Mapping Software Faults with Web Security Vulnerabilities

José Fonseca[1], Marco Vieira[2]
[1] *CISUC, University of Coimbra, IPG, Portugal*
[2]*CISUC, University of Coimbra, Portugal*
josefonseca@ipg.pt, mvieira@dei.uc.pt

Abstract

Web applications are typically developed with hard time constraints and are often deployed with critical software bugs, making them vulnerable to attacks. The classification and knowledge of the typical software bugs that lead to security vulnerabilities is of utmost importance. This paper presents a field study analyzing 679 security patches of six widely used web applications. Results are compared against other field studies on general software faults (i.e., faults not specifically related to security), showing that only a small subset of software fault types is related to security. Furthermore, the detailed analysis of the code of the patches has shown that web application vulnerabilities result from software bugs affecting a restricted collection of statements. A detailed analysis of the conditions/locations where each fault was observed in our field study is presented allowing future definition of realistic fault models that cause security vulnerabilities in web applications, which is the key element to design a realistic attack injector.

1. Introduction

Most information systems and business applications that are built nowadays (e.g. e-commerce, banking, transportation, web mail, blogs, etc.) have a web front-end. They need to be universally accessed by clients, employees and partners around the world as online trading is becoming more and more ubiquitous in the global economy. These web applications, which can be used from anywhere, also become so widely exposed that any existing security vulnerability will most probably be uncovered and exploited by hackers. Hence, the security of web applications is a major concern and it is receiving more and more attention from the research community. However, in spite of this growing awareness of security aspects at web application level [1, 2, 3, 4, 5], there is an increase in the number of reported attacks that exploit web application vulnerabilities. A recent example (August 17, 2007) of such attacks occurred in the recruitment website Monster.com where 1.6 million of personal records were stolen [6]. Numerous other data breach attacks are frequently reported and many of them are due to security problems in web applications [7, 8, 9].

According to an Accunetix [10] audit result, 70% of the 3,200 websites scanned in the past three years contain security vulnerabilities. The NTA Monitor's 2007 Annual Security Report [11] states that online risks in financial institutions have increased 16%, and 28% in publishing companies. Overall, 32% of the websites analyzed contain critical vulnerabilities that are widely known and actively exploited by hackers. This confirms that the security problem in web applications is an issue far from being solved.

Software bugs that are responsible for security vulnerabilities may have a devastating cost if exploited by hackers. Although configuration and human issues are also potential causes for vulnerabilities, the root cause of most security attacks are vulnerabilities created by software faults.

In this paper we look at 679 security patches that were developed for different versions of six widely used web applications. In order to characterize the types of faults that are most likely to lead to software vulnerabilities we classify each patch code according to the Orthogonal Defect Classification (ODC) [12, 13, 14]. This is important to better understand the potential relation between certain types of software defects and security vulnerabilities. Every patch is also inspected in depth to gather the precise characteristics of the code that was responsible for the security problem. This detailed patch information is of utmost importance for example to build a realistic attack injector. It may also be crucial for the development of automatic static code analyzers focusing on finding security vulnerabilities, the specification of guidelines for teams of security code reviewers, the evaluation of penetration test tools as well as for the creation of more secure internal policies for programming practices, among others.

For the purpose of this paper it is also important to understand if the distribution of security faults follows a similar pattern of software faults in general. Knowing the different distribution patterns can help direct the instruction of security teams, for example. Therefore, we try to correlate our results with a field study on common software faults [14]. We also compare our results with another study that injected common software faults into web applications to see if they caused security vulnerabilities [15]. The comparison with both field studies [14, 15] is important to conclude if the injection of software faults can be used to accurately simulate security defects.

The structure of the paper is as follows. Section 2 presents the classification of software faults and discusses the source data (web applications and patches) used in the field study. Section 3 presents the results of the field study, its correlation with other studies and the vulnerability fault models. Section 4 concludes the paper and suggests future work.

2. Classification of web application security patches

The web applications whose vulnerabilities were used as case study represent a large slice of the web application market and have a large community of users. The classes of vulnerabilities analyzed have a critical importance and affect most of the web applications, not just those used in this study.

The present study uses LAMP (Linux, Apache, Mysql and PHP) web applications because they include the technologies most commonly used to build web applications nowadays. LAMP software is free, fast, flexible, and has many libraries. For these reasons, it is widely adopted to build custom web applications, portals for large community of users, e-commerce applications, web administration tools, etc. Nevertheless, this kind of setup is also responsible for a large number of reports of security flaws.

The two vulnerabilities analyzed by the present study are the most critical in web applications: Cross Site Scripting (XSS) and SQL Injection (see [8] for details on these vulnerabilities). Exploits of these vulnerabilities take advantage of unchecked input fields at user interface, which allows the attacker to change the SQL commands that are sent to the database server (SQL Injection) or through the input of HTML and a scripting language (XSS).

The Open Web Application Security Project (OWASP Foundation) [8] released a report in 2007 which listed the ten most critical web application security vulnerabilities. It was based on data on vulnerability type distributions in Common Vulnerabilities and Exposures (CVE[1]) provided by Mitre Corporation [9]. According to this report, XSS is the most critical vulnerability (18.5%), followed by SQL injection (13.6%). Together they are responsible for approximately one third of all the CVE in 2006. The popularity of these attacks is related to: a) the facility in finding and exploiting such vulnerabilities; b) the importance of the assets they can disclose; and c) the level of damage they may inflict. In fact, SQL injection and XSS allow attackers to access unauthorized data (read, insert, change or delete), gain access to privileged database accounts, impersonate another user (such as the administrator), mimicry web applications, deface web pages, get access to the web server, etc.

When application vulnerabilities are discovered, software developers correct the problem releasing application updates or patches. To understand which code is responsible for security problems we based our study on patches correcting vulnerabilities. With this approach we are classifying the code that caused real security flaws.

For each web application tested, the methodology to classify the security patches is the following:

1) Verification of the patch to confirm if the version of the web application is available.
2) Analysis of the code with the vulnerability and of the code after being patched.
3) Classification of each code fix that is found in the patch. To be accurate, we followed some rules as described in section 2.4.
4) Loop through the previous steps until all available patches of the web application are analyzed.

2.1 Classification of software faults from the security vulnerability point of view

The security patch code analyzed in the present study was categorized based on the software faults classification proposed by Chillarege et al. [12, 13]. They introduced the Orthogonal Defect Classification (ODC) that is typically used to classify software faults or defects after they have been fixed. The ODC has been used to improve the software design process and it bridges the gap between statistical defect models and the causal analysis. The underlying idea is that

[1] CVE is a widely accepted list of publicly reported web application vulnerabilities. It is hosted by MITRE Corporation

knowing the root cause of software defects helps in removing their source, therefore contributing to the improvement of software quality [16]. In the present study we are only dealing with code defects, therefore we only use the ODC defect types that are directly related to the code. These defect types are the following: **Assignment** - errors in code initialization; **Checking** - errors in program logic and validation; **Interface** - errors interacting among components; **Algorithm** - need algorithm change without a design change. Although Function and Timing/Serialization are also related to the code we do not consider them because there was no example of these types in the field data.

The five classes of ODC fault types are too broad and they do not have enough detail for the precision needed by the present field study. We need to analyze the code from the point of view of the software programmer, so each of the ODC types can be detailed according to the nature of the defect [14]: **missing construct**; **wrong construct**; and **extraneous construct**. With this extension the five classes of the ODC are detailed into 62 fault types (see [14] for details). This refinement of the ODC is more focused on the concrete source of the software faults and has been used to support the fault model of the G-SWFIT tool for the emulation of software faults [14].

All the security vulnerabilities collected can be classified using only 11 fault types already identified in [14] and one extra fault type (MFC extended). They are presented in Table 1, where their correlation with the original ODC types is also shown. We defined the MFC extended fault type because there was one situation that could not be classified according to the fault types defined by Durães [14]. The MFC extended is based on the missing function call (MFC) fault type. The MFC can only be used if the return value of the function is not used elsewhere in the code. However, in web application programming there are lots of security problems because of missing functions whose purpose is to sanitize a variable. The return of these functions is the variable sanitized that will be used in the code. This important fault type can not be classified using the original MFC so, to overcome this situation we removed the restriction and created a new type named "MFC - Missing function call extended".

2.2 Web applications analyzed

One mandatory condition for this field study is that we need to have access to the source code of the web applications under analysis. The code of previous versions and the associated security patches must also be accessible. The other mandatory condition is the availability of information correlating the security fix and the specific version of the web application.

For the present study we have selected six web applications: PHP-Nuke [17], Drupal [18], PHP-Fusion [19], WordPress [20], phpMyAdmin [21] and phpBB [22]. These are representative open source web applications and, fortunately, it is possible to find enough information available about them.

Drupal, PHP-Fusion and phpBB are Web Content Management Systems (CMS). A CMS is an application that allows an individual or a community of users to easily create and administrate web sites that publish a variety of contents. The created sites can go from personal web pages and community portals to corporate and e-commerce applications. Drupal won first place at the 2007 Open Source CMS Award [23]. PHP-Fusion was one of the five award overall winner finalists at the 2007 Open Source CMS Award [23] and has a large community of users working with it. In fact, a Google search of PHP-Fusion pages using the text

Fault type	Description	ODC type
MFC	Missing function call	Algorithm
MFC extended	Missing function call extended	Algorithm
MVIV	Missing variable initialization using a value	Assignment
MIA	Missing if construct around statements	Checking
MIFS	Missing if construct plus statements	Algorithm
MLAC	Missing "AND EXPR" in expression used as branch condition	Checking
MLOC	Missing "OR EXPR" in expression used as branch condition	Checking
WVAV	Wrong value assigned to variable	Assignment
WPFV	Wrong variable used in parameter of function call	Interface
WFCS	Wrong function called with same parameters	Algorithm
ELOC	Extraneous "OR EXPR" in expression used as branch condition	Checking
EFC	Extraneous function call	Algorithm

Table 1. The 12 detected fault types observed in the field, their description and corresponding ODC fault type

"Powered by PHP-Fusion" finds over 2 million pages. Finally, phpBB is the most widely used Open Source forum solution. phpBB was the winner of the 2007 Sourceforge Community Choice Awards for Best Project for Communications [24].

PHP-Nuke is a well known web based news automation system built as a community portal. The news can be submitted by registered users and commented by the community. PHP-Nuke is quite modular and custom modules can be added to increase the number of features available. PHP-Nuke is one of the most well known CMS and it has been downloaded from the official site over 8 million times [17].

WordPress is a personal blog publishing platform that also supports the creation of easy to administrate web sites. A Google search of WordPress pages using the text "Proudly powered by WordPress", which is at the bottom of WordPress based sites, finds over 7 million pages.

phpMyAdmin is a web based MySQL administration tool. It is one of the most popular PHP applications and has a very large community of users. phpMyAdmin is available in 47 languages, is included in many Linux distributions and was the winner of the 2007 Sourceforge Community Choice Awards for Best Tool or Utility for SysAdmins [24].

Web application	Versions analyzed	# Vuln.
PHP-Nuke	6.0, 6.5, 6.9, 7.0, 7.2, 7.6, 7.7, 7.8, 7.9	292
Drupal	4.5.5, 4.5.6, 4.6.5, 4.6.6, 4.6.7, 4.6.8, 4.6.9, 4.6.10, 4.6.11, 4.7.6, 5.1	63
PHP-Fusion	6.00.106, 6.00.108, 6.00.110, 6.00.204, 6.00.206, 6.00.207, 6.00.303, 6.00.304, 6.01.4, 6.01.5, 6.01.6, 6.01.7, 6.01.8, 6.01.9, 6.01.10, 6.01.11, 6.01.12	49
WordPress	1.2.1, 1.2.2, 1.5.2-1, 2.0, 2.0.10-RC2,2.0.4, 2.0.5, 2.0.6, 2.1.2, 2.1.3 2.1.3-RC2, 2.2, 2.2.1, 2.3	110
phpMyAdmin	2.1.10, 2.4.0, 2.5.2, 2.5.6, 2.5.7PL1, 2.6.3PL1, 2.6.4, 2.6.4PL4, 2.7.0PL2, 2.8.2.4, 2.9.0, 2.9.1.1, 2.10.0.2, 2.10.1, 2.11.1.1, 2.11.1.2 and SVN revisions	104
phpBB	2.0.3, 2.0.5, 2.0.6, 2.0.6c, 2.0.7, 2.0.8, 2.0.9, 2.0.10, 2.0.16, 2.0.17	61
Total vulnerabilities analyzed		**679**

Table 2. Versions of the web application used and number of vulnerabilities analyzed

The web applications analyzed are so broadly used that they have a large number of vulnerabilities disclosed from previous versions, which are the subject of analysis of the present field study (see Table 2). The number of vulnerabilities is not constant among web applications because the quality of the code and the number of vulnerabilities publicly disclosed varies a great deal.

It is important to emphasize that all discovered vulnerabilities open a door for hackers to successfully attack any one of the millions of web sites developed with a given version of the web application. Furthermore, it is common to find a vulnerability in a specific version of a web application that also affects a large number of the previous versions of the same application. The overall situation is even worse because web site administrators do not always update the software of the site in due time when new patches and releases are available. This can be confirmed by the results of the security analyst David Kierznowski that preformed a survey showing that 49 out of 50 WordPress blogs checked did not upgrade to the last stable version and were running software with known vulnerabilities [25].

2.3 Obtaining the patch code

The availability to the public of the past collection of vulnerability patches is closely related to the policies the developers have about sharing information about older versions, especially those with security problems. Furthermore, most of the security announcements available are so vague that it is impossible to know what source files and variables are affected. Moreover, some of the information disclosed groups together other types of security vulnerabilities which are not the target of the present paper (e.g. directory traversal, remote file inclusion, cookie poisoning).

In order to gather the actual code of security patches several sources of data had to be used, such as mirror web sites, other sites with the source code, online reviews, news sites, sites related to security, hacker sites, changelog files of the application, the version control system repository, etc.

For the purpose of this study, we just need the changes made to the code of the application correcting the vulnerability problem. There is no standard way of providing the data about a security vulnerability fix; therefore, the variety of resources of information provides different presentations of the collected data. This makes it harder to perform the analysis because

the information about the fix code has to be obtained from several sources. The four main source types used in the current paper are described next:

1) **Security patch files available with information about the target version of the application.** This patch file was written to substitute just the original application file with the vulnerability leaving all the other source files intact. To obtain the code changes of these two files we used the UNIX diff command.
2) **Updated version of the web application.** Actually, this is a completely new version of the application containing all the new features and bug fixes (including security ones). This is the most common source of information but it is also the one that needs more work to be done. We have to find, amongst all the other source files of the application, the code responsible for the various security vulnerabilities addressed by this version. Additional information is needed about what source files have been updated with the security fixes. This information is commonly found in the changelog file that is distributed with the application. This changelog file consists of the summary of the changes made in the several versions of the application, including what bugs and security issues were fixed. After identifying the vulnerable source file we had to use the UNIX diff command to obtain the code changes between this file and the corresponding file from the vulnerable version of the application (usually is the previous version).
3) **Available security diff files**. This is a file containing only the code changes needed to fix a referenced vulnerability. The contents are ready to be applied to the target application using the UNIX patch command. This is all the information we need and, although this is the easiest data source to work with, it is the rarest to find.
4) **The version control system repository.** Almost all open source applications are developed using a version control system to administer the contributions of the large community of developers from around the world. With granted permissions to query the repository of the version control system we have access to all the revisions (similar to versions) of the application and corresponding changelog files. By querying the changelog we can obtain the information about the revisions of the application where the security vulnerability problems were fixed. It is then possible to obtain the security diff file using the version control system.

2.4 Patch code analysis guidelines

The patch code is analyzed according to the extension of the ODC classification, emphasizing the nature of the patch as missing, wrong, or extraneous code. Because of the different coding practices of the target applications some decisions need to be clarified. To avoid classification mistakes and for the coherent analysis of the fix code some generic guidelines were defined:

1) When the patch can fix both XSS and SQL Injection the corresponding fault type is accounted for both security vulnerabilities. For example, this occurs when a variable not properly sanitized is used in a query (allowing SQL injection) and, later on is displayed on the screen (allowing XSS). When this variable is properly sanitized both vulnerabilities will be mitigated simultaneously.
2) It is assumed that the information publicly disclosed in specialized sites is accurate and that the fix made by the programmer of the patch and made available by the company that develops the web application solves the stated problem.
3) To correct a single vulnerability several code changes may be necessary. All the changes will be considered as a series of singular fault type fixes. For example, when two functions are needed to properly sanitize a variable. Missing any of these functions makes the application vulnerable, so both of them must be taken into account.
4) When a particular code change corrects immediately several vulnerabilities, each one is considered as a singular fix. For example, suppose that: the value assigned to a specific variable may come from two sources of external inputs; and the variable is only displayed in one place without ever being sanitized. We consider that the application has two security vulnerabilities because it can be attacked from two different inputs. However to correct the problem all that is needed is sanitizing the variable just before it is displayed. In this example we consider that two security problems have been fixed, although only one change of code was needed.
5) A security vulnerability may affect several versions of the application. This happens when the code has not been changed for a long time, but it is vulnerable. The patch to fix the problem is the same for all of the versions, and therefore it is considered only one fix.

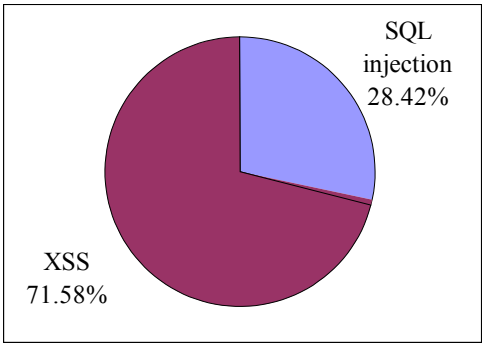

Figure 1. XSS and SQL injection distribution

By following the previous guidelines it is possible to classify almost all the code fixes. However in some situations, patching one or more vulnerabilities involved so many changes that the creation of new functions or a change in the structure of the overall piece of code was impossible to classify. This situation is usually associated with major code changes involving simultaneously security and other bug fixes. These occurrences are quite marginal (5.4%) and were not considered in our study because they are complex and difficult to analyze due to the lack of information available.

3. Results and discussion.

We have classified 679 XSS and SQL injection security fixes found in six web applications. Figure 1 shows the overall distribution of XSS and SQL injection vulnerabilities found in all the web applications analyzed. As can be seen, XSS is the most frequent type by far. This trend is also confirmed by vulnerability reports disclosed in CVE [8, 9]. One of the factors that contribute to the prevalence of XSS is that every input variable of the application is a potential attack entry point, which is not the case for SQL injection. The explanation resides in the high number of variables found that needed to be sanitized.

The distribution of the occurrences of XSS and SQL injection throughout the twelve classification fault types is shown in Figure 2. The most representative and widespread fault type is the MFC extended. This is the most common fault type, representing 72.90% of all the fault types found. The high value observed for this fault type comes from the massive use of specific functions to validate and clean data that comes from the outside of the application (user inputs, database records, files, etc.). In many cases, functions are also used to cast a variable to a numeric value, therefore preventing string injection in this variable.

The next three most common fault types are the WPFV, MIFS and WVAV. These vulnerabilities usually arise from the following main situations:

1) Missing "'" around a PHP variable in SQL queries allowing an attacker to inject a custom query (SQL injection).
2) Missing "if" around a statement. When a variable is not null it needs to be sanitized, otherwise a malicious code may be injected from the outside. This is an exploit of the PHP directive "register_globals = on" [26] which allows the injection in all sorts of variables, when the code is unsecured.

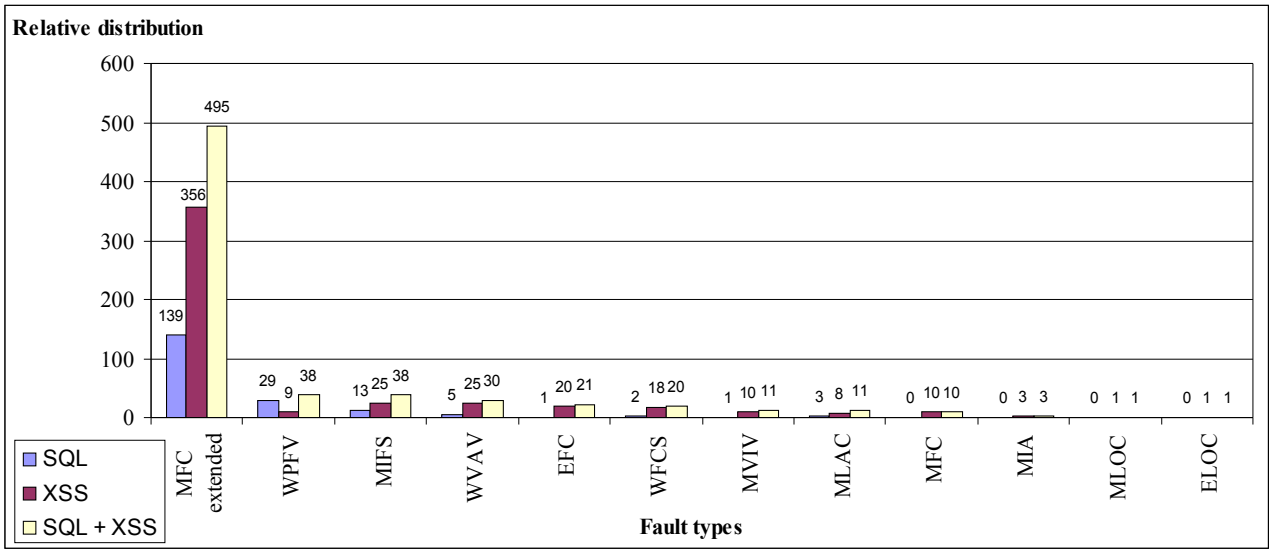

Figure 2. Vulnerability fault types summary

Web applications	PHP-Nuke		Drupal		PHP-Fusion		WordPress		phpMyAdmin		phpBB	
Fault type	**SQL**	**XSS**	**SQL**	**XSS**	**SQL**	**XSS**	**SQL**	**XSS**	**SQL**	**XSS**	**SQL**	**XSS**
MFC extended	115	135	4	36	6	15	8	82	1	57	5	31
WPFV	29			4						4		1
MIFS	5	2		2	8	6		1		12		2
WVAV	4			3						6	1	16
EFC					1					19		1
WFCS				3	1	1	1	14				
MVIV		1			1	1		4				4
MLAC				1	3	4				3		
MFC				8						2		
MIA				1		2						
MLOC		1										
ELOC				1								
Total Faults	**153**	**139**	**4**	**59**	**20**	**29**	**9**	**101**	**1**	**103**	**6**	**55**

Table 3. Detailed results

A poor regex string [2] used to filter the user input. Looking at several versions of the same program we frequently found the same regex string being updated as new attacks were discovered.

Excluding the faults types already discussed (MFC extended, WPFV, MIFS and WVAV), the remaning fault types correspond just to 11.49% of the security vulnerabilities. These fault types are EFC, WFCS, MVIV, MLAC, MFC, MIA, MLOC and ELOC.

Table 3 shows the individual results for each fault type found in all web application. All the fault types contribute to XSS, but only eight to SQL injection. Although it is relevant to know what fault types may generate vulnerabilities, the four fault types that do not contribute to SQL injection (MFC, MIA, MLOC and ELOC) only account for 2.21% of all the fault types.

A common belief is that vulnerabilities related to input validation are mainly due to missing IF constructs or even missing conditions in the IF construct. However, our field study shows that this is not the case, as the overall MISSING IF fault types (MIFS and MIA: see table 1) have a weight of 6.04%. As for the MISSING CONDITION fault types (MLAC and MLOC), they only represent 1.77%. We indeed verified that programmers do not use IF constructs to validate the input data, because of the usual complexity of the validation procedure to avoid XSS and SQL injection. The typical approach is to use a function to clean (filter) the input data and let it go through, and not to stop the program and raise an exception.

[2] A regex string describes a search pattern, according to specific syntax rules, that is used to search into another string.

3.1 Comparing security faults with generic software faults

The original ODC classification is broadly used and accepted as quite adequate for the classification of software faults. Durães [14] analyzed 668 faults from a collection of 12 representative open source C programs using the ODC, while Christmansson and Chillarege [12] studied large databases and operating systems. Each of these studies uses a different application and programming technology. Thus, it is relevant to compare our results with other field studies like [14] and [12], as shown in Table 4.

Although the values of the current field study are quite different from those of the two other studies [12, 14], the Algorithm has the highest value followed by the Assignment. The overall distribution in table 4 is quite different and reinforces the idea that the kind of mistakes leading to security vulnerabilities has a different distribution from the generic fault types. There is an increasing importance of some fault types in detriment of others when we want to analyze the

ODC defect type	# vuln.	% vuln.	% vuln. [14]	% vuln. [12]
Assignment	41	6.04%	21.4%	21.98%
Checking	16	2.36%	25%	17.48%
Interface	38	5.60%	7.3%	8.17%
Algorithm	584	86.01%	40.1%	43.41%
Function	0	0%	6.1%	8.74%

Table 4. ODC vulnerabilities

MFC - extended	A	Missing casting to numeric of one variable to itself
	B	Missing assignment of one variable to a function processing that same variable
	C	Missing casting to numeric of one variable to another variable
	D	Missing assignment of one variable to a function processing another variable
WPFV	A	Add "′" to variables inside a string argument of a SQL query
	B	Changing the regex string of a function argument
	C	Changing the sub-string of a function argument
	D	Changing the PHP superglobal variable when it is an argument of a function
MIFS	A	Missing traditional "`if…then…else`" condition
	B	Missing "`if…then…else`" condition in compact form
WVAV	A	Insertion of a new pattern in a regex string assigned to a variable
	B	Changing one value of an array or a concatenation of a new substring inside a string
	C	Changing the PHP superglobal variable when assigned to a variable
	D	Adding "′" to variables inside a string in a SQL Query assignment
	E	Destroying the variable
	F	Removing the concatenation operator "." in an assignment
	G	Replacing an array variable with a scalar variable

Table 5. fault types and corresponding sub-types

security of web applications.

Based on the fact that some common vulnerabilities found are caused by specific characteristics of the programming language (like the use of default value of the "register_globals" directive, in PHP), we believe that the type of language/technologies involved will influence the distribution of faults over the ODC types when we are analyzing security faults. Programming languages, in general, have a great concern in security and this can be seen in the new features that are being implemented in recent versions. Some of these changes imply an adjustment in the way some operations are performed by the programmer and this will certainly reflect more the results of the ODC analysis than the case of common faults.

Another important conclusion can also be derived by comparing our results with those presented by the study in [15]. In this study the authors propose a method to benchmark web application vulnerability scanners by injecting generic software faults. Although the purpose of the paper was to test fuzzers, the results also show what common fault types may produce XSS and SQL injection vulnerabilities. Analyzing the results presented in [15] we find that they could inject both vulnerabilities with one fault type that was not detected during our study. This fault type is "WLEC - Wrong logical expression used as branch condition", which generated 16.10% of the total number of faults injected. This result means that not all the possible fault types that can generate vulnerabilities are actually responsible for the security problem of web applications found in the real world.

3.2 Detailed vulnerability analysis

During the classification of the web application vulnerabilities we discovered some repeating patterns in the code. The instructions that fixed the vulnerabilities belong to a restricted subset of all the possible code structures of each fault type. To make use of this data and accommodate the precise situations

Fault types & sub-types		SQL (%)	XSS (%)	SQL+XSS (%)
MFC - extended	A	61.66	29.42	38.59
	B	7.25	35.39	27.39
	C	2.07	5.56	4.57
	D	1.04	2.88	2.36
WPFV	A	15.03	0.00	4.27
	B	1.04	1.03	1.03
	C	0.00	1.03	0.74
	D	0.00	0.62	0.44
MIFS	A	5.18	4.73	4.86
	B	1.55	0.41	0.74
WVAV	A	0.00	2.88	2.06
	B	0.00	1.23	0.88
	C	1.04	0.41	0.59
	D	1.04	0.00	0.29
	E	0.00	0.41	0.29
	F	0.00	0.21	0.15
	G	0.52	0.00	0.15
EFC		0.52	4.12	3.09
WFCS		1.04	3.70	2.95
MVIV		0.52	2.06	1.62
MLAC		1.55	1.65	1.62
MFC		0.00	2.06	1.47
MIA		0.00	0.62	0.44
MLOC		0.00	0.21	0.15
ELOC		0.00	0.21	0.15

Table 6. Occurrence of fault types and sub-types

found we defined sub-types for the four most common fault types (MFC extended, WPFV, MIFS and WVAV), as described in Table 5.

The occurrence of the fault types and the sub-types detected in the vulnerabilities analyzed is shown in Table 6. We can observe that there are some sub-types responsible for a large slice of the all the vulnerabilities. The two types with higher values belong to the MFC – extended fault sub-types A and B and together they account for 65.98% of all the vulnerabilities found. We can also find important differences between the values of the sub-types relating to XSS and SQL injection. For example, MFC – extended A is much more importance in SQL injection, but MFC – extended B is the opposite. Also WPFV A has a huge importance in SQL injection and was not found any in XSS.

In the following paragraphs we analyze in detail each fault type discussing the conditions/locations where each fault was observed in our field study. This discussion provides useful insights to support the future definition of realistic vulnerability fault models, which are essential to allow the development of realist attack injectors. Such kind of tools (which currently do not exist yet) can be potential very useful to simulate attacks by injecting realistic vulnerabilities, as help validating intrusion detection systems and other security mechanisms.

Missing function call extended (MFC extended). This fault type is typically observed in situations where the patch code consists of a missing function returning a value that will be used in the code. One important point is the fact that this defect does not cause any compiling or executing errors. The missing function is always related to the filtering one of the arguments. Whenever it has more than one argument the other arguments are the configuration of the filtering.

We define four sub-types in which the functions that change the variable type to numeric are differentiated from the other functions. This is done because there is a common exploitation of numeric variables that also allow string values.

Following are the constraints of the sub-types:

A. **Missing casting to numeric of one variable to itself** using the "`(int)`" type cast or using the "`intval()`" PHP function. It can also be considered as being the same variable "`$var`" situations like those using the "`$_GET[$var]`". This was found when the patch added the entire assignment line, for example:
"`$var=(int)$_GET[$var];`"
or when the patch replaced one variable in a string concatenation, for example:
Replace "…"`'str1'.$var'str2'";`" with
"…"`'str1'.intval($var).'str2'";`"

B. **Missing assignment of one variable to a function processing that same variable.** This was also found when replacing a variable that was part of a string concatenation with a function processing that same variable. The functions can also act as arguments of other functions. For example, when the patch added the function "`func2()`":
"`$var1 = func2(func1(($var1));`"

C. **Missing casting to numeric of one variable to another variable** using the "`(int)`" type cast or using the "`intval()`" PHP function. This was found when the patch added the entire assignment line, for example:
"`$var1=(int)$_GET[$var2];`"
or when the patch replaced one variable in a string concatenation, for example:
Replace "…"`'str1'.$var1'str2'";`" with
"…"`'str1'.intval($var2).'str2'";`"

D. **Missing assignment of one variable to a function processing another variable.** This was also found when replacing a variable that was part of a string concatenation with a function processing another variable. The functions can act as arguments of other functions. For example, when the patch added the function "`func2()`":
"`$var1 = func2(func1(($var2));`"

Wrong variable used in parameter of function call (WPFV). This is typically found when the following changes occurred in the argument of a function:

A. **Adding "'" to variables inside a string argument of a SQL query.** For example:
Replace "`func("SELECT…FROM…WHERE id=$var")`" with
"`func("SELECT…FROM…WHERE id='$var'")`"

B. **Changing the regex string of a function argument.** When the patch code is a change in the regex string of a function argument. This function can be a custom made function that processes a regex string or one of the PHP functions "`preg_replace`" and "`preg_match`". In the analyzed code the regex string was used to check a variable closely related to an input value, looking for known suspicious strings that were part of an attack.

C. **Changing the sub-string of a function**

argument. When the argument of the function is the result of the concatenation of several strings and variables and the patch code removed or changed one of them.

D. **Changing the PHP superglobal variable when it is an argument of a function.** When the argument of the function contains the PHP superglobal variable "`$_SERVER`" and it is changed. For example:
Replacing "`func($_SERVER[var1])`" with "`func($_SERVER[var2])`"

Missing if construct plus statements (MIFS). This fault type was found only when an IF condition and just one or two surrounding statements were missing.

A. **Missing traditional "if…then…else" condition.** When it is a traditional "`if…then…else`" condition, an "`elsif`" or an "`else`"
B. **Missing "if…then…else" condition in compact form.** This fault type was also found when the condition is in the compact form, for example:
"`(($var != '') ? 'true' : 'false')`"

Wrong value assigned to variable (WVAV). This is typically found when the following situations changed the variable assignment:

A. **Insertion of a new pattern in a regex string assigned to a variable.** In the analyzed code the regex string was used to check a variable closely derived from an input value, looking for known XSS attacks.
B. Missing or extraneous value in an array or a concatenation of a new substring inside a string.
C. **Changing the PHP superglobal variable when assigned to a variable.** When the variable is assigned to a PHP superglobal variable "`$_SERVER`" and it is changed. For example:
Replace "`$var1=$_SERVER[$var2];`" with "`$var1=$_SERVER[$var3];`"
D. **Adding "'" to variables inside a string in a SQL query assignment.** For example:
Replace: "`SELECT…FROM…WHERE id=$var`" with
"`SELECT…FROM…WHERE id='$var'`"
E. **Destroying the variable.** For example:
"`unset($var);`"
F. **Removing the concatenation operator "." in an assignment.** For example:
Replacing "`$var .= …`" with "`$var = …`"
G. **Replacing an array variable with a scalar variable.** For example:
Replacing "$var=`$members[$i];`" with "$var=`$memberval;`"

Extraneous function call (EFC). When this fault type was found the extraneous function returned the same data type of the argument. This was found when the function was replaced with a variable which has already been sanitized. Another situation found was the removal of a function when there is a function of a function.

Wrong function called with same parameters (WFCS). When this fault type was found the function was replaced by another function while keeping the same arguments, even when the function is the only statement in the line. In all these situations the new function was a custom made function that was already in the code or it was implemented in the patch. The new function was always related to filtering the argument.

Missing "AND EXPR" in expression used as branch condition (MLAC). When this fault type was found there was a missing AND expression inside an IF condition.

Missing variable initialization using a value (MVIV). In PHP there is no need to declare a variable and the variable stays uninitialized (with the default value) until the first assignment. Variables have a default value of their type (false, zero, empty string or an empty array). This fault type was found when there was a missing first assignment of a variable to an empty string, or an empty array.

Missing function call (MFC). This fault type was observed in the situations where the patch code consists of a missing function being the only statement in its line of code. The function did not return any value and, therefore it was not assigned to any variable. The missing function was always custom made and its implementation was most of the times created by the patch.

Missing if construct around statements (MIA). This fault type was found only when an IF condition was missing, surrounding only one statement that is already there in the code.

Missing "OR EXPR" in expression used as branch condition (MLOC). This fault type was found when there was a missing OR expression inside an IF condition.

Extraneous "OR EXPR" in expression used as branch condition (ELOC). This fault type was considered when there was an extraneous OR expression inside an IF condition.

4. Conclusion

This paper analyzes the vulnerabilities of six web applications using their past 679 security fixes as the field data. Results show that only a small subset of 12 generic software faults is responsible for all the security problems (XSS and SQL injection). We found considerable differences by comparing the distribution of the fault types of our results with studies of common software faults. We also detected that one of the Missing Function Call fault types (MFC extended) is responsible for 73% of all the security problems analyzed. The fault types are thoroughly detailed providing enough information for the definition of vulnerability fault models that can be used by researchers of realistic attack injectors.

For future work we propose other studies searching for patterns in the code responsible for security problems. These studies could follow the same methodology presented here but aimed at vulnerabilities in operating systems and their applications.

References

[1] Valeur, F., Mutz, D., Vigna, G.: "A Learning-Based Approach to the Detection of SQL Attacks". DIMVA 2005

[2] Christey, S., "Unforgivable Vulnerabilities", Black Hat Briefings 2007

[3] Zanero, S., Carettoni, L., Zanchetta, M., "Automatic Detection of Web Application Security Flaws", Black Hat Briefings 2005

[4] David, P., Stroud, R., "Conceptual Model and Architecture of MAFTIA", LAAS-CNRS, 2003

[5] Jovanovic, N., Kruegel, C., Kirda, E., "Precise Alias Analysis for Static Detection of Web Application Vulnerabilities", IEEE Symposium on Security and Privacy, 2006

[6] Vnunet, August, 2007, http://www.vnunet.com/vnunet/news/2197408/monster-kept-breach-secret-five

[7] The Privacy Rights Clearinghouse, December, 2007, http://www.privacyrights.org/ar/ChronDataBreaches.htm

[8] Stock, A., Williams, J., Wichers, D., "OWASP top 10", OWASP Foundation, July, 2007

[9] Steve, C., Martin, R., "Vulnerability Type Distributions in CVE", Mitre report, May, 2007

[10] Acunetix Ltd, February 12, 2007, http://www.acunetix.-com/news/security-audit-results.htm

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

[12] Christmansson, J., Chillarege, R. "Generation of an Error Set that Emulates Software Faults", Proc. of the 26th IEEE Fault Tolerant Computing Symposium – FCTS-26, Sendai, Japan, 1996.

[13] Chillarege, R., Bhandari, I. S., Chaar, J. K., Halliday, M. J., Moebus, D., Ray, B., Wong, M., "Orthogonal Defect Classification – A Concept for In-Process Measurement", IEEE Transactions on Software Engineering, vol. 18, no. 11, pp. 943-956, November 1992.

[14] Durães, J., Madeira, H., "Emulation of Software Faults: a Field Data Study and a Practical Approach", Transactions on Software Engeneering TSE, 2006.

[15] Fonseca, J., Vieira, M., Madeira, H., "Testing and comparing web vulnerability scanning tools for SQL injection and XSS attacks", The 13th IEEE Pacific Rim International Symposium on Dependable Computing, December 2007.

[16] Mays, R., Jones, C., Holloway, G., Strudinsky, D., "Experiences with defect prevention", IBM syst. J., vol 29, 1990

[17] PHP-Nuke, December, 2007, http://phpnuke.org/

[18] Drupal, December, 2007, http://drupal.org/

[19] PHP-Fusion, December, 2007, http://PHP-Fusion.co.uk/

[20] Wordpress, December, 2007, http://wordpress.org/

[21] phpMyadmin, December, 2007, http://www.phpmyadmin.net/

[22] phpBB, August, 2007, http://www.phpbb.com/

[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] blogsecurity.net, December, 2007, http://blogsecurity.net/wordpress/articles/article-230507/

[26] The PHP Group, December, 2007, http://pt.php.net/register_globals