Effectiveness of Automated Application Penetration Testing Tools

Alexandre Miguel Ferreira
Alexandre.MiguelFerreira@os3.nl

Harald Kleppe
Harald.Kleppe@os3.nl

February 6, 2011

Contents

Abstract

There are a wide range of tools available to assist penetration testing and security assessment of web applications. This study gives insight into the effectiveness of time spent using vulnerability scanning and penetration testing tools.

We selected a subset of the available tools and tested their abilities on our own web application, where we implemented different types of common vulnerabilities on top of an originally secure application built with PHP and MySQL.

We found that there are remarkable differences between the different tools, both in terms of quantifiable measures like vulnerabilities found but also qualitative figures such as presentation and relevance of results. Also, the results of our comparison does not comply with previous research in the sense that the detction rate was generally lower than what is seen in the results of other research.

Chapter 1

Introduction

Web enable applications is ubiquitous these days. Everything from ones social graph, shopping history and credit card info at online book stores to banking and communication with governmental institutions happens trough web applications. Ensuring confidentiality, integrity the availability of the data and services offered through these portals rely on the security measures built in to the applications. Whenever security is breached, it implies either huge economic, public relation or social impact for the organisations involved.

Auditing the security of web applications is in other words critical and should be treated a high priority. Rapid deployment of new code bearing new functionality often makes it expensive to asses the implemented security, even while utilizing tools and scanners to increase the efficiency of these process. There is already quite some research available on how different tools compare on results in terms of results alone. We will on the other hand try to shred light on how these tools can aid the automation of these tasks.

- Are automated penetration testing tools effective?
  - What and how is automated with these tools?
  - How much manual intervention is required from the results? (false positives / negatives)
  - What are the most effective tools?
  - What level of effectiveness is acceptable / necessary to properly support pentesters?

Our approach to this subject is to develop our own web application which we will use as a target for the vulnerability scanners. Building our own tool rules out the possibility that the vulnerability scanners are trained for or othervise already know of the spesific implementation of vulnerabilities availble. The scanning for vulnerabilities of our application will start of as soon as the target application is ready. Since we are focusing on automated penetration testing tools, we will refrain to teaching the tools their way around the application. Instead we will let them rely on their integrated crawling engines, which earlier research have established are quite reliable [20]. The nature in some of these tools require more interaction than others, but we will focus on the automated crawling and scanning functionality.

Chapter 2

Background

Before we start discussing our testing methodology, we will explain what a penetration test is, present an abstract model of tools specialized for this work and identify different types of vulnerabilities.

2.1 Web Application Vulnerabilities

In this section we will characterize some of the most common security problems related with web applications. This list is based on the OWASP Top 10 Project [12], that is a list with the 10 most common vulnerabilities related with web applications.

2.1.1 Injection

Injection flaws, such as SQL injection, are very common in web applications, as reported by OWASP in 2007 [11], reaching the second place, and in 2010, when it was considered the most common programming error in web applications. This kind of vulnerability occur when untrusted data is sent as part of a command or query. This data sent by the attacker can trick the interpreter when executed, for instance executing unplanned commands or changing important data.

2.1.2 Cross Site Scripting (XSS)

XSS flaws, as they are known, allow attackers to execute malicious Java Script code, pretending that the application is sending the code to the user. When a website is vulnerable to cross site scriping, an attacker is able to execute scripts in the victim's browser which can be used to hijack user's sessions, spoil web sites or possibly introduce worms, among others. It was considered the most common programming error in web applications in 2007 and the second most common in 2010 .

2.1.3 Insecure Direct Object References

When a developer displays important internal implementations to the user, such as URLs or files, he is doing a direct object reference. Without access control checking rights or other protections implemented, an attacker can adulterate these references to access unauthorized data, may cause great damage to the web application. It is a common programming error in web applications as shown by OWASP in 2007 and in 2010 .

2.1.4 Cross Site Request Forgery (CSRF)

A CSRF attack allows an attacker to send requests on behalf of a client without knowledge or intraction from the client. Subsequently can the attacker force the victim's browser to perform a hostile action, benefiting from this. These attacks can be as powerful as the web application that is being attacked. OWASP reported it as the fifth most common programming error in web applications in 2007 and in 2010 .

2.1.5 Security Misconfiguration

Security Misconfiguration is a situation where the configuration of either the application it self or the services it depend upon reveal technical information to a page visitor in the event of an error. This information can come in form of either stack traces or error messages directly from a database management system. This type of error was reported as the sixth most common vulnerability in the OWASP 2010 Top 10 list .

2.1.6 Failure to Restrict URL Access

Failure to Restrict URL Access occurs when web applications check pages access rights before presenting protected data, but do not perform this action each time the protected data is accessed. This way it is possible to an attacker access these hidden pages anyway. It appears in both 2007 and 2010 OWASP Top 10 lists.

2.2 Penetration Tests

A Penetration Test, often referred to as pentest, evaluates the security of applications by simulating malicious attacks. It is different from other types of auditing in the sense that penetration tester takes the attackers point of view, with only limited knowledge about the inner workings of an applications inner workings. This is different than, for example, a code review where the auditor has access to all aspects of the application. Penetration tests usually involve a significant amount of human intervention, but often assisted by different types of tools.

2.3 Penetration Testing Tools

Penetration testing tools are software developed to assist professionals during a penetration test. This could either mean tools that accomplished specific automated tasks or fully automated "point-and-shoot" solutions that, without human intervention, crawls the functionality in an application before trying to detect vulnerabilities.

Chapter 3

Target Application

To assess the penetration testing tools we needed an application in which we knew of vulnerabilities for the tools to find. There is long lists [4] of projects which already develop open source deliberately vulnerable applications, for example *HacmeBank* [7] and *WebGoat* [25]. We did not want to utilize these projects, for several reasons.

First, many of these projects are striving to produce applications which can teach developers security and not intended as targets for penetration testing tools. However, our most significant reason to avoid these solutions is that the penetration testing tools we will compare might be trained to know the vulnerable functionality in these applications and thereby give results that do not reflect the real situation. Another important reason to use a different application as the basis for our research is that results from scanning numerous vulnerable applications is needed to build a more complete picture of how the players in the vulnerability scanner market compares.

The application we are going to prepare have to fulfill the following requirements:

- Implement vulnerabilities in a realistic way
- Be representative of the web applications in use today in terms of functionality and of technologies used
- Have clearly defined vulnerabilities for assessment purposes

Two options are still available, either build a web application completly from scratch or build vulnerable functionality into or on top of an already working system. After analyzing these requirements we decided to implement our own application. It was built on Drupal [6], that is an open source PHP/MySQL content management system which meets all the above requirements. Drupal also strives to be easy to extend by nature, which enables us to build vulnerable functionality on top of it.

3.1 Vulnerabilities Implementation

Our target application contains implementations of a subset of the most common vulnerabilities found in web applications, already described earlier in this document. Appendix A presents an example of our application.The following subsections discuss our implementation in detail.

3.1.1 SQL Injection

The vulnerable functionality was implemented using both HTML forms and request parameters given in the URL. While the values given in the URL simply is not sanitized, and thereby could contain control characters is the second implementation also able to process more than one query. In other words giving an attacker the same permissions on the database as the web application has.

3.1.2 Cross Site Scripting (XSS)

To test this vulnerability we decided to implement it in two different ways, both explained below.

Stored XSS

Stored XSS happens when a malicious Java Script code is stored in the database. This vulnerability was implemented on a guest book page. A guest book page provides a way to receive feedback from all visitors and is also a good way to perform a stored XSS attack. The form used to submit feedback contains only a "Comment" field where we submitted our tests.

- **Unprotected Guest book**
  In this page the "Comment" field is not properly escaped, so it is really easy to exploit this vulnerability (for instance, by creating a comment containing Java Script code).
- **Protected Guest book**
  For assessment purposes, we also create a guest book completely protected against XSS. To assure that it is not possible to exploit this vulnerability in this page, we used both *mysql_real_escape_string*() and *htmlentities()* PHP functions on the "Comment" field.

Reflected XSS

The **Texts** page has this vulnerability implemented. JavaScript code embedded in the URL will be interpreted by a visitors browser. An URL like this: $\&inc =< script > alert(1) < /script >$, will execute JavaScript code which triggers an pop up message. This trivial example, but only JavaScript and an attackers imagination defines limits. URL shorteners like `http://bit.ly` and `http://goo.gl` amplify the impact of this vulnerability since it becomes impossible for a user to know if there is XSS incorporated in the shortened link.

3.1.3 Cross Site Request Forgery (CSRF)

Drupal's forms, like the log in form are already protected against CSRF. However we also implemented the **Choose your day** page that is also protected against this vulnerability, but let every other form on the page be vulnerable to this type of request. To protect it we add a unique token to a hidden field in the HTML form. This token acts as a shared secret between the server and the client browser. Anyone without knowledge of this one-time-token will not be able to successfully submit the request.

3.1.4 Path Traversal

The **Texts** page was created to test this flaw. This page includes a file that is given by a file name defined in the URL. Since there is no sanitizing on this file name, strings like *../../../db-connection-info.inc* can be given and might result in displaying the contents of the file, potentially revealing critical information.

3.1.5 Failure to Restrict URL Access

As explained before, it happens when a file, that is supposed to be accessed only by authorized users, is no properly protected. To test this kind of vulnerability we create the **Protected URL** page that, when the user is logged-in, presents an URL to another page that contains all the information about the users. However, this page does not check if the user is logged-in or not and in case the attackers know the URL to this page they can bypass the login process, making this file completely unprotected.

Chapter 4

Tool comparison

4.1 Application selection

Since we based our research on the OWASP Top 10 Project [12], the tools to be tested were restricted to only those which are related with web applications.

The following tools were tested (sorted alphabetically):

- Acunetix Web Vulnerability Scanner
- Burp Suite Pro
- Core Impact
- IBM AppScan
- NTOSpider
- Paros Free
- MileScan ParosPro Desktop
- Qualys Web Application Vulnerability Scan
- Skipfish
- w3af
- ZAProxy

This list of tools include both established tools that have been on the market for a long time, but also tools that are newer on the market. There are both open source and commercial tools, some tools are fully automated and even Software-as-a-Service offerings while others are proxies for a browser and requires some manuall intervention from the user. The next sections discuss our experiences with the tools in question and their findings.

4.2 Testing approach

When the target web application were ready were each of the scanners let loose on our application, with minimal training. For the tools we tested our self is the results based on a fully automated point-and-shoot approach, except teaching the scanners to authenticate. Some additional time were spent with a few of the tools, this is clearly noted in the text. The database were reset to a default starting point before each test. The web server were configured to only listen on the IP's of the expected scanning server to exclude any interference from other parties during the scan.

4.3 Test results

We will not compare the tools neck on neck with measures such as scan duration, total number of alerts etc. Scan duration and number of requests are inherently different because of the different approach the tools tasks. Regarding the results, one tool might excel on one specific application or on all applications built with one set of tools, while totally failing to detect vulnerabilities in other applications [24] . One scanner performing poorly in our setup might in other words just as well be superior when scanning another application and the reader should have this in mind while reading this result section, and should not be ruled out based on results from one scanned application. Although we will not compare the numbers from scan to scan directly will we discuss subjects such as scanner performance, platform constraints et cetera whenever it is noteworthy. However, on a more abstract level.

4.3.1 Acunetix Web Vulnerability Scanner 7

Overview	
Type	Automated scanner
Availability	Commercial
Price	€3100 (Consultant edition, 1year)
Version tested	7.0 Build 20101216

A Windows application with a GUI for both configuration, in-scan status and results. Teaching the scanner to authenticate with the web application is done by actually logging in to the application and mark excluded links (for ex. the log out link) in a special browser window. The vulnerability scanner itself is fully automated, but can be configured to scan for only specific vulnerability types and do it all in either Quick, Heuristic or Extensive mode which changes how many requests the scanner will make.

Acunetix detected only one SQL injection vulnerability from variables in the URL and the path traversal. Further, the results overview does not emphasize the right things in our opinion. Repeated alerts are counted as individual and the SQL injection vulnerability is reported twice and thereby appears as two alerts.

We tested scanning in both heuristic and extensive mode against our application and found the difference to be negligible both in terms of scanning time and number of requests. The SQL injection and the directory traversal were found in both scan modes. Extensive mode added another 18 alerts, which were all new instances of already mentioned vulnerabilities and thereby added no value.

4.3.2 Burp Suite Pro

Overview	
Type	Proxy with automation
Availability	Commercial
Price	€210 per user per year
Version tested	1.3.09

Burp Suite is a Java application, and thereby cross platform. It can act as a proxy and scan on demand from the user or automatically crawl and scan an application. Additional vulnerabilities were found when the proxy functionality were used to browse and scan the application.

Both XSS, both SQL injection and the directory traversal were detected, although the reflected XSS vulnerability were reported as stored and only found when Burp were used as a proxy and scanning were manually initiated on the given page. There were also made note of a "Possible XML injection" which is a false positive. This finding occurred several times, since it originates from the Drupal log in form which is present on every page before the user authenticates.

After all, we felt that the time we used with Burp Suite Pro were time very well spent. Having the scanner crawl and search the application revealed all but one of the vulnerabilities. Even though this somehow undermines the automatic crawling/scanning we do not think it obsoletes it. Letting the initial crawling/scanning do it's magic first, before adding special attention to interesting areas seems to bear fruits.

4.3.3 Paros Proxy

Overview	
Type	Proxy
Availability	Open Source
Price	Free
Version tested	3.2.13

Paros [13] is another Java application that acts as a proxy for ones browser. To perform the scan with this tool, the browser needs to be configured to utilize the proxy. Either one has to browse through the web application or start the scanner within Paros to build the list of URLs it will later scan for vulnerabilities.

Concerning the results, both XSS vulnerabilities and the printed error message were reported, although the reflected XSS vulnerability were reported as stored. A report with all the vulnerabilities found, and respective risk level, can be printed from the application. The printed error message and both of the reflected and the stored cross site scripting vulnerabilities were made note of by this tool.

4.3.4 MileSCAN ParosPro Desktop Edition

Overview	
Type	
Availability	Commercial
Price	$1,495 per year (unlimited IP license)
Version tested	1.8.0

The commercial version of Paros also acts as a proxy, but also has features to automatically crawl and scan an application. There does not seem to be very much relation between this program and the open sourced version. It is first of all a native Windows application, and not a java application as Paros Proxy, with all the pros and cons that implies. Another significant difference is its ability to interpreted JavaScript to extract links during the crawling, which is noteworthy even tough it does not add anything in our comparison where all links are in HTML.

The automated crawling and scanning with ParosPro revealed only the reflected XSS. Walking the scanner trough our web application with the browser configured with ParosPro as a proxy instead of relying on the built in scanner revealed one SQL injection vulnerability as well, but no mention of the other vulnerabilities in the application. The results report is available within ParosPro and can also be saved to a HTML website report.

4.3.5 Qualys Web Application Scanning

Overview	
Type	Software as a Service automated scanner
Availability	Commercial
Price	€500 per scan, yearly fee and fee to enable WAS
Version tested	Scanner 5.13.51-1, WAS 2.4.57-1, Web 6.16.60-1, Vulnsigs 1.27.157-2,

The Qualys tool is Software-as-a-Service solution. The interface for configuration of the application subject to be scanned is straight forward and the test results were available 15 minutes after we first logged in to the service.

Qualys Web Application Scanner detected directory traversal and reflected XSS. It also detected the SQL injection and explicitly takes note of the fact that the vulnerability will evaluate every query in the string.

The report from the Web Application Scan were a 52 pages long PDF which contains a mix of vulnerability summaries, printings of raw requests implementing the vulnerabilities found and general description of the vulnerability types. The results them self are drowned in a document that intended to impress both directors and technicians at once. Extracting the actual results from the report thereby requires quite some scrolling and generally feels cumbersome.

As noted in other papers [20], the newsletter introducing this product states "An embedded Web crawler parses HTML and some JavaScript to extract links". This is noteworthy because of only *some* JavaScript is evaluated. This could potentially keep the scanner from crawling, and thereby also scanning big chunks of potentially vulnerable functionality in an application. However, this have not affected the results in our test since our application provide all links without use of JavaScript

4.3.6 Skipfish

Overview	
Type	Automated scanner
Availability	Open source
Price	Free
Version tested	1.84b

Skipfish is a no-nonsense tool written in C. It is fully automated in the sense that it crawls the website itself and there is no need for or opportunity for human interaction with the scan flow. Only source code is available at the project website, but the compilation is quick, well documented and only depends on a small set of widely available libraries. The tool itself is started from the command line with a set of switches to set values affecting the scan properties.

Running Skipfish with an extensive word list means several millions requests have to be made. Skipfish itself is performing very good and has a quite small footprint, so performance are not likely to be an issue on the scanning host. Although it might mean trouble if the service one want to scan is slow. In our test setup with two virtual machines on the same LAN, we had around 285 requests per second which gave a total scanning time of just more than 4 hours while other tools completes in minutes. This scan resulted in Skipfish detecting the reflected XSS, path traversal, on SQL injection and the printed error message. Also, Skipfish were the only tool to differentiate between the forms that are CSRF protected and those who are not.

We also gave Skipfish a spin without the brute forcing; the test finished in under four minutes and it still made notice of both the SQL injection, directory traversal and the printed error message. In other words much more information relative to the amount of time spent on scanning.

4.3.7 w3af

Overview	
Type	Automated scanner
Availability	Open source
Price	Free
Version tested	1.0 rc5

W3af [23] is an cross-platform tool written in the Python. However, it is not that easy to use. It has a lot o profiles that can be chosen to perform the tests, but it turns out that some of them generate a lot of errors. After a lot of attempts to start testing our web application with the *OWASP Top 10* profile, without any success, we decided to run the *audit high risk* profile. We just needed to indicate the target site and start the scan.

With this tool only 2 out of 8 vulnerabilities were found, SQL injection (in URL) and printed error message. This tool does not provide a way to save the report.

High risk vulnerabilities like both XSS flaws were not reported, something that we were not expecting. Also the poor performance, comparing with tools like Skipfish (open source tool) or Burp Suite Pro (commercial tool), makes this one of the less reliable tools.

4.3.8 ZAProxy

Overview	
Type	Proxy
Availability	Open source
Price	Free
Version tested	1.2.0

ZAProxy [26] is based on Paros, so it is also a Java application. Some improvements were done to Paros, most of them related with menus others related with the application performance, such as the passive scanner that now can look for new vulnerabilities, where between them we can find weak authentication.

To perform the tests with this tool, all the steps followed with Paros have to be done here to.

As in Paros, only the two XSS vulnerabilities and the printed error message were reported, also with the reflected XSS vulnerability reported as stored. With this tool, we can also print a report with all the vulnerabilities found, and respective risk level.

After configuring the tool to perform the scan exactly the same way as we did with Paros, the results were no different. However different risk levels were reported by each of the scanners for the same vulnerability. For instance with Paros were XSS reported has "Medium Risk" while with ZAProxy it were reported has "High Risk". One could argue that both of these are notions valid, since the important part is that the vulnerability is detected.

Since Paros Proxy is no longer maintained, ZAProxy is the natural choice. But there is no clear distinction between them and the results they provide.

4.4 Reference tools

Due to availability problems because of a combination of licensing issues and time constraints where we not able to get first hand experience with all the tools our self. We believe the results from these tests are valuable even though we are not able to compare these tools directly to those on the previous list because it puts the findings of the other tools in perspective. They scanned the exact same application and they had the exact same opportunities to detect vulnerabilities.

4.4.1 Core Impact

Core impact found only SQL injection vulnerabilities. The report produced from the scan is PDF file containing necessary information about the vulnerable URL and requests made which incorporates the exploits. This PDF report includes some text describing why SQL injection is dangerous etc, while it is lacking a clear overview over what type of vulnerabilities and where they where found.

Both Core Impact, IBM AppScan and NTOSpider are in the price range of €10.000 per user per year.

4.4.2 IBM AppScan

As IBM is generally acknowledged as one of the giants, large expectations are tied to their products. And AppScan proves to be one of the tools that point out most vulnerabilities in our comparison. Both SQL injection vulnerabilities, the reflected XSS, directory traversal and the printed error message is made note of.

4.4.3 NTOSpider

The report from NTOSpider scan [10] comes in from of a set of website which can produce number of different types of reports and summaries. Remediation reports for developers or DBA's, a number of different best practice and compliance reports and summaries with different focuses. It even guesstimates the cost of fixing the problems it found in our application to be between $3000 and $3750, numbers that are seemingly taken straight out of the blue.

Other studies [21] have shown that NTOSpider is a strong tool, but it failed to prove this strength in our test. Only SQL injection and the printed error message were detected. Each of the SQL injection vulnerabilities were also reported three times each, appearing in the summary as six vulnerabilities, instead of two which is the case.

4.4.4 A yet to be released commercial tool

We also had a tool still under development scanning our application. This tool more or less failed to provide any results, only discovering the directory traversal vulnerability. From what we have been able to sort out is it their crawler which holds it back, filling up the 2000 link limit with useless links. We decided not to name this tool because we do not believe it necessarily will reflect the nature of the tool once it is released in the future. But we think the point of what seems to be a failing crawler in this scanner is interesting to note, as it significantly affects the result.

4.5 Results summary

We expected to see a much higher detection rate by the scanners than we did. Most of the tools considered in this study also have been evaluated in other studies and turned out as much more reliable [20] [21] [19]. It seems that the properties of the target web application that is subject to scanning significantly affects the findings of the tools. An overview of the vulnerabilities detected by the scanners in our comparison can be found in Appendix B.

Based on what we have seen during the work with this project, does it seem that the abilities of the scanners to crawl an application is their acillies heel. Further is that what makes our results so different from other comparisons in terms of detected vulnerabilities. Although our application utilitzes a totally valid URL scheme which is fully functioning in a normal browser does it seem like the tools had difficulties getting around the application. This scheme includes a slash in the part of the URL that decides which page the URL returns. Such as */drupal/?q=node/1* which is in fact the default setup in Drupal, the basis for our applicaion.

Although our comparison because of this might not paint the most accurate picture of the tools in total does it shed light on another very important aspect. Namely that one have to watch these tools closely in their work and be very cautious about trusting their process blindly.

There is in other words a clear consensus that the automatic crawling functionality in these tools still have room for improvement. And the fact that they cannot be relied upon heavily affects the integrity of the final results scanners as well. One reason for the fact that the scanners had problems crawling our application might have been it's URL structure. Although it is fully functioning in a browser, the usage of variables in the URL might have implied problems for the tools.

Chapter 5

Conclusions

We do not consider any of these tools to have passed our test. While our results show a much lower detection rate than other research has two different implications. First, these results alone does not give a correct image of how these tools perform in general. Most importantly, these results shred light on the importance of comparing this type of scanners on more than one application but also that there seem to be flaws in many of them that severly

While these tools fail to point out a satisfying amount of the vulnerabilities, we still feel that they could add value to an audit process. Utilizing these tools in the initial part of the process would add value both in the sense that they are able to give an auditor of an overview of the situation and possible disclosure of vulnerabilities. Adding some manual intervention to the process of using these products should be considered insted of a blind point-and-shoot approach as it seems to enhance the results significantly.

That said, from our experiences, both BurpSuite Pro and Skipfish feels like the tools giving most back in terms of results compared to their requirements of time and licensing expenses. The effectiveness of the time spent with these all of these tools very much depends on the reliability of their findings. As it does not seem that they can be used to determine an application as secure, they can still be used to point an auditor to insecure functionality. And that would definitively enhance the efficiency of an audition proccess if the results are clear and correct.

5.1 Further research

5.1.1 Crawling abilities

Looking into how the crawling functionality of these tools compare seems to us as the natural next step as this is essential to the end result for an automated penetration testing tool. Developing a custom application with several types of links and functionality only available after successfully submitted forms with hooks in the application code to detect exactly where each of the tools crawled and even which page referred them to each page. This would give the user much more insight into how trustworthy the different tools are at different types of applications.

5.1.2 Selective scanning

Several of these tools have selective scanning methods, which reduces the amount of requests and thereby scanning time. How good are these heuristic scanning methods? If a tool detects a vulnerability in a single form, will it still detect it if there are 10 forms on the same page, of which only one is vulnerable?

Chapter 6

Acknowledgments

We would like to thank Coen Steenbeek and Gijs Hollestelle at Deloitte Enterprise Risk Services for taking the time to be our supervisors in this project. Also thanks to their colleague Nick Kirtley for a number of helpful discussions and different Deloitte offices for letting us use tools they where in possession of licenses.

Appendix A

Web application example

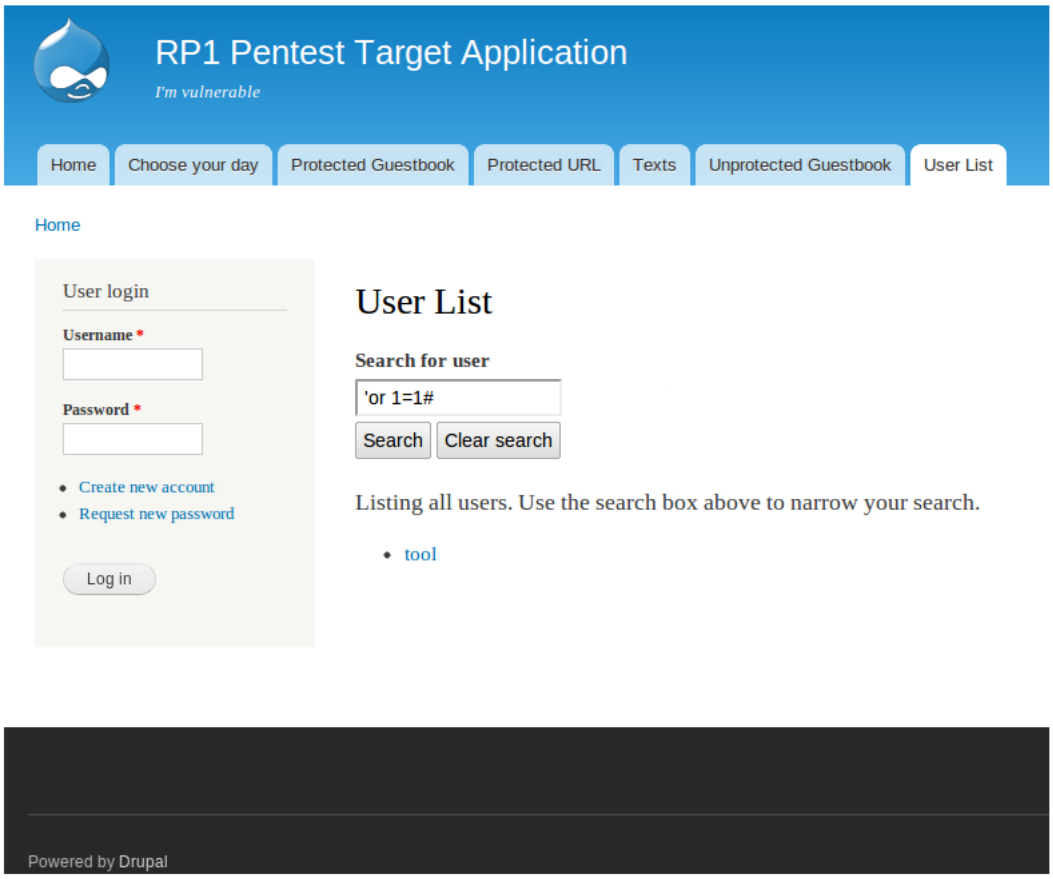

Figure A.1: Example of SQL Injection in our web application

Appendix B

Scanner result overview

The following table presents an overview of the vulnerabilities found by each of the scanners tested. The table lists the scanners alphabeticly, with the commercial tools above the horizontal line and the open source tools underneith it.

	Path traversal	CSRF	Reflected XSS	Stored XSS	Failure to restrict URL access	SQL Injection (in url)	SQL injection (in HTML form)	Printed error message
Acunetix	Found					Found		
Burp Suite Pro			Found	Found		Found	Found	
Core Impact						Found	Found	
IBM AppScan	Found		Found			Found	Found	Found
NTOSpider						Found	Found	Found
ParosPro			Found				Found	
Qualys	Found		Found			Found		Found
Paros			Found	Found				Found
Skipfish	Found	Found	Found			Found		Found
W3AF							Found	Found
ZAProxy			Found	Found				Found

Table B.1: Overview over vulnerabilities reported by each of the scanners. Commercial scanners above the seperating line and open source below, sorted alphabeticly by scanner product name

Bibliography

[1] Acunetix Web Vulnerability Scanner, http://www.acunetix.com/.

[2] Burp Suite Pro, http://www.portswigger.net/burp/.

[3] Core, http://www.coresecurity.com/.

[4] Deliberately Insecure Web Applications For Learning Web App Security, http://www.irongeek.com/i.php?page=security/deliberately-insecure-web-applications-for-learning-web-app-security

[5] Doup, A., Cova, M. and Vigna, G., *Why Johnny Cant Pentest: An Analysis of Black-box Web Vulnerability Scanners*.

[6] Drupal, http://drupal.org/.

[7] Hacme Bank v2.0, http://www.mcafee.com/us/downloads/free-tools/hacme-bank.aspx, May 19, 2006.

[8] IBM AppScan, http://www.ibm.com/us/en/sandbox/ver1/.

[9] Ke, J., Yang, C. and Ahn, T., *Using w3af to Achieve Automated Penetration Testing By Live DVD / Live USB*.

[10] NTOSpider, http://www.ntobjectives.com/ntospider.

[11] Open Web Application Security Project (OWASP): OWASP Top Ten Project, http://www.owasp.org/index.php/Top_10_2007, 2007.

[12] Open Web Application Security Project (OWASP): OWASP Top Ten Project, http://www.owasp.org/index.php/Top_10_2010-Main, 2010.

[13] Paros, http://www.parosproxy.org/index.shtml.

[14] ParosPro, http://www.milescan.com/hk/.

[15] Qualys, *Vulnerability Management For Dummies*, 2008.

[16] Qualys Web Application Vulnerability Scan, http://www.qualys.com/.

[17] RP1 Pentest Target Application - I'm vulnerable, http://target.warsaw.practicum.os3.nl/drupal/.

[18] Skipfish, http://code.google.com/p/skipfish/.

[19] Doupe, A.,Cova, M., Giovanni, V. (2010) *Why Johnny Can't Pentest: An analysis of blackbox web vulnerability scanners*.

[20] Suto, L., *Analyzing the Accuracy and Time Costs of Web Application Security Scanners*, San Francisco, February 2010.

[21] Suto, L., *Analyzing the Effectiveness and Coverage of Web Application Security Scanners*, San Francisco, October 2007.

[22] Vieira, M., Antunes, N. and Madeira, H., *Using Web Security Scanners to Detect Vulnerabilities in Web Services*, June 2009.

[23] w3af, http://w3af.sourceforge.net/.

[24] Web Application Scanners Accuracy Assessment - Freeware & Open Source Scanners, http://sectooladdict.blogspot.com/2010/12/web-application-scanner-benchmark.html

[25] WebGoat, http://code.google.com/p/webgoat/.

[26] ZAProxy, http://code.google.com/p/zaproxy/.