyathome.org/imhoneypot` .

We also provide a service that does not require user registration. Users can submit URLs they receive in instant messages to correlate them with our database. As mentioned before, suspicious URLs usually contain the target's username and, thus, searching for an identical URL in our database would rarely result in a match. Therefore, the service searches our database for any URL that has the same top level domain with that of the submitted URL, which is an indication that they might belong to the same campaign. If a match is found the user is presented with a small report containing the date the URL was first caught by HoneyBuddy and the category it was assigned by our classifier. Based on our findings we assign the submitted URLs with a value of how likely they are to still pose a threat to users depending on the time window between being collected by HoneyBuddy and being submitted by the user.

7 Conclusion

The document presented several early warning services, collectively forming the WOMBAT Early Warning System. The services are separate, but – as previous chapters show – they are in fact connected in a number of ways thanks to the cooperation between consortium members. The systems make use of the same datasets (e.g. Wepawet, HoneySpider Network, Anubis). Some use similar interfaces (FIRE and Exposure). The integration of the HoneySpider Network with FIRE even merited a separate chapter.

It is clear, why the systems do not naturally blend well together. FIRE and Exposure could, in principle, be remade into a single, modular system – they share the same general audience (mostly security specialists) and while the monitored phenomena differ, the presentation of results follows similar design. The BANOMAD system, however, has a completely different scope and audience – it is a system aimed specifically at monitoring of the security of Internet banking. The HoneyBuddy myIMhoneypot needs an even simpler interface, as its target group includes average Internet (specifically Instant Messaging) users.

The presented early warning solutions are innovative and useful. They offer previously unavailable alerting capability. The presented results show that the systems work well. The research presented in this deliverable completes the Early Warning System part of Workpackage 5.

Bibliography

[1] CAPTCHA: Telling Humans and Computers Apart Automatically. `https://captcha.net/`.

[2] Vulnerability in PNG Processing Could Allow Remote Code Execution. `http://www.microsoft.com/technet/security/bulletin/MS05-009.mspx`.

[3] W32.Bropia. `http://www.symantec.com/security_response/writeup.jsp?docid=2005-012013-2855-99&tabid=2`.

[4] Internet Systems Consortium. `https://sie.isc.org/`, 2011.

[5] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster. Building a Dynamic Reputation System for DNS. In *19th Usenix Security Symposium*, 2010.

[6] U. Bayer, C. Kruegel, and E. Kirda. TTAnalyze: A Tool for Analyzing Malware. In *15th European Institute for Computer Antivirus Research (EICAR 2006) Annual Conference*, April 2006.

[7] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi. EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis. In *18th Symposium on Network and Distributed System Security (NDSS)*, 2011.

[8] M. Cova, C. Kruegel, and G. Vigna. Detection and analysis of drive-by-download attacks and malicious javascript code. In *Proceedings of the 19th international conference on World wide web*, WWW, 2010.

[9] B. Krebs. Naming and Shaming 'Bad' ISPs. `http://krebsonsecurity.com/2010/03/naming-and-shaming-bad-isps/`, 2010.

[10] B. Krebs. Takedowns: The Shuns and Stuns That Take the Fight to the Enemy. *McAfee Security Jouranl*, (6), 2010.

[11] R. Spoor, P. Kijewski, and C. Overes. The honeyspider network: Fighting client-side threats. In *First, Vancouver*, 2010.

[12] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna. Your botnet is my botnet: Analysis of a botnet takeover. In *ACM Conference on Computer and Communication Security (CCS)*, 2009.

[13] B. Stone-Gross, A. Moser, C. Kruegel, K. Almaroth, and E. Kirda. FIRE: FInding Rogue nEtworks. In *Annual Computer Security Applications Conference (ACSAC)*, 2009.

[14] T.Holz, C. Gorecki, K. Rieck, and F. Freiling. Measuring and Detecting Fast-Flux Service Networks. In *Annual Network and Distributed System Security Symposium (NDSS)*, 2008.