2010 IEEE 21st International Symposium on Software Reliability Engineering

# The Web Attacker Perspective – A Field Study

José Fonseca
CISUC, University of Coimbra /
Polytechnic Institute of Guarda, Portugal
josefonseca@ipg.pt

Marco Vieira, Henrique Madeira
CISUC, University of Coimbra
Coimbra, Portugal
mvieira@dei.uc.pt, henrique@dei.uc.pt

***Abstract*—Web applications are a fundamental pillar of today's globalized world. Society depends and relies on them for business and daily life. However, web applications are under constant attack by hackers that exploit their vulnerabilities to access valuable assets and disrupt business. Many studies and reports on web application security problems analyze the victim's perspective by detailing the vulnerabilities publicly disclosed. In this paper we present a field study on the attacker's perspective by looking at over 300 real exploits used by hackers to attack web applications. Results show that SQL injection and Remote File Inclusion are the two most frequently used exploits and that hackers prefer easier rather than complicated attack techniques. Exploit and vulnerability data are also correlated to show that, although there are many types of vulnerabilities out there, only few are interesting enough for attackers to obtain what they want the most: *root shell access* and *admin passwords*.**

***Keywords- Security; Exploit; Vulnerability; Web application; Field study***

## I. INTRODUCTION

In less than two decades, the World Wide Web was able to radically change the way people communicate and do business. From individuals to large organizations, everyone uses the web. In fact, web applications quickly spread all over the world in the form of personal web sites, blogs, news, social networks, webmails, forums, e-commerce applications, among others. In developed countries, even critical infrastructures like water supply, power supply, banking, insurance, stock market, retail, communications, defense, etc. rely on networks, on the web and on applications that run in these distributed environments.

As the importance of the assets accessed and managed by web applications increases, so does the natural interest of malicious minds in exploiting this new streak. Frequently, web applications developed with a strong focus on functionality and usability find themselves under heavy attack by hackers and organized crime, exploiting their weaknesses and vulnerabilities [1, 2, 3, 4]. The pressure to take advantage of web application assets is huge, thus it is not a surprise to see numerous reports of successful security breaches and exploitations [5, 6, 7].

After years of uncontrolled software development processes and practices, we now face the challenge of securing millions of existing web applications and developing new ones with good security embedded. The high number of regulations put into place by governments and corporations in recent years reflects the increasing concern top managers now have about web security. However, there are some significant factors that still make securing web applications a task hard to fulfill. Some examples are the fast growing market, their high exposure to attacks and the general lack of knowledge or experience in the area of security from those who develop and manage these applications.

In spite of all security-related efforts, web applications are typically deployed with security vulnerabilities that make them vulnerable to attacks. This suggests that web developers and researchers still need to know more about vulnerabilities and attacks to mitigate them more effectively. Web application vulnerability analysis has been addressed by recent studies from several points of view [1, 4, 8, 9, 10, 11]. The attacker's perspective has also been of some focus in the literature ([12, 13, 14, 15], among others), but mainly from empirical data gathered by the authors highlighting social networking and what could be obtained from attacking specific vulnerabilities. Some studies analyze the attack from the victim's perspective, like the proposal of a taxonomy to classify attacks based on their similarities [16] and the analysis of the attack traces from *HoneyPots* to separate the attack types [17]. There is, however, a lack of knowledge about existing exploits and their correlation with the targeted vulnerabilities.

Important aspects that help understand web application attacks are what vulnerabilities are exploited, what assets hackers usually target, how these attacks are performed and the techniques actually used to execute them. This valuable data can be obtained by analyzing real attacks on web applications and the tools used to execute these attacks. In this paper we address the security of web applications focusing on the attacker's perspective. To have a broader view of the attacker panorama we analyzed over three hundred real exploits targeting vulnerabilities of six widely used LAMP (Linux, Apache, Mysql and PHP) web applications. These exploits are publicly available in a hacker related site [18] and they have been downloaded over three million times from that site by potential attackers. Some of the exploits have also been adapted as modules of the Metasploit framework, widely used for generic penetration testing and vulnerability exploitation [19].

In this field study, exploits are analyzed from various dimensions to understand what types of vulnerabilities attackers prefer, what the goals of attacks are and how they are performed. The exploit data is also compared with vulnerability data of web applications to help unveil some behaviors, like whether the most common vulnerabilities are the ones that hackers prefer to attack.

The information resulting from this study can be used in security related scenarios, to help directing security practitioners to the most common attack types, to better protect the assets and to properly configure their environment. In fact, results confirm and enforce that some well-known security measures can prevent some real devastating attacks. For example, implementing policies like giving the lowest privilege to Operating System (OS) users that own network services, using strong passwords or cease using the *register_globals = 1* PHP directive can prevent many exploits from achieving their goal. Finally, field study data can be valuable to improve security mechanisms, like the payload generator of a web application attack injector [20], the training of penetration testers and the procedures they use in the process.

The outline of this paper is as follows. Next section presents the target web applications and the field study methodology. Section III details the types of exploits found. Section IV analyses the field data and discusses the results. Section V concludes the paper.

## II. Field Study On Web Application Exploits

A vulnerability is a weakness (an internal software bug) that may be exploited to cause harm, although its presence does not cause harm by itself [21]. However, a vulnerability is a precondition for an attack (a malicious external fault) to cause an error and possibly subsequent failures [22]. The exploit is the piece of code that is used to maliciously take advantage of a given vulnerability.

A web application exploit may be as simple as a specially crafted URL or as complex as an automated program with hundreds of lines of code that can be compiled and executed. Failures may occur due to attacks performed on a weakness in the security (vulnerability) of the application, which allows a malicious user to bypass security attributes like Authentication, Integrity, Non-repudiation, Confidentiality, Availability and Authorization [12]. This malicious action allows the attacker to gain access and to tamper with inappropriate resources and assets within the web application or the server computer: unauthorized access to data like credit card numbers and passwords, use privileged database accounts, impersonate another user (such as the administrator), mimic web applications, deface web pages, obtain access to the web server as the root user, install malicious programs and backdoors, etc.

In this section we present the target web applications used in the field study and the methodology followed to analyze the exploits of their vulnerabilities.

### A. Web Applications

This field study analyses the exploits of six widely used and well-known web applications: PHP-Nuke (phpnuke.org), Drupal (drupal.org), PHP-Fusion (php-fusion.co.uk), WordPress (wordpress.org), phpMyAdmin (phpmyadmin.net) and phpBB (phpbb.com). Drupal, PHP-Fusion and phpBB are Web Content Management Systems (CMS). CMS is an application that allows an individual or a community of users to easily create and administer web sites that publish a variety of contents. PHP-Nuke is a well-known web based news automation system built as a community portal. News can be submitted by registered users and commented by the community. WordPress is a personal blog publishing platform that also supports the creation of easy to administer web sites. phpMyAdmin is a web based MySQL administration tool. It is one of the most popular PHP applications and has a huge deployment base.

The web applications considered have a large community of users and belong to a class of applications that has a large spectrum of adoption. They have also won several prizes (some are Sourceforge and Open Source CMS finalists and winners [23, 24]) and are considered among the best in their class. All these web applications are developed using LAMP (Linux, Apache, Mysql and PHP), which is a combination of the most common technologies used by web applications around the world. Linux is mostly used as the chosen OS for servers, MySQL is the world's most popular database, Apache is a leader in web servers and PHP web sites have about 1/3 of the worldwide market share [25].

These same web applications were also previously used in a field study on security vulnerabilities [8], in which the authors analyzed the classes of vulnerabilities that are the most frequent in web applications. Using the same web applications makes it possible to correlate exploit results with information about the vulnerabilities that are being exploited. This can be done at least for XSS and SQL Injection vulnerabilities, which are the two most common vulnerabilities in web applications [1] and were the focus of the field study presented in [8].

### B. Field Study Methodology

The field study consisted in the analysis of pieces of code developed to take advantage of vulnerabilities in web applications. These are the kind of exploits used by hackers and *script kiddies* to attack widely spread web applications, like the ones considered in this work. As the goal is to examine the inner workings of the exploits, for the analysis we need their source code. This may seem a big constraint, but it is quite common to find web application exploits available in their source code version. One reason for this is because they are usually developed in scripting languages, like PERL, PHP or HTML. Another reason seems to be the developers' will to raise their rank within the hacker community for their accomplishments, which may justify the presence of the source code of exploits developed using compiled languages, like C, C# or ASP.

The web application exploits were obtained from the Milw0rm web site [18]. This is a hacker related site containing around ten thousand exploits and whose contributors belong to several hacking groups. It is one of the most popular exploit databases and it is the largest that we are aware of. The site has a collection of exploits of vulnerabilities already fixed, but has also some *0 day* vulnerabilities, for which no solution is available yet. Many of the exploits present in the Milw0rm site are also distributed by other hacker and security related sites, like the RedOracle (redoracle.com), SecurityReason

(securityreason.com), SecurityFocus (securityfocus.com) and osvdb (osvdb.org). Some Milw0rm exploits are available as modules of the Metasploit framework [19], which is widely used by hackers and security practitioners for penetration testing and vulnerability detection.

In order to have a coherent data set, we gathered all the exploits (for the target web applications) from the Milw0rm database at the same time, so that no exploit was added after we started our analysis. Although we were also comparing the exploits with the vulnerabilities of the field study presented in [8], we did not restrict our search to XSS and SQL Injection, as our goal was to have a holistic view of the distribution of all types of exploits. We also did not restrict the exploit release date in order to be able to get a picture of the variation of the exploits over time.

All the exploits collected were manually analyzed in detail. This allowed us to obtain inside information about how they were built, including: the language used, the vulnerable variables attacked, the attack entry point, etc. This also helped us to confirm whether the exploit type classification referred by the Milw0rm site is correct.

By analyzing the source code we can accurately classify each exploit. In the files downloaded, we found cases in which several exploits were depicted together. These cases are usually (but not always) differentiated from the others by having the text "Multiple Vulnerabilities" somewhere in their names. When each of these exploits attacks different vulnerabilities in its own specific way (XSS, SQL Injection, etc.), we considered them as representing different exploits. On the other side, when we found in the same exploit file similar ways to exploit the same vulnerability we accounted them as a single exploit. As an example let us consider a SQL Injection vulnerability that allows the attacker to get access to a list of user names using a specific attack string. Let us also assume that this vulnerability can be exploited to obtain the list of passwords just by replacing the query's *user name* field with the *password* field in the same attack string. We accounted situations like these as a single exploit and we classified the level of damage as the most damaging (in this case, the disclosure of both user names and passwords). This decision also makes sense if we think about some complex exploits (that account as a single exploit), like blind SQL Injection, where the attack is done in several stages exploiting the same vulnerable variable in a loop over and over until all the valuable field data is finally obtained.

### C. *Field Study Remarks*

Using the guidelines presented above, we collected 312 single exploits that overall had been downloaded, at the time of this study 3,249,484 times. The quantity of exploits used is certainly a subset of all the exploits developed for the target web applications. Moreover, we are also aware that there may be some other exploits outside the Milw0rm site. However, using exploits from a single repository does not mean that they come from a restricted elite of hackers. In fact, anyone can upload an exploit to the Milw0rm site and the data analyzed came from 118 different authors.

As final remarks of our methodology, we should emphasize that we assume the number of downloads of the exploits shown in the Milw0rm site to be merely indicative. We cannot nor intend to assure that it represents the strict number of times the exploits were used, or even downloaded from the web. We know that some downloads were not used to attack any site (like we did); some attackers can download the same exploit from other sites (like those sites referred earlier); the exploits can be shared by other channels like the IRC (highly used by hackers) or email; and they can also be used in specific attack tools, like the Metasploit [19].

Despite these remarks, that are unavoidable and omnipresent in every web related study, the data used is real and the field study results are repeatable. They are not easily generalized, at least for web applications not using the LAMP technology. More data should be analyzed to have a broader knowledge of how attacks are being performed on other web applications we use every day. However, our results do contribute to improve web application security, because they unveil how some vulnerabilities are really being exploited, which by itself should be enough to trigger the awareness for new security procedures.

## III. Field Study Vulnerabilities and Exploits

The Open Web Application Security Project Foundation (OWASP) released a report on the ten most critical web application security risks [10], based on the vulnerability data provided by Mitre Corporation. In this section we present the seven types of exploits we found in our field study and how they relate to the OWASP Top 10 list of web application risks. We shall see that most of the attacks also exploit the most common vulnerabilities.

#### *1) Bruteforce/Dictionary*

These exploits attack the third most common web application risk of the OWASP top ten 2010 list [10]: "Broken Authentication and Session Management". We found only one exploit of this nature, developed by DarkFig [18], and it attacks the weaknesses in the passwords typically used in web applications [26] and the application logic. In fact, it is common to find applications that allow users to try endless passwords without any sort of counter measures, like maximum number of attempts, response delay on error or using a two-factor authentication (a password and an additional authentication item like a token) [27].

The exploit we found gives the attacker the option of choosing to use a brute force or a dictionary attack:

- The **brute force attack** can be further configured to use lower case or upper case alphabetic characters or even numeric characters. The starting length of the password to be guessed is also configurable. The exploit does simple iterations until it guesses the correct password of the victim whose user name in the application must be known previously. In spite of being able to crack any user's password it is, obviously, widely used to try to obtain the administrator password, which is usually associated to typical usernames like *admin*, *root*, *webmaster*, *webadmin*, *administrator*, among others.
- The **dictionary attack** uses a file of common passwords (including common application shipped

default ones) to try to check if any one of them is the correct one. This is certainly the quickest method, if the dictionary is well chosen [26].

*2) Admin Takeover*

These exploits attack the eighth most common web application risk of the OWASP top ten 2010 list [10]: "Failure to Restrict URL Access". They attack a bug in the application logic that allows unauthenticated or normal users to execute a function that should only be accessible to administrators. We found three exploits like this that allow any registered user to create a new administrator account.

For example, the "PHP-Nuke v7.4 admin exploit (old exploit)", by Silentium [18], uses the following C code as parameters of the *admin.php* page:

```
add_aid=%s&add_name=morte&add_pwd=%s&S
ubmit=Create+Admin
```

This C code, whose *%s* parts are replaced by the *name* and *password* provided by the attacker, makes PHP-Nuke create a new administrator account. Afterwards, the attacker can log in using this new account and take complete control of the web application. A regular user should not have access to the *admin.php* page. To prevent these kinds of problem, developers should check every web page for proper authentication and authorization.

*3) SQL Injection*

These exploits attack the most common web application risk of the OWASP top ten 2010 list [10]: "Injection". Except the exploits described previously, all the other exploits we analyzed take advantage of unchecked input fields at user interface.

In the case of SQL Injection, the exploit alters the SQL query that is sent to the back-end database to manipulate sensible data. We found that nearly 2/3 of the 102 SQL Injection exploits analyzed use the SQL *UNION* clause to obtain database data. The idea of this particular attack is to append a second query to an already existing one for which the results will be displayed by the web browser.

For example, the "PHP-Fusion Mod TI (id) Remote SQL Injection Vulnerability", by IRCRASH [18] is an exploit for the PHP-Fusion application. This web application has a function that is used to show in the browser the blog that has the identification number given by the input GET variable *id*. This variable is vulnerable to SQL Injection and the exploit attacks the variable like this:

```
id=-9999
+union+select+0,1,2,user_name,
user_password,5+from+fusion_users/*
```

The + signs are just the replacements of the SPACE character which cannot be used in a URL. In some other exploits the *%20* is also used, which corresponds to the hexadecimal of the ASCII value of the SPACE character.

This is a typical SQL Injection attack where a negative number is passed to the vulnerable variable so that the first query does not return any row. Next, the exploit adds the SQL *UNION* clause and the malicious select that allows obtaining the list of user names and respective passwords. The last part of the malicious string is the /*, which is the indication of the start of a comment in the MySQL language (also used in many other databases). This comment sign disables the rest of the original SQL query so that the parser does not raise any error, therefore rejecting the attack.

Although the passwords displayed may be encrypted with the MD5 or SHA1 hashes, most times they can be cracked using a dictionary attack [26, 28], and rainbow tables (pre-generated data set of hashes) to speed up the attack. SQL Injection is a very important threat, but it is easily defended using parameterized interfaces in the web application program, through prepared statements and stored database procedures [10, 12, 27].

*4) Cross Site Scripting (XSS)*

These exploits attack the second most common web application risk of the OWASP top ten 2010 list [10]: "Cross Site Scripting (XSS)". We found 16 exploits and although we did not find it as frequently as the SQL Injection, XSS vulnerability is usually reported as being more common [1, 10]. To attack a XSS vulnerability, the exploit tweaks the vulnerable input variable with a text containing a special crafted HTML or scripting language (usually JavaScript).

XSS exploitation may allow the attacker to do web site defacement, steal application cookies allowing the impersonation of the victim in the vulnerable web site, etc. However, most of XSS exploits analyzed target a more dangerous issue: the remote execution of OS commands in the web server machine as the root user, if the web server is not properly configured (which is usually the case, given the huge number of attacks like this).

An example of this type of attack for the Drupal application is the "Drupal < 5.1 (post comments) Remote Command Execution Exploit v2", by str0ke (who is also one of the members of the Milw0rm hacking group) [18]. It exploits the possibility of posting comments about a given blog message, which is a common feature in CMS applications. The problem here is that previews of these comments are not validated, and they are sent directly to the PHP interpreter that executes them and displays the result in the attacker web browser. The exploit uses the ability to input arbitrary characters in the *comment* variable to inject a PHP malicious string, like this:

```
comment=<?passthru('.$byte.');?>
```

The *passthru()* PHP function, similar to the *exec()* or *system()* functions, or even the ` (backtick operator), executes an external command on behalf of the web server OS user. The functions *passthru()* and *exec()* even allow the execution of multiple commands split with semi-colons. The *$byte* is the PHP variable that contains the external command the attacker wants to execute. With this exploit the attacker can freely manipulate the web server as if he was the owner.

*5) Remote File Inclusion (RFI)*

We found 171 RFI exploits, making them the most common type. They attack the third most common vulnerability of the OWASP top ten 2007 list [9]: "Malicious File Execution". Like some of the most dangerous XSS exploits analyzed, the RFI allows arbitrary code execution on the server. This is considered one of the methods used by hackers to create *botnets* and serve malware worldwide [7].

To exemplify how this exploit operates, let us consider the "phpBB Spider Friendly Module <= 1.3.10 File Include Exploit", by Kacper [18]. This attack exploits the vulnerable variable *phpbb_root_path*. The phpBB uses this variable extensively in the source code in sentences like this:

```
require($phpbb_root_path . 'config.' .
$phpEx);
```

Besides the *require()*, RFI exploits can use all PHP file and stream functions: *include()*, *include_once()*, *require()*, *require_once()*, *fopen()*, *imagecreatefromXXX()*, *file()*, *file_get_contents()*, *copy()*, *delete()*, *unlink()*, *upload_tmp_dir()*, *$_FILES* and *move_uploaded_file()* [9].

In a normal execution of the application, the file represented by the *$phpbb_root_path* is concatenated with *config.php* (*$phpEx* variabe is *php* by default) to be parsed and executed by the PHP interpreter. Although not intended by the normal use of the phpBB, it is possible to assign to the *$phpbb_root_path* variable something like *http://www.evilsite.com/shell.txt*. Using this in the *require()* function makes the PHP interpreter try to execute the file *http://www.evilsite.com/shell.txtconfig.php* (which is not the final goal of the attacker). To overcome the concatenation of the *config.php* to the malicious input, the exploit appends at the end the *%00* character. This technique is called null byte injection (because the *%00* character marks the end of the string) and it makes the PHP interpreter discard what comes next in the string. In this example, the file *shell.txt* can be a simple PHP file stored in the site *http://www.evilsite.com* controlled by the attacker, with the following code:

```
<?passthru($_GET["cmd"]);die;?>
```

This file executes in the victim's server computer the command that is passed by the GET parameter *cmd*. So, to exploit this vulnerability the attacker can use in the URL of the phpBB application something like this:

```
phpbb_root_path=http://www.evilsite.co
m/shell.txt?ls%20-las%00
```

In this example, the *ls -las* UNIX command is executed and the result is sent to the web browser of the attacker. However the payload could be any other command, like the *netcat* (commonly referred to as the Swiss army knife of networking), a program that can make the server listen to a particular *port* and run a program like the UNIX *shell*. This way, the attacker can, at any time, connect remotely to the server computer through that *port*.

Before 2007, the exploitation of this vulnerability type was very common in PHP web applications due to weaknesses in the default configuration shipped with PHP. Nowadays PHP default configuration is much safer than it was back in 2007 and critical configuration variables are now deprecated, not available or they have safer default values (e.g. *allow_url_fopen*, *allow_url_include*, *register_globals*). The support of remote file access for some functions used by hackers to perform RFI is also restricted [29, 13]. These PHP improvements may explain the decrease of the current importance of RFI vulnerabilities and their removal from the OWASP top ten 2010 list [10].

### 6) *Local File Inclusion (LFI)*

We found 18 of these exploits and they attack the fourth most common web application risk of the OWASP top ten 2010 list [10]: "Insecure Direct Object Reference". The LFI allows the attacker to obtain the contents of files stored in the server. This is usually achieved by maliciously altering the value of a vulnerable variable containing the path of the target file. For example, the exploit "Wordpress Plugin Page Flip Image Gallery <= 0.2.2 Remote FD Vuln", by GoLd_M [18], is able to obtain the server *passwd* file (the file with the user name and password hashes of the Linux server machine). This is done assigning to the vulnerable *book_id* variable the relative path of the target file. For example:

```
book_id=../../../../etc/passwd%00123
```

At the end of the malicious string there is, once again, the null byte injection (using the *%00*). This attack bypasses the naïve use of string concatenation to protect from malicious injection, as explained previously. Obviously, if the web server is correctly configured then it is not possible to obtain files from unwanted locations in the file system. However, even in these situations it is usually possible to get valuable information from the root of the web application, like source code files, configuration files, backup files, etc. If the access is not correctly restricted and the application is not well managed, the information that can be obtained may be enough for the attacker to access critical information (e.g., web application user names and passwords that are sometimes stored in a XML file in the web server).

### 7) *HTTP Response Splitting (HTTP-Splitting)*

These exploits were never present in the OWASP top 10 lists. In the Mitre definition, they are referred as [1]: "CRLF (Carriage Return and Line Feed) injection". The HTTP-Splitting attacks web applications in places where there is a failure to sanitize HTTP headers for CRLF (*%0D%0A* or *\r\n*) sequences. We found only a single attack of this type.

Using this technique, it is possible to force the server to consider that the web browser output consists of two different HTTP responses, where the latter is the attack [30]. To accomplish this objective, the attacker alters the HTTP response, for example by adding a *Content-Length: 0* to one of the HTTP header or POST parameters. Although this is a normal HTTP header parameter, using it in this way forces the premature end the current HTTP response. Then, the attacker places another HTTP response (the malicious one) containing a new header and a defaced HTML page that may trigger the victim into thinking he was browsing the original web application while he is, for example, interacting with the attacker's web page (page hijacking). The user may be then tricked to supply sensitive data, like passwords or credit card information.

TABLE I. EXPLOIT DISTRIBUTION OVER SIX WIDELY USED WEB APPLICATIONS

| | XSS | SQL Injection | RFI | LFI | Admin Takeover | HTTP-Splitting | Bruteforce/Dictionary | Total |
|---|---|---|---|---|---|---|---|---|
| **Drupal** | 4 | 0 | 1 | 0 | 0 | 0 | 0 | **5** |
| **PHP-Fusion** | 1 | 23 | 0 | 1 | 0 | 0 | 0 | **25** |
| **PHP-Nuke** | 1 | 29 | 22 | 5 | 1 | 0 | 0 | **58** |
| **phpBB** | 5 | 18 | 129 | 8 | 0 | 0 | 1 | **161** |
| **phpMyAdmin** | 0 | 2 | 2 | 1 | 0 | 0 | 0 | **5** |
| **Wordpress** | 5 | 30 | 17 | 3 | 2 | 1 | 0 | **58** |
| **Total** | **16** | **102** | **171** | **18** | **3** | **1** | **1** | **312** |

## IV. RESULTS AND ANALYSIS

In this section, we present and discuss the results of the field study from different points of view, so that we can get a profile of the attacker's perspective.

### A. *Exploit Distribution*

Table 1 shows the overall distribution of the exploits analyzed across the web applications considered in the study. As shown, the four most common vulnerabilities exploited are RFI, SQL Injection, LFI and XSS. RFI and SQL Injection are undoubtedly the most important in the scenario we evaluated as they were found in five out of the six web applications analyzed and they contributed to 88% of all exploits. We also observe that SQL Injection and RFI are the most wanted by the attackers, as they account for 83% of the downloads, according to Fig. 1, that shows the overall number of downloads for each exploit type. This is understandable, as RFI exploits typically provide remote access to the server through a root shell [7] and SQL Injection is almost the only way to get a direct access to the back-end database (which may enable disclosure of user names and passwords). However, from Fig. 1, we see that SQL Injection exploits are slightly preferred over RFI. These SQL Injection attacks give hackers what they want (access to data) and there are plenty of vulnerabilities allowing this type of attacks on the target web applications [8].

In the lower end of the hacker's usage we have Admin Takeover, HTTP-Splitting and Bruteforce/Dictionary. In spite of the lower number of exploits released, they have a high download rate, as can be seen in Fig. 2. For example, the single Bruteforce/Dictionary exploit found in our study was downloaded over 45 thousand times. It is a PERL program aimed to attack the user password using a brute force or a dictionary attack. Although it is meant to exploit the phpBB application, it can be easily adapted to other applications where a password cracking technique is needed, and this may justify part of the interest in downloading it. However, the most downloaded exploit (56 thousand times) was a SQL Injection one also affecting phpBB.

The exploits analyzed have an evolution over time, from 2003 to 2009 with its top at 2006, which can be seen in Fig. 3. It seems that exploits for the web applications analyzed started losing momentum after 2006. This also applies to the number of downloads of the exploits, which started very low in 2003, reached its peak in 2006 and since then they have been decreasing. In the case of SQL Injection exploits, for example, they have been increasing until 2008, which was the *best* year with 45 exploits, but that number also decreased in 2009 to 12 exploits, which is lower than the 20 exploits back in 2007.

Analyzing the spike seen in 2006 we found that 91% is due to the 123 RFI exploits that were developed in that year. Strangely, 2006 was the first year where RFI exploits were observed and the development of these types of exploits quickly decreased over the years, reaching only 5 in 2009. Some researchers believe that changes introduced in the PHP language are the reason for this trend [10] and they may well be, because they clearly addressed these types of problems. This observation shows the importance of the programming language in the security of the application.

To attack the application the hacker uses its weakest point, like plug-ins that are usually not so well developed or tested as the main application. We found that nearly 58% of exploits target external modules (Fig. 4). This exploitation can even be found in plug-ins that are supposed to bring a higher level of security to the web application. For example, we found security plug-ins *phpBB_Security* and *phpBB*

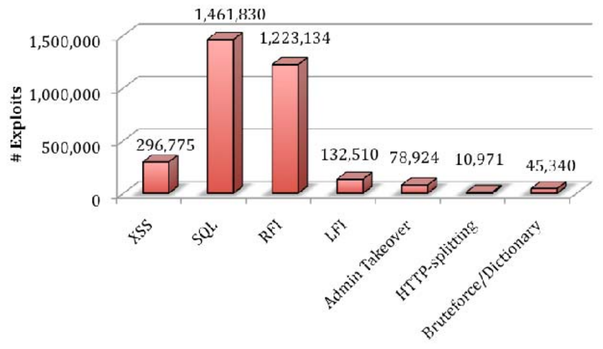


Figure 1. Exploits downloads

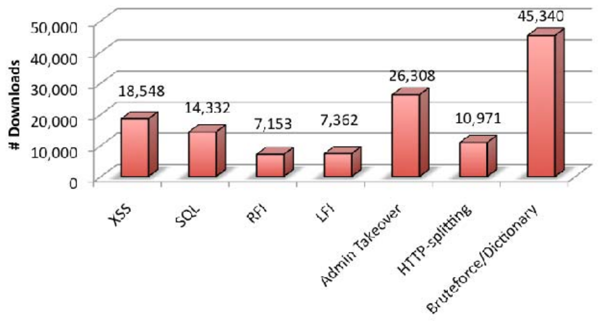
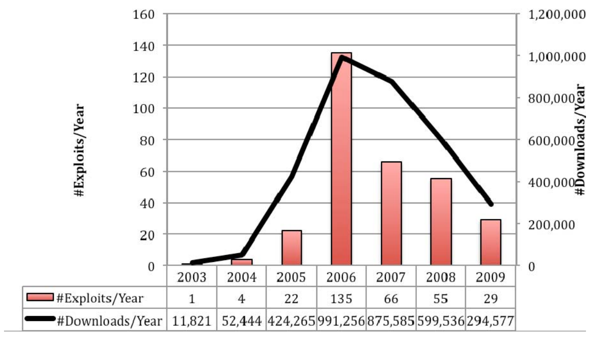


Figure 2. Average number of downloads per exploit



| | 2003 | 2004 | 2005 | 2006 | 2007 | 2008 | 2009 |
|---|---|---|---|---|---|---|---|
| 🟥 #Exploits/Year | 1 | 4 | 22 | 135 | 66 | 55 | 29 |
| #Downloads/Year | 11,821 | 52,444 | 424,265 | 991,256 | 875,585 | 599,536 | 294,577 |

Figure 3. Exploits over time

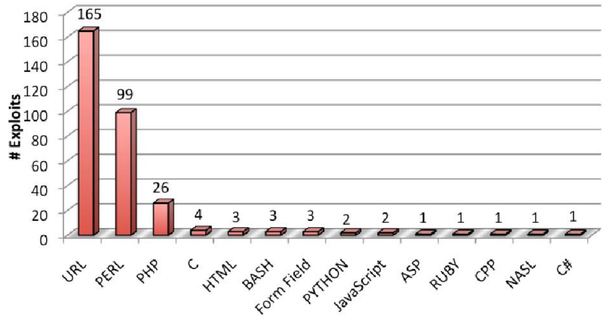


Figure 5. Technologies used in the development of the exploits

*Security Suite* to have RFI vulnerabilities that were exploited, allowing an attacker to execute code in the context of the web server process. These are paradox situations where it is the presence of the security plug-in that compromises the whole system they were supposed to protect, in a cascading manner.

## B. *Exploit Technologies*

Although hackers may need skilled resources to find vulnerabilities in web applications, they do not have to know much about programming languages to build a common exploit. Regarding the technologies used to develop the exploits we found that a simple maliciously crafted URL is all that it is needed in more than half the times (Fig. 5). In particular, GET variables are easier to probe for vulnerabilities and to write exploits than POST variables, because all it usually takes is to place the malicious string as the input of the vulnerable variable directly in the URL. By altering the URL, the attacker can control GET variables in a low security environment. This usually seems to be the case, as we found that unchecked GET variables account for 64% of the vulnerabilities exploited. On the other hand, POST variables need some sort of programming language that can build HTTP requests in order to be exploited in an automated fashion, which adds extra complexity.

For more elaborate exploits that show the result of the attack in a user-friendly manner, both GET and POST related vulnerabilities are similar in complexity. Other times, a programming language is really needed to actually perform and automate the attack. This is common in cases where the exploit includes counter measures against security protections built into the web application or due to the specific nature of the vulnerability. For example, in typical blind SQL Injection attacks, exploits try to discover every character of the user password, one at a time, and the final result is shown to the attacker only at the end.

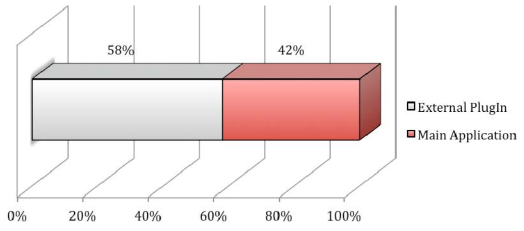


Figure 4. Exploits in main application vs. external plugins

PERL and PHP are the most chosen programming languages to develop the exploits analyzed (Fig. 5). These are powerful scripting languages that can be easily used in different operating systems, without cross-compilation issues. We found that most of the exploits developed with a programming language have the code well commented, with usage notes and they try to be as user friendly as possible. This makes them well suited for the occasional hacker and *script kiddies*.

Six of the exploits analyzed are already part of the collection of exploits of the Metasploit framework [19]. These exploits can, therefore, also be executed by every user of the Metasploit framework, which can be freely downloaded and is present in some Linux distributions devoted for security testing, like the BackTrack and the Whoppix. From the Milw0rm site only, these six exploits are responsible for more than 126 thousand downloads.

## C. *Severity Analysis*

The main reason to develop an exploit is the benefit that can be achieved by using it against the vulnerable web application. This characterizes the severity of the exploit, viewed from the web application perspective. It represents the level of damage the exploit can inflict to their victims. To classify the exploit severity we used the Payment Card Industry Data Security Standard (PCI-DSS) taxonomy [27], which is one of the security standards most used nowadays, mainly in e-business and e-commerce applications. This standard classifies vulnerabilities in 5 levels, from 1 to 5. The high-level vulnerabilities are designated 5, 4 or 3, in decreasing order. To be compliant with the PCI-DSS standard the application cannot have any one of these high-level vulnerabilities. The following points present a brief description of the PCI-DSS severity levels, in line with our interpretation of the level of damage that may be obtained with the exploits analyzed:

- **Level 5** - Allows the attacker to get read and write access to the remote computer as an administrator or to the web application database as a DBA.
- **Level 4** - Allows the attacker to get read and write access as a regular user (a user that is not an administrator nor a DBA) to the remote computer or to the web application database.

- **Level 3** – Allows the access to remote assets without permission to change contents. The attacker may see remote data (like security settings), files or browsing the directory tree.
- **Level 2** - Exposes sensitive information about the running programs and services on the server side.
- **Level 1** - Exposes data like server's open ports.

In Fig. 6 we present the details of the classification of the exploits analyzed according to their PCI-DSS severity. We see that 97% of the exploits attack the web application or the server in the most damaging way. The number of exploits that are classified in a level less than 5 is merely residual.

Analyzing in detail the exploits classified with level 5 severity, we found that 97% of them could have been classified with level 4, with little system configuration change. In fact, even without altering the application code, configuration and policy decisions of the administrator of the remote server or web application can make the difference between level 5 and level 4 or lower for the vast majority of exploits. The two most relevant of these best practices issues that we found, which are also extensively discussed in security literature [12, 27], are the following:

- **Running the web server process with the lowest privilege needed** - if the web server OS user has root privileges, when the account is compromised, the attacker can also deploy a remote shell as root. We found this in 61% of level 5 severity issues. An OS user with the same responsibilities, but having only the least amount of privileges needed to perform his job conveys a level 4 severity, as he would not run as the root user. Preventing the execution of external commands from the web application completely eliminates the root shell problem. Further configuration improvements, like restricting the access to the OS file system, reduces to level 4 or below even more 5% of level 5 issues.
- **Using a strong administrator account password** - a weak password may be easily cracked, while a strong password may be impossible to obtain using current state of the art technologies. Passwords are really wanted by hackers, and we found 31% of level 5 cases due to the exploitation of vulnerabilities that expose database passwords. Normally, users do not provide strong passwords when they register in online web applications. This is a well-known security concern and has been discussed over the years. Recent results of the whitepaper on consumer passwords from Imperva confirm this generalized idea [26]. The authors analyzed the largest password breach ever, containing around 32 million real passwords leaked from the RockYou social network application maker. The study shows that users tend to choose very weak passwords and it estimates that an automated attack can crack one password every second, corresponding to 111 attempts, if using a carefully chosen dictionary. This may also justify the high number of downloads of the Bruteforce/Dictionary exploit described earlier.

## D. *Vulnerability and Exploit Correlation for XSS and SQL Injection*

XSS and SQL Injection are the two most common vulnerabilities in web applications [1, 10] and they are also very relevant in the exploits analyzed. Fig. 7 shows the comparison between XSS and SQL Injection in web applications. The leftmost bar presents the distribution of the occurrences of XSS and SQL injection in the **exploits** we analyzed; the center bar shows the **vulnerabilities** from the field study presented in [8] for the same target applications; and in the rightmost bar presents the **vulnerabilities** disclosed in the MITRE report [1, 4].

Analyzing Fig. 7 we can see that web applications are more likely to have XSS than SQL Injection vulnerabilities. However, when we look at the number of exploits developed, less than 14% are for XSS. It seems that, although it is easier to find XSS, hackers prefer to exploit SQL Injection. We also obtained 17% of XSS downloads and 83% of SQL Injection (Fig. 1), when comparing only these two exploits.

Fig. 8 compares the XSS and SQL Injection exploits found for each web application with the number of vulnerabilities analyzed in [8]. Values are expressed in percentages so that we can easily compare and discuss the relative distribution of exploits and vulnerabilities. We see little relation between the distribution of vulnerabilities and the respective developed exploits. We do not have further data that can help justify this observation, but we can present

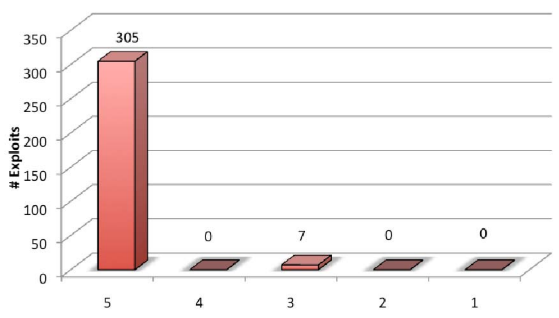


Figure 6. PCI severity level of the exploits analyzed

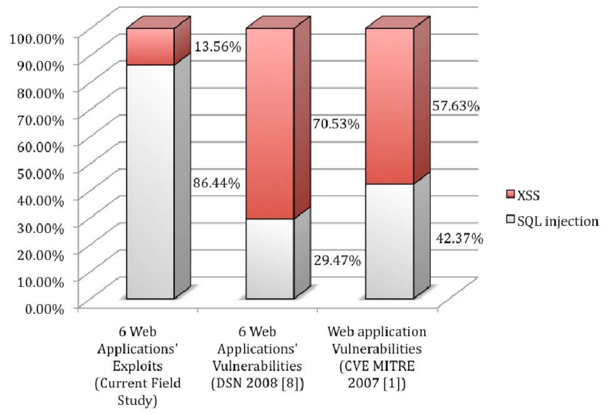


Figure 7. XSS vs. SQL Injection vulnerabilities and exploits

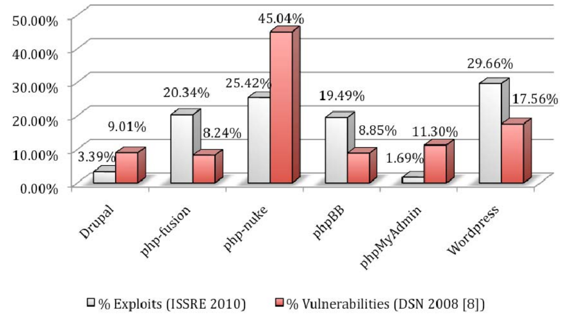


Figure 8. Vulnerabilities and exploits across web applications

some thoughts on this matter:

- Some of the reported vulnerabilities may be too hard to be exploited or not be exploitable at all as the web application may have other protection schemes or the originating bug may have been mistakenly reported as a vulnerability.
- Some of the vulnerabilities found may only be exploited in a manner that does not give attackers access to the resources they want.
- When hackers have many vulnerabilities at their disposal, they do not have to exploit all of them, as their goal may be achieved by exploiting only a subset of those vulnerabilities.
- Some web applications are more interesting to exploit than others. For example, phpMyAdmin exploits are usually only available to those who have privileges to administer MySQL databases. If a vulnerability can only be exploited after having granted access to the admin pages, only the admin can benefit from it. However, the phpMyAdmin admin already has full access to the databases, making the use of the exploit pointless.
- Some web applications have a larger market share than others. From a cost/benefit ratio, it is more interesting to invest time and money exploiting those who give a better perspective of success in a wider number of possible victims.

## *E. Analysis of the Attacker Perspective*

As expected, we observed that an attacker typically tries to exploit vulnerabilities that can empower him with an administrator account, whether it is an OS root shell or a DBA database password. However, it was a surprise that this accounted for so many exploits (97%, as seen in Fig. 6). There are vulnerabilities that may provide a path to assets more sensitive than others and these are clearly what hackers are looking for to develop their exploits. This demand for high-damaging exploits may help justify why the number of vulnerabilities and exploits do not always have a direct relationship (Fig. 8). By comparing XSS with SQL Injection we see that the attacker normally prefers SQL Injection (Fig. 1), although XSS is typically easier to find: apparently, the access to the database records pays off the extra effort.

Our results show that in a vast majority of vulnerable web applications, the attacker can get access to the server, exploiting LFI and RFI vulnerabilities. Besides good configuration practices [12, 27, 29, 13], security practitioners are advised to build a combination of web application, network and OS defense layers. This is known as Defense In Depth [31].

Although there are some complex exploits written in a programming language, surprisingly the attacker can achieve its objectives nearly half of the time with simple resources, like a crafted URL (Fig. 5). We also observed the search for simplicity reflected by the higher number of GET variables attacked when compared with POST variables.

To improve the attack success ratio, hackers not only want to increase the ability to access important assets, but also to decrease the chance of getting caught. They want to attack their victims remaining hidden behind another identity or by executing the attack stealthy. To achieve this, they can use public computers (from schools, libraries, cyber cafés, etc.), computers from *botnets* they own (they remotely control) or even use *anonymizer* proxy server solutions, like Tor (torproject.org) and UltraSurf (ultrareach.com). An *anonymizer* proxy does not transmit the client IP information and does not store any information about the client when he is surfing the web. When using these kinds of proxies in a web attack, there are no traces that can help identify the attacker. It allows the attacker to stay anonymous, even when digital forensics is in place. The possibility to configure the use of proxies in the attack is available in some more elaborated exploits, which shows the concern of hackers about this matter.

During our analysis, it was possible to observe the consecutive exploitation of the same vulnerable variable over and over again. For example, in the phpBB web application, we found exploits of the *phpbb_root_path* variable from February 2006 to April 2007. During this period, 110 exploits were developed to attack that same variable (this is the reason for the high number of RFI exploits in 2006). Many of these exploits targeted different external modules of the phpBB application, which also raises the question about the quality of external plug-ins and components, which may compromise the security of the whole application. In fact, to achieve higher security, software developers should perform a thorough analysis and have a holistic thinking of the problem. These 110 RFI exploits target a vulnerable variable with the same name, but in different source code files. It seems that the PHP in these files was written reutilizing common code, maybe using *copy & paste*. The discovery of a vulnerability should trigger the developers into searching for problems in other locations too.

We also found in many situations that, after the fix, a slight change in the old exploit is all that is needed to bypass the new counter-measures and build a new exploit. This suggests that developers should revise the review procedures used during the correction of the vulnerabilities [12, 27, 32]. For the attackers, the weak actions done by the developers are welcomed. They only have to tweak the exploit and probe for the same vulnerabilities in all the places where the target variables are in use.

## V. Conclusion

This paper analyzes over three hundred exploits, of six widely used LAMP web applications, which were downloaded by possible attackers over three million times. The exploit code and metadata was manually reviewed focusing on aspects that contribute to profile typical operations and the hacker perspective. Results provide insights that can be used to improve web application security from software developer to administrator actions.

We can see that attackers benefit from poorly developed web applications and weak server and application configurations. Most of the exploits analyzed are just crafted URLs or simple scripts in PERL or PHP that allow even non-experts to deploy them easily. However, their critical damage can also be mitigated using common security best practices, such as using the latest software versions (e.g. newer PHP releases address some critical security problems), using strong passwords and deploying services with low privileges.

We found that attackers typically use only a restricted subset of the existing vulnerabilities as RFI and SQL Injection account for almost 90% of exploits analyzed. In fact, by applying a limited set of procedures that exploits the most common vulnerabilities, hackers target the most important assets of web applications, which are administrator accounts and remote shell access to servers. This is inline with other studies that concluded that only a small percentage of software bug types accounted for the vast majority of web application vulnerabilities.

Observations show that security procedures executed when an exploit is found are far from being effective, and they must be urgently addressed. Results can also be useful for those interested in understanding what web attackers want and how they operate. This helps to improve security related mechanisms, like attack simulators, penetration testing tools, procedures and training.

As future work this seminal study should be extended to capture the relationship between exploit sites, cover web applications developed with other technologies and correlate them to see if attackers follow a common pattern.

## References

[1] Steve Christey, Robert Martin, “Vulnerability type distributions in CVE”, Mitre report, May, 2007

[2] Stefano Zanero, Luca Carettoni, Manuel Zanchetta, “Automatic detection of web application security flaws”, Black Hat Briefings, 2005

[3] Nenad Jovanovic, Christopher Kruegel, Engin Kirda, “Precise alias analysis for static detection of web application vulnerabilities”, IEEE Symposium on Security and Privacy, pp. 27-36, Ottawa, Ontario, Canada, 2006

[4] Steve Christey, “Unforgivable vulnerabilities”, The MITRE Corporation, Black Hat Briefings, August 2007

[5] Vnunet, August, 2007, http://www.vnunet.com/vnunet/news/2197408/monster-keptbreach-secret-five

[6] NTA, May, 2007, http://www.nta-monitor.com/posts/2007/05/annualsecurityreport.html

[7] Gadi Evron, Kfir Damari, Noam Rathaus, “Web server botnets and hosting farms as attack platforms”, Virus Bulletin, 2007

[8] José Fonseca, Marco Vieira, “Mapping software faults with web security vulnerabilities”, International Conference on Dependable Systems and Networks, pp. 257-266, Anchorage, USA, 2008

[9] Andrew Stock, Jeff Williams, Dave Wichers, “OWASP top 10”, OWASP Foundation, July, 2007

[10] Jeff Williams, Dave Wichers, “OWASP top 10 – 2010”, OWASP Foundation, April, 2010

[11] Prasanth Anbalagan, Mladen Vouk, “Towards a Unifying Approach in Understanding Security Problems”, International Symposium on Software Reliability Engineering, pp. 136-145, Mysuru, India, 2009

[12] Michael Howard, David LeBlanc, “Writing secure code”, Microsoft Press, 2003

[13] Shaun Clowes, “A study in scarlet, exploiting common vulnerabilities in PHP applications”, Blackhat Briefings Asia, 2001

[14] Kevin Mitnick, William Simon, “The art of deception: controlling the human element of security”, 1st ed., Wiley, 2002

[15] Seth Fogie, Jeremiah Grossman, Robert Hansen, Anton Rageattacks, “XSS attacks: cross site scripting exploits and defense”, Syngress, 2007

[16] Gonzalo Álvarez, Slobodan Petrovic. “A new taxonomy of web attacks suitable for efficient encoding”. Computers and Security, vol. 22, issue 5, pp. 435-449, July 2003

[17] Michel Cukier, Robin Berthier, Susmit Panjwani, Stephanie Tan, “A statistical analysis of attack data to separate attacks”. International Conference on Dependable Systems and Networks, pp. 383-392, Philadelphia, USA, 2006

[18] Milw0rm, April, 2010, http://www.milw0rm.com

[19] David Maynor, “Metasploit Toolkit for Penetration Testing, Exploit Development, and Vulnerability Research”, Syngress, 2007

[20] José Fonseca, Marco Vieira, Henrique Madeira, “Vulnerability & attack injection for web applications”, International Conference on Dependable Systems and Networks, pp. 93-102, Lisbon, Portugal, 2009

[21] Ivan Victor Krsul, “Software vulnerability analysis”, PhD Thesis, Purdue University, 1998

[22] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, Carl Landwehr, “Basic concepts and taxonomy of dependable and secure computing”, IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 1, pp. 11-33, Jan.-Mar. 2004

[23] Packt Publishing Ltd, December, 2007, http://www.packtpub.com

[24] sourceforge, December, 2007, http://sourceforge.net/community/index.php/2007/08/01/community-choice-awards-winners/

[25] Nexen.net, October, 2008, http://www.nexen.net/chiffres_cles/phpversion/18824-php_statistics_for_october_2008.php

[26] Imperva Application Defense Center, “Consumer password worst practices”, 2010

[27] PCI Security Standards Council, “Payment card industry (PCI) data security standard, Requirements and Security Assessment Procedures”, version 1.2, 2008

[28] Openwall Project, John the Ripper password cracker, April, 2010 http://www.openwall.com/john

[29] PHP runtime configuration, April, 2010, http://php.net/manual/en/filesystem.configuration.php

[30] Common Weaknesses Enumeration Definitions, April, 2010, http://cwe.mitre.org/data/definitions/113.html

[31] NSA, “Defense in depth”, http://www.nsa.gov/ia/_files/support/defenseindepth.pdf, 2004

[32] José Fonseca, Marco Vieira, Henrique Madeira, “Training security assurance teams using vulnerability injection", Pacific Rim Dependable Computing Conference, pp. 297-304, Taipei, Taiwan, 2008