Performance Evaluation of Web Application Security Scanners for More Effective Defense

Chanchala Joshi, Umesh Kumar Singh

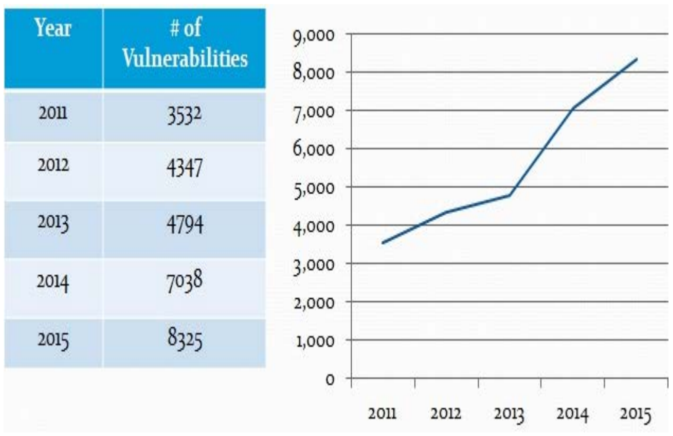
Institute of Computer Science, Vikram University, Ujjain, M.P. India

***Abstract*-** With more and more people becoming Internet users there have been great increase in using Web in all areas of life, including communication, education and shopping. And as a result of these changes the security concerns have also grown. The web application vulnerability scanners help reduce these security concerns in Web-based applications. In today's market a large number of web scanners are available. Although these tools are available in the market but question is how efficient they are to address security concerns in WEB applications? The primary objective of the paper is to study the effectiveness of the scanners and to try to identify common types of vulnerabilities in web services environment. Also, one of the prominent objectives is to provide defense measures, which secure the application significantly.

To compare vulnerability detection rate of different scanners, it is important to have an independent test suite. This paper describes a web application, which is intended to be used to evaluate the efficiency of Netsparker and Acunetix web application vulnerability scanners. For several vulnerabilities presented in this application, we also explain defense measures, which secure the application significantly. The results of web application evaluation identify the most challenging vulnerabilities for scanner to detect, and compare the effectiveness of scanners. The assessment results can suggest areas that require further research to improve scanner's detection rate.

I. Introduction

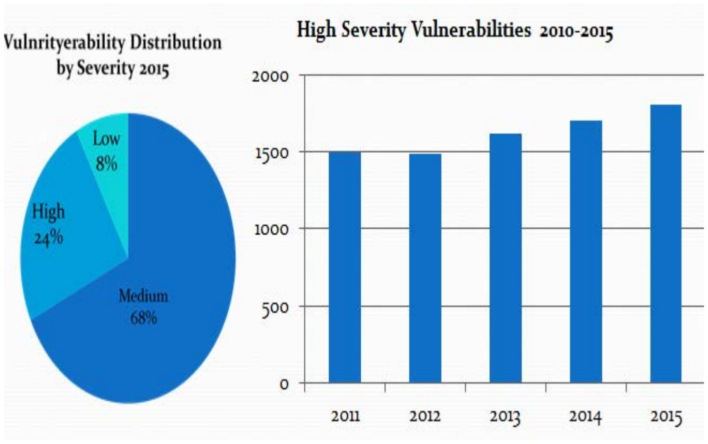
Today's Web applications contain dangerous security flaws. The global distribution of these applications makes them prone to attacks that uncover and maliciously exploit a variety of security vulnerabilities [1]. ISO 27005 defines vulnerability as "a weakness of an asset or group of assets that can be exploited by one or more threats where an asset is anything that can has value to the organization, its business operations and their continuity, including information resources that support the organization's mission" [2]. According to National Vulnerability Database (NVD) [3] the number of vulnerabilities has approximately three times increased since 2011. This is shown in Figure1.



Year	# of Vulnerabilities
2011	3532
2012	4347
2013	4794
2014	7038
2015	8325

Figure: Number of Vulnerabilities 2010-2015

Although OWASP's 2012 report indicated that investment in security was increasing but NTA Monitor's 2014 Web Application Security Report demonstrated that Web security had actually decreased compared to the previous year. In fact, Web application vulnerabilities represent huge problems for companies and organizations. According to WhiteHat Security's most recent Website Security Statistics Report, 63 percent of assessed websites are vulnerable, each having an average of six unsolved flaws [4]. These vulnerabilities create and feed an underground economy based on attacking and stealing data and resources. Figure 2 shows the vulnerability distribution by severity of the year 2015.



The web application vulnerability scanners help reduce these security concerns in Web-based applications. In today's market a large number of web application-scanning tools are available, e.g. Acunetix, Netsparker, Appscan etc. Although these tools are available in the market but question becomes how efficient they are to address security concerns in WEB applications? To compare vulnerability detection rate of different scanners, it is important to have an independent test suite. 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., IBM Rational AppScan[5] and HP WebInspect[6]) but there are also some free application scanners (e.g., Acunetix[7], Netsparker[8], Burp Suite[9], Foundstone WSDigger[10] and Wsfuzzer[11]) with limited use, as they lack most of the functionalities of their commercial counterparts.

This paper describes a web application, which is intended to be used to evaluate the efficiency of Netsparker, Burp Suite and Acunetix web application vulnerability scanners. The application implements real life scenarios for OWASP Top Ten Security Risks [12]. For several vulnerabilities presented in this application, we also explain defense measures, which secure the application significantly.

1.1 OWASP Web Application Security Risks

The OWASP security community has released its annual report in 2015 capturing the top risks in web application development as a combination of the probability of an event and its consequence [12].
The list of the top risks in web applications is as follows:
A1 Injection
A2 Broken Authentication and Session Management (XSS)
A3 Cross Site Scripting (XSS)
A4 Insecure Direct Object References
A5 Security Misconfiguration
A6 Sensitive Data Exposure
A7 Missing Function Level Access Control
A8 Cross Site Request Forgery (CSRF)
A9 Using Components with Known Vulnerabilities
A10 Unvalidated Redirects and Forwards

The two most common risks in the Web environment are SQL injection, which lets attackers alter SQL queries sent to a database and cross-site scripting (XSS). Injection attacks take advantage of improperly coded applications to insert and execute attacker-specified commands, enabling access to critical data and resources. XSS vulnerabilities exist when an application sends user-supplied data to a Web browser without first validating or encoding that content.

In web application described in this paper, we implement OWASP top vulnerabilities A1, A2, A3 and A5.

In this paper we used two free web application vulnerability scanners to identify security flaws in web application. Our main objective is to study the effectiveness of the scanners and to try to identify common types of vulnerabilities in web application environments. In summary, our practical experiment 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?

II. Experimental details

In Broad our experimental study consisted of five steps:

i. **Web Application:** Design a web application that implements all the vulnerabilities from OWASP Top Ten report also select publically available web application services.
ii. **Vulnerability Scanner**: Select the free web application vulnerability scanners.
iii. **Execution**: Use the vulnerability scanners to scan the services to identify potential vulnerabilities.
iv. **Verification**: Perform manual testing to confirm that the vulnerabilities identified by the scanners do exist (i.e., are not false positives).
v. **Analysis**: Analyze the results obtained and systematize the lessons learned.

There are several existing web applications to demonstrate common web application vulnerabilities such as “HacMe” series [13] and “WebGoat” [14]. “WebGoat” is mainly used in educational purposes. But we want to implement vulnerabilities from OWASP Top Ten report, which is not possible with these web applications. Because of these drawbacks of available applications, there is a need to have an independent Web Application, which implements OWASP Top Ten vulnerabilities, to be used to test these web scanners. We design a web application (“shopatujjain”) to simulate the steps a regular user goes through while using a dynamic web page and replicates the behavior. The availability of source code and the control over server results provides better evaluation of web application scanners.

Main functionalities of the application are:

- ➢ First a user creates an account and provides his/her personal data including shipping address and credit card details.
- ➢ Second he/she selects the product and stores his selection in personal shopping cart.
- ➢ Later when the user decides to make the purchase an invoice is placed in queue for further processing.
- ➢ In addition to that the user can add reviews to products and read other customer’s opinions, newsletters and subscribe to mailing list.

III. Methodology

The “shopatujjain” Web Application is PHP based application, which is deployed on Apache Tomcat Server. It uses database on MySQL to store the data for the web site in its tables. The application uses PHP to present the user interface. It also uses HTML, CSS, JavaScript, and AJAX technologies. The presence of such technologies as AJAX and JavaScript in our web application gives additional opportunities. JavaScript is widely used in modern web applications and it is important to analyze the behavior of tools and their ability to parse JavaScript code.

The web application developed is based on OWASP Top Ten report of 2014. In this section we go over the characteristics of vulnerabilities presented in the Web Application.

1.1 SQL Injection Vulnerability

User has provided his/her credentials, username and password via web application. Web application has stored the user data to the SQL server. An attacker crafts HTTP requests that are sent to the web server to inject commands to the SQL server in order to gain system level access [15]. The vulnerable web application allows this malicious code to be placed on an SQL server, thus making it possible for the attacker to use SQLI commands to get user account credentials.

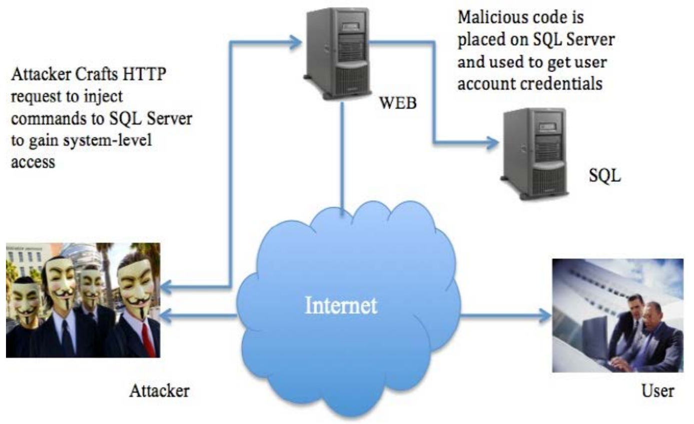

Attacker Crafts HTTP request to inject commands to SQL Server to gain system-level access
WEB
Malicious code is placed on SQL Server and used to get user account credentials
SQL
Internet
Attacker
User

Figure Hacking Strategy of SQLI

Exploiting SQLI vulnerability: During SQLI Attack, a malicious string is used as an input to a function that calls an SQL query, which is executed immediately. In this way, the injection result is reflected right away, thus the vulnerability is called Reflected SQLI vulnerability.

For example, recoverPassword function is intended to recover the user's password based on his/her answer to a security question.

String ***recoverPassword****(String emailAddress, String answer){*
…
String query = "SELECT Password FROM v_UserPass WHERE (v_UserPass.EmailAddress = "' + emailAddress + "' AND v_UserPass.Answer = "' +
answer + "') ";
…
}
Payload:
emailAddress=test%40test.com%27%29 -- &answer=anycolor

In recoverPassword function, concatenation is used to create dynamic SQL query. An attacker can easily impersonate a site user and recover a victim's password by commenting out the part of the query using '--' single-line comment indicator [15].

1.2 Broken Authentication and Session Management Vulnerability

The user authentication on the web typically involves the use of a user's ID and password. When the authentication mechanism does not provide enough protection, an attacker can try to obtain credentials by using different techniques or some other combination. Simple password recovery mechanisms can become victims of a social engineer who manipulates a user into revealing confidential information.

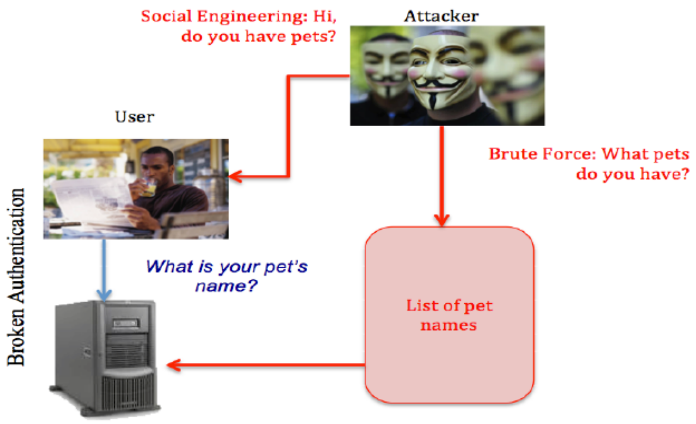

Social Engineering: Hi, do you have pets?
Attacker
User
Brute Force: What pets do you have?
Broken Authentication
What is your pet's name?
List of pet names

Figure: Two ways to bypass Broken Authentication

Exploiting Broken Authentication Vulnerability

The password recovery mechanism is based on a secret question and answer. A user provides the name of the city, when he/she was born and his/her password is immediately displayed on a web page without further verifications. Using social engineering, an attacker can guess the country. Then by using a dictionary method, the attacker finds the city and obtains the victim's credentials. Brute force attack is widely used to obtain log-in credentials, session identifiers, and credit card information with the help of brute force tools [9]. Attackers can use these tools and proxy applications such as BurpSuite to access a user's private information.

Brute force attack is very simple:

1. The intercepted request is sent to the Intruder application
2. The parameter, which is supposed to be brute forced, is selected.
3. The payloads are formed and configured to be used in the task.
4. The attack begins.

1.3 Cross Site Scripting Vulnerability

Cross Site Scripting (XSS) vulnerability occurs when there is a possibility of injection of malicious code in web application. Thus, the XSS flaw is as a result of not validated or sanitized input parameters. There are three types of XSS: Non-Persistent, called Reflected XSS; Persistent or Stored XSS; and Document Object Model (DOM)-based [16].

- **Non-Persistent XSS Vulnerability:** This vulnerability occurs when a web application accepts an attacker's malicious request that is then echoed into the application's response in an unsafe way.
- **Persistent XSS Vulnerability:** This vulnerability occurs when a web application accepts the attacker's malicious request, stores it in a data source, and later displays the information from the request to a wide range of users.
- **DOM-Based XSS Vulnerability:** This vulnerability doesn't involve server validation. The attack works on

a web browser, avoiding the server side [16]. The DOM 'environment' in the victim's browser is modified by original client-side script, and as a result of that, the payload is executed.

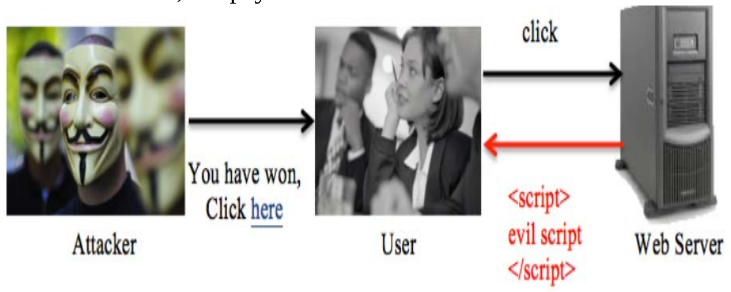


Figure: Non-Persistent XSS Vulnerability

Exploiting XSS Vulnerability

XSS vulnerabilities are exploited by using XSS attacks. XSS attacks are usually divided into three categories: Non-Persistent or Reflected XSS Attack; Persistent or Stored XSS Attack; and DOM-Based XSS Attack [16].

- **Non-Persistent or Reflected XSS Attack:** User registration information is saved in an online store database after 'creditCardNumber' parameter is validated on the server side. No input inspection for 'firstName' parameter is performed.

<form action="registrationServlet" method=post>
First Name <input type="text" name="firstName" value="${newUser.firstName}">
Card number <input type="text" name="creditCardNumber">
<input type="button" value="Continue">
</form>
Payload:
firstName=John"'><script>alert("firstName parameter is vulnerable")</script>&creditCardNumber=1234

If the credit card number is incorrect, 'firstName' value will be reflected on the web page.

1.4 Security Misconfiguration Vulnerability

This type of vulnerability occurs when application, frameworks, application server, web server, database server, and platform configurations are not securely defined to prevent unintentional leakage of information. For example, a web application can use the GET method in an HTTP request for transferring password information. But while using the GET method, the browser encodes form data into a URL. Since form data is in the URL, it is displayed in the browser's address bar, and information leakage occurs.

GET http://www.vulnerableApp.com/updateUserPassword?password =falsepass HTTP/1.1
Host: vulnerableApp.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:11.0) Gecko/20100101 Firefox/11.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,/*;q=0.8*
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Proxy-Connection: keep-alive
Referer: http:// vulnerableApp.com/displayAccountPassword
Cookie: JSESSIONID=98224C7236B39895384AD3A760E405AB

While using the POST method, form data appears within the message body of the HTTP request, not the URL. Thus, password information is not revealed. To avoid security misconfiguration vulnerability in the above example, the password should be transferred via POST method.

IV. Defense Mechanisms against Web Vulnerability and Secure Coding Techniques

Preventing vulnerabilities in web applications is extremely important due to the high number of attacks. The best way to prevent vulnerabilities in applications is to write secure code. According to Computer Emergency Response Team, or CERT, at the Software Engineering Institute at Carnegie-Mellon University, the following Top 10 Secure Coding Practices [17] are vital to security.

1. Proper implementation of Input Validation helps to avoid most of the web application vulnerabilities. But, on the other hand, handling each input in isolation to avoid unexpected command line arguments, user controlled files, and other suspicious input is a complex task, and as a result, the validation may be omitted.
2. Warnings and Error messages can suggest the places of possible security flaws for both developers and an attacker. Static and dynamic analysis tools can detect and eliminate the vulnerabilities.
3. Strong web application architecture helps to enforce security policies.
4. Simple design helps to avoid errors that can be made during implementation, configuration, and use.
5. To simplify the access mechanism, by default the access is denied. In other words, "Everything not explicitly permitted is forbidden."
6. To continue the ideas in points 4 and 5, the principle of least privilege is introduced, which suggests the execution of a process using the least set of privileges necessary to complete the job.
7. Before data is processed, it should be sanitized. The un-validated data could be the cause of SQL, command, or other injection attacks.
8. In- depth defense mechanisms help to improve security by adding layers of multiple defensive strategies, so that if one layer of defense turns out to be inadequate, another layer of defense can prevent a security flaw from becoming an exploitable vulnerability, and/or limit the consequences of a successful exploit.
9. Quality Assurance is the key point in security of the software. There are different techniques to improve reliability of the application, like using source code analysis tools, penetration testing tools, and independent review of the system.
10. A secure coding standard should be adopted. Programmers should develop and/or apply a secure coding standard for the target development language and platform.

With Top 10 Secure Coding Practices for each vulnerability type, we provide the defense mechanism for top four vulnerabilities out of top ten OWASP vulnerabilities.

1.5 SQLI Defense

Server Side defense using Prepared Statement [18] is the most effective way to protect from SQL Injections, because it ensures that intent of query is not changed. For example, the insertPassword(User user) function adds a new record to UserPass table in "shopatujjain" application database, when a new customer is registering his/her account.

```
public static int insertPassword(User user) {
ConnectionPool pool = ConnectionPool.getInstance();
Connection connection = pool.getConnection();
PreparedStatement ps = null;
ResultSet rs = null;
String query ="INSERT INTO UserPass (EmailAddress, Password, Answer) VALUES (?, ?, ?)";
try {
ps = connection.prepareStatement(query);
ps.setString(1, user.getEmailAddress());
ps.setString(2, user.getPassword());
ps.setString(3, user.getAnswer());
return ps.executeUpdate();
} catch (SQLException e) {
e.printStackTrace();
return 0;
} finally {
DBUtil.closeResultSet(rs);
DBUtil.closePreparedStatement(ps);
pool.freeConnection(connection);
}
}
```

In this example, PreparedStatement object is used with parameters. Before executing the query, all special characters will be escaped. All SQL functions, those that are not intended to be exploited while stress testing [19] the application, are developed using PreparedStatements.

1.6 Cross-Site Scripting (XSS) Defense

For prevention code injection attacks, including SQLI and XSS, all user data should be validated. There are several main rules that should be followed to increase security:

- Check the data type and set length limits on any form fields on your site.
- Encode or escape the data where it is used in your application to ensure that the browser treats the possibly dangerous content as text, and not as active content that could be executed.

From a security perspective, however, client-side validation is not effective, because it doesn't provide protection for server-side code. An attacker can easily bypass the clientside using proxies.

1.7 Security Misconfiguration Defense

Maintaining security settings of the application, frameworks, application server, web server, database server, and platform is a very complex problem. Web servers are frequent targets of attacks, so when trying to secure web servers, the following aspects should be taken into account [20]:

- Configuration
- Web content and server-side applications
- Operating System
- Documentation

Example:

HTTP server is subject to Slow type HTTP Attack [21]. There is number of steps to protect against this attack pattern [22].

The RequestReadTimeout directive value should be set to limit the time a client may take to send the request [23].

The implementation of defense mechanisms is an important part of the code analysis that is performed to increase the security of a web application. Some vulnerability can be exploited only if an attacker performs several steps successively or in specific order.

V. Observations

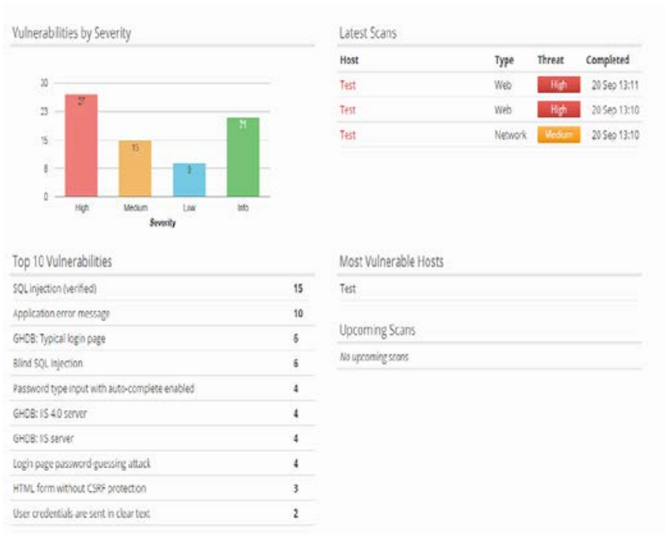
A customer cannot feel fully secured while using an application as long as there is a possibility of losing some personal information or other confidential data. Firstly, as many security flaws as possible should be discovered in order to secure a web application. To improve the success rate of discovering application flaws Web Application Vulnerability Scanners (WAVS) are used. WAVS are tools that most closely mimic web application attacks. These tools cannot guarantee that their use will eliminate the flaws completely, but they can make the application more secure. Web Application Security Scanner Functional Specification Version 1.0 [24] in 2008 defined a list of requirements that all WAVS must provide:

- Identify all types of vulnerabilities listed.
- Report an attack that demonstrates the vulnerability.
- Specify the attack by providing script location, inputs, and context.
- Identify the vulnerability with a name semantically equivalent.
- Be able to authenticate itself to the application and maintain logged-in state.
- Have an acceptably low False Positive rate.

In this paper three prominent free Web Application Security Scanners (Acunetix, Netsparker and Burp Suite) are used for vulnerabilities detection. The scanning results of Web Application Vulnerability Scanners are as follows:

1.8 Acunetix

Acunetix Web Vulnerability Scanner (WVS) [7] is an automated web application security testing tool that audits web applications by checking for vulnerabilities like SQL Injections, Cross-Site Scripting and other exploitable hacking vulnerabilities. In general, Acunetix WVS scans any website or web application that is accessible via a web browser and uses the HTTP/HTTPS protocol.



Latest Scans

Host	Type	Threat	Completed
Test	Web	High	20 Sep 13:11
Test	Web	High	20 Sep 13:10
Test	Network	Medium	20 Sep 13:10

Top 10 Vulnerabilities

SQL injection (verified)	15
Application error message	10
GHDB: Typical login page	6
Blind SQL injection	6
Password type input with auto-complete enabled	4
GHDB: IIS 4.0 server	4
GHDB: IIS server	4
Login page password guessing attack	4
HTML form without CSRF protection	3
User credentials are sent in clear text	2

Most Vulnerable Hosts

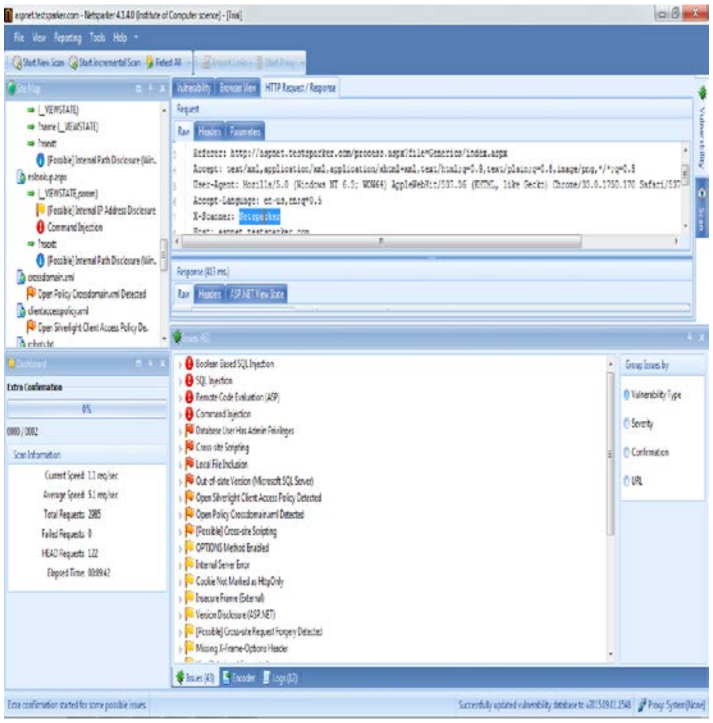
Test

Upcoming Scans

No upcoming scans

1.9 Netsparker

Netsparker does not require a brief knowledge to use the tool, it has a very good user interface, and it does a decent job detecting the most important vulnerabilities [8]. It has good reporting features that are easy to read and intuitively designed. Moreover it has ability to confirm detected vulnerabilities. This feature can be a real time saver as the tester does not need to validate those vulnerabilities that have been confirmed by Netsparker. The scanning results of Netsparker are shown below



```
Referer: http://aspnet.testsparker.com/process.aspx?file=Generics/index.aspx
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/33.0.1750.170 Safari/537
Accept-Language: en-us,en;q=0.5
X-Scanner: Netsparker
Host: aspnet.testsparker.com
```

- [_VIEWSTATE]
- 'Name [_VIEWSTATE]
- 'Invalid
- [Possible] Internal Path Disclosure (Win...
- rssbook.aspx
- [_VIEWSTATE_power]
- [Possible] Internal IP Address Disclosure
- Command Injection
- 'Invalid
- [Possible] Internal Path Disclosure (Win...
- crossdomain.xml
- Open Policy Crossdomain.xml Detected
- clientaccesspolicy.xml
- Open Silverlight Client Access Policy De...
- robots.txt

0000 / 0002

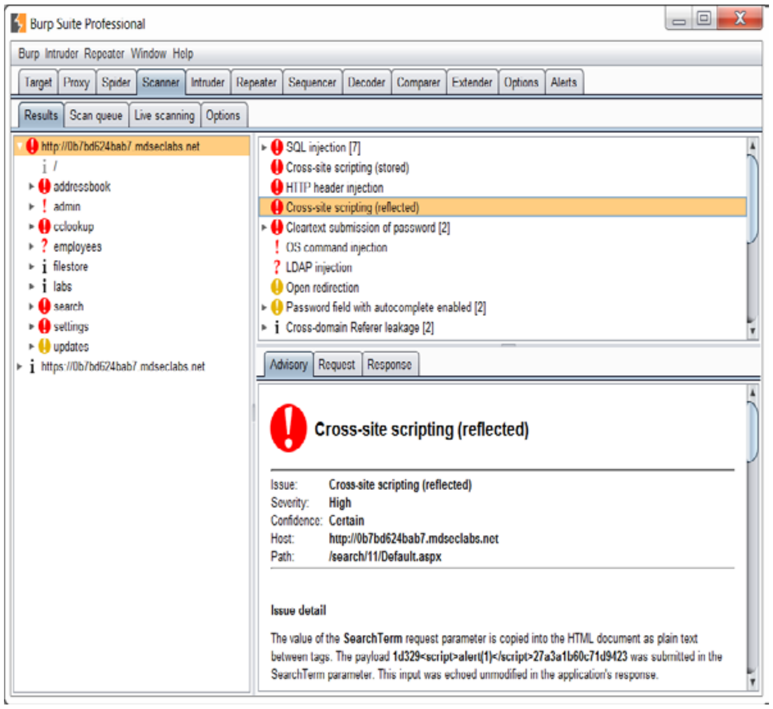
Current Speed: 1.1 req/sec
Average Speed: 5.1 req/sec
Total Requests: 2985
Failed Requests: 0
HEAD Requests: 122
Elapsed Time: 00:09:42

- Boolean Based SQL Injection
- SQL Injection
- Remote Code Evaluation (ASP)
- Command Injection
- Database User Has Admin Privileges
- Cross-site Scripting
- Local File Inclusion
- Out-of-date Version (Microsoft SQL Server)
- Open Silverlight Client Access Policy Detected
- Open Policy Crossdomain.xml Detected
- [Possible] Cross-site Scripting
- OPTIONS Method Enabled
- Internal Server Error
- Cookie Not Marked as HttpOnly
- Insecure Frame (External)
- Version Disclosure (ASP.NET)
- [Possible] Cross-site Request Forgery Detected
- Missing X-Frame-Options Header

Extra confirmation started for some possible issues. Successfully updated vulnerability database to v.2015.09.01.1548 Proxy: System(None)

1.10Burp Suite

Burp is easy to use and intuitive, allowing new users to begin working right away. Burp is also highly configurable, and contains numerous powerful features to assist the most experienced testers with their work. The scanning results of Burp Suite are shown below



- http://0b7bd624bab7.mdseclabs.net
  - /
  - addressbook
  - admin
  - cclookup
  - employees
  - filestore
  - labs
  - search
  - settings
  - updates
- https://0b7bd624bab7.mdseclabs.net

- SQL injection [7]
- Cross-site scripting (stored)
- HTTP header injection
- Cross-site scripting (reflected)
- Cleartext submission of password [2]
- OS command injection
- LDAP injection
- Open redirection
- Password field with autocomplete enabled [2]
- Cross-domain Referer leakage [2]

Cross-site scripting (reflected)

Issue:	Cross-site scripting (reflected)
Severity:	High
Confidence:	Certain
Host:	http://0b7bd624bab7.mdseclabs.net
Path:	/search/11/Default.aspx

Issue detail

The value of the **SearchTerm** request parameter is copied into the HTML document as plain text between tags. The payload **1d329<script>alert(1)</script>27a3a1b60c71d9423** was submitted in the SearchTerm parameter. This input was echoed unmodified in the application's response.

There are some free WAVS available in the market. We reviewed three of them: Acunetix Web Application Scanner (WAS), Netsparker Web Vulnerability Scanner and Burp Suite Web Vulnerability Scanner (WVS). All WAVS follow the common strategy: firstly they crawl the victim web site, then they create and insert payloads, and finally they analyze the response. We have chosen these scanners because they provide the feature that; they identify all types of vulnerabilities listed in OWASP Top Ten report

VI. Evaluation of Web Application Vulnerability Scanners

The results of Web Vulnerability Scanners Acunetix, Netsparker and Burp Suite are shown in Table 1. The Table contains the following data:

- The first column represents the serial number.
- The second column represents the vulnerability number taken from Top Ten OWASP Vulnerabilities.
- The third column represents the vulnerabilities presented in the test suite.
- The fourth column shows the different types of a vulnerability presented in the third column.
- The fifth column contains the number of vulnerabilities detected by Acunetix WAVS.
- The sixth column contains the number of vulnerabilities detected by Netsparker WAVS.
- The last column represents the number of vulnerabilities detected by Burp Suite WAVS.

Table: Results of WAVS assessment

S No	OWASP report 2015 Number	OWASP Vulnerabilities	Vulnerability Type	Acunetix	Netsparker	Burp Suite
1	A1	SQL Injection		15	4	7
2	A2	Broken Authentication and Session Management	Password Guessing	5	0	2
			Brute Force	1	1	0
3	A3	Cross Site Scripting	Non-Persistent XSS	9	9	2
			Persistent XSS	1	3	1
			DOM XSS	3	1	0
4	A5	Security Misconfiguration	Password sent via GET Method	5	5	5
			Web Server DDoS	2	0	2
			Sensitive Data display	0	4	2
Total				40	27	18

The Table 1 reports the vulnerabilities that were detected by web application scanners. As seen from the Table 1 all the tool tools missed some weaknesses. The analysis of why the scanners missed certain vulnerabilities is as follows

1.11SQL Injection: Acunetix Scanner is able to discover all SQL Injection vulnerabilities. But Netsparker and Burp Suite scanners are failed to find some SQL Injection vulnerabilities, which are not executed immediately.

1.12Broken Authentication and Session Management: Both Netsparker and Burp Suite scanners were not able to find the vulnerability.

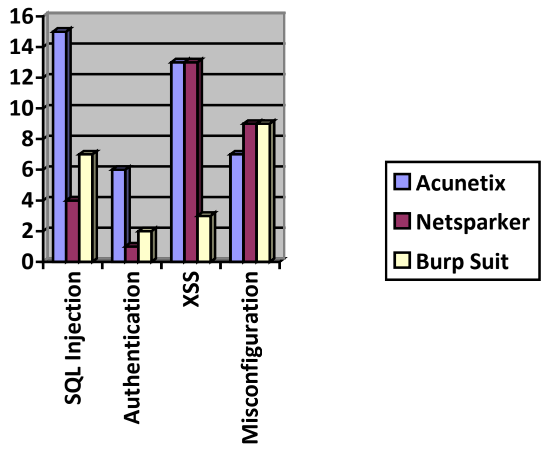
1.13Cross-Site Scripting: Acunetix and Netsparker Scanners discovered all NonPersistent XSS vulnerabilities. Burp Suite scanner result is very poor. Most of the Persistent XSS and DOM XSS vulnerabilities were missed by all scanners.

1.14Security Misconfiguration: All the scanners are able to find the vulnerability Password get via GET Method. Acunetix Scanner missed Sensitive Data Display vulnerability.

VII. Conclusions

This paper described OWASP Top 10 Security Risks implemented in the web application, which was used as a testset for evaluation of effectiveness of Acunetix web application vulnerability scanners, Netsparker web application vulnerability scanners and Burp Suite web application vulnerability scanners. We chose four vulnerabilities from Top 10 OWASP Security Risks for evaluation of three prominent Web Application Vulnerability Scanners. The evaluation of three prominent Web Application Vulnerability Scanners is done by analyzing the results that is obtained from the execution of web scanners against the vulnerable web application, then comparing the number of detected vulnerabilities.

The comparison of the three chosen scanners shown by the following graph:



The result show that both Acunetix and Netsparker scanners able to discover cross site scripting XSS but Burp Suit results was very poor. For SQL Injection Acunetix detect all the vulnerabilities. Scan results of Acunetix WAVS for Broken Authentication and Session Management vulnerabilities are better than other two scanners. But Security Misconfiguration vulnerabilities are not properly discovered by Acunetix, in this case the result of Netsparker and Burp Suit Scanners are better. The results show that the crawling has been significantly improved, although there are still limitations that affect the detection rate of such vulnerabilities as SQLI and XSS.

For several vulnerabilities presented in this application, we also explain defense measures, which secure the application significantly. The results of web application evaluation identify the most challenging vulnerabilities for scanner to detect, and compare the effectiveness of scanners. The assessment results can suggest areas that require further research to improve scanner's detection rate.

REFERENCES

[1] Sarasan S. "Detection and Prevention of Web Application Security Attacks", International Journal of Advanced Electrical and Electronics Engineering, (IJAEEE), ISSN (Print) : 2278-8948, Volume-2, Issue-3, 2013, pp. 29- 34.

[2] International Organization for Standardization and International Electrotechnical Commission. ISO/IEC 27001:2005, Information technology – security techniques – information security management systems – requirements, 2005.

[3] National Vulnerability Database, http://nvd.nist.gov

[4] N. Antunes and M. Vieira, "Enhancing Penetration Testing with Attack Signatures and Interface Monitoring for the Detection of Injection Vulnerabilities in Web Services," Proc. IEEE Int'l Conf. Services Computing (SCC 11), IEEE CS, 2011, pp. 104-111.

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

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

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

[8] Netsparker Web Vulnerability Scanner, 2012, https://www.netsparker.com/web-vulnerability-scanner/

[9] Burp Suit Web Vulnerability Scanner, https://portswigger.net/burp/

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

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

[12] https://www.owasp.org/images/0/0f/OWASP_T10_-_2015_rc1.pdf

[13] Foundstone Hacme Series. McAfee Corp

[14] WebGoat Project. OWASP. http://www.owasp.org/index.php/Category:OWASP WebGoat Project

[15] K. K. Mookhey, Nilesh Burghate, Detection of SQL Injection and Cross-site Scripting Attacks, Symantec Connect Community, 02 November 2010

[16] J. Weinberger, P. Saxena, D. Akhawe, M. Finifter, R. Shin, and D. Song, "A Systematic Analysis of XSS Sanitization in Web Application Frameworks", University of California, Berkeley, 2011

[17] The OWASP Foundation, "OWASP Top Ten Web Application Security Risks", http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project, 2015

[18] Oracle Documentation. "Using Prepared Statements", 2011. Retrieved 2012 from: http://docs.oracle.com/javase/tutorial/jdbc/basics/prepared.html

[19] Yang Guang, J. J., & Jipeng, H. "System modules interaction based stress testing model", 2014. The Second International Conference on Computer Engineering and Applications, (pp. 138-141) Bali Island

[20] Neto, A. A., Duraes, J., Vieira, M., & Madeira, H. "Assessing and Comparing Security of Web Servers", 2008. 14th IEEE Pacific International Symposium on Dependable Computing. IEEE Computer Society

[21] Shekyan, S. Qualys Community. "Identifying Slow HTTP Attack Vulnerabilities on Web Applications", 2013

[22] Shekyan, S. Qualys Community. "How to Protect Against Slow HTTP Attacks", 2014

[23] Apache Software Foundation. "Security Tips, V 2.5", 2011. Retrieved 2014, from: http://httpd.apache.org/docs/2.0/misc/security_tips.html

[24] Black, P. E., Fong, E., Okun, V., & Gaucher, R. National Institute of Standards and Technology (NIST). "Software Assurance Tools: Web Application Security Scanner Functional Specification"

[25] Vieira M, Antunes N, Madeira H. "Using Web Security Scanners to Detect Vulnerabilities in Web Services", Coimbra - 2015

AUTHORS

First Author – Chanchala Joshi, Institute of Computer Science Vikram University, Ujjain, M.P. India, chanchala.joshi@gmail.com
Second Author – Umesh Kumar Singh, Institute of Computer Science, Vikram University Ujjain, M.P. India umeshksingh1@gmail.com