# Behind the Scenes of Online Attacks: an Analysis of Exploitation Behaviors on the Web

Davide Canali
*EURECOM, France*
*canali@eurecom.fr*

Davide Balzarotti
*EURECOM, France*
*balzarotti@eurecom.fr*

## Abstract

*Web attacks are nowadays one of the major threats on the Internet, and several studies have analyzed them, providing details on how they are performed and how they spread. However, no study seems to have sufficiently analyzed the typical behavior of an attacker* after *a website has been compromised.*

*This paper presents the design, implementation, and deployment of a network of 500 fully functional honeypot websites, 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 100 days of experiments, our system automatically collected, normalized, and clustered over 85,000 files that were created during approximately 6,000 attacks. Labeling the clusters allowed us to draw a general picture of the attack landscape, identifying the behavior behind each action performed both during and after the exploitation of a web application.*

## 1 Introduction

Web attacks are one of the most important sources of loss of financial and intellectual property. In the last years, such attacks have been evolving in number and sophistication, targeting governments and high profile companies, stealing valuable personal user information and causing financial losses of millions of euros. Moreover, the number of people browsing the web through computers, tablets and smartphones is constantly increasing, making web-related attacks a very appealing target for criminals.

This trend is also reflected in the topic of academic research. In fact, a quick look at the papers published in the last few years shows how a large number of them cover web-related attacks and defenses. Some of these studies focus on common vulnerabilities related to web applications, web servers, or web browsers, and on the way these components get compromised. Others dissect and analyze the internals of specific attack campaigns [13, 5, 17], or propose 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. However, there is still a missing piece in the puzzle. In fact, no academic work seems to have sufficiently detailed the behavior of an average attacker *during* and *after* a website is compromised. Sometimes the attackers are only after the information stored in the service itself, for instance when the goal is to steal user credentials through a SQL injection. But 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, 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 which no previous work was done in this direction of research is that almost all of the existing projects based on web honeypots use fake, or 'mock' applications. This means that no real attacks can be performed and thus, in the general case, that all the steps that would commonly be performed by the attacker after the exploitation will be missed.

As a result, to better understand the motivation of the various classes of attackers, antivirus companies have often relied on the information reported by their clients. For example, in a recent survey conducted by Commtouch and the StopBadware organization [7], 600 owners of compromised websites have been asked to fill a questionnaire to report what the attacker did after exploiting the website. The results are interesting, but the approach cannot be automated, it is difficult to repeat, and there is no guarantee that the users (most of the time not experts in security) were able to successfully distinguish one class of attack from the other.

In this paper we provide, for the first time, a comprehensive and aggregate study of the behavior of attackers on the web. We focus our analysis on two separate aspects: i) the exploitation phase, in which we investigate how attacks are perfomed until the point where the application is compromised, and ii) the post-exploitation phase, in which we examine what attackers do after they take control of the application. The first part deals with methods and techniques (i.e., the *"how"*) used to attack web applications, while the second part tries to infer the reasons and goals (i.e., the *"why"*) behind such attacks.

For this reason, in this paper we do not analyze common SQL injections or cross-site scripting vulnerabilities. Instead, our honeypot is tailored to attract and monitor criminals that are interested in gaining (and maintaining) control of web applications. Our results show interesting trends on the way in which the majority of such attacks are performed in the wild. For example, we identify 4 separate phases and 13 different goals that are commonly pursued by the attackers. Within the limits of the available space, we also provide some insights into a few interesting attack scenarios that we identified during the operation of our honeypots.

The remainder of the paper is organized as follows: in Section 2 we explore the current state of the art concerning web honeypots and the detection and analysis of web attacks. Section 3 describes the architecture of the honeypot network we deployed for our study; Section 4 gives more details about the deployment of the system and the way we collected data during our experiments. Finally, Section 5 and Section 6 summarize the results of our study in term of exploitation and post-exploitation behaviors. Section 7 concludes the paper and provides ideas on future directions in the field.

## 2 Related Work

Honeypots are nowadays the tool of choice to detect attacks and suspicious behaviors on the Internet. They can be classified in two categories: client honeypots, which detect exploits by actively visiting websites or executing files, and server honeypots, which attract the attackers by exposing one or more vulnerable (or apparently vulnerable) services.

In this study, we are mainly interested in the second category, since our aim is to study the behavior of attackers after a web service has been compromised. Several server-side honeypots have been proposed in the past years, allowing for the deployment of honeypots for virtually any possible service. In particular, we can distinguish two main classes: high-interaction and low-interaction honeypots. The first only *simulate* services, and thus can observe incoming attacks but cannot be really exploited. These honeypots usually have limited capabilities, but are very useful to gather information about network probes and automated attack activities. Examples of these are honeyd [21], Leurre.com [20] and SGNET [16], which are able to emulate several operating systems and services. High-interaction honeypots [19], on the other hand, present to the attacker a fully functional environment that can be exploited. This kind of honeypot is much more useful to get insights into the modus operandi of attackers, but usually comes with high setup and maintenance costs. Due to the fact that they can be exploited, high-interaction honeypots are usually deployed as virtual machines, allowing their original state to be restored after a compromise.

The study of attacks against web applications is often done through the deployment of *web* honeypots. Examples of low-interaction web honeypots are the Google Hack Honeypot [3] (designed to attract attackers that use search engines to find vulnerable web applications), Glastopf [24] and the DShield Web Honeypot project [4], all based on the idea of using templates or patterns in order to mimic several vulnerable web applications. Another interesting approach for creating low interaction web honeypots has been proposed by John et al. [14]: with the aid of search engines' logs, this system is able to identify malicious queries from attackers and automatically generate and deploy honeypot pages responding to the observed search criteria. Unfortunately, the results that can be collected by low-interaction solutions are limited to visits from crawlers and automated scripts. Any manual interaction with the system will be missed, because humans can quickly realize the system is a trap and not a real functional application. Apart from this, the study presented in [14] collected some interesting insights about automated attacks. For example, the authors found 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 heat-seeking 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 authors also proposed a setup that is similar to the one adopted in this paper, but they decided to not implement it due to the their concerns about the possibility for attackers to use infected honeypot machines as a stepping stone for other attacks. We explain how we deal with this aspect in Section 3.1.

If interested in studying the real behavior of attackers, one has to take a different approach based on high interaction honeypots. A first attempt in this direction was done by the HIHAT toolkit [18]. Unfortunately, the evaluation of the tool did not contain any interesting finding, as it was run for few days only and the honeypot received only 8000 hits, mostly from benign crawlers. To the best of our knowledge, our study is the first large scale evaluation of the post-exploitation behavior of attackers on the web.

However, some similar work has been done on categorizing the attackers' behavior on interactive shells of high-interaction honeypots running SSH [19, 23]. Some interesting findings of these studies are that attackers seem to specialize their machines for some specific tasks (i.e., scans and SSH bruteforce attacks are run from machines that are different from the ones used for intrusion), and that many of them do not act as knowledgeable users, using very similar attack methods and sequences of commands, suggesting that most attackers are actually following cookbooks that can be found on the Internet. Also, the commands issued on these SSH honeypots highlight that the main activities performed on the systems were checking the software configuration, and trying to install malicious software, such as botnet scripts. As we describe in Section 6, we also observed similar behaviors in our study.

Finally, part of our study concerns the categorization of files uploaded to our honeypots. Several papers have been published on how to detect similarities between source code files, especially for plagiarism detection [6, 26]. Other similarity frameworks have been proposed for the detection of similarities between images and other multimedia formats, mostly for the same purpose. Unfortunately, we saw a great variety of files uploaded to our honeypots, and many of them consisted in obfuscated source code (that renders most plagiarism detection methods useless), binary data or archives. Also, many of the proposed plagiarism detection tools and algorithms are very resource-demanding, and difficult to apply to large datasets. These reasons make the plagiarism detection approaches unsuitable for our needs. The problem of classifying and fingerprinting files of any type has, however, been studied in the area of forensics. In particular, some studies based on the idea of similarity digest have been published in the last few years [15, 25]. 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 proposed in [15, 25], for our work.

# 3 HoneyProxy

Our honeypot system is composed of a number of websites (500 in our experiments), each containing the installation of five among the most common - and notoriously vulnerable - content management systems, 17 pre-installed PHP web shells, and a static web site.

We mitigated the problem of managing a large number of independent installations by hosting all the web applications in our facilities, in seven isolated virtual machines running on a VMWare Server. On the hosting provider side we installed only an ad-hoc proxy script (*HoneyProxy*) in charge of forwarding all the received traffic to the right VM on our server. This allowed us to centralize the data collection while still being able to distinguish the requests from distinct hosts. A high-level overview of the system is shown in Figure 1.

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.

- *X-Server-Path*: this custom header is set by the 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/`

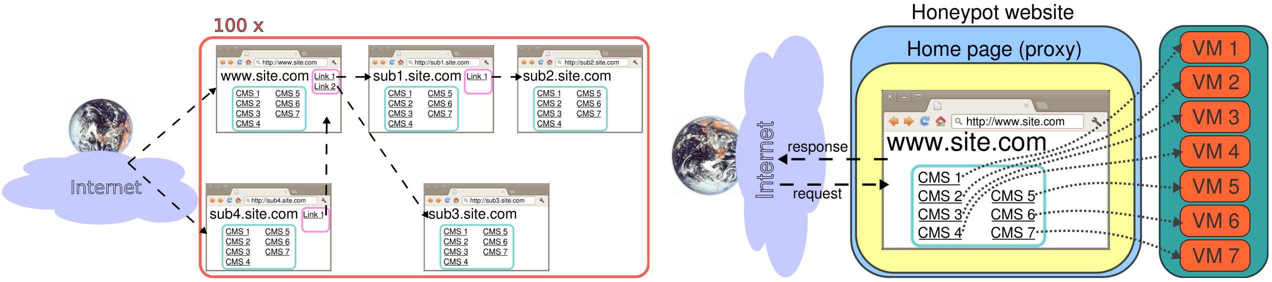
These two headers are transmitted for tracking purposes only between the hosting provider's webserver and the honeypot VM's webserver, and thus are not visible to the users of the HoneyProxy.

## 3.1 Containment

Each virtual machine was properly set up to contain the attackers and prevent them from causing any harm outside our honeypot. In particular, we blocked outgoing connections (which could otherwise result in attacks to external hosts), patched the source code of the vulnerable blog and forum applications to hide messages posted by spammers (that could result in advertising malicious links), and tuned the filesystem privileges to allow attackers to perpetrate their attacks, but not to take control of the machine or to modify the main source files of each application. Still, the danger of hosting malicious files uploaded by attackers exists, and we tackle this problem by restoring every virtual machine to its pristine state at regular time intervals.

In the following lines, we briefly explain the possible abuses that can be perpetrated on a honeypot machine and present our way to prevent or mitigate them.

- *Gaining high privileges on the machine*. We tackle this problem by using virtual machines with up-to-date software and security patches. In each virtual



(a) Architecture of the system - high level.

(b) Architecture of the system - detail.

**Figure 1. High-level architecture of the system.**

machine, the web server and all exposed services run as non privileged user. Of course, this solution does not guarantee a protection against new 0-day attacks, but we did our best to limit the attack surface, having only 3 services running on the machine (apache,sshd,mysqld), among which only the web server is exposed to the Internet. We considered the possibility of a 0-day attack against apache fairly remote, and, may it happen, a vast majority of the Internet will be exposed to it as well.

- *Using the honeypot machine as a stepping stone to launch attacks or email campaigns*. This is probably the most important concern that has to be addressed before deploying a fully functional honeypot machine. In our case, we used regular `iptables` rules to block (and log) all outgoing traffic from the virtual machines, except for already established connections. One exception to this rule is the IRC port (6667). We will explain this in more detail in sections 4 and 6.

- *Hosting and distributing illegal content(e.g., phishing pages)*. It is difficult to prevent this threat when applications have remote file upload vulnerabilities. However, it is possible to mitigate the risk of distributing illegal content by limiting the privileges of directories in which files can be uploaded and preventing the modification of all the existing HTML and PHP files. In addition, we also monitor every change on the VM file systems, and whenever a file change is detected, the system takes a snapshot of it. The virtual machine is then restored, at regular intervals, to its original snapshot, thus preventing potentially harmful content from being delivered to victims or indexed by search engines.

- *Illegally promoting goods or services (e.g., spam links)*. Another issue is raised by applications that, as part of their basic way of working, allow users to write and publish comments or posts. This is the case for any blog or forum CMS. These applications are often an easy target for spammers, as we will show in section 5.3.1, and when hosting an honeypot it is important to make sure that links and posts that are posted by bots do not reach any end user or do not get indexed by search engines. We solved this problem by modifying the source code of the blog and forum applications (namely, Wordpress and Simple Machines Forum), commenting out the snippets of code responsible of showing the content of posts. With this modification, it was still possible for attackers to post messages (and for us to collect them), but navigating the posts or comments will only show blank messages.

These countermeasures are limiting the information we can collect with our honeypot (e.g., in the case in which an attacker uploads a back-connect script that is blocked by our firewall), but we believe they are necessary to prevent our infrastructure to be misused for malicious purposes.

## 3.2 Data Collection and Analysis

Our analysis of the attackers' behavior is based on two sources of information: the logs of the incoming HTTP requests, and the files that are modified or generated by the attackers after they obtain access to the compromised machines.

We built some tools for the analysis of HTTP request logs, allowing us to identify known benign crawlers, known attacks on our web applications, as well as obtaining detailed statistics (number and type of requests received, User-Agent, IP address and geolocalization of every visitor, analysis of the 'Referer' header, and analysis of the inter-arrival time between requests). Our analysis tools also allow us to normalize the time of attack relatively to the timezone of the attacker, and to detect possible correlations between attacks (e.g., an automated script infecting a web application uploading a file, followed by another IP visiting the uploaded file from another IP address). We also developed a parser for the HTTP request logs of the most commonly used PHP web shells, allowing us to extract the requested commands and understand what the attacker was doing on our systems.

We employed two sources of uploaded or modified files: webserver logs and file snapshots from monitored directories. Webserver logs are the primary source of uploaded files, as every file upload processed by our honeypots is fully logged on the apache mod_security logs. File snapshots from monitored directories on the virtual machines, instead, are the primary source for files that are modified or generated on the machine, or about archives or encrypted files that are decompressed on the system. The total number of files we were able to extract from these sources was 85,567, of which 34,259 unique.

Given the high number of unique files we collected, a manual file analysis was practically infeasible. Therefore, in order to ease the analysis of the collected data, we first separate files according to their types, and then apply similarity clustering to see how many of them actually differ from each other in a substantial way. This allows us to identify common practices in the underground communities, such as redistributing the same attack or phishing scripts after changing the owner's name, the login credentials, or after inserting a backdoor.

First of all we employed the *file* Linux utility to categorize files and group them in 10 macro-categories: source code, picture, executable, data, archive, text, HTML document, link, multimedia, and other.

We then observed that many files in the same category only differ for few bytes (often whitespaces due to cut&paste) or to different text included in source code comments. Therefore, to improve the results of our comparison, we first pre-processed each file and transformed it to a normalized form. As part of the normalization process, we removed all double spaces, tabs and new line characters, we removed all comments (both C-style and bash-style), and we normalized new lines and stripped out email addresses appearing in the code. For HTML files, we used the *html2text* utility to strip out all HTML tags as well.

PHP files underwent an additional pre-processing step. We noticed that a large amount of PHP files that were uploaded to our honeypots as result of an exploitation were obfuscated. For files in this form it is very difficult, even with automated tools, to detect similarities among similar files encoded in different ways. In order to overcome this issue, we built an automatic PHP deobfuscation tool based on the *evalhook* PHP extension [10], a module that hooks every call to dynamic code evaluation functions, allowing for step-by-step deobfuscation of PHP code. We deployed our tool on a virtual machine with no network access (to avoid launching attacks or scans against remote machines, as some obfuscated scripts could start remote connections or attacks upon execution) and, for each file with at least one level of deobfuscation (i.e., nested call to *eval()*), we saved its deobfuscated code.

Our approach allowed us to deobfuscate almost all the PHP files that were obfuscated using regular built-in features of the language (e.g., gzip and base64 encoding and decoding, dynamic code evaluation using the *eval()* function). The only obfuscated PHP files we were not able to decode were those terminating with an error (often because of syntax errors) and those encoded with specialized commercial tools, such as Zend Optimizer or ionCube PHP Encoder. However, we observed only three samples encoded with these tools.

In total, we successfully deobfuscated 1,217 distinct files, accounting for 24% of the source code we collected. Interestingly, each file was normally encoded multiple times and required an average of 9 rounds of de-obfuscation to retrieve the original PHP code (with few samples that required a stunning 101 rounds).

**3.2.1 Similarity Clustering.** Once the normalization step was completed, we computed two similarity measures between any given couple of files in the same category, using two state-of-the-art tools for (binary data) similarity detection: *ssdeep* [15] and *sdhash* [25]. We then applied a simple agglomerative clustering algorithm to cluster all files whose similarity score was greater than 0.5 into the same group.

We discarded files for which our analysis was not able to find any similar element. For the remaining part, we performed a manual analysis to categorize each cluster according to its purpose. Since files had already been grouped by similarity, only the analysis (i.e., opening and inspecting the content) of one file per group was necessary. During this phase, we were able to define several file categories, allowing us to better understand the intentions of the attackers. Moreover, this step allowed us to gain some insights on a number of interesting attack cases, some of which are reported in the following sections as short in-depth examples.

## 4 System Deployment

The 500 honeyproxy have been deployed on shared hosting plans[1] chosen from eight of the most popular international web hosting providers on the Internet (from USA, France, Germany, and the Netherlands). In order for our HoneyProxy to work properly, each provider had to support the use of the cURL libraries through PHP, and allow outgoing connections to ports other than 80 and 443.

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 across the hosting providers. On each hosting

[1] This is usually the most economical hosting option, and consists in having a website hosted on a web server where many other websites reside and share the machine's resources.

provider, we configured 4 additional subdomains for every domain, thus having 5 distinct websites (to preserve the anonymity of our honeypot, hereinafter we will simply call them www.site.com, sub1.site.com, sub2.site.com, sub3.site.com, sub4.site.com) 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, as already proposed by Müter et al. [18] for a similar purpose.

We used a modified version of the *ftp-deploy* script [11] to upload, in batch, a customized PHP proxy to each of the 500 websites in our possession. This simplified the deployment and update of the PHP proxy, and uniformed the way in which we upload files to each hosting service[2], Thanks to a combination of `.htaccess`, ModRewrite, and cURL, we were 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 blank error page shown to the user. Not taking into account possible timing attacks or intrusions on the web hosting provider's servers, there was no way for a visitor to understand that he was talking 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 linking structure is not the same for every subdomain, as can be noticed taking a closer look at Figure 1(a). Indeed, each subdomain links to at most 2 different subdomains under its same domain. We put in place this small linking graph with the aim of detecting possible malicious traffic from systems that automatically follow links and perform automated attacks or scans.

### 4.1 Installed Web Applications

We installed a total of 5 vulnerable CMSs on 7 distinct Virtual Machines. The Content Management Systems were chosen among the most known and vulnerable ones at the time we started our deployment. 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 version 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.

Table 1 describes the vulnerable applications installed on the 7 virtual machines, along with their publication date and the list of their known and exploitable vulnerabilities. We have installed two instances of WordPress 2.8, one with CAPTCHA protection on comments, and one without CAPTCHA protection, in order to see if there are attackers that register fake accounts by hand, or systems that are capable of automatically solve CAPTCHAs. This does not seem to be the case, since we did not receive any post on the CAPTCHA-protected blog. Therefore, we will not discuss it any further in the rest of the paper.

### 4.2 Data Collection

We collected 100 days of logs on our virtual machines, starting December 23rd, 2011. All the results presented in our work derive from the analysis of the logs of these 7 machines.

Overall, 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 [14]. Moreover, we also extracted over 85,000 files that were uploaded or modified during attacks against our web sites.

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. These two views are described in more detail in the next two Sections.

## 5 Exploitation and Post-Exploitation Behaviors

In order to better analyze the behavior of attackers lured by our honeypots, we decided to divide each attack in four different phases: discovery, reconnaissance, exploitation, and post-exploitation. The *Discovery* phase describes how attackers find their targets, e.g. by querying a search engine 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 through an anonymization proxy. In the *Exploitation* phase we describe the number

[2] Shared web hosting services from different providers usually come with their own custom administrative web interface and directory structure, and very few of them offer ssh access or other 'advanced' management options. Thus, the only possible way to automate the deployment of the websites was to use FTP, the only protocol supported by every provider.

| VM # | CMS, version | Plugins | Description | Vulnerabilities |
|---|---|---|---|---|
| 1 | phpMyAdmin, 3.0.1.1 | - | MySQL database manager | PHP code injection |
| 2 | osCommerce, 2.2-RC2a | - | Online shop | 2 remote file upload, arbitrary admin password modification |
| 3 | Joomla, 1.5.0 | com_graphics, tinymce | Generic/multipurpose portal | XSS, arbitrary admin password modification, remote file upload, local file inclusion |
| 4 | Wordpress, 2.8 | kino,<br>amphion lite theme | Blog (non moderated comments) | Remote file include,<br>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 (partially working) |
| 6 | PHP web shells,<br>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, 2.8 | kino,<br>amphion lite theme | Blog (captcha-protected comments) | Remote file include,<br>admin password reset |

**Table 1. Applications installed on the honeypot virtual machines, together with a brief description and a list of their known and exploitable vulnerabilities.**

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 is hard to present all possible combinations of behaviors. Not all phases are always present in each attack (e.g., reconnaissance and exploitation can be performed in a single monolithic step), some of the visits never lead to any actual attack, and sometimes it is just impossible to link together different actions performed by the same attacker with different IP addresses. However, by extracting the most common patterns from the data collected at each stage, we can identify the "typical attack profile" observed in our experiment. Such profile can be summarized as follows:

1. 69.8% of the attacks start with a *scout bot* visiting the page. The scout often tries to hide its User Agent or disguise as a legitimate browser or search engine crawler.

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 30), 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).

Figure 2 shows a quick summary of some of the characteristics of each phase.[3] More information and statistics are reported in the rest of the section. Then, based on the analysis of the files uploaded or modified during the exploitation and post-exploitation phases, in Section 6 we will try

[3] The picture does not count the traffic towards the open forum, because its extremely large number of connections compared with other attacks would have completely dominated the statistics.

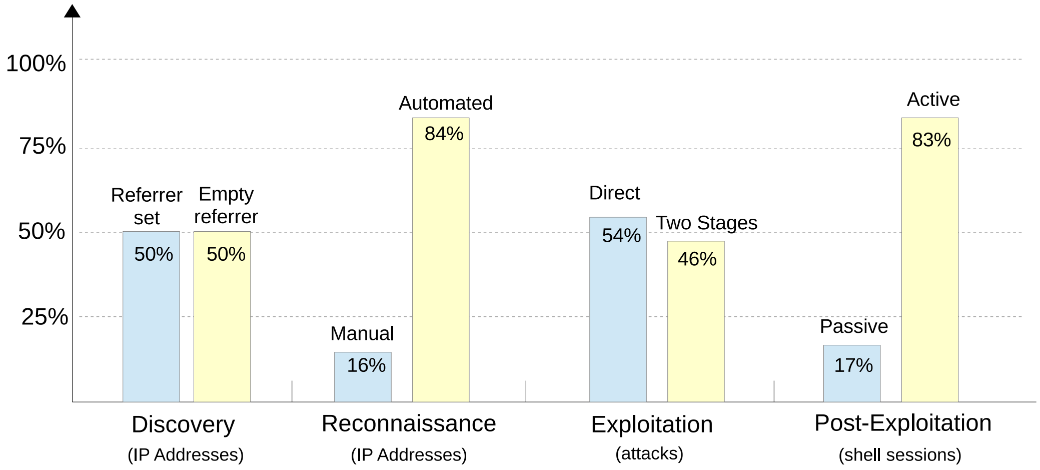


**Figure 2. Overview of the four phases of an attack**

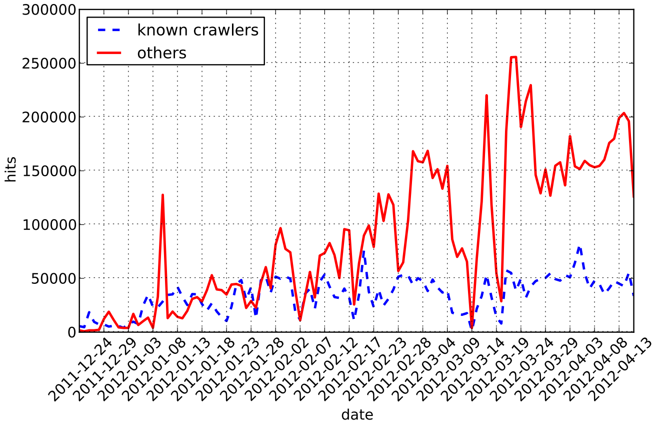


**Figure 3. Volume of HTTP requests received by out honeypots during the study.**

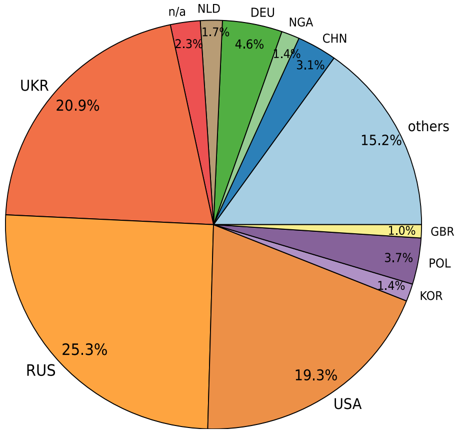


**Figure 4. Amount of requests, by issuing country.**

to summarize the different goals and motivations behind the attacks we observed in our experiments.

## 5.1 Discovery

The very first HTTP request hit our honeypot proxies only 10 minutes after the deployment, from Googlebot. The first direct request on one IP address of our virtual machines (running on port 8002) 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 [2, 1] 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 UserAgent 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 (e.g, John et al. [14] used a more complex technique to identify 16 web crawlers), 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 Figure 3. 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.

When plotting these general statistics we also identified a number of suspicious spikes in the access patterns. 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 [4] and exploited a file upload vulnerability to upload a malicious PHP script to the honeypot. Figure 4 summarizes the visits received by our honeypot (benign crawlers excluded), grouped by their geolocalization.

**5.1.1 Referer Analysis.** The analysis of the Referer HTTP header (whenever available) helped us identify how visitors were able to find our honeypots on the web. Based on the results, we can distinguish two main categories of users: criminals using search engines to find vulnerable applications, and victims of phishing attacks following links posted in emails and public forums (an example of this phenomenon is discussed in Section 6.8).

A total of 66,449 visitors reached our honeypot pages with the Referer header set. The domains that appear most frequently as referrers are search engines, followed by web mails and public forums. Google is leading with 17,156 entries. Other important search engines used by the attackers to locate our websites, were Yandex (1,016), Bing (263), and Yahoo (98). A total of 7,325 visitors arrived from web mail services (4,776 from SFR, 972 from Facebook, 944 were from Yahoo!Mail, 493 from Live.com, 407 from AOL Mail, and 108 from comcast.net). Finally, 15,746 requests originated from several public web forums, partially belonging to hacking communities, and partially just targeted by spam bots.

Finally, we extracted search queries (also known as 'dorks', when used for malicious purposes) from Referer headers set by the most common web search engines. Our analysis shows that the search terms used by attackers highly depend on the application deployed on the honeypot. For example, the most common dork that was used to reach our Joomla web application contained the words *'joomla allows you'*, while the Simple Machines Forum was often reached by searching *'powered by smf'*. Our machine containing public web shells was often reached via dorks like *'inurl:c99.php'*, *'[cyber anarchy shell]'* or even *'[ftp buteforcer] [security info] [processes] [mysql] [php-code] [encoder] [backdoor] [back-connection] [home] [enumerate] [md5-lookup] [word-lists] [milw0rm it!] [search] [self-kill] [about]'*. The latter query, even though very long, was used more than 150 times to reach our machine with web shells. It was probably preferred to searching via `'intitle:'` or `'inurl:'` because script names and titles are often customized by attackers and as such searching for their textual content may return more results than searching for fixed url patterns or page titles. Some specialized search engines appear to be used as well, such as devilfinder.com, which was adopted in 141 cases to reach some of the shells on our machines. This search engine claims to show more low-ranking results than common search engines, not to store any search data, and to return up to 300 results on the same web page, making it very suitable for attackers willing to search for dorks and collect long lists of vulnerable websites.

## 5.2 Reconnaissance

After removing the legitimate crawlers, the largest part of the traffic received by our honeypots was from unidentified sources, many of which were responsible of sending automated HTTP requests. We found these sources to be responsible for the majority of attacks and spam messages targeting our honeypots during the study.

However, distinguishing attackers that manually visited our applications from the ones that employed automated scout bots is not easy. We applied the following three rules to flag the automated requests:

- *Inter-arrival time*. If requests from the same IP address arrive at a frequency higher than a certain threshold, we consider the traffic as originated from a possible malicious bot.

- *Request of images*. Automated systems, and especially those having to optimize their speed, almost never request images or other presentation-related content from websites. Scanning web logs for visitors that never request images or CSS content is thus an easy way of spotting possible automated scanners.

- *Subdomain visit pattern*. As described in Section 4, each web site we deployed consisted in a number of sub-domains linked together according to a predetermined pattern. If the same IP accesses them in a short time frame, following our patterns, then it is likely to be an automated crawler.

[4] Since UserAgent information can be easily spoofed, we cannot prove our assumptions about the browser and tools run by the attacker, and his or her locale, are correct.

For example, after removing the benign crawlers, a total of 9.5M hits were received by systems who did not request any image, against 1.8M from system that also requested images and presentation content. On the contrary, only 641 IP addresses (responsible for 13.4K hits) visited our websites by following our links in a precise access pattern. Among them, 60% followed a breadth first approach.

85% of the automated requests were directed to our forum web application, and were responsible for registering fake user profiles and posting spam messages. Of the remaining 1.4M requests directed to the six remaining honeypot applications, 95K were mimicking the User-Agent of known search engines, and 264K switched between multiple User-Agents over time. The remaining requests did not contain any suspicious User-Agent string, did not follow paths between domains, neither requested images. As such, we classified them as unknown (possibly benign) bots.

### 5.3 Exploitation

The first important activity to do in order to detect exploitation attempts was parsing the log files in search of attack traces. Luckily, knowing already the vulnerabilities affecting our web applications allowed us to quickly and reliably scan for attacks in our logs using a set of regular expressions.

Overall, we logged 444 distinct exploitation sessions. An interesting finding is that 310 of them adopted two or more different User-Agent strings, appearing in short sequence from the same IP address. As explained in the beginning of Section 5, this often happens when attackers employ a combination of scout bots and automatic attack scripts in order to speed up attacks and quickly find new targets. In particular, in two thirds (294) of the total exploitation sessions we observed, the User-Agent used for the exploitation was the one associated to the LibWWW Perl library (`libwww/perl`).

In some of these exploitation sessions, the attacker tried to disguise her tools and browser as known benign bots. Some crawler User-Agent strings that were often used during exploitation sessions were: *FreeWebMonitoring*, *Gigabot/3.0*, *gsa-crawler*, *IlTrovatore-Setaccio/1.2*, *bingbot/2.0;*, and *Googlebot/2.1*.

The most remarkable side effect of every exploitation session is the upload or modification of files on the victim machine. Quite surprisingly, we noticed that when an exploitation session uploads a file, the file is uploaded in average 9.75 times. This strange behavior can be explained by the fact that most of the exploitation tools are automated, and since the attacker does not check in real-time whether each exploit succeeded or not, uploading the same file multiple times can increase the chance for the file to be successfully uploaded at least once.

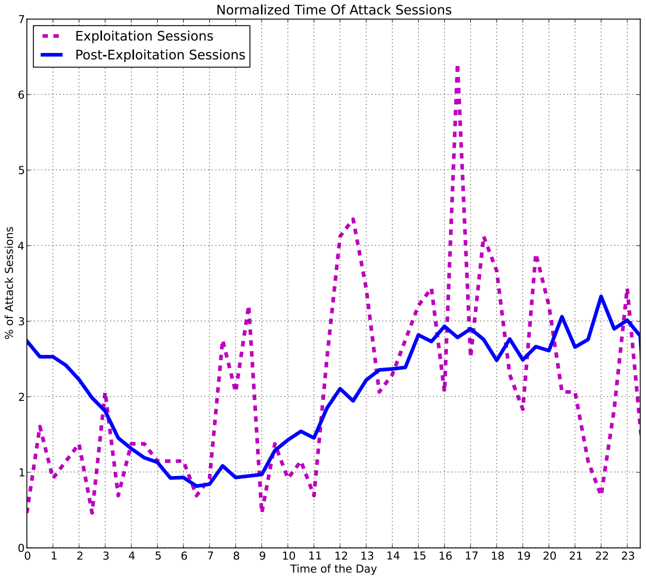


**Figure 5. Normalized times distribution for attack sessions**

Using the approach presented in Section 3.2, we automatically categorized the files uploaded to our honeypots as a result of exploiting vulnerable services. We then correlated information about each attack session with the categorization results for the collected files. Results of this phase show that the files uploaded during attack sessions consist, in 45.75% of the cases, in web shells, in 17.25% of the cases in phishing files (single HTML pages or complete phishing kits), in 1.75% of the cases in scripts that automatically try to download and execute files from remote URLs, and in 1.5% of the cases in scripts for local information gathering. Finally, 32.75% of the uploaded files were not categorized by our system, either because they were not similar to anything else that we observed, or because they were multimedia files and pictures (e.g., images or soundtracks for defacement pages) that were not relevant for our study.

Figure 5 shows the normalized times of the attacks received by our honeypots. The values were computed by adjusting the actual time of the attack with the timezone extracted from the IP geolocalization. As such, our normalization does not reflect the correct value in case the attacker is proxying its connection through an IP in a different part of the world. However, the graph shows a clear daylight trend for both the exploitation and post-exploitation phases. In particular, for the interactive sessions we observed fewer attacks performed between 4am and 10am, when probably also the criminals need to get some sleep. Interestingly, also the exploitation phase, that is mostly automated, shows a similar trend (even though not as clear). This could be the consequence of scans performed through botnet infected machines, some of which are probably turned off by their users during the night.

Searching our attack logs for information about attackers reaching directly our virtual machines, without passing through the honeypot proxies, we found that a small, but still significant number of attacks were carried out directly against the ip:port of our honeypots. In particular, we found 25 of such attack sessions against our e-commerce web honeypot and 19 against our machine hosting the web shells and the static website. In both cases, the attacker may have used a previous exploit to extract the IP of our machines (stored in a osCommerce configuration file that was often downloaded by many attackers, or by inspecting the machine through an interactive shell) and use this information in the following attacks.

**5.3.1 Posts.** Since the 1st day of operation, our forum application received a very large amount of traffic. Most of it was from automated spamming bots that kept flooding the forum with fake registrations and spam messages. We analyzed every snapshot of the machine's database in order to extract information about the forum's posts and the URLs that were embedded in each of them. This allowed us to identify and categorize several spam and link farming campaigns, as well as finding some rogue practices such as selling forum accounts.

A total of 68,201 unique messages were posted on the forum during our study, by 15,753 users using 3,144 unique IP addresses. Daily statistics on the forum show trends that are typical of medium to high traffic message boards: an average of 604 posts per day (with a max of 3085), with an average of 232 online users during peak hours (max 403).

Even more surprising than the number of posts is the number of new users registered to the forum: 1907 per day in average, and reaching a peak of 14,400 on March 23, 2012. This phenomenon was so common that 33.8% of the IP addresses that performed actions on our forum were responsible of creating at least one fake account, but never posted any message. This finding suggests there are some incentives for criminals to perform automatic user registrations, perhaps making this task even more profitable than the spamming activity itself. Our hypothesis is that, in some cases, forum accounts can be sold in bulk to other actors in the black market. We indeed found 1,260 fake accounts that were created from an IP address and then used few days later by other, different IPs, to post messages. This does not necessarily validate our hypothesis, but shows at least that forum spamming has become a complex ecosystem and it is difficult, nowadays, to find only a single actor behind a spam or link farming campaign.

A closer look at the geolocation of IP addresses responsible for registering users and posting to the forum shows that most of them are from the United States or Eastern Europe countries (mostly Russia, Ukraine, Poland, Latvia, Romania). A total of 6687 distinct IP addresses were active on our forum (that is, posted at least one message or registered one or more accounts). Among these, 36.8% were associated to locations in the US, while 24.6% came from Eastern European countries. The country coverage drastically changes if we consider only IP addresses that posted at least one message to the forum. In this case, IPs from the United States represent, alone, 62.3% of all the IP addresses responsible for posting messages (Eastern Europe IPs in this case represent 21.2% of the total).

Finally, we performed a simple categorization on all the messages posted on the forum, based on the presence of certain keywords. This allowed us to quickly identify common spam topics and campaigns. Thanks to this method, we were able to automatically categorize 63,763 messages (93.5% of the total).

The trends we extracted from message topics show clearly that the most common category is drugs (55% of the categorized messages, and showing peaks of 2000 messages per day), followed by search engine optimization (SEO) and electronics (11%), adult content (8%), health care and home safety (6%).

All the links inserted in the forum posts underwent an in-depth analysis using two automated, state-of-the-art tools for the detection of malicious web pages, namely Google Safe Browsing [22] and Wepawet [8]. The detection results of these two tools show that, on the 221,423 URLs we extracted from the forum posts, a small but not insignificant fraction (2248, roughly 1 out of 100) consisted in malicious or possibly harmful links.

## 5.4 Post-Exploitation

The post-exploitation phase includes the analysis of the interaction between the attackers and the compromised machines. In our case, this is done through the web shells installed during the exploitation phase or, to increase the collected data, through the access to the public shells that we already pre-installed in our virtual machines.

The analysis of the post-exploitation phase deserves special attention since it is made of interactive sessions in which the attackers can issue arbitrary commands. However, these web shells do not have any notion of session: they just receive commands via HTTP requests and provide the responses in a state-less fashion.

During our experiments we received a total of 74,497 shell commands. These varied from simple file system navigation commands, to file inspection and editing, up to complex tasks as uploading new files or performing network scans.

To better understand what this number represents, we decided to group together individual commands in virtual "interactive sessions" every time they are issued from the same IP, and the idle time between consecutive commands is less than 5 minutes.

According to this definition, we registered 232 interactive sessions as a consequence of one of the exploited services, and 8268 in our pre-installed shells[5]. The average session duration was of 5 minutes and 37 seconds, however, we registered 9 sessions lasting more than one hour each. The longest, in terms of commands issued to the system, was from a user in Saudi Arabia that sent 663 commands to the shell, including the manual editing of several files.

Interestingly, one of the most common actions performed by users during an attack is the upload of a custom shell, even if the attacker broke into the system using a shell that was already available on the website. The reason for this is that attackers know that, with a high probability, shells installed by others will contain backdoors and most likely leak information to their owner. In addition to the 17 web shells supported by our tools, we also identified the HTTP patterns associated to the most common custom shells uploaded by the attackers, so that we could parse the majority of commands issued to them.

In 83% of the cases, attackers tried to use at least one active command (uploading or editing a file, changing file permissions, creating files or directories, scanning hosts, killing a process, connecting to a database, sending emails, etc.). The remaining sessions were purely passive, with the attackers only browsing our system and downloading source and configuration files.

Finally, in 61% of the sessions the attackers uploaded a new file, and in 50% of them they tried to modify a file already on the machine (in 13% of the cases to perform a defacement). Regarding individual commands, the most commonly executed were the ones related to listing and reading files and directories, followed by editing files, uploading files, running commands on the system, listing the processes running on the system, and downloading files.

## 6 Attackers Goals

In this section we shift the focus from the way the attacks are performed to the motivation behind them. In other words, we try to understand what criminals do after they compromise a web application. Do they install a botnet? Do they try to gain administrator privileges on the host? Do they modify the code of the application and insert backdoors or malicious iFrames?

| **File Type** | **Clustered** | **Not Clustered** | Clusters |
|---|---|---|---|
| Archive | 335 (82.6%) | 71 (17.4%) | 159 |
| Data | 221 (62.5%) | 133 (37.5%) | 87 |
| Executable | 102 (82.3%) | 22 (17.7%) | 41 |
| HTML doc | 4341 (100.0%) | 0 (0%) | 822 |
| Image | 1703 (81.9%) | 374 (18.1%) | 811 |
| Source code | 3791 (100.0%) | 0 (0%) | 482 |
| Text | 886 (43.8%) | 1138 (56.2%) | 219 |
| Various | 118 (65.9%) | 61 (34.1%) | 42 |
| Total | 11,497 (86.5%) | 1799 (13.5%) | 2663 |

**Table 2. Results of clustering**

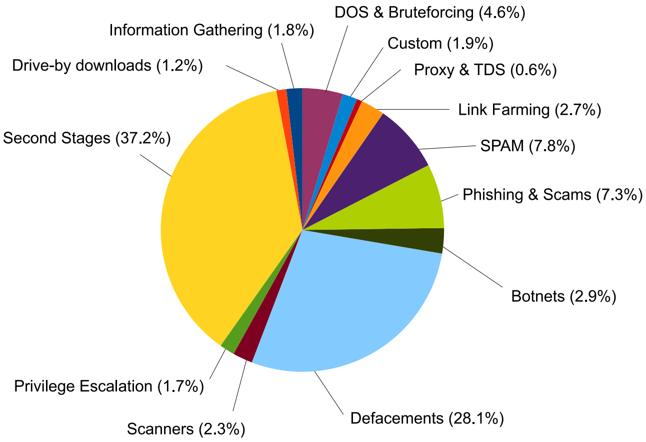


**Figure 6. Attack behavior, based on unique files uploaded**

To answer these questions, we analyzed the files uploaded during the exploitation phase, and the ones created or modified during the post-exploitation phase. We normalized each file content as explained in Section 3, and we clustered them together according to their similarity. Finally, we manually labeled each cluster, to identify the "purpose" of the files. The results of the clustering are summarized in table 2 and cover, in total, 86.4% of the unique files collected by our honeypots. For them, Figure 6 shows the distribution of the file categories[6]. For example, 1.7% of the unique files we observed in our experiments were used to try to escalate the privileges on the compromised machine. This is different from saying that 1.7% of the attackers tried to escalate the privileges of the machine. Unfortunately, linking the files to the attacks in which they were used is not always possible. Therefore, we computed an estimation of the attackers that performed a certain action by identifying each unique IP that uploaded a certain file during an

[5] For the pre-installed shells, we also removed sessions that contained very fast sequences of commands or that did not fetch images on the pages, because they could have been the result of crawlers visiting our public pages. Since shells uploaded by attackers were not linked from any page, we did not apply this filtering to them.

[6] We removed from the graph the irrelevant and damaged documents, that accounted in total for 10% of the files.

attack. Identifying an attacker only based on his or her IP address is not always correct, but still provides a reasonable approximation. Thus, if we say that a certain category has an *estimated attackers ratio* of 20%, it means that 1 attacker out of 5 uploaded at least one file of that category during his or her operation.

Only 14% of the attackers uploaded multiple files belonging at least to two separate categories. This means that most of the attacks have a precise goal, or that attackers often change their IP addresses, making it very hard for us to track them.

In the rest of the section, we briefly introduce each of the 13 categories.

### 6.1 Information gathering

| | |
|---|---|
| Unique files ratio | 1.8% |
| Estimated attackers ratio | 2.2% |

These files consist mainly in automated scripts for the analysis of the compromised system, and are often used as a first stage of a manual attack, in which the attacker tries to gather information on the attacked system before proceeding with other malicious actions. In general, we observed a number of attackers using scripts to search, archive, and download several system configuration files.

For example, an attack using such tools hit our honeypots on April 7, 2012. The attacker, using a normal browser and coming from a Malaysian IP address, uploaded a script called `allsoft.pl`. Once executed, the script scans the system for a list of directories containing configuration files of known CMSs (e.g., Wordpress, Joomla, WHM, phpBB, vBulletin, . . . ), creates a tar archive containing all the files it was able to find, and returns to the attacker a link to the created archive, that can thus be easily downloaded. The script iterates on both the users and the possible multiple home directories in the system trying to gather information from as many accounts as possible on the attacked machine.

### 6.2 Drive-by Downloads

| | |
|---|---|
| Unique files ratio | 1.2% |
| Estimated attackers ratio | 1.1% |

We have witnessed few attacks that aimed at creating drive-by download webpages, by inserting custom exploit code in the HTML source of the web pages of our honeypots, or by uploading documents that contain exploits for known browser vulnerabilities. This kind of activity is aimed at exploiting users visiting the website, typically to convert their machines in bots that can be later used for a large spectrum of illicit activity.

An example of such attacks was the *intu.html* web page uploaded to one of our honeypots on February 28th, 2012. When opened, the page shows *'Intuit Market. Loading your order, please wait...'*. Behind the scenes, a malicious javascript loads an iframe pointing to a document hosted at `twistedtarts.net`. This document is malicious and contains two exploits, for CVE-2010-0188 and CVE-2010-1885. Wepawet [8] reported the document as malicious on the same day this webpage was uploaded to our honeypots.

### 6.3 Second Stages

| | |
|---|---|
| Unique files ratio | 37.2% |
| Estimated attackers ratio | 49.4% |

This category includes downloaders (programs designed to download and execute another file), uploaders (web pages that can be used to remotely upload other files), web shells, and backdoors included in already existing documents. These are the tools of choice for attackers to perform manual web-based attacks. The reason is that such tools allow either to upload any file to the victim machine, or to issue arbitrary commands as if the attacker was logged in to one of the server's terminals. The majority of the attacks logged by our honeypot adopted a mix of web shells and custom scripts to try to hack the machine and install malicious software on it.

An example of this behavior is the attack that started at 6:50 am (GMT) on January 1st, 2012. An IP address from Englewood, Colorado, with an User-Agent set to *'blackberry8520_ver1_subvodafone'* connected directly to our honeypot virtual machine running osCommerce and exploited a file upload vulnerability, uploading several different PHP scripts, all of them launching IRC bots connecting to different IRC servers. The same person also uploaded a PHP shell, and used it to download the configuration file of the CMS installed on the machine.

The fact that the attacker was not connecting through our HoneyProxy infrastructure but directly to our IP address was unusual, and attracted our attention. Searching backwards in our logs starting the date of the attack, we found out that less than 24 hours before, an automated system with an User-Agent set to *'bingbot/2.0'* connected to one of our websites from another IP address from Englewood, Colorado, exploited a vulnerability and downloaded the osCommerce configuration file, which contains the real IP of our virtual machine hosting the e-commerce web application.

### 6.4 Privilege Escalation

| | |
|---|---|
| Unique files ratio | 1.7% |
| Estimated attackers ratio | 2.2% |

Privilege escalation exploits are among the oldest types of exploits in the computer security history, but are still among the most sought after, as they allow an attacker to gain administrator privileges and thus full control of vulnerable machines. Successfully executing a privilege escalation exploit on server machines used in a shared web hosting environment would make the attacker in the position to modify the files of every website hosted on the server, possibly allowing for mass exploitations of hundreds or even thousands of websites at the same time.

An example of such kind of attack hit our honeypots on February 9, 2012. An attacker with an Hungarian IP address uploaded a file called `mempodipper.c` to our machine hosting the web shells, and used one of the shells to try to compile its source code with *gcc*. The machine had no available compiler, thus, less than 5 minutes later, the attacker uploaded a pre-compiled ELF binary named `mempodipper`, and tried to execute it through one of the shells. We found this exploit to be for a very recent vulnerability, the CVE-2012-0056, published less than 20 days before this attack. At the time of the attack, the exploit for this vulnerability, titled *Linux Local Privilege Escalation via SUID /proc/pid/mem Write* was already publicly available [27]. However, the kernel of our virtual machines was not vulnerable to it.

## 6.5 Scanners

| | |
|---|---|
| Unique files ratio | 2.3% |
| Estimated attackers ratio | 2.8% |

This kind of activity is performed to find other local or remote vulnerable target websites that could possibly be exploited by the attacker. For example, FTP scanning, querying search engines using 'dorks', or trying to list all the domain names being hosted on the machine belong to this category.

A concrete example is the *trdomain.php* page, uploaded to one of our honeypots on December 26th, from a Turkish IP address. It contains a local domain name scanner, that pulls the domain names configured on the machine from the local configuration files (such as named.conf), gets their PageRank from Google, as well as their document root and their owner's username, and returns a web page with a list containing all this information. The title of the page is *'Domain ve User ListeLiyici — by W£ßRooT '*; as of today, searching such title on the web still yields many results, showing that this kind of attack is very common and wide spread.

## 6.6 Defacements

| | |
|---|---|
| Unique files ratio | 28.1% |
| Estimated attackers ratio | 27.7% |

Attacks of this kind are among the most frequent ones on our honeypots. In this kind of attack, the attackers modify existing web pages on the honeypot, or upload new pages with the purpose of claiming their responsibility for hacking the website. Usually, but not always, the claims are accompanied by religious or politic propaganda, or by funny or shocking images. Many of the attackers performing such attacks even insert links to their personal website or Facebook page, where one can see they are mainly teenagers looking for fame and bragging in front of their friends.

One of the many defacements attacks that hit our honeypots happened around 8 pm GMT on the 6th of March. Somebody connecting from a German IP address found one of the hidden shells in our machine hosting the static website, and used it to edit one of the static html pages hosted on the machine. The code of the page was thus uploaded using copy-and-paste in a textarea provided by the web shell. The defacement page contained a short slogan from the author, an animated javascript text slowly unveiling a Portuguese quote, and a set of links to the personal Twitter pages of each member of the hacking crew, some of which had more than 1000 tweets and several hundred followers. Quickly looking at these Twitter profiles, we found out that all the members are actively posting their defacements on their profile pages. Apparently, they do so in order to build some sort of reputation. This is confirmed by the URL they posted as a personal webpage on Twitter, a web page from the `zone-h.org` website, reporting statistics about previous defacements of the crew. The statistics are quite impressive: at the time of writing the whole crew has claimed more than 41,600 defacements starting July 20, 2011, of which almost 500 are on important websites with high reputation (governative websites, universities, multinational corporations, etc.).

Thanks to attacks like this we found out that it is common practice among attackers to advertise their defacements on publicly accessible 'defacement' showcases, such as the one on the `zone-h.org` website. It seems that some of these people are really in a sort of competition in order to show off their presumed skills at hacking websites, and our honeypot domains were often reported as trophies by several groups.

## 6.7 Botnets

| | |
|---|---|
| Unique files ratio | 28.1% |
| Estimated attackers ratio | 27.7% |

Several attackers, after exploiting our honeypots, tried to make our servers join an IRC botnet by uploading dedicated PHP or Perl scripts.

Two of the honeypot virtual machines, and specifically those with the most severe vulnerabilities, allowing attackers to upload and run arbitrary files on the server, have been set up to allow outgoing connections to port 6667 (IRC). We did so in order to monitor IRC botnet activity launched by an eventual attacker on our machines. We allowed connections only to port 6667, allowing thus only botnets running on the standard IRC port to connect to their management chat rooms. To avoid being tracked down by bot masters, every connection to the IRC port was tunneled through a privacy-protected VPN that anonymized our real IP address. No other outgoing connections were allowed from the machines, in order to avoid the possibility for our machines to launch attacks or scans against other hosts.

Our expectations proved to be correct, and we indeed logged several connections from our two machines to IRC command and control servers. The analysis of the packet traces showed some interesting information.

First of all, we were expecting IRC botnets to be quite rare nowadays, given the relatively high number of web-based exploit packs circulating on the black market. However, the analysis of the files that were uploaded on our honeypots showed an opposite trend, with about 200 distinct scripts launching IRC bots.

Another interesting observation is that, apparently, most of these IRC botnets are operated by young teenagers, as some IRC logs show. Some of the bot masters even put links to their Facebook or Twitter profiles in order to show off with their friends. Despite being run by youngsters, however, most of our connection logs show IRC rooms with hundreds to thousands of bots (the biggest IRC botnet we observed was comprised of 11900 bots).

While some logs showed us some of the bot masters attacking rivals on other IRC servers (which we considered a typical script-kiddie behavior), we were interested to see that these young people already deal with money and are able to use (and probably develop themselves) automated tools for searching on search engines and exploiting web vulnerabilities. We received a number of commands to perform DoS attacks, search engines scans using dorks, automatic mass exploitations, and instructions to report back usernames and passwords, as well as credit card credentials, stolen from exploited websites.

A final interesting finding, supported by the language used in the IRC logs and by an analysis of the IP addresses used for the upload of the IRC script, was that the majority of these IRC botnets were installed by users from South-Eastern asian countries (mostly Malaysia and Indonesia).

### 6.8 Phishing

| | |
|---|---|
| Unique files ratio | 7.3% |
| Estimated attackers ratio | 6.3% |

Phishing is one of the most dangerous activities that online criminals perform nowadays. We found proof of many attempts to install phishing pages or phishing kits on our honeypots. This kind of activity is always profit-driven; the vast majority of phishing websites are replicas of online banking websites, but we also collected few examples of online email portal phishing and even a handful of web pages mimicking ISPs and airline companies' websites.

During the 100 days of operation, our honeypots collected a total of 470 phishing-related files, 129 of which were complete phishing packages (archives often containing a full phishing website installation, including images, CSS files, and the phishing scripts themselves). Susprisingly, Nigeria seems to be a very active country for this kind of attacks, with Nigerian IP addresses responsible for approximately 45% of the phishing attacks logged by our honeypots.

An interesting case was logged by our honeypots starting on March 27th. Analyzing the Referer header of the requests received by our websites, we found 4776 requests, from 1762 different IP addresses, reaching our pages with the referer set to the mail servers of `sfr.fr`, one of the major French ISPs. Inspecting the webserver logs, we found out that all the HTTP requests having a Referer from sfr.fr requested only two png images. Both files had been uploaded to our honeypots on the 24th of March; when the first hit from SFR arrived, the virtual machines had already been cleaned up several times, but we found the original version of the pictures in our snapshots of uploaded files. Surprisingly, the pictures showed a message resembling a regular communication from SFR's customer service. All the users that hit our honeypots with a Referer from sfr.fr had thus received a phishing email containing links to the two png files, and their web client was only trying to download and show them the contents of the email.

### 6.9 Spamming and message flooding

| | |
|---|---|
| Unique files ratio | 7.8% |
| Estimated attackers ratio | 9.3% |

Many users still seem to use spam as a technique to make profit on the Internet. Some of the scripts we found are indeed mailers, i.e., scripts used to send out spam to a large number of recipients in an automated way. Some other scripts were email or SMS flooders, that are instead used for launching DoS attacks.

Our honeypots collected around 600 such scripts. As an example, on February 21st, a script called *a1.php* was uploaded from a Nigerian IP address. This script is a highly customizable mailer, and allows sending spam to a list of recipients in plain text or HTML format, with many options. It can also be configured to log in to a remote SMTP server in order to send spam through an authenticated account, and to disconnect and reconnect to the server after a certain threshold of sent emails is reached, probably with the purpose of avoiding bans.

### 6.10 Link Farming & Black Hat SEO

| | |
|---|---|
| Unique files ratio | 2.7% |
| Estimated attackers ratio | 1.0% |

Link farms are groups of web sites linking to each other, usually creating web pages with a very dense link structure, whose aim is to boost the search engine ranking of the web sites of the group. Black-hat SEO, instead, refers to using illicit or unethical techniques, such as cloaking, to boost the search engine ranking of a website, or to manipulate the way in which search engines and their spiders see and categorize a web pages. If we exclude automated posts on the forum web application, where a high percentage of posts contained links to link farming networks, this kind of behavior has not been observed very frequently on our honeypots.

An interesting attack that created a big amount of web pages on our honeypots was launched on March 19th. Somebody installed an fully functional CMS, comprising hundreds of static html pages, to one of our honeypots. All the generated pages were installed on the *images/rf/* subdirectory of our e-commerce web application, and contained russian text, along with images, CSS and JavaScript files used for presentation purposes. This page structure seems to be generated through a blog or CMS creation engine, as all the pages have a very dense link structure and point one another using absolute links (that had been customized and contained our honeypot website's domain name). We expect this to be part of an attempt to create a link farming network, or simply to be a marketing campaign for some counterfeit goods, as most of the pages we analyzed were actually advertising the sale of replica watches.

Finally, on a smaller scale, we also saw some attackers creating pages with ads or inserting links to partner sites on their uploaded pages. The reason for this is still making profit out of ads, or improving their or their partners' ranking on search engines.

### 6.11 Proxying and traffic redirection

| | |
|---|---|
| Unique files ratio | 0.6% |
| Estimated attackers ratio | 0.6% |

Online criminals always look for reliable ways to hide their tracks, and as time goes by, it becomes more and more difficult to rely only on open proxy networks, the TOR network, or open redirection web pages to conduct malicious activities. In fact, these services are often overloaded with (malicious) traffic and as such have very bad average performances and are very likely to be monitored by the authorities. In this scenario, the possibility of tunneling traffic on infected hosts seems idyllic, as it is quite easy to turn a webserver into a proxy, and often webservers running on hosting providers premises have high bandwidths, making them a very valuable target. We saw some attackers uploading proxy scripts or traffic redirection systems (TDS) to our honeypots, for the purpose of redirecting traffic anonymously (proxies) or redirecting users to malicious sources or affiliate websites (TDSs).

As an example, an archive of 504KB was uploaded on one of our honeypots on February 22, 2012. The archive contained a proxy tool called *VPSProxy*, publicly available at *http://wonted.ru/programms/vpsproxy/*; it is a PHP proxy fully controllable through a GUI client. Apparently, among all its features, if installed on more than one server, the tool makes it easy for the person using it to bounce between different connections. We believe tools like this can be very useful to criminals trying to hide their traces on the Internet.

### 6.12 Custom attacks

| | |
|---|---|
| Unique files ratio | 1.9% |
| Estimated attackers ratio | 2.6% |

This category groups all attacks that were either built on purpose for exploiting specific services, or that had no other matching category. For example, attacks in this category include programs whose aim is to scan and exploit vulnerable web services running on the server, such as the *config.php* script that was uploaded to one of our websites on April the 9th. This PHP script presents a panel for finding and attacking 9 of the most known Content Management Systems: if any of these is found on the machine, the attacker can automatically tamper with its configuration. The tool also contained other scripts to launch local and remote exploits.

### 6.13 DOS & Bruteforcing tools

| | |
|---|---|
| Unique files ratio | 4.6% |
| Estimated attackers ratio | 2.9% |

This category includes programs that launch Denial of Service or bruteforce attacks against specific applications and services (e.g., bruteforcing tools for FTP or web services, UDP and TCP flooding scripts).

An interesting example of this kind of behavior was the email bruteforce script that was uploaded to one of our honeypots on April 7, 2012. An IP address from Azerbaijan used a web shell to upload a file called *n.php* and a wordlist containing 1508 words, called *word.txt*. The *n.php* file, once executed, uses the cURL PHP libraries to connect to the `box.az` email portal and the uses the wordlist to bruteforce the password for a specific username that was hardcoded in the program. Our honeypots actually logged the upload of *n.php* several times, to three different domains. The attacker tried multiple times to execute the script (10 times in 16 minutes) and to edit it (4 times) as if looking for an error in the code. In reality, the script traffic was simply blocked by our firewall.

## 7 Conclusions

In this paper we described the implementation and deployment of a honeypot network based on a number of real, vulnerable web applications. Using the collected data, we studied the behavior of the attackers before, during, and after they compromise their targets.

The results of our study provide interesting insights on the current state of exploitation behaviors on the web. On one side, we were able to confirm known trends for certain classes of attacks, such as the prevalence of eastern European countries in comment spamming activity, and the fact that many of the scam and phishing campaigns are still operated by criminals in African countries [12]. Pharmaceutical ads appear to be the most common subject among spam and comment spamming activities, as found by other recent studies [9].

On the other hand, we were also able to observe and study a large number of manual attacks, as well as many infections aimed at turning webservers into IRC bots. This suggests that some of the threats that are often considered outdated are actually still very popular (in particular between young criminals) and are still responsible for a large fraction of the attacks against vulnerable websites.

We are currently working toward a completely automated system that can monitor the honeypot in realtime, identify and categorize each attack, and update a dashboard with the most recent trends and exploitation goals.

## 8 Acknowledgements

The research leading to these results was partially funded from the EU Seventh Framework Programme (FP7/2007-2013) under grant agreement n°257007.

## References

[1] IP Addresses of Search Engine Spiders. `http://www.iplists.com/`.

[2] Robots IP Address Ranges. `http://chceme.info/ips/`.

[3] Google Hack Honeypot. `http://ghh.sourceforge.net/`, 2005.

[4] Dshield web honeypot project. `https://sites.google.com/site/webhoneypotsite/`, 2009.

[5] J. Caballero, C. Grier, C. Kreibich, and V. Paxson. Measuring pay-per-install: The commoditization of malware distribution. In *Proceedings of the USENIX Security Symposium*, 2011.

[6] X. Chen, B. Francia, M. Li, B. Mckinnon, and A. Seker. Shared information and program plagiarism detection. *Information Theory, IEEE Transactions on*, 50(7):1545–1551, 2004.

[7] s. Commtouch. Compromised Websites: An Owner's Perspective. `http://stopbadware.org/pdfs/compromised-websites-an-owners-perspective.pdf`, february 2012.

[8] M. Cova, C. Kruegel, and G. Vigna. Detection and Analysis of Drive-by-Download Attacks and Malicious JavaScript Code. In *Proceedings of the International World Wide Web Conference (WWW)*, 2010.

[9] Cyberoam Technologies and Commtouch. Internet Threats Trend Report October 2012. `http://www.cyberoam.com/downloads/ThreatReports/Q32012InternetThreats.pdf`, october 2012.

[10] S. Esser. evalhook. `http://www.php-security.org/downloads/evalhook-0.1.tar.gz`, may 2010.

[11] M. Hofer and S. Hofer. ftp-deploy. `http://bitgarten.ch/projects/ftp-deploy/`, 2007.

[12] Imperva Inc. Imperva's Web Application Attack Report. `http://www.imperva.com/docs/HII_Web_Application_Attack_Report_Ed2.pdf`, january 2012.

[13] J. P. John, F. Yu, Y. Xie, A. Krishnamurthy, and M. Abadi. deSEO: Combating Search-Result Poisoning. In *Proceedings of the USENIX Security Symposium*, 2011.

[14] J. P. John, F. Yu, Y. Xie, A. Krishnamurthy, and M. Abadi. Heat-seeking honeypots: design and experience. In *Proceedings of the International World Wide Web Conference (WWW)*, 2011.

[15] J. Kornblum. Identifying almost identical files using context triggered piecewise hashing. *Digital Investigation*, 3, Supplement(0):91 – 97, 2006.

[16] C. Leita and M. Dacier. Sgnet: A worldwide deployable framework to support the analysis of malware threat models. In *Dependable Computing Conference, 2008. EDCC 2008. Seventh European*, may 2008.

[17] T. Moore and R. Clayton. Evil searching: Compromise and recompromise of internet hosts for phishing. In *Financial Cryptography*, pages 256–272, 2009.

[18] M. Müter, F. Freiling, T. Holz, and J. Matthews. A generic toolkit for converting web applications into high-interaction honeypots, 2007.

[19] V. Nicomette, M. Kaâniche, E. Alata, and M. Herrb. Set-up and deployment of a high-interaction honeypot: experiment and lessons learned. *Journal in Computer Virology*, june 2010.

[20] F. Pouget, M. Dacier, and V. H. Pham. V.h.: Leurre.com: on the advantages of deploying a large scale distributed honeypot platform. In *In: ECCE 2005, E-Crime and Computer Conference*, pages 29–30, 2005.

[21] N. Provos. A virtual honeypot framework. In *Proceedings of the USENIX Security Symposium*, pages 1–14, 2004.

[22] N. Provos, P. Mavrommatis, M. A. Rajab, and F. Monrose. All Your iFrames Point to Us. In *Proceedings of the USENIX Security Symposium*, 2008.

[23] D. Ramsbrock, R. Berthier, and M. Cukier. Profiling attacker behavior following ssh compromises. In *in Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2007.

[24] L. Rist, S. Vetsch, M. Koßin, and M. Mauer. Glastopf. `http://honeynet.org/files/KYT-Glastopf-Final_v1.pdf`, november 2010.

[25] V. Roussev. Data fingerprinting with similarity digests. In K.-P. Chow and S. Shenoi, editors, *Advances in Digital Forensics VI*, volume 337 of *IFIP Advances in Information and Communication Technology*, pages 207–226. Springer Boston, 2010.

[26] A. Saebjornsen, J. Willcock, T. Panas, D. Quinlan, and Z. Su. Detecting code clones in binary executables. In *Proceedings of the eighteenth international symposium on Software testing and analysis*, ISSTA '09, pages 117–128. ACM, 2009.

[27] zx2c4. Linux Local Privilege Escalation via SUID /proc/pid/mem Write. `http://blog.zx2c4.com/749`, january 2012.