POLITECNICO DI TORINO

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

Web Honeypot 2.0 Deployment of a Fully Functional Honeypot Network and an Analysis of Attackers Behaviors on the Web

Relatori:
prof. Antonio Lioy
prof. Davide Balzarotti

Candidato:
Maurizio ABBA'

ANNO ACCADEMICO 2013-2014

Sommario

1-Come faccio a cambiare lingua!? ho inserito il select language appena prima di begin document ma non cambia nulla..

The first section of this thesis presents the design, implementation, and deployment of a network of 500 fully functional honeypot web-sites, hosting a range of different services, whose aim is to attract attackers and collect information on what they do during and after their attacks.

In the second chapter, we present a platform for the automatic collection, normalization and clustering of files attackers uploaded or modified on the honey-pot network. We also implemented a web interface for monitoring the honeypot network and for a visual representation of clusters, whose design will also be discussed in this chapter.

In the last part of this work we show the results obtained by collecting, normalizing and clustering uploaded files, showing the most common behaviors of web attacks in the real world. We also present some examples of files we collected during our experiments.

Finally, we try to analyze the reasons and goals of such attacks.

All experiments have been realized in Eurecom Institute (Sophia Antipolis, France), with the support of Trend MicroSystem for the websites hosting service.

NOTE: Maybe insert some results, like total number of files uploaded or a brief introduction to common trends (maybe just numbers) to do at the end

Indice

Capitolo 1

Introduction

1.1 Motivations

Web attacks are nowadays considered the most important source of loss of financial and intellectual property. Thanks to the high availability of an Internet connection in the modern world, these kind of attacks are getting more and more common, targeting not only institutions and high-profile companies, but also medium-small size companies owning a web-server and common users, causing overall a massive stealing of valuable personal user information and financial losses in the order of millions of Euros.

Moreover, the increase of different vehicles for browsing the web, like tablets and smartphones, makes web-related attacks a very appealing target for criminals and at the same time a more difficult challenge for securing all possible holes which could allow a potential attacker to take control of the system.

This trend is also reflected in the topic of academic research. In fact, it can be shown as in the last few years a large number of papers, seminars and workshops published in the most important conferences all over the world cover web-related attacks and defenses.

It is possible to categorize these studies in three main sectors:

- analysis of vulnerabilities related to web applications, web servers, or web browsers, and on the way these components get compromised;
- dissection and analysis of the internals of specific attack campaigns;
- proposal of new protection mechanisms to mitigate existing attacks.

The result is that almost all the web infections panorama has been studied in detail: how attackers scan the web or use Google dorks to find vulnerable applications, how they run automated attacks, and how they deliver malicious content to the final users.

There is still a missing piece in this scenario: there are only a few academic works which sufficiently detail the behavior of an average attacker during and after a website is compromised. Even if there are cases where the attacker is only interested in some informations stored in the service itself, like the dump of a database following an SQL injection, in the majority of the cases the attacker wants to maintain access to the compromised machine and include it as part of a larger malicious infrastructure (e.g., to act as a C&C server for a botnet or to deliver malicious documents to the users who visit the page).

While the recent literature often focuses on catchy topics and future trends, such as drive-by-downloads and black-hat SEO, this is just the tip of the iceberg. In fact, there is a wide variety of malicious activities performed on the Internet on a daily basis, with goals that are often different from those of the high-profile cyber criminals who attract the media and the security firms' attention.

The main reason for the lack of previous works in this direction of research is that almost all existing projects (further details will be provided in the next section) of web honeypots use fake applications. With this kind of approach no real attacks can be actually preformed and all steps commonly performed by the attacker after the vulnerability exploitation are missing.

Other than academic works, security companies often relied on informations provided by clients in order to better understand the motivation of the various classes of attackers. For example, in a recent survey conducted by Commtouch and the StopBadware organization [1], 600 owners of compromised websites have been asked to fill a questionnaire to report what the attacker did after exploiting the website. This is obviously a very naive approach, as different owners have different understanding of how the attack affected their website, with the consequence of having partial or completely wrong informations. Furthermore, a survey is difficult to serialize and automatize in order to extract valuable informations for research purposes.

This work points to fulfill this hole: first we present a fully functional honeypot network, where applications are real and completely exploitable by attackers. Then we analyze the behavior of the attackers, analyzing the files uploaded and the most common methods and techniques used. Finally we try to understand the reasons and probable goals behind such attacks.

The web application deployed are specifically tailored in order to be attractive for the attacker interested in gaining and maintaining control over the machine after the exploitation, privileging specific vulnerabilities aimed to allow this kind of behavior, such remote file upload and administrator password reset, rather than other common vulnerabilities more focused on information gathering, like SQL injection, or user's credentials stealing, like XSS.

1.2 Related Works

A honeypot is one of the most common automated system available for security researcher to investigate real world trend and phenomena on widespread networks.

The concept of a honeypot on a network first began in 1999 when Lance Spitzner, founder of the Honeynet Project [2], published the paper "To Build a Honeypot", where he defines a honeypot as:

> A honeypot is a high interaction fake agent that simulates a production network and configured such that all activity is monitored, recorded and, in a degree, discreetly regulated.

Speaking about Internet network, we can basically classify honeypots in two different categories:

Client Honeypots, which look for malicious servers and detect exploits by actively connecting to servers, downloading and executing files;

Server Honeypots, which attract attackers by exposing one or more known vulnerable services.

Our research focused on the second category, as we wanted to investigate attackers behaviors after a web application has been compromised.

We can divide server honeypots in two subcategories, according to their behavior toward the users of the service:

Low-Interaction honeypots: as can be guessed by its name, this kind of honeypots only simulates the application without actually deploying any real service (and therefore can only observe attacks and can't really be exploited). These honeypots usually have limited capabilities but can be useful for detecting network probes or automated attack services (e.g., malicious Search engines or common dorks). Examples of these are Honeyd [5], Leurre.com [3] and SGNET [4], which are able to emulate several operating systems and services;

High-Interaction honeypots: this class of honeypots offers a fully functional environment that can be completely compromised by the attacker. Attacker's behavior, commands executed and uploaded files can be fully tracked, offering a more useful insight into its modus operandi, but with higher maintenance costs. These honeypots are usually deployed as virtual machine, allowing for a fast restore of the original state after the system has been compromised. An example of this honeypot system on ssh service can be found here [6].

The study of attacks against web applications is often done through the deployment of web honeypots. Several different approaches has been considered in the deployment of low-interaction honeypot networks, as Glastopf [10] and the DShield Web Honeypot Project [8], based on the idea of using templates and patterns in order to mimic several vulnerable web application, or the Google Hack honeypot [7], designed to attract attackers who use search engines to find vulnerable web applications. Another interesting approach has been proposed by John et al. [11] using search engines logs in order to identify malicious queries (queries aimed to list vulnerable web applications, commonly known as "dorks") and to automatically generate and deploy honeypot pages responding to the observed search criteria. This latter study in particular showed some interesting results: the authors found out that the median time for honeypot pages to be attacked after they have been crawled by a search engine spider is 12 days, and that local file disclosure vulnerabilities seem to be the most sought after by attackers, accounting to more than 40% of the malicious requests received by their honeypots. Other very common attack patterns were trying to access specific files (e.g.web application installation scripts), and looking for remote file inclusion vulnerabilities. A common characteristic of all these patterns is that they are very suitable for an automatic attack, as they only require to access some fixed paths or trying to inject precomputed data in URL query strings.

The main weakness in all these systems is their inability to disguise themselves as real websites to manual attackers. Low interaction honeypots, in fact, can collect data from crawlers and automated script, but human attackers can quickly realize that the system is a trap and not a real functional application (no images present, no text present etc.).

The only real attempt for realizing high-interaction web honeypots has been done by HIHAT [12] toolkit. However, the results from this experiment did not contain any interesting finding, as the machines run for few days only and the honeypot received only 8000 hits, mostly from benign crawlers.

Nevertheless, some work on categorizing the attackers' behavior has been done on interactive shells of high-interaction honeypots running SSH, like in the work from V. Nicomette et al[6]. Another study, performed by D. Ramsbrock et al. [13], used the very same system in order to perform a first profiling of attackers. Their results showed how attackers seem to separate phases assigning different tasks to different machines (i.e., scans and SSH bruteforce attacks are run from machines that are different from the ones used for intrusion), and that most of the attacks follow a rigid list of passages, using very similar attack methods and sequences of commands, suggesting that most attackers are actually following cookbooks that can be found on the Internet. Looking at the commands issued on these SSH honeypots this study shows how the main activities performed on the systems were checking the software configuration, and trying to install malicious software, such as a botnet scripts or a backdoor, in order to keep in contact with the victim machine after the compromisation.

Finally, our work does not only concern the profiling of attackersb́ehavior, but also the categorization of files uploaded to our honeypots. Several studies have been proposed regarding the automatic detection of similarities between source code files and binaries, especially for plagiarism detection, as in the work of Chen et al. [14] or in the work of Saebjornsen et al. [15] regarding binary executables. Over the years several other frameworks have been published for different formats (especially images and other multimedia formats), mostly with the same purpose.

The difference between the datasets used in these studies and in our case study is that we have much more variety in files. Our collection includes examples of multimedia, image, source code and binary files. Furthermore, many of the source code files use some degree of obfuscation, making plagiarism detection methods useless. On top of that we needed a method able to compare large datasets of files in a very quick way, as the clustering should be done daily against the whole past collection, while plagiarism detection is very resource and time demanding.

The problem of classifying and fingerprinting files of any type in a decent amount of time and using as less computing power as possible has, however, been studied carefully in the area of forensics. In particular, some tools based on the idea of similarity digest have been published in the last few years, like Ssdeep [16] and Sdhash [17]. These approaches have been proven to be reliable and fast with regard to the detection of similarities between files of any kind, being based on the byte-stream representation of data. We chose to follow this approach, and use the two tools mentioned before for our work.

Capitolo 2

HoneyProxy: The Honeypot Platform

2.1 General Structure

Our honeypot system is composed of a number of websites (500 in our experiments), each containing the installation of seven among the most common - and notoriously vulnerable - content management systems, and a static web site linked to 17 pre-installed web shells among the most common ones that can be found on the Internet, like c99 and priv57r.

A Generic overview of a website can be seen in figure 2.1

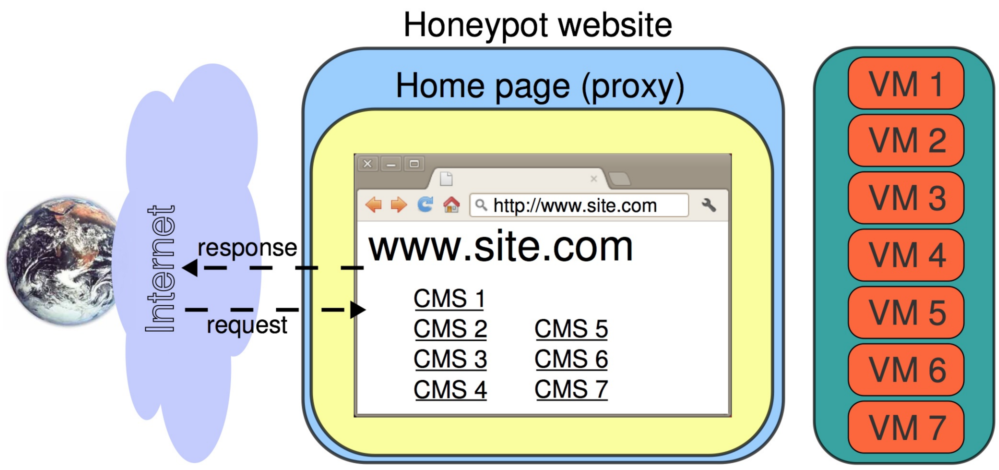


Figura 2.1: Overview of a website.

As can be seen, the system is composed by two main parts: A VMWare server, located in our facilities, hosting our web applications, and a total of 500 replicated web proxies hosted by TrendMicro distributed over eight different locations across the world, connecting the servers to the Internet. Any request reaching one of the web proxies is forwarded to our server, and the suitable response is sent back to the proxy and from there to the final user.

To make our honeypots reachable from web users, we purchased 100 bulk domain names on GoDaddy.com with privacy protection. The domains were equally distributed among the .com, .org, and .net TLDs, and assigned evenly among the locations. On each of the eight locations, we configured four additional subdomains for every domain, obtaining five distinct websites (e.g, on domain

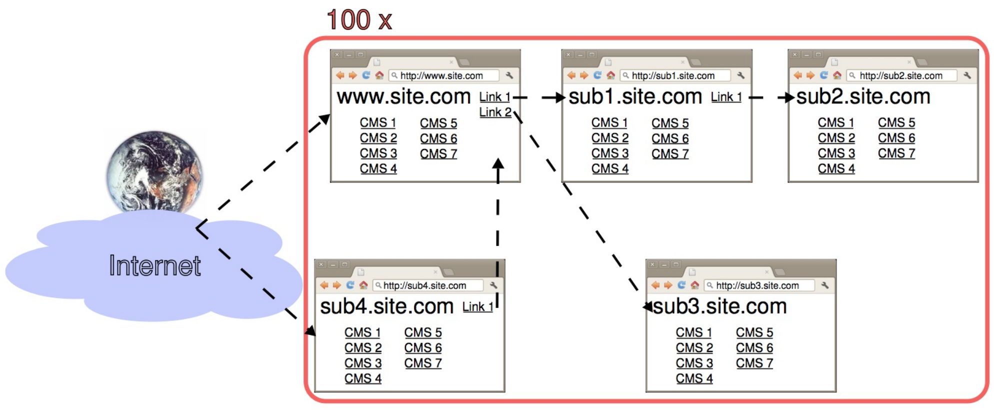


Figura 2.2: General schema of the honeypot platform.

site.com we have www.site.com, sub1.site.com, sub2.site.com, sub3.site.com and sub4.site.com). A schema of this structure is shown in fugure 2.2

Finally, we advertised the 500 domains on the home page of the authors and on the research group's website by means of transparent links. This approach, initially proposed by Müter et al. [12], consist in adding on high-reputation web pages links not visible by the user to other pages, so that the position of these latter ones in the Google index will increase.

In order to manage such a high-number of different machines in a comfortable way we used a modified version of *ftp-deploy* script to upload, in batch, every file we needed to each of the 500 websites in our possession. This greatly simplified the deployment and the update/upgrade of the system, and solved one of the main issues developers need to solve during the deployment of replicated content over different providers. Even if the hosting services can always be traced back to Trend MicroSystems, in fact, each of them has its own directory structure and specific web interface, discouraging the usage of ssh and other advanced management options in order to deploy contents. Thus the only way to easily perform this action was to use FTP protocol, the only one supported by every provider.

Looking more closer to figure 2.2, It can be noticed as the linking structure is not the same for every subdomain. Indeed, each subdomain links to at most 2 different subdomains under its same domain. The aim of this particular structure of the linking graph is to detect possible malicious traffic from systems that automatically follow links and perform automated attacks or scans.

2.2 The Web Applications

We deployed a total of eight web servers, using seven web applications and one static web site linked to 17 pre-installed webshells. Every web application is based on an openSource Content Management System among the most used across the Internet, each one based on php version 5 and using (if needed) MySQL version 5 as RDBMS. For each CMS, we chose a version with a high number of reported vulnerabilities, or at least with a critical one that would allow the attacker to take full control of the application. We also limited our choice to versions no more than 5 years old in order to ensure our websites are still of interest to attackers. Our choice was guided by the belief that attackers are always looking for low-hanging fruits. On the other hand, our honeypots will probably miss sophisticated and unconventional attacks, mostly targeted to high profile organizations or well known websites. However, these attacks are not easy to study with simple honeypot infrastructures and are therefore outside the scope of our study.

NOTE: Depending on the output, maybe it's better to create different subsubsection for each web application, or leave it as it is right now. NO I LIKE WITH DESCRIPTION.
I like the idea of putting a reference with a link to the exploit at the bottom, from exploit-db.com.

However i don't like it to have such a englight over the page, maybe removing url? leaving it like a normal text?

phpMyAdmin: phpMyAdmin [19] is an openSource tool written in PHP for the administration of a MySQL database over the WWW. The version installed is the 3.0.1.1 disguised as version 2.6.4-rc1. Both these versions present a critical PHP code injection vulnerability in the /scripts/setup.php; a page needed during the installation process of the CMS which should be manually uninstalled at the end of it, allowing an attacker to inject a payload that can execute arbitrary PHP commands. Even if both versions are vulnerable to this exploit, a research over various underground forums showed as the one for the version 2.6.4-rc1 is far more well-known with respect to the 3.0.1.1 one (even if they are identical), thus we disguised our version to the one more suitable for attackers (the 2.6.4-rc1 is no more available for downloads on phpMyAdmin website). Reference:

- `http://www.exploit-db.com/exploits/8921/`

osCommerce: osCommerce [20] is one of the most used openSource e-commerce and online store-management frameworks. Our specific version is the 2.2, which contains several critical vulnerabilities, from arbitrary file upload to modification of admin user and password by anonymous user. Reference:

- `http://www.exploit-db.com/exploits/16899/`
- `http://www.exploit-db.com/exploits/9566/`
- `http://www.exploit-db.com/exploits/10707/`
- `http://www.exploit-db.com/exploits/12801/`

Joomla!: Joomla! [21] is an openSource general-purpose framework. The version we used is the 1.5, with the addition of two plugins, tinybrowser 1.0 (a plugin for listing directories and file management) and com_graphics 1.0.16 (magnify ability to the page). This web application suffers from several vulnerabilities both server and client-side, admin password reset, remote file inclusion, local file inclusion and multiple XSS (Cross-Site scripting) cause by bad input filtering. Reference:

- `http://www.exploit-db.com/exploits/6234/`
- `http://www.exploit-db.com/exploits/16091/`
- `http://www.exploit-db.com/exploits/12430/`
- `http://www.exploit-db.com/exploits/9926/`

Wordpress: Wordpress [22] is a popular CMS aimed to the easy creation of a blog. Our version is the 2.8, with one plugin installed, kino version 1.1, a calendar/event manager, and a theme, amphion-lite. Both the plugin and the theme rely on the file timthumb.php, vulnerable from Remote File Inclusion vulnerability. This is a website which allowed to post comments, therefore we needed to take care of removing offensive/illegal content from the posts. Anyone can register to the website, and the default user role is Contributor, allowing creation of posts with links but not file uploads. Posts are sanitized from links and then become viewable to guests. For this particular machine we had to authorize sending emails because of the registration procedure, but we stop email scams by the analysis of the content of each single mail the server is willing to send.

Reference:

- `http://www.exploit-db.com/exploits/17872/`

SMF: SMF (acronym of Simple Machines Forum [23]) is an openSource software for developing a forum web application, where users can open threads and post comments. Our specific version is the 1.1.3. As in the case of the Wordpress machine, we applied the same approach for managing registration via mail and for checking posts on the forum. This application suffers from several vulnerabilities, the most important ones are a remote PHP code execution vulnerability, a stored XSS and an information-disclosure vulnerability. Reference:

- http://www.exploit-db.com/exploits/10274/

Static website: We also created a static website composed only by HTML pages with a hidden directory containing 17 different web-shells. These web-shells have been taken from underground forums and past experiments, and are fully equipped in order to perform any operation an attacker would like to perform, from shell commands execution to full database dump. They are indexed by search engine and can therefore easily be found by an appropriate dork.

Wordpress: Because of the dominant position of Wordpress CMS on the market, we deployed another web application based on this system. This specific machine run an updated version of Wordpress (3.4) with two plugins, wpstorecart and nextgengallery. Both these plugins suffer from Arbitrary File Upload vulnerability.

- http://www.exploit-db.com/exploits/19023/
- http://1337day.com/exploit/description/20352

Drupal: Drupal [24] is the third most common CMS present on the web, after Wordpress and Joomla!. We installed version 7 of this application with IMCE plugin (Image manager). This machine does not present a standard vulnerability, but rather a mis-configuration: IMCE allows for remotely upload files inside a specific directory, and its access should be denied for every user but the administrator. In this case this page is usable by every user, and it's also indexed by search engines.

Table 2.1 shows a summary of the web applications used and its vulnerabilities.

2.3 The Web Proxy

The Web Proxy is in charge of receiving the various requests from the Internet, creating a unique channel for the communication between the user and the vulnerable webserver and propagating the requests to the gateway. All web applications are hosted in our facilities, in eight isolated virtual machines running on a VMWare server. On the proxy hosting side we installed only an ad-hoc proxy tool (HoneyProxy) in charge of forwarding all the received traffic to the right virtual machine on our server.

We crafted a custom install of the Apache webserver using modified .htaccess file, ModRewrite module, cURL module and php.ini configuration files, in order to be able to transparently forward the user requests to the appropriate URL on the corresponding virtual machine. Any attempt to read a non-existing resource, or to access the proxy page itself would result in a usual 404 error page shown to the user. Not taking into account possible timing attacks or intrusions on the proxy hosting servers, there is no way for a visitor to understand that he is speaking to a proxy.

The HoneyProxy system installed on every website is composed of an index file, the PHP proxy script itself and a configuration file. The index file is the home page of the website, and it links to the vulnerable web applications and to other honeypot websites, based on the contents of the configuration file. The proxy uses sessions in order to serve the appropriate page to each user, and it also scans each response received from the web application before forwarding it in order to substitute each domain with the one initially requested by the user.

Furthermore, The PHP proxy adds two custom headers to each request it receives from a visitor:

X-Forwarded-For: this standard header, which is used in general by proxies, is set to the real IP address of the client. In case the client arrives with this header already set, the final X-Forwarded-For will list all the previous IPs seen, keeping thus track of all the proxies traversed by the client. This is necessary in case the user passes through another proxy before reaching ours.

VM #	*CMS, version*	*Plugins*	*Description*	*Vulnerabilities*
1	phpMyAdmin, 3.0.1.1	-	MySQL database manager	PHP code injection
2	osCommerce, 2.2	-	Online shop	2 remote file upload, arbitrary admin password modification
3	Joomla, 1.5.0	com_graphics, tinymce	Generic / multi-purpose portal	XSS, arbitrary admin password modification, remote file upload, local file inclusion
4	Wordpress, 2.8	kino, amphion lite theme	Blog (non moderated comments)	Remote file include, admin password reset
5	Simple Machines Forum (SMF), 1.1.3	-	Forum (non moderated posts)	HTML injection in posts, stored XSS, blind SQL injection, local file include
6	PHP web shells, static site	-	Static site and 17 PHP shells (reachable through hidden links)	PHP shells allow to run any kind of commands on the host
7	Wordpress, 3.4	wpstorecart, nextgengallery	Blog	Arbitrary File Upload
8	Drupal, 7.0	IMCE	Generic / multi-purpose portal	Remote File Upload

Tabella 2.1: Summary of the web applications deployed on the system.

X-Server-Path: this custom header is set by our PHP proxy in order to make it possible, for us, to understand the domain of provenance of the request when analyzing the request logs on the virtual machines. An example of such an entry is: X-Server-Path: http://sub1.site.com/. This entry will be used to substitute all references to other resources inside each page before sending it to the user.

These two headers are transmitted only between the hosting webserver and the honeypot VM's webserver, and thus they are not visible to the users of the HoneyProxy.

2.4 The Gateway

The Gateway is in charge of sorting the requests coming from the proxy to the appropriate webserver. This is the only communication channel between the vulnerable web applications and the outside world, and it's therefore a critical point for the infrastructure. This machine runs a very basic Debian version, with a complex routing table and a complete set of firewall rules based on iptables. We provided this machine with two connections to the Internet: the main one, where the requests to/from the honeypots are going through, which is based on a point-to-point VPN connection to the proxy, and a secondary line which is directly connected to the outside world via a DSL connection.

The general approach on which the gateway relies in order to solve connections is the following:

Requests coming from the proxy: this requests are coming from the point-to-point VPN connection. The proxy already changed the request in such a way that the destination port will be specific to the web application the request is directed to. The proxy simply receives the request, changes the destination port to the usual 80 and sends the request to the appropriate web application.

Request coming from a web application: when the gateway receives a request coming from one of the webservers, it tries to understand its nature. If it's a HTTP response, it will forward it to the Proxy as a normal gateway, if it's something else (a beginning SSH connection, a raw IP packet or something else) it immediately drop the request and dump it on a log which will be analyzed daily. There are two exceptions to this behavior, explained in the following points.

DNS requests coming from a web application: DNS requests represent an exception to the normal behavior. This kind of requests are forwarded to the Internet by the gateway using the secondary line. This behavior allows for the request to be solved without passing through the web proxy. The Proxy, in fact, is not hosted in our facilities and some malicious DNS could be blocked by the hosting provider. In this way we are sure that every DNS request is solved independently from the target of the request.

HTTP requests coming from a web application: this represents the second class of exceptions to the normal behavior of the Gateway. In this case, in fact, we want to log the content of the HTTP request, which would be impossible if we instantaneously drop the first packet received. The natural sequence of actions performed during such an event is, in fact:

- an initial TCP SYN packet from the client to the server;
- the server will answer with a TCP SYN/ACK packet;
- the client will send a TCP ACK packet and the actual HTTP request;

Because we are interested in the last one of these packets, we used an extension of iptables called "conntrack" [25] which allows for tracking the entire connection and stop (and log) it when the first HTTP request is performed.

2.5 The VMWare Server

The VMWare server is the container of our web applications. Each web application is encapsulated in a different virtual machine hosted on this server. This allows for a fast recovery of any web application after a successful attack; this recovery is performed daily. Our aim is to provide a properly safe environment for each web application, guaranteeing not only the undetectability of the presence of the virtual machine (the attacker should think to face a normal web server, not a honeypot), but also disallowing any action that can harm any other system outside our honeypots (e.g., in case a DoS script toward another server is run from one of our machines). Furthermore, we needed to prevent our honeypots to host any malicious or illegal content that could be dangerous for any client visiting the webpage. We studied every possible threat that could endanger our honeypot system, and we tackle each of these problems separately.

Gaining high privileges on the machine

We tackled this problem by using a double protection. First, all our web application run in a completely updated VMWare virtual machine, which is considered pretty safe (the last known escaping vulnerability is CVE-2008-0923 by Core Security Technologies [18]). On this virtual machine there is a further protection, as the vulnerable processes run inside an LXC container. This technology provides an operating system-level virtualization that has its own process and network space. Basically we are building multiple cages one inside the other in order to guarantee that no possible attacks can reach our real machine. Inside this container, we installed only basic linux tools (in particular, they are python, perl, wget, cURL, build-essentials) and the two services needed for the web application to work, apache and mysql. This approach does not guarantee absolute protection, but in order to reach the real servers an attacker should need at least one 0-day exploit for apache, one for LXC and one for VMWare, which is obviously a very remote possibility.

Using the honeypot machine as a stepping stone to launch attacks or email campaigns

This is probably the most important threat a high-interaction honeypot system should face when it's been deployed on the Internet. We wanted to trick the attacker into feeling that his system works, but without actually doing any real action instructed by the malicious user. Outgoing connections starting from our machine are blocked by a specific set of *iptables* rules. Furthermore, the connMark plugin for iptables allowed us to fool the attacker into thinking the machine can actually connect to the outside. As mentioned before, we allow the first and second packet in the TCP three-way handshake , but we disallow the final third packet. This provokes most of the tools used to establish a TCP connection (wget, cURL, netCat) to display a *"Connection Established"* message, tricking the attacker into thinking that the outside server is suffering from connectivity problems. Furthermore, this allowed us to collect not only data relative to the IP of the target server, but also specific queries, as specific web addresses of malicious scripts. We also set a limit to the maximum number of connections that can be established in this way between one of our servers and an outside world in order to prevent our servers to be used as a SYN flood Denial Of Service attack starting platform.

Hosting and distributing illegal content(e.g., phishing pages)

It is virtually impossible to completely disable this threat if our web applications have remote file upload vulnerability. An attacker could always upload a phishing pack and then advertise the link to it using another server, fooling the victims into the malicious page. In any case, we mitigate this risk by limiting the directories where files can be uploaded and preventing any modification to existing files on the machine, so that attackers can't, for example, replace the index page of our web servers. As a precautionary measure, we also monitor every change happened in the LXC container file system and in the VM. If such a modification is detected, the system takes a snapshot of the VM. Each VM is restored to its original snapshot at regular intervals, preventing potentially harmful content from being delivered to victims or indexed by search engines.

Illegal promotion of goods or services (e.g., spam links)

Some of our applications allowed for some degree of interaction with users from the Internet, like insert comments to blog entries or publish posts in case of the forum. These kind of applications are heavily targeted by spammers, and we wanted to be sure that links and posts published by malicious users would not reach regular users or be indexed by search engines. Luckily for us this activity is almost fully accomplished by automatic scripts, therefore we were able to directly modify the source code of the interested web application, commenting out each snippet of code responsible of showing any spam link or related content. In this way, attackers can still publish content (and we log every single post they make), but a manual check of the webpage would show blank message instead of the malicious' payloads.

Capitolo 3

Data collection, analysis and visualization

3.1 Introduction

We developed a specific set of softwares in charge of the collection and analysis of data received from the honeypot system. We basically have two kinds of data collected:

Requests: Once the attacker exploits a vulnerability on a honeypot, he will probably try to download files from other servers on the Internet or perform other types of requests to some external sources. We collect most of these data through the Gateway. The only exception is represented by HTTPS connections: this kind of connection won't show up on the log at the gateway level, as the SSL handshake can't be completed (the connection is dropped before). We inserted a backdoor inside the OpenSSL library, precisely inside the SSL write function, in order to log in clear text the content of the request.

Files: We deployed most of the vulnerable web servers with an Arbitrary Upload File vulnerability, because we are interested in collecting files attackers daily use during their hacking activity. Other than original files uploaded, we also want to detect if the same file is used by different attackers, and track any modification happened on the file while being used from different people.

The whole software is completely written in Python, using a MySQL database to store informations.

3.2 Acquisition

The acquisition of data is a pivotal part of the architecture: as mentioned before, requests are collected via live-logging using the log chain of iptables, while HTTPS requests are directly logged by our backdoor; regarding uploaded files, we use two different systems of acquisition:

Original files are collected by a direct analysis of the requests: because of the fact we know the vulnerability, we can easily scan all HTTP requests looking for specific patterns that are used during the exploitation phase: by looking at the content of these requests, we can build the original content of the uploaded file;

Modified files are collected at constant interval: every five minutes our system takes a snapshot of the files present on the web server, compares this snapshot with the “basic” one (the snapshot that will be used for the reboot of the machine every day), takes all files that are not present in the latter one and compares them to the one saved before during the same day: if any new file show up, or if an old file has been modified, the new version of the file is saved;

This system allows us to collect every file uploaded and most of the modifications happened on them during the day. Other than saving these files in a specific folder related to the day of acquisition and the web application they were uploaded to, we also save their md5 inside a specific database. In case that file was already present in our system, it will be immediately deleted, in order to optimize storage space.

3.3 Analysis

We perform different analysis both over the uploaded files and the requests, like deobfuscation, categorization and cross-checking with other databases.

3.3.1 Analysis of Requests

Even if we collect and store in a database all kinds of requests we receive, we decided to analyze in more details only HTTP/HTTPS requests for the moment, further analysis will may be performed in the future. First of all, we collect the complete URLs from the HTTP/HTTPS requests, and we integrate these data with a list of URLs we extract directly from the uploaded files. We filter this list from a range of blacklisted domains, like youtube, facebook and image hosting websites. We send this list of urls to another machine, which will perform a GET request for all these URLs by using a DSL connection. In this way, even if the webserver we are connecting to is owned by an attacker who is checking his logs, there is no possibility for him to discover our presence. All the files collected in this way are added to the group of files collected during the day, ready for being analyzed.

3.3.2 Analysis of Files

Our aim is to categorize files according to their similarity, in order to identify not only its general category (web Shell, bruteforce script etc) but also it's subtype/family. This objective can be accomplished only by examining every single file and making it suitable for a clusterization. The analysis is performed through various steps, each of them will be explained in details.

Type Categorization

This categorization is related to the intrinsic type of the file, like PHP script, HTML document etc. We used the libMagic [26] library, the same used by the *file* Unix command to identify the filetype of an entity, with a particular improvement: many attackers, in fact, disguise their attackers script by tampering the signature responsible for the recognition of the type, adding to the normal one another one for making them appear as images, in order to trick basic automatic filetype checkers present on CMSs. In order to comply with this behavior, we determine the type of a certain file once, and if it's categorized as Image, we repeat the same operation by removing the first 16 bytes of the element. In this way, if any other signature has been added, we can retrieve the correct filetype, and in the opposite case we are simply getting as output "data".

If the file is categorized as Image, Audio or Video it is immediately removed by the system in order to save storage space.

Deobfuscation

Deobfuscation of PHP scripts is a fundamental part of the analysis process: more than 70% of the PHP scripts we receive are in fact obfuscated in several ways, from a simple hex encoding in order to fool eventual Web Application Firewall up to complex obfuscating systems provided by specific softwares, like php-crypt [27]. For very easy obfuscation systems, like hex-encoding of the ascii characters or url-encoding of them, we can easily obtain the deobfuscated version by using the decode feature of string library present in Python. However, most of the obfuscation scripts work by performing several operations on a string and then evaluating it in order to run some code. We created *Transformer*, a software aimed for the secure deobfuscation of PHP files to handle this situation.

In PHP (up to version 5), we basically have two different ways for evaluating a script:

- *eval* command, which takes a string as input, and do a direct evaluation of the script (example: `eval('echo "Hello World";');`);
- *preg_replace* command, when used with “e” inside the pattern input as PREG modifier. In this case, the command looks for a certain pattern, replaces it with the replacement input, and then automatically evaluates the resulting output
(example: `preg_replace("/.*/e",'print("Hello World")',"");`)

In order to bypass WAFs (Web Application Firewalls) and IDS (Intrusion Detection Systems), which are signature-based softwares, attackers tend to obfuscate their code, which will be deobfuscated and evaluated at runtime using one of these two functions. We also noticed, during our analyses, that is not uncommon to find, inside PHP files, a small base64-encoded string, which is evaluated at runtime. The code, deobfuscated, reveals in most of the cases to be a *sendmail* function directed to the creator of the file (who is often different from the person who is actually using it), in order to notify him about a new target uploaded.

There are several ways for deobfuscating a PHP file, from a web service [28] to the *EvalHook* PHP extension developed by Stefan Esser [29]. All of them revealed some problems for being used in our case:

- we need to deobfuscate a huge amount of files per day, and the web service can't cope with this requirement;
- evalhook library works by actually executing the non-deobfuscated code, an action we want to avoid in order not to cause any harm to our system.

Our software, *Transformer*, is able to hook eval-related calls found inside the source-code, go backward from the position of the call looking for possible transformations of the input parameters, create a new PHP file containing the strictly necessary for the deobfuscation and finally deobfuscate the code, replacing the call with the deobfuscated result inside the original file. This operation is performed recursively, in order to cope with several layers of obfuscation, everything without any need for user interactions. It must be noticed that we are not performing a global parsing of a PHP file, because the language itself cause this operation to be really challenging and time/performance consuming.

By using this approach we reached fairly high success rates (on our experiments, more than 80%), with the advantage of being completely safe from executing harmful code. We enforce our security by running this code in a Virtual Environment, which is restored on a daily basis.

Normalization

Our aim is to clusterize files based on their similarity, and one of the most important operations for improving our classification system has been to perform a normalization of them. Because of the different nature of the files received, from source code to HTML pages and executables, we developed several rules for removing “user-specific” parts according to the type of file currently analyzed. For example, when dealing with PHP source code files we are using the following set of rules:

- removal of extra whitespaces, normalizing every line for having single whitespaces and single new-line character (“\n”);
- removal of empty PHP tags resulting from deobfuscation (sequences of “¡?php ?¿” or “¡? ?¿”);
- removal of C-style comments, both in single line version (using “//”) and multiline (“/*...*/”);

Because most of these files are dealing with HTML tags, we also remove any sign of user-specific tags or graphical tags inside the code, removing “¡style¿” contents, “¡meta¿” ones and HTML comments.

Last, we remove any sign of user-specific content, as phone numbers, e-mails and urls present inside the file. The urls will be saved for further analysis (cfr. 2.3.1).

3.3.3 Clustering

What we obtain from *Transformer* output is a normalized and anonymized file, which is ready to be clustered by our system. We use three different systems for clustering files: sdhash, ssdeep, md5.

Sdhash

Sdhash is our first resource for computing similarity between files. Based on the working of Vassil Roussev [17], this tool produce a digest for each file. In details, it tries to find from every file the features (64-byte sequences) that have the lowest empirical probability of being encountered by chance. Each of the selected features is hashed and placed into a Bloom filter (a probabilistic set representation). When a filter reaches its capacity, a new filter is created until all the features are accommodated. Thus, a similarity digest consists of a sequence of Bloom filters and its length is about 2-3% of the length of the input. Bloom filters have predictable probabilistic properties, which allow for two filter to be directly compared using a Hamming distance-based measure. The result gives an estimate of the fraction of features that the two filters have in common that are not due digital investigation. To compare two digests, for each of the filters in the first digest, the maximum match among the filters of the second is found. The resulting matches are then averaged.

This technique have been proved as extremely successful for text-based files, and based on the results from Roussev (ref 3.1) we chose a threshold value of 70 for considering two files as similar.

The only disadvantages of sdhash computation are the fact that the resulting hash size is directly proportional to the size of the file, which can be a problem when storing the hash (as we do), and the minimum file size requested for sdhash to work, which is 4096 B, a value that can't be satisfied when dealing with small configuration files uploaded (e.g., a .htaccess file).

Ssdeep

Ssdeep is one of the best-known techniques for detecting similar files. It's been invented by Jesse Kornblum [16] in 2006 and based on the work on spam detection performed by Andrew Tridgell [30]. The system is based on the concept of piecewise hashing. A piecewise hashing uses an arbitrary hashing algorithm (in ssdeep case is FNV) to create many checksums for a file instead of just one. Rather than generating a single hash for the entire file, a hash is generated for many

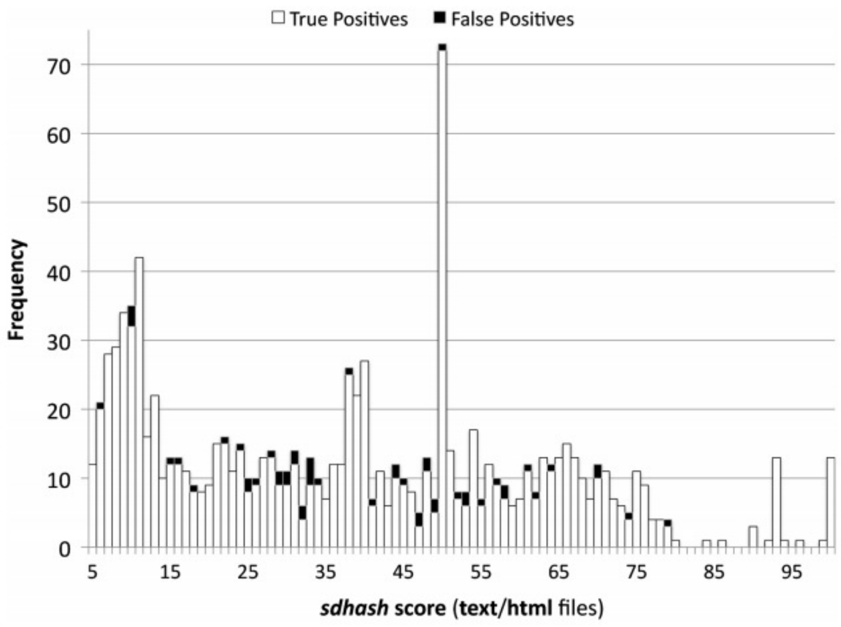


Figura 3.1: Results of sdhash similarity score with HTML files.

discrete fixed-size segments of the file. Once we obtained all these pieces, ssdeep uses a rolling hash algorithm for creating a Context Triggered Piecewise Hash (CTPH). Such hashes can be compared in order to identify ordered homologous sequences between files.

This approach produces worse results than sdhash (ref 3.2), but it has the advantage of obtaining a fixed-size hash (which is easier to store in a database) and it accepts files of a smaller size, 2048 B.

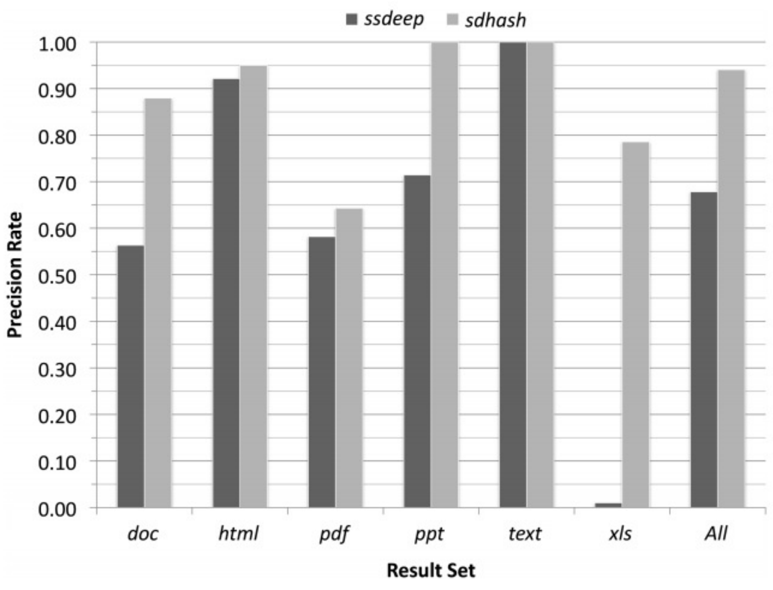


Figura 3.2: Comparison of the precision rate of ssdeep and sdhash over different file types

MD5

MD5 is a well-known technique for creating cryptographic hash of files. Designed by Ron Rivest and published by IETF as RFC 1321 [31], this technique is based on different non-linear functions computed on fixed size input in order to achieve a 128 bit hash value.

This is the last technique we use for computing similarities: whenever the file is too short to apply sdhash or ssdeep, we use this algorithm in order to find if the normalized output of the file is the same of another file, showing its similarity.

3.4 Visualization

Our aim is to create a honeypot platform which can be easily deployed, updated and monitored by the administrator of the system. In order to accomplish the last action we created a web interface for monitoring the health of the system and to obtain feedbacks from the analysis of data.

3.4.1 Back-End Architecture

We chose to use *Node.Js* as platform for developing the webserver. *Node.Js* is advertised on its website [32] as

> Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

Its particular event-driven, non-blocking model, in fact, allows us to perform database queries and file analysis operations without involving waiting time for the end-user point of view. Furthermore, its modular nature and easiness of deployment make the replication process fast and reliable.

We wanted to create a platform which does rely on a minimum number of external addition (other to the webserver) and which can perform several parallel queries to different databases for computing rankings and other operations. The system relies on *Express* [33] Framework, which is specifically aimed for minimality and flexibility, and uses only a few other modules, in the specific *mysql* driver for Database communications and *sanitizer*, a sanitizer library for strings.

Another requirement we imposed to our software is the possibility for its working in a private network, disconnected from the internet. For this reason, every library is provided directly by the webserver and there is no need to rely to any external server for the web interface to work.

3.4.2 Front-End Architecture

The Front-End architecture deeply relies on Javascript. We wanted to allow a good interactivity for the operator, who should be able to go from a high-level view to the single file analysis in a couple of clicks.

The interface uses Bootstrap [35] framework: this is one of the most used front-end frameworks for web development, as it is able to create nice and clean web pages by creating a powerful templating system. It also manage interactions with end-users through several add-ons, like modals and accordions.

Graphs displaying is performed by using D3 library [34], acronym for Data-Driven Document, a library for displaying graphs as collections of SVG elements. Thanks to its emphasis on web standards, this library will work on all modern browsers (starting from Internet Explorer 8), without any issue, and avoiding to rely on propetary frameworks for graph displaying. Other than static graphs like pie charts, this library is able to use heuristic algorithms in order to show force-directed graphs, which have been one of our minimal requirements when we decided to create the web interface.

3.4.3 Overview

Our dashboard should be able to give the following informations to the administrator: an overview over the current situation inside the system (files received, requests etc), an interactive graph showing in a clear way the various clusters, files the system failed to clusterize, stats about the system (requests, common referers, attackers provenance..). In the following I will give three examples of the functionalities of the interface.

The homepage(ref 3.3) of the web interface displays several pie charts, the most important being the number of file received during the day, divided among three categories, oldFiles (files that were already present in our databases), newFiles (new files received for which our clusterization system already categorized) and newUnknownFiles (new files that our system was unable to clusterize), and the result of our categorization for the files received during the day. Clicking on a slice of a pie displays a list of files belonging to that tranche, with the possibility to display details for each file.

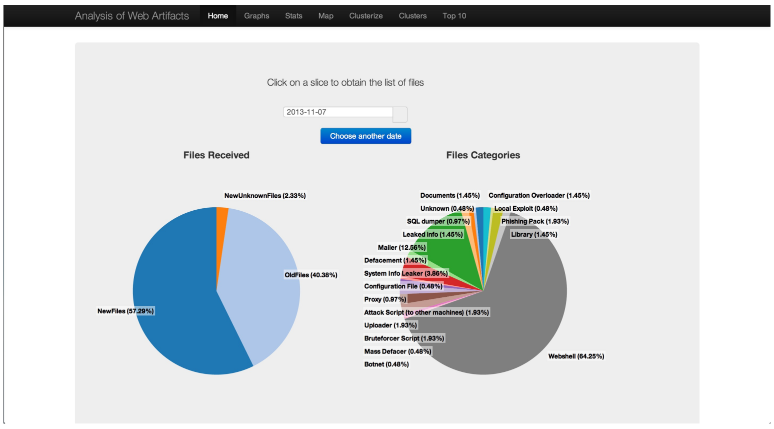


Figura 3.3: Screenshot of the web interface home page

The Graph page (ref 3.3) displays a force-directed graph showing the different clusters. It is possible to choose several parameters for the display, like the categories to be displayed and the file types. We chose to display files received only in the last 15 days with respect to the date chosen, in order not to overload the browser (the map is interactive and it's built client-side).

We can display details about a single file through the single file web page. We can see several details about the file, type, clusters, date received etc. We also have the possibility to write a comment about the file. The most interesting feature we provided on this page is the possibility to use "Transformer" over the single file, obtaining a rapid feedback over how our system managed the single file and in order to get a better understanding over the code.

NOTE: images will be displayed in a single page because there are too many, maybe remove one of them

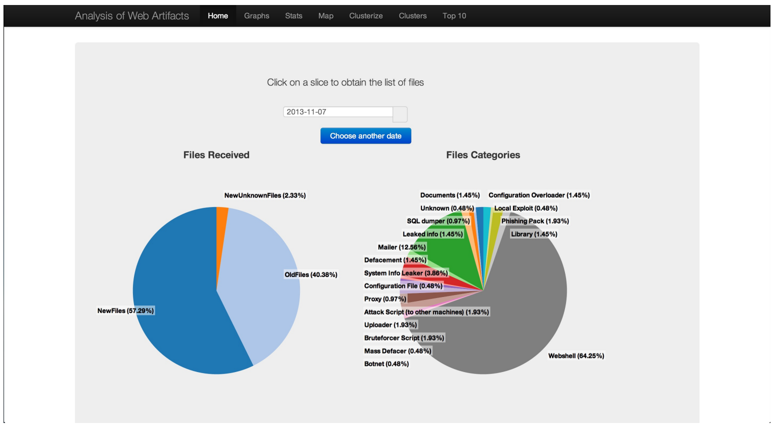
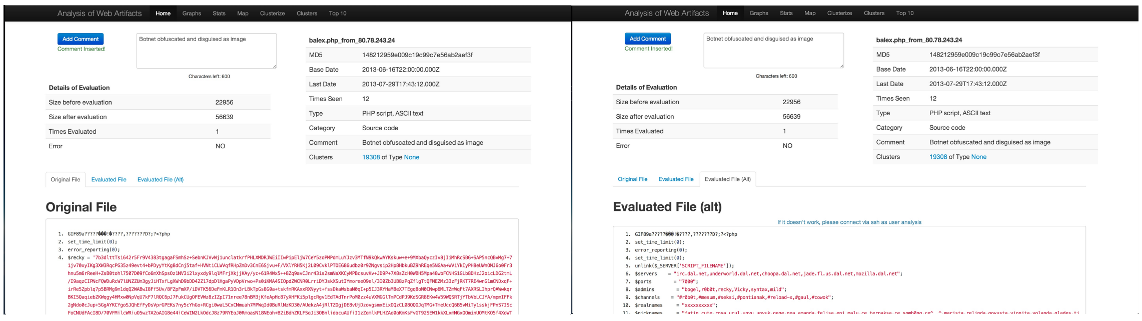


Figura 3.4: Screenshot of the web interface home page



Details of Evaluation

Size before evaluation	22956
Size after evaluation	56639
Times Evaluated	1
Error	NO

balex.php_from_80.78.243.24

MD5	148212959e009c19c99c7e56ab2aef3f
Base Date	2013-06-16T22:00:00.000Z
Last Date	2013-07-29T17:43:12.000Z
Times Seen	12
Type	PHP script, ASCII text
Category	Source code
Comment	Botnet obfuscated and disguised as image
Clusters	19308 of Type None

Original File

```
GIF89a?????���!�????,???????D?;?<?php
set_time_limit(0);
error_reporting(0);
$recky = '7b3dlttTsi642r5Fr9V4383tgagaFSmhSz+SebnKJVvWj1unclatkrfPHLXMDRJWEiIIwPipEljW7CeY5zoPMPdmLuYJzv3MTfN9kQkwAYKskuw+e+9MXbaQyczIv8jIiMhRcSBG+SAP5ncQBvMg7+7
1jv70xyIKg3XW3RqcPG35z49evt4+bPDyyYtKg8dCnj5taf+HVNtiCLWVqfRHpZmDv3CnE6Sjvu+F/VXlYRH5Kj2L09CvklPTOEG86udbz0r9ZNgvsip2Hp8HbkuBZ9hREqe5NGAa+WViYkIyPH8eUWnOMJ6o0Fr3
hnu5m6rReeH+ZsB0tohl7507D09fCo6mXhSpsOz1NV3i2lxyxdy9lqlMFrjXkjjKAy/yc+61R4Wx5++8Zq9avCJnr43is2smNaXKCyMPBcsuvKv+JD9P+7X8sZcH0WBH5Mpa48wbFCNHS1GLb8DHzJ2oicLDG2tmL
/I9aqzCIPNcFQWDuRcW7lUNZZUm3gyJiHTxfLgXWhD9bDD42Z17dpDlHgaPyVDpVrwo+Ps0iKMA4SIOpdZWCNR0LrriDYJskXSutIYmoreeO9el/3I0Zb3UB8zPqZflgTtQFMEZMz33zFjRKT7RE4wnG1mCNDxqF+
irRe5Zpblq7p5BRMg9m1dqQ2WABwI8Ff5Uv/8FZpFmXP/iDVTK56DeFmKLR1On3rLBkTpGs8G0a+tskfmRKAxxRXNyyt+fssDkaWsbaN0qI+p5IJ3RYHaM8eX7TEgq8oM0CNwp6MLTZmWqFt7AXRSLIhprQAWwkqr
BKI5QaqiebZKWqgy4HMxw0NpVqU7kF7lRQC6pJ7FukCUgOFEVWz8zIZpI71nree78nBM3jKfeApHc87yXHFKi5plgcRgv1EdTAdTnrPoM0zz4uVXMGGlTmPCdPJ9KdSGR8EKw4W59WQSRTjYTbVbLCJYA/mpmIFFk
2gNdo0cJup+5GgAYKCYgoSJQhEfFyDsVprGPEKs7ny5cYhGo+RCgi0waL5CxCNmuah7MPWqId0BuRlNzKD30/AUekzA4jRlTZOgjDE8vUjDzovgsmxEixOQzCL08QQOJqTMG+TmeUccQ6B5vMiTy1sskjFPnS7ISc
FoCNUdFAcI8D/70VFMilcWRiuO5wzTA2pAIG8e44iCeWIN2LkOdcJ8z79RYEqJ0RmqasN18NEqh+B2iBdhZKLFSqJi3OBnlidqcuAUfiI1zZqmlkPLHZAo0qKmKsFvGT92SEW1kkXLxmNGxOOminUOMtKO5f4XoWT
```

(a) Before deobfuscation

Details of Evaluation

Size before evaluation	22956
Size after evaluation	56639
Times Evaluated	1
Error	NO

balex.php_from_80.78.243.24

MD5	148212959e009c19c99c7e56ab2aef3f
Base Date	2013-06-16T22:00:00.000Z
Last Date	2013-07-29T17:43:12.000Z
Times Seen	12
Type	PHP script, ASCII text
Category	Source code
Comment	Botnet obfuscated and disguised as image
Clusters	19308 of Type None

Evaluated File (alt)

If it doesn't work, please connect via ssh as user analysis

```
GIF89a?????���!�????,???????D?;?<?php
set_time_limit(0);
error_reporting(0);
set_time_limit(0);
unlink($_SERVER['SCRIPT_FILENAME']);
$servers    = "irc.dal.net,underworld.dal.net,choopa.dal.net,jade.fl.us.dal.net,mozilla.dal.net";
$ports       = "7000";
$admins       = "bogel,r0b0t,recky,Vicky,syntax,mild";
$channels    = "#r0b0t,#mesum,#seksi,#pontianak,#reload-x,#gaul,#cowok";
$realnames     = "xxxxxxxxxx";
$nicknames     = "fatin,cute,rosa,ucul,unyu,unyuk,gege,gea,amanda,felisa,eni,malu,ce,terpaksa,ce,somb0ng,ce^,^,marista,relinda,goyusta,vionita,yolanda,glades,ti
```

(b) After deobfuscation

Figura 3.5: Example of the Single File web page

Capitolo 4

Results

4.1 Overview

The experiments run for a period of 100 days, from the August 1st 2013 to November 9th, 2013. During this period, we collected 9.5 GB of raw HTTP requests, consisting in approximately 11.0M GET and 1.9M POST. Our honeypots were visited by more than 73,000 different IP addresses, spanning 178 countries and presenting themselves with more than 11,000 distinct User-Agents. This is over one order of magnitude larger than what has been observed in the previous study by John et al. on low interaction web-application honeypots [11]. Moreover, we also extracted over 110,000 files that were uploaded or modified during attacks against our web sites, 85,000 of them are unique. There are two different ways to look at the data we collected: one is to identify and study the attacks looking at the web server logs, and the other one is to try to associate a goal to each of them by analyzing the uploaded and modified files. In this chapter we will look at the first part, while in the next chapter we will give some examples of uploaded files, in order to understand attacker's goals.

NOTE: maybe remove Overview, as it's closed in itself (we are starting with another section for obtaining nice subsections based on the different phases)

4.2 Exploitation and Post-Exploitation Behaviors

While analyzing the behavior of attackers lured by our honeypots, we identified four different phases commonly present in an attack session: discovery, reconnaissance, exploitation, and post-exploitation. The *Discovery* phase describes how attackers find their targets, e.g. by querying a search engine (using a dork) or by simply scanning IP addresses. The *Reconnaissance* phase contains information related to the way in which the pages were visited, for instance by using automated crawlers or by manual access, and if this manual access is performed via visual browser or command line tool, and if the attacker is using an anonymization proxy. In the *Exploitation* phase we describe the number and types of actual attacks performed against our web applications. Some of the attacks reach their final goal themselves (for instance by changing a page to redirect to a malicious website), while others are only uploading a second stage. In this case, the uploaded file is often a web shell that is later used by the attacker to manually log in to the compromised system and continue the attack. We refer to this later stage as the *Post-Exploitation* phase.

It must be noticed, however, that not all phases are present in every attack: some of them can be joined in one step (e.g., reconnaissance and exploitation are often performed in one single action), some of them are simply not present (e.g, post-exploitation), some visits do not lead to an actual attack (error in attackers requests, or incomplete file upload), and sometimes it is just impossible to link together different actions performed by the same attacker with different IP addresses.

Neverthless, by extracting the most common patterns from the data collected at each stage, we can identify the “typical attack profile” observed during our experiments with the following sequence:

1. 69.8% of the attacks start with a scout bot visiting the page. The scout often tries to hide its User Agent (removing directly the header) or disguise themselves as a crawler search (the most used being *GoogleBot*);

2. Few seconds after the scout has identified the page as an interesting target, a second automated system (hereinafter exploitation bot) visits the page and executes the real exploit. This is often a separate script that does not fake the user agent, therefore often appearing with strings such as *libwww/perl*.

3. If the vulnerability allows the attacker to upload a file, in 46% of the cases the exploitation bot uploads a web shell. Moreover, the majority of the attacks upload the same file multiple times (in average 9, and sometimes up to 40), probably to be sure that the attack was successful.

4. After an average of 3 hours and 26 minutes, the attacker logs into the machine using the previously uploaded shell. The average login time for an attacker interactive session is 5 minutes and 37 seconds.

While this represents the most common behavior extracted from our dataset, many other combinations were observed as well - some of which are described in the rest of the section. Finally, it is important to mention that the attack behavior may change depending on the application and on the vulnerability that is exploited. Therefore, we should say that the previous description summarizes the most common behavior of attacks against osCommerce 2.2 (the web application that received by far the largest number of attacks among our honeypots). A particular notice must be performed regarding the SMF application: this application suffered from heavy traffic, provoked by automated bots, and we preferred to exclude this application from our statistics in order to produce more reliable results. A specific discussion over the content of messages posted on the forum will be performed later. In figure 4.1 we show an overview of the four phases and its characteristics.

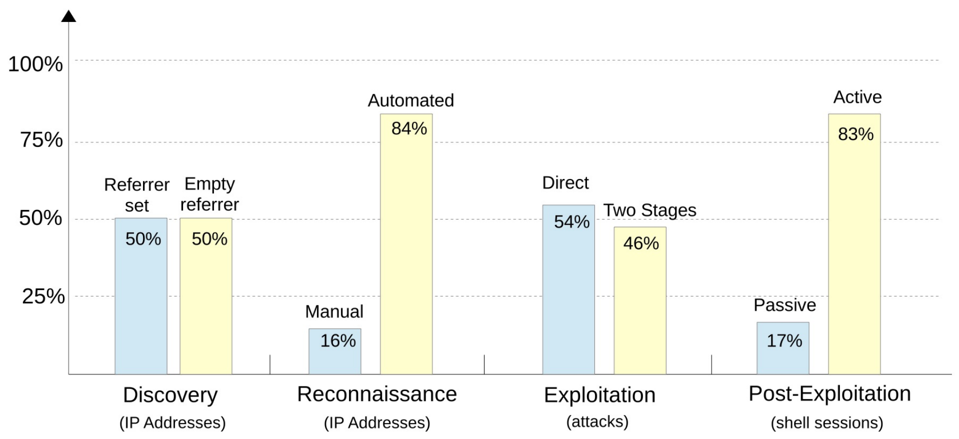


Figura 4.1: Overview of the four phases of an attack

4.2.1 Discovery

The very first HTTP request hit our HoneyProxy only 10 minutes after the deployment, from “Googlebot”. The first request not related to a benign crawler came after 1 hour and 50 minutes.

During the first few days, most of the traffic was caused by benign web crawlers. Therefore, we designed a simple solution to filter out benign crawler-generated traffic from the remaining traffic. Since HTTP headers alone are not trustable (e.g., attackers often use User Agents such as “Googlebot” in their scripts) we collected public information available on bots and we combined them with information extracted from our logs and validated with WHOIS results in order to identify crawlers from known companies. By combining User Agent strings and the IP address ranges associated to known companies, we were able to identify with certainty 14 different crawlers, originating from 1965 different IPs. Even though this is not a complete list, it was able to successfully filter out most of the traffic generated by benign crawlers.

Some statistics about the origin of the requests is shown in ADD A GRAPH, TWO LINES WITH DATES, KNOWN CRAWLER AND TOTAL REQUESTS. The amount of legitimate crawler requests is more or less stable in time, while, as time goes by and the honeypot websites get indexed by search engines and linked on hacking forums or on link farming networks, the number of requests by malicious bots or non-crawlers has an almost linear increase. While looking at these statistics we also noticed a number of suspicious spikes in the number of accesses. In several cases, one of our web applications was visited, in few hours, by several thousands of unique IP addresses (compared with an average of 192 per day), a clear indication that a botnet was used to scan our sites.

Interestingly, we observed the first suspicious activity only 2 hours and 10 minutes after the deployment of our system, when our forum web application started receiving few automated registrations. However, the first posts on the forum appeared only four days later, on December 27th. Even more surprising was the fact that the first visit from a non-crawler coincided with the first attack: 4 hours 30 minutes after the deployment of the honeypots, a browser with Polish locale visited our osCommerce web application and exploited a file upload vulnerability to upload a malicious PHP script to the honeypot. We also examined the IP source address in order to understand the more active countries, as we can see from 4.2 Russia, Ukraine and USA are in order the top three countries for number of requests. HEATMAP OR PIE CHART?.

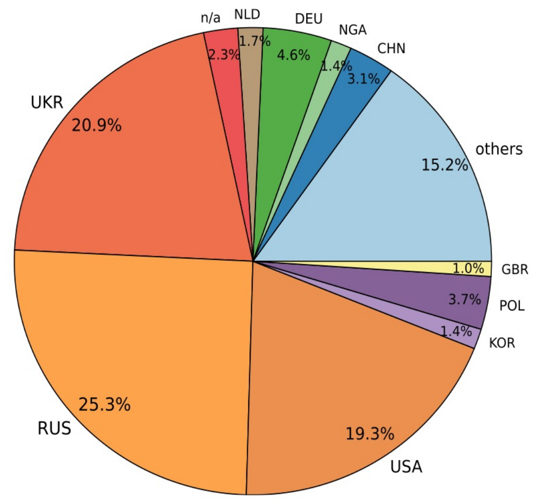


Figura 4.2: Number of requests divided by country

Capitolo 5

Risultati

Inserire in questo capitolo i risultati conseguiti, cercando di analizzarli – se possibile – in modo quantitativo.

Capitolo 6

Conclusioni

Qui si inseriscono brevi conclusioni sul lavoro svolto, senza ripetere inutilmente il sommario.

Si possono evidenziare i punti di forza e quelli di debolezza, nonché i possibili sviluppi futuri o attività da svolgere per migliorare i risultati.

Bibliografia

[1] Stop BadAware Survey for Compromised Websites http://www.stopbadware.org/files/compromised-websites-an-owners-perspective.pdf

[2] The Honeynet project, http://www.honeynet.org/

[3] F.Pouget, M.Dacier, V.H.Pham "Leurre.com: on the advantages of deploying a large scale distributed honeypot platform" ECCE 2005, E-Crime and Computer Conference, March 29-30 2005, Monaco, France pp. 29-30

[4] C.Leita, M.Dacier, "Sgnet: A worldwide deployable framework to support the analysis of malware threat models" EDCC 2008, 7th European Dependable Computing Conference, Kaunas, Lituania, May 7-9, 2008 DOI 10.1109/EDCC-7.2008.15

[5] N. Provos, "A virtual honeypot framework", USENIX Security Symposium, San Diego (California - USA), August 9-13, 2004, pp. 1-14

[6] V. Nicomette, M. Kaaniche, E. Alata, and M. Herrb, "Set-up and Deployment of a High Interaction Honeypot: Experiment and Lessons Learned", Journal in Computer Virology, vol. 7, no. 2, May 2011, pp. 143-157, DOI 10.1007/s11416-010-0144-2

[7] Google Hack Honeypot, http://ghh.sourceforge.net

[8] DShield web honeypot project, https://sites.google.com/site/webhoneypotsite/

[9] To Build A Honeypot, Lance Spitzner, http://www.spitzner.net/honeypot.html

[10] Glastopf project, http://honeynet.org/files/KYT-Glastopf-Final_v1.pdf

[11] J. P. John, F. Yu, Y. Xie, A. Krishnamurthy, M. Abadi, "Heat-seeking honeypots: design and experience", International World Wide Web Conference (WWW), New York (NY - USA), March-April 2011, pp. 207-216, DOI 10.1145/1963405.1963437

[12] M. MuÃ¼ter, F. Freiling, T. Holz, and J. Matthews, "A generic toolkit for converting web applications into high-interaction honeypots" Recent Advances in Intrusion Detection (RAID), pp. 154-170, Gold Coast (Australia), September 5-7, 2007.

[13] D.Ramsbrock, R.Berthier, and M.Cukier, "Profiling attacker behavior following ssh compromises" IEEE/IFIP International Conference on Dependable Systems and Networks, 2007. DOI 10.1109/DSN.2007.76

[14] X. Chen, B. Francia, M. Li, B. Mckinnon, and A. Seker, "Shared information and program plagiarism detection" IEEE Transactions on Information Theory, Vol. 50, No. 7, July 2004, pp. 1545-1551, DOI 10.1109/TIT.2004.830793

[15] A. Saebjornsen, J. Willcock, T. Panas, D. Quinlan, and Z. Su, "Detecting code clones in binary executables", International Symposium on Software testing and analysis, ISSTA 2009, Chicago (Illinois - USA), July 19-23, 2009, pp. 117-128, DOI 10.1145/1572272.1572287

[16] J. Kornblum, "Identifying almost identical files using context triggered piecewise hashing", Digital Investigation, Vol. 3, Supplement(0) 2006. pp. 91-97, DOI :10.1016/j.diin.2006.06.015

[17] V. Roussev, "Data fingerprinting with similarity digests" Advances in Digital Forensics VI, volume 337, Springer Boston, 2010, pp. 207-226, DOI 10.1007/978-3-642-15506-2_15

[18] VMWare escape, http://www.coresecurity.com/content/advisory-vmware

[19] phpMyAdmin reference page, http://www.phpmyadmin.net

[20] osCommerce reference page, http://www.oscommerce.com

[21] Joomla! reference page, http://www.joomla.org

[22] Wordpress reference page, http://www.wordpress.com

[23] SMF reference page, http://www.simplemachines.org

[24] Drupal reference page, http://www.drupal.org

[25] Conntrack reference page `http://www.netfilter.org/projects/libnetfilter_conntrack/index.html`
[26] LibMagic Debian package page `http://packages.debian.org/unstable/libdevel/libmagic-dev`
[27] php-crypt home page `http://www.php-crypt.com`
[28] PHP deobfuscation web service `https://www.whitefirdesign.com/tools/deobfuscate-php-hack-code.html`
[29] EvalHook PHP extension, by Stefan Esser `http://php-security.org/2010/05/13/article-decoding-a-user-space-encoded-php-script`
[30] SpamSum spam detection system, first example of context-triggered hash `https://www.samba.org/ftp/unpacked/junkcode/spamsum/`
[31] MD5 RFC 1321 `http://www.ietf.org/rfc/rfc1321.txt`
[32] Node.Js homepage `http://nodejs.org/`
[33] Express framework homepage `http://expressjs.com/`
[34] d3 homepage `http://d3js.org/`
[35] Bootstrap homepage `http://getbootstrap.com/2.3.2/`
[36] G.Cabiddu, E.Cesena, R.Sassu, D.Vernizzi, G.Ramunno, A.Lioy, "Trusted Platform Agent", IEEE Software, Vol. 28, No. 2, March-April 2011, pp. 35-41, DOI 10.1109/MS.2010.160
[37] A.Lioy, G.Ramunno, "Trusted Computing" nel libro "Handbook of Information and Communication Security" a cura di P.Stavroulakis, M.Stamp, Springer, 2010, pp. 697-717, DOI 10.1007/978-3-642-04117-4_32
[38] The OpenSSL project, `http://www.openssl.org/`
[39] T.Dierks, E.Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC-5246, August 2008
[40] Ross J. Anderson, "Security engineering", Wiley, 2008, ISBN: 978-0-470-06852-6