FACULDADE DE CIÊNCIAS
E TECNOLOGIA DA
UNIVERSIDADE DE
COIMBRA

DEPARTAMENTO DE
ENGENHARIA
INFORMÁTICA

Using Web Security Scanners to Detect Vulnerabilities in Web Services

Marco Vieira
Nuno Antunes
Henrique Madeira

Internal Report
DEI

Coimbra - 2009

Vieira, M., Antunes, N., Madeira, H., “Using Web Security Scanners to Detect Vulnerabilities in Web Services”, (accepted) IEEE/IFIP Intl Conf. on Dependable Systems and Networks, DSN 2009, Lisbon, Portugal, June 2009; Internal report available at http://eden.dei.uc.pt/~mvieira

Using Web Security Scanners to Detect Vulnerabilities in Web Services

Marco Vieira, Nuno Antunes, and Henrique Madeira
CISUC, Department of Informatics Engineering
University of Coimbra – Portugal
mvieira@dei.uc.pt, nmsa@student.dei.uc.pt, henrique@dei.uc.pt

Paper category: ***Practical Experience Report***

Contact author: Marco Vieira
Address: DEI/FCTUC – Polo2
Universidade de Coimbra
3030-290 Coimbra, Portugal
Phone/fax: +351 239 790 000 / +351 239 701 266

The material included in this paper has been cleared through authors' affiliations.
Word count: ~4900
Keywords: Web services, security, vulnerabilities, vulnerability scanners, penetration testing.

Abstract

Although web services are becoming business-critical components, they are often deployed with critical software bugs that can be maliciously explored. Web vulnerability scanners allow detecting security vulnerabilities in web services by stressing the service from the point of view of an attacker. However, research and practice show that different scanners have different performance on vulnerabilities detection. In this paper we present an experimental evaluation of security vulnerabilities in 300 publicly available web services. Four well know vulnerability scanners have been used to identify security flaws in web services implementations. A large number of vulnerabilities has been observed (177), which confirms that many services are deployed without proper security testing. Additionally, the differences in the vulnerabilities detected and the high number of false-positives (35% and 40% in two cases) and low coverage (less than 20% for two of the scanners) observed highlight the limitations of web vulnerability scanners on detecting security vulnerabilities in web services.

1. Introduction

Ranging from on-line stores to large corporations, web services are increasingly becoming a strategic vehicle for data exchange and content distribution [1].

Web services provide a simple interface between a provider and a consumer [1] and are supported by a complex software infrastructure, which typically includes an application server, the operating system and a set of external systems (e.g., databases, payment gateways, etc). The Simple Object Access Protocol (SOAP) [1] is used for exchanging XML-based messages between the consumer and the provider over the network (using for example http or https protocols). In each interaction the consumer (client) sends a request SOAP message to the provider (server). After processing the request, the server sends a response message to the client with the results. A web service may include several operations (in practice, each operation is a method with several input parameters) and is described using WSDL (Web Services Definition Language) [2], which is a XML format used to generate server and client code, and for configuration. A broker is used to enable applications to find web services.

Web services become so widely exposed that any existing security vulnerability will most probably be uncovered and exploited by hackers. To prevent vulnerabilities, developers should apply best coding practices, perform security reviews of the code, execute penetration tests, use code vulnerability analyzers, etc. However, many times developers focus on the implementation of functionalities and on satisfying the user's requirements and disregard security aspects. Additionally, numerous developers are not specialized on security and the common time-to-market constraints limit an in depth test for security vulnerabilities.

Security vulnerabilities like SQL Injection and XPath Injection seem particularly relevant in web services as they are directly related to the way the web service code is structured [3][4]. Basically, SQL Injection and XPath Injection attacks take advantage of improper coded applications to change SQL commands that are sent to the database or tamper XPath queries used to access parts of an XML document.

There are two main approaches to test web applications for vulnerabilities:

– **White box testing**: consists of the analysis of the source code of the web application. This can be done manually or by using code analysis tools like FORTIFY [5], Ounce [6] or Pixy [7]. The problem is that exhaustive source code analysis may be difficult and cannot find all security flaws because of

the complexity of the code.

– **Black box testing**: consists in the analyses of the execution of the application in search for vulnerabilities. In this approach, also known as penetration testing, the scanner does not know the internals of the web application and it uses fuzzing techniques over the web HTTP requests.

Web vulnerability scanners are often regarded as an easy way to test applications against vulnerabilities. In fact, vulnerability scanners provide an automatic way to search for vulnerabilities avoiding the repetitive and tedious task of doing hundreds or even thousands of tests by hand for each vulnerability type. Most of these scanners are commercial tools (e.g., Acunetix Web Vulnerability Scanner [8], IBM Rational AppScan [9], and HP WebInspect [10]), but there are also some free application scanners (e.g., Foundstone WSDigger [11] and wsfuzzer [12]) with limited use, as they lack most of the functionalities of their commercial counterparts.

Previous research suggests that the effectiveness of scanners in the detection of vulnerabilities varies a lot. In [13] it is proposed a method to evaluate and benchmark automatic web vulnerability scanners using software fault injection techniques. Software faults are injected in the application code and the web vulnerability scanning tool under evaluation is executed, showing their strengths and weaknesses concerning coverage of vulnerability detection and false positives. Three leading commercial scanning tools were evaluated and the results showed that in general the coverage is low and the percentage of false positives is very high. However, this study was focused on a specific family of applications, namely database centric web based applications written in PHP, and the results obtained cannot be easily generalized, especially if we take into account the specificities of web services environments.

In this work we used four commercial vulnerability scanners (including two different versions of a given brand) to identify security flaws in the implementation of 300 publicly available web services. The goal was to study the effectiveness of the scanners and to try to identify common types of vulnerabilities in web services environments. The results obtained showed that many of the services tested were deployed without proper security testing as a large number of vulnerabilities has been detected (177). A key observation was that different scanners detect different vulnerabilities, which indicates that the coverage of an individual scanner is far from being perfect. The rate of false-positives (vulnerabilities detected that did not exist) and coverage (vulnerabilities that existed and were not detected) observed highlight some limitations of web vulnerability scanners on detecting vulnerabilities in web services.

In summary, this practical experience report focuses on the following three questions:

– What is the coverage of the vulnerability scanners tested when used in a web services environment?
– What is the false-positive rate of the web vulnerability scanners tested when used in a web services environment?
– What are the most common types of vulnerabilities in web services environments?

The structure of the paper is as follows. The next section presents the experimental study. Section 3 presents and discusses the results. Section 4 concludes the paper and presents ideas for future work.

2. The experimental study

Our experimental study consisted of four steps:

1. **Preparation**: select the vulnerability scanners and a large set of publicly available web services.
2. **Execution**: use the vulnerability scanners to scan the services to identify potential vulnerabilities.
3. **Verification**: perform manual testing to confirm that the vulnerabilities identified by the scanners do exist (i.e., are not false positives).
4. **Analysis**: analyze the results obtained and systematize the lessons learned.

2.1. Web services tested

A set of 300 publicly available services was tested. The following bullets point out some important aspects concerning the web services selected:

– Several technologies were considered, including .NET, Java, and Delphi.
– Web services are owned by different relevant parties, including Microsoft, Google, and Xara.
– Some web services implement the same functionally (e.g., Text Disguise and Free Captcha Service).
– A small number of web services are used in real businesses in the field (e.g., Portuguese Postal Office Orders Cost, Mexican Postal Codes).

The web services selection was as random as possible. The following paragraphs describe the web services selection process.

The first step was to identify a large set of web services. The first source was a web site that lists publicly available web services (*http://xmethods.net/*). From this list we have selected the first set of 450 web services. Then we used the Web Services Search Engine (*http://seekda.com/*) to discover additional services. seekda is a portal that enables searching for public web services based on the services' description. To discover services we have used a large set of generic

keywords, ranging from popular tags (e.g., *business*, *tourism*, *commercial*, *university*, etc) to queries with countries (e.g., *country:AR*, *country:PT*, *country:US*, etc) and keywords related to company names (e.g., *Oracle*, *Sun*, *Microsoft*, *Google*, *Acunetix*).

The resulting list included 6180 web services. As it was not possible to test such a large set of services we decided to randomly select 300 services from this list. However, some of the services initially selected had to be discarded due to several reasons, namely:

- **Invalid/malformed WSDL**: at least one of the scanners could not parse the WSDL description.
- **Unable to retrieve WSDL**: at least one of the scanners could not find the WSDL.
- **No methods found**: the web service has no operations to scan.
- **Authentication required**: some operations require authentication, which means that the service cannot be fully tested. This prevents the observation of vulnerabilities in potentially interesting functionalities of services (the private functionalities), which would obviously be of high interest.
- **Unhandled exception**: the tests abort and/or one of the scanners closes unexpectedly.
- **Communication errors**: at least one of the scanners cannot complete requests due to problems in HTTP Request or SSL Connection Error.
- **Scanning problems**: at least one of the scanners reported a generic error like "unable to add transaction" or "unable to render object".
- **Testing duration**: due to practical reasons, we decided to drop the services for which the scanning process takes more than two hours.

As the goal was to test 300 web services, the services discarded were replaced by others randomly selected from the initial list. 316 services were explored and had to be discarded due to: invalid/malformed WSDL (117); unable to retrieve WSDL (30); no methods found(20); authentication required (40); unhandled exception (19); communication errors (50); scanning problems (24); and testing duration (15).

Due to space restrictions, the list of web services tested is not included in this paper. It is available at [14] together with the detailed results obtained.

2.2. Vulnerability scanners studied

In the present study we have tested four commercial web vulnerability scanners widely used, including two different versions of a specific brand. The following paragraphs briefly introduce these scanners.

HP WebInspect "*performs web application security testing and assessment for today's complex web applications, built on emerging Web 2.0 technologies. HP WebInspect delivers fast scanning capabilities, broad security assessment coverage and accurate web application security scanning results*" [10]. This tool includes pioneering assessment technology, including simultaneous crawl and audit (SCA) and concurrent application scanning. It is a broad application that can be applied for penetration testing in web-based applications.

IBM Rational AppScan "*is a leading suite of automated Web application security and compliance assessment tools that scan for common application vulnerabilities*" [9]. This tool is suitable for users ranging from non-security experts to advanced users that can develop extensions for customized scanning environments. IBM AppScan can be used for security testing in web applications, including web services.

Acunetix Web Vulnerability Scanner "*is an automated web application security testing tool that audits your web applications by checking for exploitable hacking vulnerabilities*" [8]. Besides web services, Acunetix Web Vulnerability Scanner can be applied for security testing in web applications in general.

For the results presentation we have decided not to mention the brand and the versions of these scanners to assure neutrality and because commercial licenses do not allow in general the publication of tool evaluation results. This way, the scanners presented above are referred in the rest of this paper as VS1.1, VS1.2, VS 2, and VS3 (without any order in particular). Vulnerability scanners VS1.1 and VS1.2 refer to the two versions of the same product.

3. Results and discussion

The following subsections present and discuss the results obtained and the lessons learned during this study. Due to space constraints we do not present the detailed results (i.e., results for the individual services). Interested readers can obtain them at [14].

3.1. Overall results analysis

Table 1 presents the overall results of the study. For each scanner it is presented the total number of vulnerabilities and the number of services in which those vulnerabilities were found. The scanners pointed six different types of vulnerabilities, namely:

- **SQL Injection**: it is possible *"to alter the construction of backend SQL statements"* [15]. An attacker can read or modify database data and, in some cases, execute database administration operations or commands in the system [4].
- **XPath Injection**: it is possible to modify an XPath query to *"be parsed in a way differing from the*

Vulnerability Types	VS1.1		VS1.2		VS2		VS3	
	# Vuln.	# WS	# Vuln.	# WS	# Vuln.	# WS	# Vuln.	# WS
SQL Injection	217	38	225	38	25	5	35	11
XPath Injection	10	1	10	1				
Code Execution	1	1	1	1				
Possible Parameter Based Buffer Overflow							4	3
Possible Username or Password Disclosure							47	3
Possible Server Path Disclosure							17	5
Total	**228**	**40**	**236**	**40**	**25**	**5**	**103**	**22**

Table 1. Overall results.

programmer's intention" [15]. Attackers may gain access to information in XML documents [4].

- **Code Execution**: it is possible to manipulate the application inputs to trigger server-side code execution [4]. An attacker can exploit this vulnerability to execute malicious code in the server machine.
- **Buffer Overflow**: it is possible to manipulate inputs in such a way that causes buffer allocation problems, including overwriting of parts of the memory [4]. An attacker can exploit this causing Denial of Service or, in worst cases, *"alter application flow and force unintended actions"* [15].
- **Username/Password Disclosure**: the web service response contains information related to usernames and/or passwords. An attacker can use this information to get access to private data [4].
- **Server Path Disclosure**: the response contains a fully qualified path name to the root of the server storage system. An attacker can use this info to discover the server file system structure and devise other security attacks [4].

An additional result provided by the scanners is related to application errors obtained due to invalid parameters. Scanners classify these situations as low importance security issues. In fact, these errors are due to the fact that input parameters are not correctly checked (i.e., the service may have robustness problems). As shown in Figure 1, the four scanners detected different application errors (although the two versions of the same scanner have similar results). This is obviously dependent on the tests performed by each scanner. Although in few cases application errors can be maliciously explored (e.g., if the error message discloses sensitive information about the service code or structure), in most cases they are not related to security vulnerabilities. Thus, in the present work we have decided to exclude them from further analysis.

As we can see in Table 1, different scanners report different types of vulnerabilities. The only exception is SQL Injection that is reported by all scanners. This is a first indicator that scanners implement different forms of penetration tests and that the results from different scanners may be difficult to compare. Nevertheless, results suggest that in order to have a good coverage the user will have to use several scanners instead of relying in only one.

Scanners VS1.1 and VS1.2 (two different versions of the same brand) are the only ones that detected XPath Injection vulnerabilities. An important aspect is that, when compared to SQL Injection, the number of XPath-related vulnerabilities is quite small. In fact, XPath vulnerabilities were detected in a single service, indicating that most web services make use of a database instead of XML documents to store information.

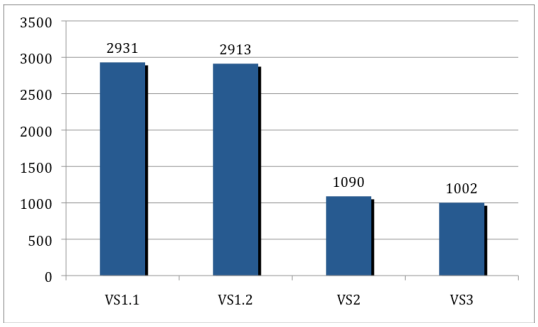


Figure 1. Application errors detected.

Scanners VS1.1 and VS1.2 detected a code execution vulnerability. This is a particularly critical vulnerability that allows attackers to execute code in the server. After discovering this vulnerability we performed some manual tests and we were amazed by the possibility of executing commands (e.g., '*cat /etc/passwd*', '*ls -la*') and get the corresponding answer in a readable format.

VS3 was the only one pointing vulnerabilities related to buffer overflow, username and password disclosure, and server path disclosure.

As mentioned before, SQL Injection is the only type of vulnerability that was detected by the four scanners used. However, different scanners reported different vulnerabilities in different web services. In fact, the number of SQL Injection vulnerabilities detected by VS1.1 and VS1.2 is much higher than the number of vulnerabilities detected by VS2 and VS3. Thus we decided to look at this vulnerability type in more detail.

The intersection areas of the circles in Figure 2 represent the number of vulnerabilities detected by more than one scanner (the number of vulnerabilities detected is shown; zero is the value when no number is presented). Note that the area of each circle is roughly proportional to the number of vulnerabilities detected, but there is no correspondence between the size of the intersection areas and the number of vulnerabilities (it is too complex to represent graphically).

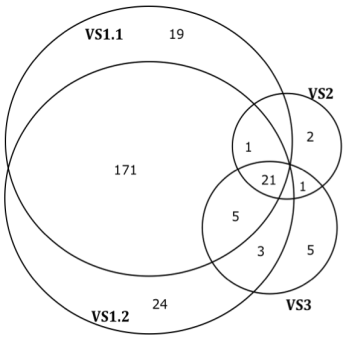


Figure 2. SQL Injection vulnerabilities.

Figure 2 clearly shows that the four scanners detected different sets of SQL Injection vulnerabilities and the differences are considerable, pointing again to relatively low coverage of each vulnerability scanner individually. In fact, even for VS1.1 and VS1.2, two consecutive versions of the same scanner, there are considerable differences. VS1.1, the older version, detected 19 SQL Injection vulnerabilities that were not detected by VS1.2. On the other hand, VS1.2 detected 27 vulnerabilities that were not detected by VS1.1.

3.2. False positives analysis

The results presented so far do not consider false positives (i.e., situations where scanners detected a vulnerability that in the reality does not exist). However, it is well known that false positives are very difficult to avoid. This way, we decided manually confirm the existence (or not) of each vulnerability detected.

Confirming the existence of a vulnerability without having access to the source code is a difficult task. Thus, we defined a set of rules and corresponding checks to classify the vulnerabilities detected by the scanners in three groups: a) Confirmed false positives, b) Confirmed vulnerabilities, and c) Doubtful.

Detected vulnerabilities are classified as false positives if meet one of the following cases:

- For SQL Injection vulnerabilities, if the error/answer obtained is related to an application robustness problem and not to a SQL command (e.g., a *NumberFormatException*).
- The error/value in the web service response is not caused by the elements "injected" by the scanner. In other words, the same problem occurs when the service is executed with valid inputs.
- For path and username/password disclosure, the information returned by the service is equal to the information submitted by the client (e.g., the vulnerability scanner) when invoking the web service. In other words, there is no information disclosure.

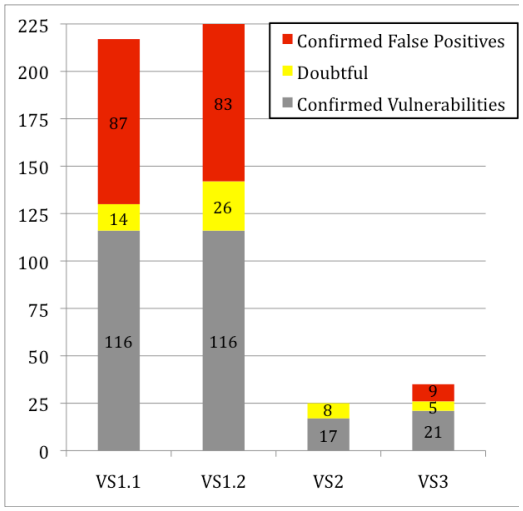
Detected vulnerabilities are classified as confirmed vulnerabilities if satisfy one of the next conditions:

1. For SQL Injection vulnerabilities, if it is possible to observe that the SQL command executed was invalidated by the values "injected" by the scanner (or manually). This is possible if the SQL command or part of it is included in the web service response (e.g., stack trace).
2. For SQL Injection vulnerabilities, if the "injected" values lead to exceptions raised by the database server.
3. If it is possible to access unauthorized services or web pages (e.g., by breaking the authentication process using SQL Injection).
4. For Path disclosure, if it is possible to observe the location of folders and files in the server.
5. For XPath Injection, if the "injected" values lead to exceptions raised by the XPath parser.
6. For Buffer Overflow, if the server does not answer to the request or raises an exception specifically related to buffer overflow.

If none of these rules can be applied then there is no way to confirm whether a vulnerability really exists or not. These cases were classified as doubtful. Figure 3 shows the results for SQL Injection vulnerabilities.

As we can see, the number of vulnerabilities that we were not able to confirm (doubtful cases) is low for VS1.1, VS1.2, and VS3 (always less than 15%), but considerably high for VS2 (32%). This means that the false positive results are relatively accurate for the first three scanners, but it is an optimistic figure (zero false positives) for scanner VS2. Obviously, we can also read the false positive results shown in Figure 3 as a range, going from an optimistic value (confirmed false positives) to a pessimistic value (confirmed false positives + doubtful cases).

The number of (confirmed) false-positives is high



Scanner	% FP
VS1.1	40.09
VS1.2	36.89
V2	0
VS3	25.71

Scanner	% Doubtful
VS1.1	6.45
VS1.2	11.56
V2	32
VS3	14.29

Figure 3. False positives observed for SQL Injection.

for scanners VS1.1 and VS1.2, and is also high for VS3, in relative terms. Scanner VS2 shows zero confirmed false positives, but it detected a fair percentage (8 out of 25) of vulnerabilities that were classified as doubtful, thus a pessimistic interpretation of results is that 8 out of 25 vulnerabilities may be false positives. Obviously, the low number of vulnerabilities detected by VS2 and VS3 (25 and 35 respectively) also limits the absolute number of false positives.

Table 2 presents the false positive results for the other vulnerabilities. In this case, we were able to confirm the existence (or inexistence) of all vulnerabilities and no doubts remained. An interesting aspect is that all XPath injection and Code Execution vulnerabilities were confirmed. On the other hand, all vulnerabilities related to username and password disclosure were in fact false positives (in all cases the username/information information returned is equal to the one sent by the scanner).

Vulnerability	Scanners	Confirmed	F. P.
XPath Injection	VS1.1 & VS1.2	10	0
Code Execution	VS1.1 & VS1.2	1	0
Buffer Overflow	VS3	1	3
Username/Password Discl.	VS3	0	47
Server Path Disclosure	VS3	16	1

Table 2. False positives.

Figure 4 presents the SQL Injection vulnerabilities intersections after removing the false positives. The doubtful situations were in this case considered as existing vulnerabilities (i.e., optimistic assumption from the point of view of scanners detection effectiveness). Results clearly show that, even if we manually remove the false positives, the four scanners still report different vulnerabilities. An interesting result is that three vulnerabilities were detected by VS1.1 and were not detected VS1.2 (the newer version of the scanner). The reverse also happens for 15 vulnerabilities, which is expectable as a newer version is expected to detected more vulnerabilities than an older one (but that should happen without missing any of the vulnerabilities identified by the older version, which was not the case). This results called our attention and we tried to identify

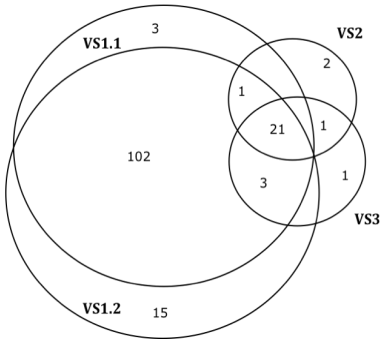


Figure 4. SQL Injection vulnerabilities without FP.

the reasons. After analyzing the detailed results we concluded that all of these 18 vulnerabilities are in the group of the doubtful ones (maybe they are really false positives, but we were not able to demonstrate that), preventing us from drawing a definitive conclusion.

3.3. Coverage analysis

A key aspect is to understand the coverage of the vulnerabilities detected. Coverage compares the number of vulnerabilities detected against the total number of vulnerabilities. Obviously, in our case it is impossible to know how many vulnerabilities were not disclosed by any of the scanners (we do not have access to the source code). Thus, it is not possible to calculate the coverage. However, it is still possible to make a relative comparison based on the data available.

In practice, we know the total number of vulnerabilities detected (which correspond to the union of the vulnerabilities detected by the four scanners after removing the false positives) and the number of vulnerabilities detected by each individual scanner. Based on this information it is possible to get an optimistic coverage indicator for each scanner (i.e., the real coverage will be lower than the value presented). Obviously, this is relevant only for SQL Injection vulnerabilities as it is the only type that is detected by all the scanners.

Table 3 presents the coverage results. As shown, 149 different SQL Injection vulnerabilities were detected (as before, we decided to include the doubtful situations as existing vulnerabilities). Each scanner detected a subgroup of these vulnerabilities, resulting in partial detection coverage. VS1.1 and VS1.2 present quite good results. On the other hand, the coverage of VS2 and VS3 is very low.

Scanner	# SQL Injection Vuln.	Coverage %
VS1.1	130	87.2%
VS1.2	142	95.3%
VS2	25	16.8%
VS3	26	17.4%
Total	**149**	**100.0%**

Table 3. Coverage for SQL Injection.

3.4. Lessons learned

The results presented before allowed us to observe some interesting aspects. The first observation is that different scanners detected different types of faults. SQL Injection was the only type that was detected by all scanners. The two scanners of the same brand (VS1.1 and VS1.2) were the only ones that detected XPath and code execution vulnerabilities. Only one scanner (VS3) detected vulnerabilities related to buffer overflow, username and password disclosure, and

server path disclosure. VS2 only detected SQL Injection vulnerabilities.

SQL Injection vulnerabilities are the dominant type in the web services tested (see Figure 5). However, different scanners detected different vulnerabilities of this type. In fact, VS1.1 and VS1.2 detected a huge number of vulnerabilities (215 and 225 respectively) while VS2 and VS3 detected a very low number (25 and 35 respectively).

A key observation was a very large number of false positives. In fact, for three of the scanners the percentage of false positives was more than 25%. VS2 presented zero false positives, but 8 out of the 25 SQL Injection vulnerabilities detected by this scanner remained as doubtful (i.e., could not be manually confirmed as real vulnerabilities nor as false positives).

A very low coverage, lower than 18%, was observed for two of the scanners (VS2 and VS3), while the other two scanners (VS1.1 and VS1.2) present a coverage superior to 87%. Note that this value represents an optimistic coverage, as the real coverage of the tested scanners (at least for the 300 web services used in the experiments) is definitely lower than the value observed.

Figure 5 presents the final distribution of vulnerabilities per type, after removing the confirmed false positives but including the doubtful cases (i.e., optimistic evaluation of the scanners). As the doubtful cases only affect the SQL Injection, it means that the number of SQL injection vulnerabilities could be overestimated. Scanners have found 177 different vulnerabilities in 25 different services, which represent approximately 8.33% of the tested services. As mentioned before, the predominant vulnerability is SQL Injection, representing 84.18% of the vulnerabilities found. This is a very important observation due to the high number of cases found and the high severity of this vulnerability.

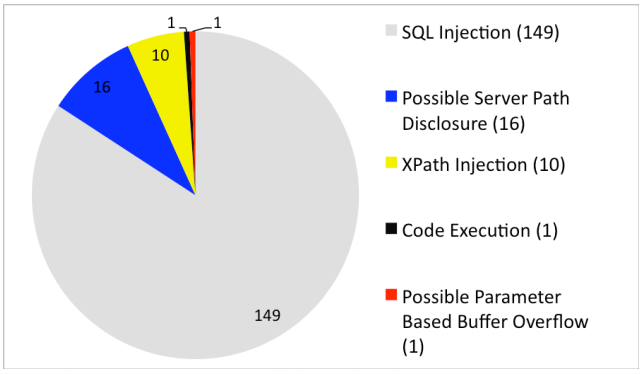


Figure 5. Vulnerabilities distributed per type.

4. Conclusions

In this paper we presented an experimental study on using four commercial vulnerability scanners to detect security vulnerabilities in 300 publicly available web services, covering a diversity of technologies used to implement the web services and including services used in real business. A large number of vulnerabilities was observed, confirming that many services are deployed without proper security testing.

Results showed that selecting a vulnerability scanner for web services is a very difficult task. First, different scanners detect different types of vulnerabilities. Second, the number of false positives is quite high, which reduces the confidence on the precision of the vulnerabilities detected. Finally, the coverage is in some cases very low, suggesting that many vulnerabilities probably remain undetected. A final observation is that SQL Injection vulnerabilities are prevalent in the web services tested as they represent more than 84% of all vulnerabilities detected.

Future research include new experiments covering web service functionalities that require authentication, performed over a selected set of representative web services to which it is possible to get access rights under agreement with owners.

References

[1] D. A. Chappel, T. Jewell, “Java Web Services: Using Java in Service-Oriented Architectures”, O'Reilly. 2002

[2] E. Christensen et al., “Web Services Description Language (WSDL) 1.1”. 2001, W3C

[3] Stock, A., Williams, J., Wichers, D., “OWASP top 10”, OWASP Foundation, July, 2007

[4] Stuttard, D., Pinto, M., “The Web Application Hacker's Handbook: Discovering and Exploiting Security Flaws”, Wiley, ISBN-10: 0470170778, October 2007

[5] FORTIFY - http://www.fortifysoftware.com/

[6] Ounce, http://www.ouncelabs.com/

[7] Pixy, http://pixybox.seclab.tuwien.ac.at/pixy/

[8] Acunetix Web Vulnerability Scanner, 2008, http://www.acunetix.com/vulnerability-scanner/

[9] IBM Rational AppScan, 2008, http://www-01.ibm.com/software/awdtools/appscan/

[10] HP WebInspect, 2008, http://www.hp.com

[11] Foundstone WSDigger, 2008, http://www.foundstone.com/us/resources/proddesc/wsdigger.htm

[12] wsfuzzer, 2008, http://www.neurofuzz.com/modules/software/wsfuzzer.php

[13] Fonseca, J., Vieira, M., Madeira, H., "Testing and comparing web vulnerability scanning tools for SQL injection and XSS attacks", 13th IEEE Pacific Rim Dependable Computing Conference (PRDC 2007), Melbourne, Victoria, Australia, December 2007

[14] Antunes, N., Vieira, M., Madeira, H., “Web Services Vulnerabilities”, 2008, http://eden.dei.uc.pt/~mvieira

[15] Web Application Security Consortium, 2008, http://www.webappsec.org/projects/threat/classes_of_attack.shtml