Pen Testing for Web Applications

Ahmad Al-Ahmad, Yarmouk University, Jordan

Belal Abu Ata, Yarmouk University, Jordan

Abdullah H. S. Wahbeh, Dakota State University, USA

ABSTRACT

As many Web applications are developed daily and used extensively, it becomes important for developers and testers to improve these application securities. Pen testing is a technique that helps these developers and testers to ensure that the security levels of their Web application are at acceptable level to be used safely. Different tools are available for Pen testing Web applications; in this paper the authors compared six Pen testing tools for Web applications. The main goal of these tests is to check whether there are any security vulnerabilities in Web applications. A list of faults injected into set of Web pages is used in order to check if tools can find them as they are claimed. Test results showed that these tools are not efficient and developers should not depend solely on them.

Keywords: *Developers, Faults Injection, Pen Testing, Security Vulnerabilities, Web Application, Web Pages*

INTRODUCTION

Nowadays, Web applications are used worldwide by users for their personal needs but what make Web applications more and more important are the business intentions to use the Web. Web application become one of the most core business assists that must be surrounded by special security guards that are not less than any other assets as stated by Pan and Li (2009). Security issues may harm the benefits of businesses, but here comes the story; these are not ordinary guards, they are IT experts with high salaries, and the enemies here are also different, they are everywhere and nowhere, aided by the same defense methods you use that are Information technologies.

Currently, most of the businesses provide services to its customers using the Web and many of them depend on the Web to do their business in order to make benefits from the global accessibility of the Web. For example, E-commerce is a novel commerce model based on computer network; the E-commerce model depends mainly on money and whenever you find money in any filed, risks become higher and new type of attacker who have a mission to steal money from you not just for fun and self-motivation will be founded (Antunes & Vieira, 2009).

As demonstrated by Guo, Yu, and Chiueh (2005) the wide usage of Web applications and services poses new security challenges on developers and testers, hundreds of new

DOI: 10.4018/jitwe.2012070101

vulnerabilities are being discovered annually and dozens of new patches are being released monthly. So, sensitive data manipulated for these applications must be protected against the attackers who are trying to find vulnerabilities in this kind of applications; these vulnerabilities came from many sources starting from bad wiring code, servers used and firewall.

Keeping the data secure and ensuring that the application is available need extensive search over the test cases. Time challenges make it not practical to test the application code and find vulnerabilities using white box testing; even it's the best way. Black box testing reduces testing effort and can be automatically repeated extensively to find new vulnerabilities beside the face its meets the time criteria needed by the Web application testing. Pen testing is black box testing method which tries to act like an attacker and use scientific method to find vulnerabilities before the attackers do this using tools and sometimes manually, as defined by Kwon et al. (2005): Pen testing is a goal-oriented method similar to "catch-the-flag" that attempts to gain privileged access to a system using pre-conditional means that a potential attacker could manipulate.

Several studies (Pan & Li, 2009; Antunes & Vieira,2009; Orloff, Petukhov, & Kozlov, 2008; Zhao, Zheng, & Chen, 2009; Fonseca, Vieira, & Madeira 2007) addressed web application vulnerabilities and Pen testing. Many Web vulnerabilities such as SQL Injection (Viera et al., 2009; Livshits & Lam, 2005; Halfond, Viegas, & Orso, 2006), XPath Injection (Viera, Antunes, & Madeira, 2009), Cross-site scripting (Livshits & Lam, 2005; Erlingsson, Livshits, & Xie, 2007), Path traversal (Livshits & Lam, 2005), HTTP response splitting (Livshits & Lam, 2005), and Command injection (Livshits & Lam, 2005; Jovanovic, Kruegel, & Kirda, 2006) exist and one of the useful solutions for such vulnerabilities is using Pen testing. Pen testing is new field of study highly needed and interesting for both technical and researcher. With many tools and no standards in this field it is complicated for the tester to start pen testing. Setting Pen testing steps and what is the information needed for each step along with the tools to aid testers in doing their job are ambiguous and still depend on the tester and the application framework used in the development process. Without code, setting as just a user is pen testing in practice. Repeating testing process many times beside all what has been discussed make the tools used in pen testing process very important and effect the overall testing process.

The main objective of this research is to conduct Pen testing on a selected Web application using several tools in order to compare the tools themselves, finding drawbacks in these tools and suggest solution for the testing process.

The remainder of the paper is organized as follows: Related works are described in the first section. The overall methodology followed is presented in the second section. Implementation, experimentation and results evaluation are expressed in third section. Finally, we close our paper with conclusion and future works in the last section.

RELATED WORKS

Pen testing for Web applications has been addressed by several researchers. In this section we review some related studies in the field of Pen testing and Web application security.

Orloff (2011) addressed Common Web security vulnerabilities such as cross-site scripting (XSS) and SQL injection. The authors demonstrated that it's clear that it's hard for ordinarily developer to take this task of finding vulnerabilities in Web application and this task need special knowledgeable team but with tools the developer can find many of vulnerabilities in Web application in an easy way. The authors studied open sourced tools (WebScrab and Paros) that help tester find vulnerabilities developed by Open Web Application Security Project (OWASP). Orloff (2011) demonstrated that testers also need manual checking to find vulnerabilities and they should insert SQL queries and scripts and find the result of such action. The author concluded that developers need to know what the attacker looks for and how they attack to solve security problems and prevent the attacker from causing troubles to Web sites.

Moraes, De Abreu, and Martins (2009) studied and analyzed faults founded in Web based application developed using Java language in order to classify these faults. The main goal of the authors is to improve error injection mechanisms used to inject faults into Web application in order to test or to compare between testing tools. Authors mainly depend on the IBM fault classification work and another two works which classify faults for C language. Three Web applications are used to test, find faults, then fix these faults; this cycle continued until no more faults are found and records the faults discovered in order to map these faults. The authors found that 64% of faults discovered are not classified and labeled by “other” when mapped to the IBM classification list and other extended lists. Authors concluded that IBM fault classification list need to be extended to cover other languages like Java, JavaScript and HMTL.

Ismail, Etoh, and Kadobayashi (2004) studied one of the most important vulnerabilities namely; Cross-site scripting (XSS), that threaten Web application user data. Authors tried to find solution that protect user data from attackers who tries to steal information by injecting XSS code into application and steal user cookies that may include sensitive data. A system proposed with three modes, where each mode in the system tries to find the XSS code. Authors found that the proposed system is efficient and work will to find XSS that use HTML tags, but it miss the XSS that use other JavaScript and VB code.

Shahriar and Zulkernine (2009) proposed seven criteria namely; vulnerability coverage, source of test cases, test generation method, level of testing, granularity of test cases, tool automation, and target applications to analyze program security testing techniques Authors compared and contrasted 20 program security testing techniques based on these criteria. Authors showed that every technique has a different perspective about vulnerabilities and this perspective is narrow and does not cover all vulnerabilities proposed. Also, most of the techniques do not cover white box and black box but mostly they supply only one of them which lead to less coverage of the vulnerabilities. They also demonstrated that results will provide practical information for security practitioners in choosing the most appropriate tools.

Antunes and Vieira (2009) addressed the SQL injection vulnerabilities and proposed a new tool that can be used to help developer to find SQL vulnerabilities in Web application more efficiently than the available commercial tools. Authors judged the proposed tool using the results of testing 262 Web services using a commercial tool and the proposed tool. Results showed that commercial tool found more vulnerabilities than the proposed tool. They classified the result manually into three sets: vulnerabilities, non-vulnerabilities and Doubtful. Results showed that many of the vulnerabilities in the commercial tools results are non-vulnerabilities; they are mostly robustness problems or problems in the Web service itself. Results of the tools showed that the proposed tool is more efficient and found 93% of the vulnerabilities in Web services tested.

Kie˙zun, Guo, and Ernst (2009) proposed a new technique that overcomes previous techniques limitations to find vulnerabilities. Authors built a tool called Ardilla which represents the proposed technique that consist of four major steps. Authors evaluated Ardilla against several criteria and results compared against results generated from previous tests in order to compare between the proposed technique and old techniques. Authors found that Ardilla found all vulnerabilities listed in the previous results, in less time and with less configuration requirement than previous tools. Authors discussed some limitations of the proposed technique resulted from the fact that the tool only generate tests and attack sensitive flows and did not take in mind other paths which reduce vulnerabilities coverage; however, results still showed that its more efficient than previous techniques.

MeiJunin (2009) proposed technique for SQL injection vulnerability detection. To implement the proposed technique, author used a combined approach of static analysis, runtime detection and automatic testing. Results are compared with results from previous researches, authors found that proposed techniques do not show false vulnerabilities, but it does not show any vulnerabilities that is not in the result of old techniques. Authors concluded that the proposed technique provides the developer with a list of true vulnerabilities but it does not find vulnerabilities not covered in old techniques.

Huang, Tsail, Lee, and Kuo (2004) Proposed a methodology to reduce the harm caused by testing the applications by defining three modes to test the application depending on the test requirement chosen by the developer. The three test mode which depends on time and risk, range from heavy mode to relax mode to safe mode. Authors used the three modes to test the application in order to show, the percentage of covered vulnerabilities for each mode and results show that heavy mode found 80% of all vulnerabilities and need long time approximated in days, relax mode found 58.4% of vulnerabilities and take 48 hours of testing and safe mode found only 2% with short time approximate in minutes. Finally; authors combined the three modes in a tool called WAVES that allow the user to choose one of them to test Web applications.

METHODOLOGY

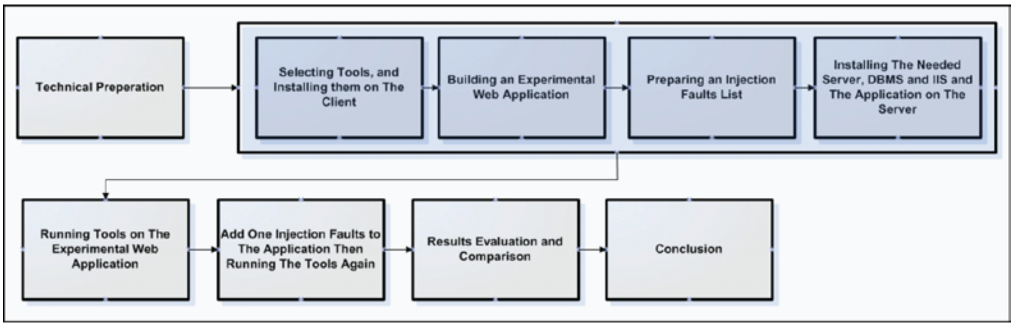
In this section we present the methodology that will be followed in order to achieve the goal of the study. Figure 1 shows the steps that make up the overall methodology.

Technical Preparations

In this step all technical issues will be discovered and studied well in order to find tools, built the experimental Web application, prepare the injection fault list and install needed application in order to run the Web application on the selected tools.

- **Finding Tools and Installing Them:** The first step in the technical preparation phase is to find the required tools required to complete the experimentation phase. A number of commercial tools and free tools are selected in order to find out more about different results obtained by these tools. The list of tools to be installed and used includes; IBM Rational AppScan, ParosPro, Netsparker, N-Stalker, Sandcat, and Grendel-Scan.
- **Building an Experimental Web Application:** A Web application is built in order to run the selected testing tool; it is built using a number of Web pages that allows injecting errors inside the application. Ac-

Figure 1. Pen testing methodology


Technical Preperation
Selecting Tools, and Installing them on The Client
Building an Experimental Web Application
Preparing an Injection Faults List
Installing The Needed Server, DBMS and IIS and The Application on The Server
Running Tools on The Experimental Web Application
Add One Injection Faults to The Application Then Running The Tools Again
Results Evaluation and Comparison
Conclusion

cordingly; the developed Web application will be a set of pages connected to each other with content added specially to be injected with faults, for each fault a special page will be developed in order to run the test against all the tools. The Web application is developed using ASP.net, VB.net along with HTML and JavaScript.

- **Prepare Fault Injection List:** Table 1 shows the list of faults developed by IBM. This list is developed in order to provide information about the general error that developer always misses and is developed using the C language. According to Moraes et al. (2009) this list does not support HTML and JavaScript, where these are in fact the languages mainly used to develop Web applications. For this reason a new developed list consist of the errors in IBM list and other errors from other researches, experienced developers, and attackers found on the Web.
- **Installing Software Needed to Run the Application:** For the Web application that uses a database and works on Windows operating system we need to install and configure a set of soft wares; according to the developed Web application specification we need to use IIS SQL server 2005 to run the Web application.

Running Clean Test

Initially; all tools are run against the built Web site in order to get zero results; this is an important step to incrementally record results. There is a need to start from an empty result list and then inject errors one by one in order to find which tool found that error and which did not. To start clean test we will keep all pages in the Web application with no functionality and run the test in all main pages specified for errors.

Adding Faults and Run Test Tools Again

In this step we will pick one of the errors and inject it in the application. Using one error at a time is important in order to record exactly the result of the tools and how it defers from the clean test. After injecting the error we will run the tools to test the application again and record the results. Then we will remove that error from the application and then pick a new error and inject it in the application and run the test tools again. This process will be repeated until all errors in the error list tested.

Table 1. IBM fault list

Fault Type	Description
MIFS	Missing “If (cond) {statement(s))}”
MFC	Missing function call
MLAC	Missing “AND EXPR” in expression used as branch condition
MIA	Missing “If (cond)” surrounding statement(s
MLPC	Missing small and localized part of the algorithm
MVAE	Missing variable assignment using an expression
WLEC	Wrong logical expression used as a branch condition
WVAV	Wrong value assigned to a value
MVI	Missing variable initialization
MVAV	Missing variable assignment using a value
MAEP	wrong arithmetic expression used in parameter of function call
MPFV	Wrong variable used in parameter of function call

Comparing Tools Results

Once all the errors are injected into the Web application; results will be recorded and a comparison between these results from the different tools is conducted.

RESULTS AND EVALUATION

In this section real experiments for testing the developed Web application are conducted starting with the clean page, followed by another page (one at a time) that contains faults from the selected fault list, result for each page will be recorded and an overall comparison is done.

As mentioned before a set of pen testing tools for Web application have been selected and installed in order to conduct the experiments, the list of tools was selected carefully in a way that it contains commercial and free tools in order to find results from the both domain. These tools claim that they can find vulnerabilities in Web applications. Tools names and descriptions are presented in Table 2.

1. **IBM Rational AppScan:** Is a tool with full functionality that supports all type of vulnerabilities, and uses an easy to use user interface without any configurations needs.
2. **ParosPro version:** A tool that provides full functionality for different type of vulnerabilities, but requires JAVA to work correctly, ParosPro has an easy to use interface without any configurations.
3. **Netsparker version:** A tool with an easy to use interface and limited functionality for SQL injection and cross site injection beside static test which in fact fully cover our needs.
4. **N-Stalker:** Is a free tool with full functionality that provide easy to use user interface beside more features added for stress testing and proxy management and testing.
5. **Sandcat:** A tool that provides easy to use user interface and provide unique feature that does not exist on any tool by allowing the user to select what browser to emulate the test with.
6. **Grendel-Scan:** Is a Java based tool that requires a special configuration by changing the proxy setting for the browser to local-host with port 8082, because this tool listens to the traffic of the browser to test Web application we used version.

Table 3 shows the list of faults to be injected into the experimental Web application. This list does not depend on IBM fault list; therefore, we have chosen our list from our experience, beside the knowledge we got from previous research, developers and attacker.

Test procedure is done starting by a clean test on a Web page that does not contain any code; after clean test the faults will be examined one at a time.

Clean Test

Clean test is conducted to make sure that we starts from a Web application that does not

Table 2. List of tools used

Tools	Company	Commercial/Free
IBM Rational AppScan	IBM	Commercial
ParosPro	Mile Scan	Commercial
Netsparker	Mavituna Security	Commercial
N-Stalker	N-Stalker	free
Sandcat	Syhunt Cyber-Security Company	free
Grendel-Scan	written by David Byrne	free

Table 3. List of faults

SQL Injection using Dynamic Strings
SQL Injection in Stored Procedures
SQL Injection using URL
Persistent Attack XSS
Non-Persistent Attack XSS
Denial of service targeting a specific user
Denial of service targeting the Web server
Path Traversal

contain any error, so if an error injected into the application the tool as it claim should report it in the results. In clean test we used a Web page that has no functionality and no code; this Web page will be our start point for next injected faults. The expected number of faults is zero for all tools. Clean test results are shown in Table 4.

As Table 3 demonstrates; some tools opposed the proposed assumption where all tools should reports zero faults. N-Stalker, Netsparker and Sandcat reports 4, 3, and 1 fault respectively.

SQL Injection using Dynamic Strings

Dynamic SQL strings are important technique used in Web applications in authentication page. In this case; this techniques can be easily attacked by attacker to gain unauthorized access to Web application. This fault will be injected by using SQL query string that is built dynamically using data inserted by the user, this dynamic query string is easy to be attacked by inserting SQL commands inside the data entered by the user. Two text boxes are used to receive data from the user; this data will be used to build the SQL query as follows:

Dim sqlStr = “select * from login where userName=’” + UserName.Text + “‘and password=’” + password.Text + “‘”.

This kind of queries can be attacked by inserting a SQL command in the text boxes that are used to build the query string. As in our case; the built page can be attacked by inserting a valid username let’s say user1 and using a code for the password like (‘anything’ OR ‘=’) so the result query will be:

```
"select * from login where userName=
'user1' and password='anything' OR
''='' ".
```

Table 4. Clean test results

Tools	Number of Faults Found	Injected Fault (Found/Not Found)
IBM Rational AppScan	0	No
ParosPro	0	No
Netsparker	3	No
N-Stalker	4	No
Sandcat	1	No
Grendel-Scan	0	No

The condition injected ‘’=” is always true so the access will be granted. Tools are run on the page that contains this fault and result is shown in Table 5.

Table 5 shows that all the tools reports a number of faults, where IBM Rational, ParosPro and Grendl-Scan reports 1 fault each, on the other hand Sandcat, N-Stalker and Netsparker reports 2, 4 and 6 faults respectively. In terms of the injected faults, only N-Stalker detects that faults and none of the other five tools detect it.

SQL Injection in Stored Procedures

Using Stored Procedures in Web application is an important technique widely used by developers. Stored procedure that uses data entered from the user can be attacked like the dynamic SQL strings. In our page that contains this fault we used a stored procedure to validate user name and password, the user name and password will be entered by user and passed to the procedure as parameters by the application. The stored procedure code:

```
@Username varchar(50) = '',@Password
varchar(50) = ''
DECLARE @command varchar(100)
set @command = 'select * from login
where userName = ''' +@Username + '''
and password = ''' + @Password + ''' '
EXEC (@command).
```

In this procedure @Username and @Password are two parameters entered by the user and passed form the Web page to the procedure. This code can be attacked in many ways like the previous fault. For example if the attacker entered "‘; DROP TABLE Accounts” as a password this will cause the SQL drop the table called Accounts.

Tools are run on the page that contains this fault and result showed in Table 6.

Table 6 also shows that only the N-Stalker tool found the injected fault and none of the other five tools found it. In terms of number of faults found; IBM Rational, Sandcat, and Grendel-Scan found only 1 fault, where ParosPro found 2 faults; and both Netsparker and N-Stalker found 6 faults each.

SQL Injection using URL

A URL can be used to pass parameters between pages in Web application, these parameters can be used at the destination page in SQL queries which make it important to study and ensure that this technique is secured. This kind of faults will be injected using two pages called, source and destination; where the source page will passes data to the destination page using URL and this data will be used in a query at the destination page. Attackers can attack the Web application by editing the URL of the destination page; this will cause SQL to generate error page that contain the path of the application, this information can be used by the attacker to initiate more advance attacks.

For example if the destination page URL sent by source:

Table 5. SQL injection results

Tools	Number of Faults Found	Injected Fault (Found/Not Found)
IBM Rational AppScan	1	NO
ParosPro	1	NO
Netsparker	6	NO
N-Stalker	4	Yes
Sandcat	2	NO
Grendel-Scan	1	NO

Table 6. SQL injection in stored procedures results

Tools	Number of Faults Found	Injected Fault (Found/Not Found)
IBM Rational AppScan	1	NO
ParosPro	2	NO
Netsparker	6	NO
N-Stalker	6	YES
Sandcat	1	NO
Grendel-Scan	1	NO

```
GRAD_FAULT_3_des.aspx?f1=ahmad&f2=pass
word&f3=ahmad.
```

Such URL contains the name of the column used by SQL query. The attacker can change this URL and cause the application to generate error that helps the attacker to attack the application. Results of this kind of faults are represented in Table 7.

As the results show, the IBM rational tool detects the injected faults where no other tools detect it. Looking at the number of faults founded on the web page, Grendel detect 1 fault, followed by Scancat and ParosPro with 2 faults then Netsparker and N-Stalker with 4 faults and finally, IBM rational with 5 faults detected.

Persistent Attack XSS

Persistent Attack XSS is a cross site scripting vulnerability that can be used by attackers to insert scripts into the application using the database. This fault will be injected by using a page that get data from the user and store it in the database without any check on the content of this data. This kind of data provided by the user could be used from the application in building pages; so if the data inserted by the user is a script code like:

```
<SCRIPT>document.location='http://
attackerhost.example/cookiesteal.
cgi?'+document.cookie</SCRIPT>.
```

The built page will contain this script which will steal the cookies of the user when he/she opens the page that contains this script. Results of such kind of faults are shown in Table 8.

As Table 8 shows, two tools only founded the injected faults namely, N-Stalker and Grendel-Scan. In terms of number of faults founded, Grendel and IBM rational founded 1 faults followed by ParosPro, Sandcat, N-Stalker and Netsparker with 2, 3, 5, and 6 faults detected.

Table 7. SQL injection using URL results

Tools	Number of Faults Found	Injected Fault (Found/Not Found)
IBM Rational AppScan	5	YES
ParosPro	2	NO
Netsparker	4	NO
N-Stalker	4	NO
Sandcat	2	NO
Grendel-Scan	1	NO

Table 8. Persistent attack XSS results

Tools	Number of Faults Found	Injected Fault (Found/Not Found)
IBM Rational AppScan	1	NO
ParosPro	2	NO
Netsparker	6	NO
N-Stalker	5	YES
Sandcat	3	NO
Grendel-Scan	1	YES

Non-Persistent Attack XSS

Non-Persistent Attack XSS is also a cross site scripting vulnerability that can be used by attackers to insert scripts into the application; where the script is inserted into the HTTP request which used to build HTTP reply. This fault will be injected by using page that uses data entered by the user to generate a response, in this page the data entered by the user will be used to get full user name and create a script; so if the attacker change the full name data field in the database into script like:

```
<SCRIPT>document.location= 'http://at-
tackerhost.example</SCRIPT>
```

The response will redirect the user to the attacker Web site. Results for such kind of faults are shown in Table 9.

As Table 9 demonstrates the same two namely, N-Stalker and Grendel-Scan founded the newly injected faults. In terms of number of faults founded, Grendel and IBM rational also founded 1 faults followed by ParosPro, Sandcat, Netsparker, and N-Stalker with 2, 3, 6, and 9 faults detected.

Denial of Service Targeting a Specific User

Most Web application use counters for the number of failed login trials and block the user from entering the system after specific number of trials. Attackers may use this feature to attack specific user by blocking him using wrong password many times. This fault will be injected by using page that used to verify the user name and password and check number or trials, this page will prevent the user from log in after 1 time trying wrong password. For the page that contains such fault; two faults are detected, one for the SQL Injection using Dynamic Strings and the other for Denial of service targeting a specific user. Tools are run on the page and results are shown in Table 10.

Table 9. Non-persistent attack XSS results

Tools	Number of Faults Found	Injected Fault (Found/Not Found)
IBM Rational AppScan	1	NO
ParosPro	2	NO
Netsparker	6	NO
N-Stalker	9	YES
Sandcat	3	NO
Grendel-Scan	1	YES

Table 10. Denial of service targeting a specific user results

Tools	Number of Faults Found	Injected Fault (Found/Not Found)
IBM Rational AppScan	1	NO
ParosPro	1	NO
Netsparker	6	NO
N-Stalker	9	NO
Sandcat	3	NO
Grendel-Scan	1	NO

Denial of Service Targeting the Web Server

Web applications have limits in term of number of users and amount of data, these limits can be used by attackers to create attacks that increase load on Web application cause it to fail. This fault will be injected by using a page that opens connection to database without closing it; if the code is called many times, the page will go down which prevent this service from all users. We used a default memory pool size and the tool expect the maximum number of IIS connections to 101 connection which can be used in the comparisons, so if you open the application 101 times the application will be down and stop working. Results for such kind of faults are shown in Table 11.

Path Traversal

Developer's first mission is to not helping attackers; they sometimes help attackers without their knowledge by providing information about the application to attackers. This information leak we are interested here is the information that the attacker gain from error pages of the system when it encounter a fault. Developer need to handle all faults and provide general error page that does not provide any information on the structure or the technology used and most important path of the application on the server which can be used by attackers to initiate advance attacks. This fault will be tested by providing the tools with invalid URL, this invalid URL will cause the application to generate error page on the server. Results for such kind of faults are shown in Table 12.

Comparing the results for all tools under the test, Table 13 clearly show that free tools which are the last three tools in Table 13 provide better results from the other commercial tools, where the coverage percentage for N-Stalker, Sandcat and Grendel is 50%, 12% and 25% respectively; where the commercial tools, namely; IBM, ParosPro, and NetSparker achieved only 25%, 0% and 12% respectively.

Table 11. Denial of service targeting the web server results

Tools	Number of Faults Found	Injected Fault (Found/Not Found)
IBM Rational AppScan	1	NO
ParosPro	2	NO
Netsparker	6	NO
N-Stalker	9	NO
Sandcat	3	NO
Grendel-Scan	3	NO

Table 12. Path traversal results

Tools	Number of Faults Found	Injected Fault (Found/Not Found)
IBM Rational AppScan	14	YES
ParosPro	0	NO
Netsparker	6	YES
N-Stalker	2	NO
Sandcat	2	YES
Grendel-Scan	1	NO

More, we can find out that the free tools provide less false faults in its results with an average of 89% where the commercial tools achieved about 97%.

Comparing results with other research; Antunes and Vieira (2009) found that none of pen testing tool found more that 51% of the injected faults which is close to 50% which is the number we found. Fonseca et al. (2007) found that different tools provide different results, and tools provide high false faults ranging from 20% to 77%; our results showed that false faults ranges from 80% to 100%, this difference between results cannot be explained because Fonseca et al. (2007) provides no details about the tools in their research plus they use IBM fault list which is not built to work with our Web application and need modification in order to be used for it.

CONCLUSION

This paper addressed Pen testing using several commercial and free tools. Results showed that each tool provides different results and the highest percentage of faults founded by the tools is 50% which makes tools usage inefficient to test Web application against security vulnerabilities. Also, results from vulnerability Web application testing tools are not trustworthy and the developer should not even look at them and that these results are misleading. Finally we found that good programming habits and more attention on the behavior of the developers who written the code is the best way to reduce vulnerabilities; white box testing take more time than black box but in fact it is the most efficient way to find vulnerability for Web application.

In the future, we will try to build a fault list that cumulate all faults founded in the Web

Table 13. Summary

Tools	Coverage Percentage	False Fault Percentage	True Fault Percentage
IBM Rational AppScan	25%	92%	8%
ParosPro	0%	100%	0%
Netsparker	12%	98%	2%
N-Stalker	50%	93%	7%
Sandcat	12%	95%	5%
Grendel-Scan	25%	80%	20%

from both developer's experience and attacker's experience to help developer and companies building the testing tools. This list will be an alternative to the IBM fault list and be specific for Web applications.

REFERENCES

Antunes, N., & Vieira, M. (2009). Comparing the effectiveness of penetration testing and static code analysis on the detection of SQL injection vulnerabilities in web services. In *Proceedings of the 15th IEEE Pacific Rim International Symposium on Dependable Computing.*

Antunes, N., & Vieira, M. (2009). Detecting SQL injection vulnerabilities in web services. In *Proceedings of the Fourth Latin-American Symposium on Dependable Computing.*

Erlingsson, U., Livshits, B., & Xie, Y. (2007). End-to-end web application security. In *Proceedings of the 11th USENIX Workshop on Hot Topics in Operating Systems*, San Diego, CA (pp.1-6).

Fonseca, J., Vieira, M., & Madeira, H. (2007). Testing and comparing Web vulnerability scanning tools for SQL injection and XSS attacks. In *Proceedings of the 13th Pacific Rim International Symposium on Dependable Computing* (pp. 365-372).

Guo, F., Yu, Y., & Chiueh, T. (2005). Automated and safe vulnerability assessment. In *Proceedings of the 21st Annual Computer Security Applications Conference.*

Halfond, W. G. J., Viegas, J., & Orso, A. (2006). A classification of SQL injection attacks and countermeasures. In *Proceedings of the International Symposium on Secure Software Engineering.*

Huang, Y., Tsail, G., Lee, D. T., & Kuo, S. (2004). Non-detrimental web application security scanning. In *Proceedings of the 15th International Symposium on Software Reliability Engineering.*

Ismail, O., Etoh, M., & Kadobayashi, Y. (2004). A proposal and implementation of automatic detection/collection system for cross-site scripting vulnerability. In *Proceedings of the 18th International Conference on Advanced Information Networking and Application.*

Jovanovic, N., Kruegel, C., & Kirda, E. (2006, May 21-24). Pixy: A static analysis tool for detecting web application vulnerabilities (Short Paper). In *Proceedings of the IEEE Symposium on Security and Privacy* (pp. 258-263).

Kie˙zun, A., Guo, P. J., & Ernst, M. D. (2009, May 16-24). Automatic creation of SQL injection and cross-site scripting attacks. In *Proceedings of the 31st International Conference on Software Engineering*, Vancouver, BC, Canada.

Kwon, O., Lee, S. M., Lee, H., Kim, J., Kim, S. C., Nam, G., & Park, J. G. (2005). HackSim: An automation of penetration testing for remote buffer overflow vulnerabilities. In C. Kim (Ed.), *Proceedings of the International Conference on Information Networking* (LNCS 3391, pp. 652-661).

Livshits, V., & Lam, M. S. (2005). Finding security vulnerabilities in java applications with static analysis. In *Proceedings of the 14th Conference on USENIX Security Symposium*, Baltimore, MD (p. 18).

Mei, J. (2009). An approach for SQL injection vulnerability detection. In *Proceedings of the Sixth International Conference on Information Technology: New Generations.*

Moraes, R., de Abreu, B. T., & Martins, E. (2009). Mapping web-based applications failures to faults. In *Proceedings of the Fourth Latin-American Symposium on Dependable Computing.*

Orloff, J. (2011) *Web application security: Testing for vulnerabilities*. Retrieved from http://www.ibm.com/developerWorks

Pan, W., & Li, W. (2009). A penetration testing method for e-commerce authentication system security. In *Proceedings of the International Conference on Management of e-Commerce and e-Government.*

Petukhov, A., & Kozlov, D. (2008, May 12-22). Detecting security vulnerabilities in web applications using dynamic analysis with penetration testing, OWASP. In *Proceedings of the Application Security Conference*, Ghent, Belgium.

Shahriar, H., & Zulkernine, M. (2009). Automatic testing of program security vulnerabilities. In *Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference*.

Vieira, M., Antunes, N., & Madeira, H. (2009). Using web security scanners to detect vulnerabilities in web services. In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks*, Lisbon, Portugal.

Zhao, G., Zheng, W., Zhao, J., & Chen, H. (2009). An heuristic method for web-service program security testing. *Proceedings of the Fourth ChinaGrid Annual Conference*, Yantai, Shandong, China (pp. 139-144).