Web Security

Software Security – DV2546

Enugurthi Madhukar
9506077537
maen16@student.bth.se

Siddhartha Srinadhuni
9508286854
sisr16@student.bth.se

TASK 1 - Vulnerability repository

Out of the several software repositories that have been mentioned, National vulnerability database has been chosen. The data yielding out of this source enables the user to automate the vulnerability management, measurement of security and acquiescence [1]. As this is United States government repository and the standards are maintained for managing the vulnerabilities of data, we have been motivated to check from this source.

Description of the Vulnerability.

The selected vulnerability has a CVE identifier as CVE-2016-2873 which is an exposure with regards to security in *IBM QRadar SIEM* 7.1*. The vulnerability in this case is identified as SQL Injection. An attacker irrespective of his geo-location could send specially crafted SQL statements which would in turn allow the attacker to add, modify, delete the information in the Back-end database [1].

**IBM QRadar SIEM* is a security intelligence platform that allows to protect any organization from cyber threats and attacks by adopting security analytics [2].

Risks of SQL Injection:

- The consequences of SQL Injection are loss of confidentiality, authentication problems, access to unauthorized information and loss of integrity [3].
- This vulnerability can be further exploited and the attacker can also become the administrator of the database server[4].

Countermeasures [5]:

The traditional method to prevent SQL Injection is to handle the attacks as an input validation problem accept characters from whilelist of safe values. Whilelisting is said to be effective but since the maintenance cost is relatively high, parameterized SQL statements are considered to be an optimal way to handle this vulnerability. Defining the code and then passing the parameters of the query is called parameterized SQL statements.

Stored procedure calls, White List Input Validation, Escaping All User Supplied Input are also considered to be feasible solutions for SQL Injection attack.

Since this is a database centric attack, check should be taken while filtering the escape characters before passing them into the SQL statement.

TASK 2 : WEBGOAT

[12] CODE QUALITY:

Poor code quality leads to unpredictable behaviour. From the user's perspective it often

regards to the usability factor. It gives an opportunity for the attacker to attack the system in many ways.

For this task, the exploitation was carried out with the source code of the HTML page . The source code was viewed by clicking right click on the html page and selecting view page source After the code was inspected , we found the admin's password i.e) adminpw and we typed it on the login page and it worked as we were logged in as admin. The result is shown in figure

1.

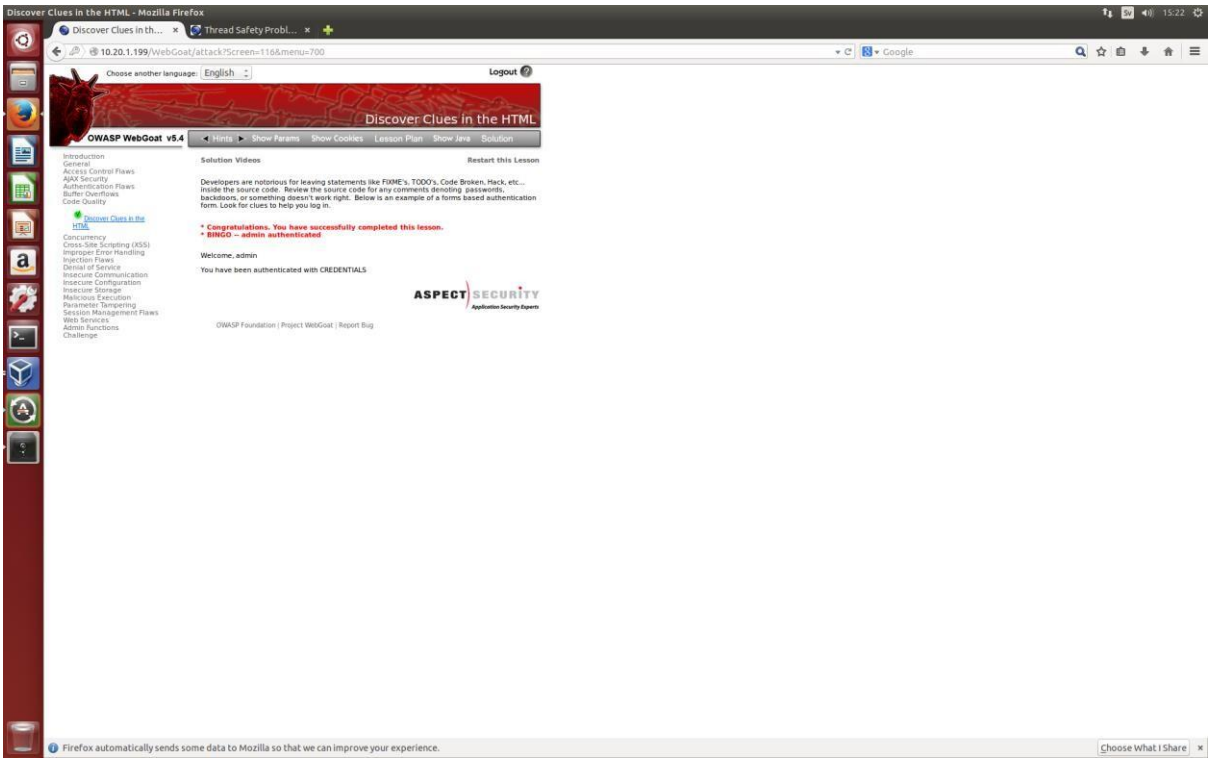

Figure 1

Countermeasures :[6]

The source code of the HTML page should have limitations in showing the context , when the code is being developed , certain validations are to be given to the code. This can be done by giving the developers certain set of rules before programing the context. Use of Automated tools to check the code quality for security concerns can also be done

CONCURRENCY :

In this Vulnerability we exploit thread safety problems. For exploitation, as given in the html page we opened two browsers and typed the two user names “jeff” and “dave” separately in two pages and clicked submit simultaneously in the span of a second . In the end result only one of the name’s account information was displayed in the screen and it was dave and jeff’s name wasn’t displayed. This showcases the concurrency vulnerability and its known as race condition. The result is shown in figure 2.

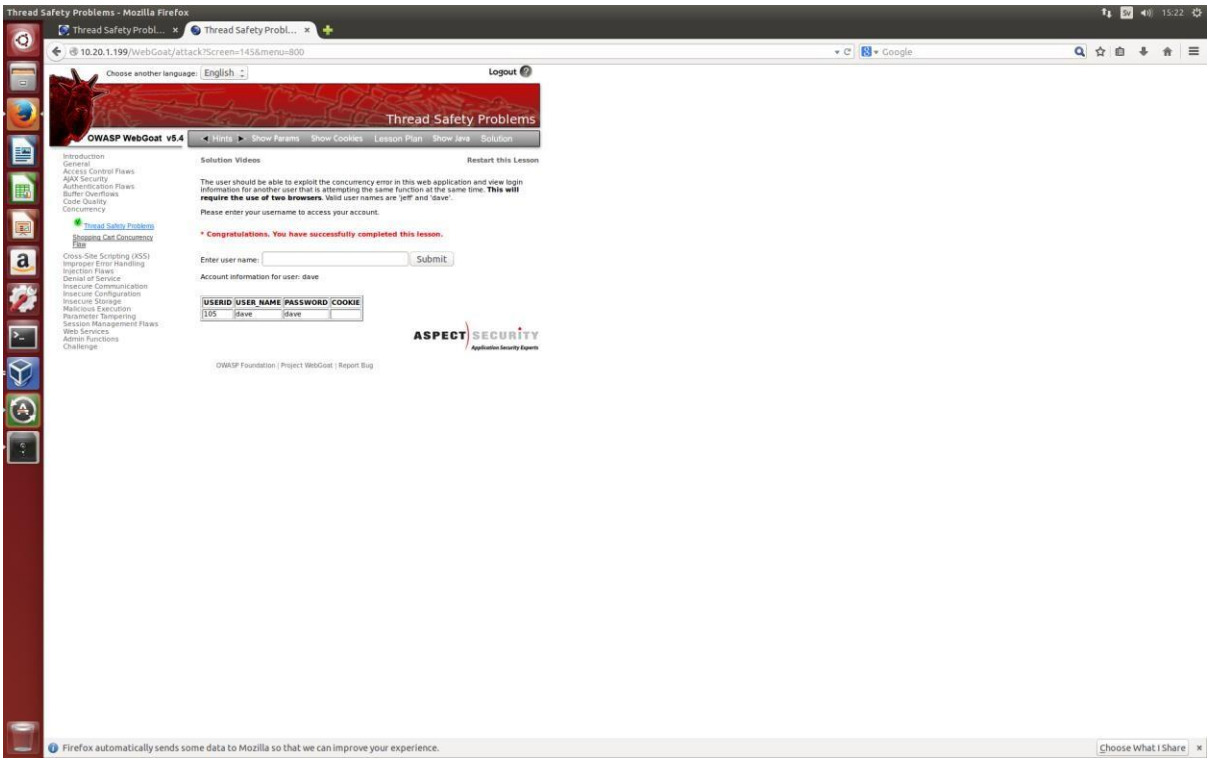

Figure 2

Countermeasures **:[6]**

The test cases should be validated by certain currency control mechanisms such as serializability , recoverability , distributed serializability . By implementation of these mechanisms race around conditions can be avoided.

AUTHENTICATION FLAWS:

For this vulnerability , we exploited forgot password section. For this task as mentioned in the html page we entered the username as admin and submitted. Then for Webgoat password

recovery it asked for the favourite color of admin. Then through brute force we typed different colors and submitted, and by typing color ‘green’ admin’s password was retrieved as shown in figure 3

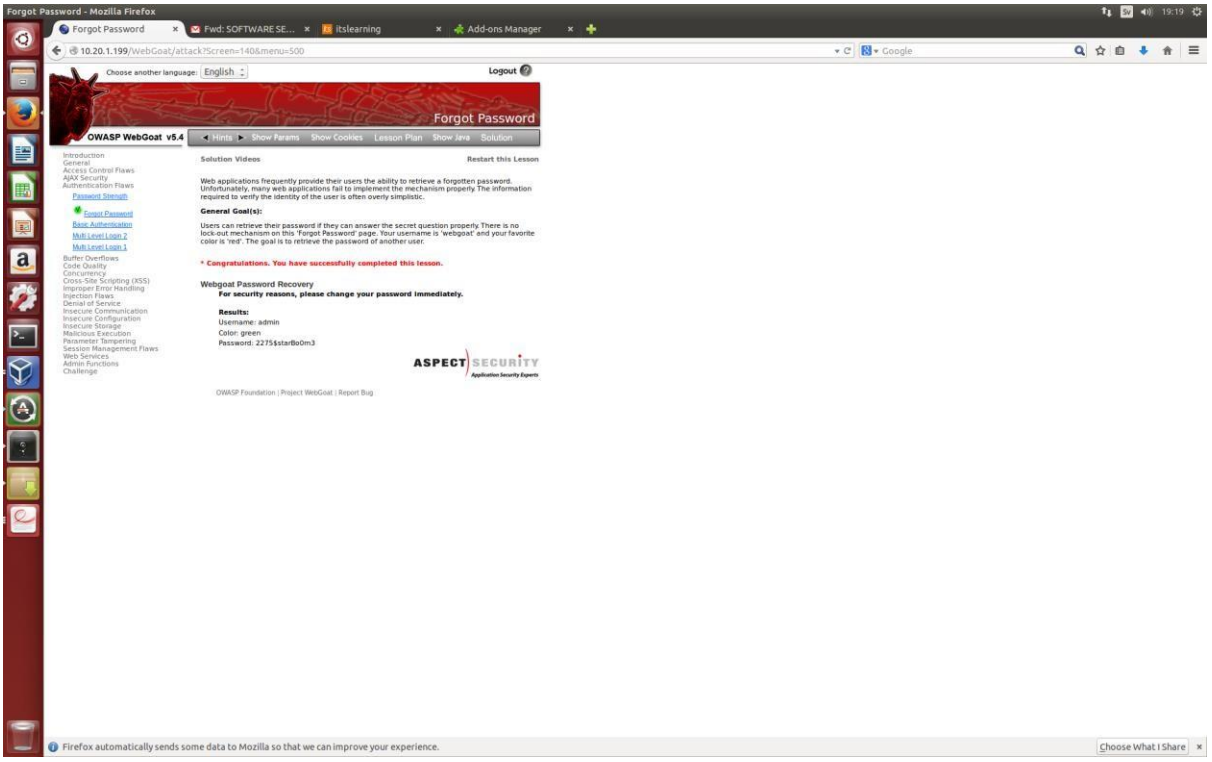

Figure 3

Countermeasures

:

The security question should be strong enough for the attacker to guess. While doing brute-force attacks we perform the actions many number of times. The number of attempts should be validated.

TASK 3 : MUTILLIDAE

1. Injection(SQl) - SQLi – Extract Data – User Info(SQL)

   In this vulnerability we perform SQL injection to extract info of all the users. The exploitation was done by performing SQL injection by referring from this source as follows [6]. The characters such as single-quote, back-slash, double-hyphen, forward- slash. Injections are done on the basis of true and false values

   ‘ or 1 = 1 - -

   Both username and password was the above text . By clicking view account details we get the users info as shown in figure 4

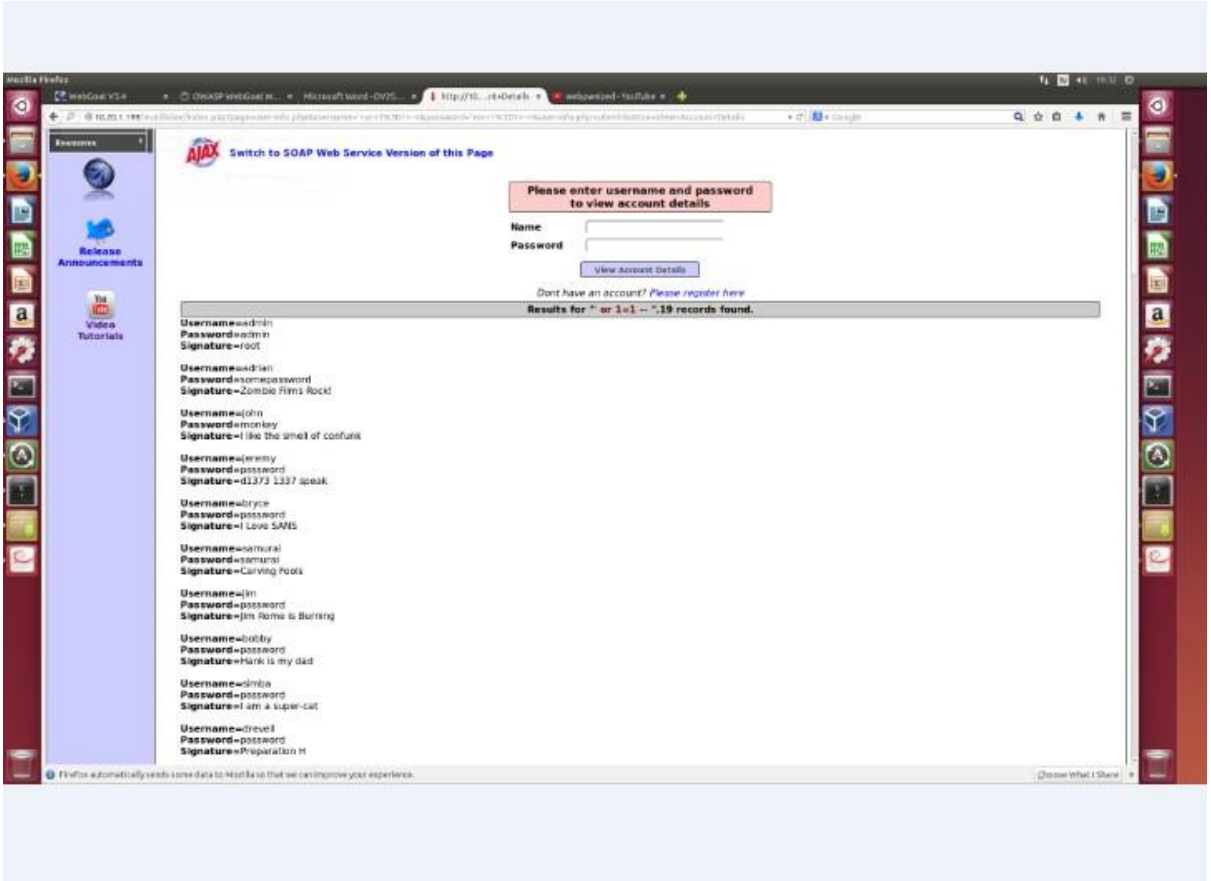

Figure 4

2. Injection(SQl) – SQLi – Bypass Authentication – Login

In this vulnerability we perform SQL Injection by bypassing authentication. The exploitation was done by performing SQL injection with reference from the source [6]as follows :

‘ or ‘1’ = ‘1

Both username and password is the same text and by clicking login we get signed in as admin as shown in figure 5

Figure 5

Countermeasures for 1 & 2 :[7]

1. Dynamic SQL queries should be prevented
2. Safe character set should be used eliminating the special characters
3. Input validations should be whitelisted
4. Specific escape syntax should be used to escape from special characters

3. Broken Authentication and Session management – Authentication Bypass – Via Cookies

In this vulnerability ,authentication bypass is exploited via cookies as follows :

First a user is registered as username : sunny and password : 123 and we login to mutillidae. Then we installed cookie manager + to check cookies and we see in the cookie uid the value was given as 24 , so by brute force we edited it to 1 and we saved it. By refreshing the website admin was logged into mutillidae and hence the attack is performed. The steps as shown in figure 6 , 7

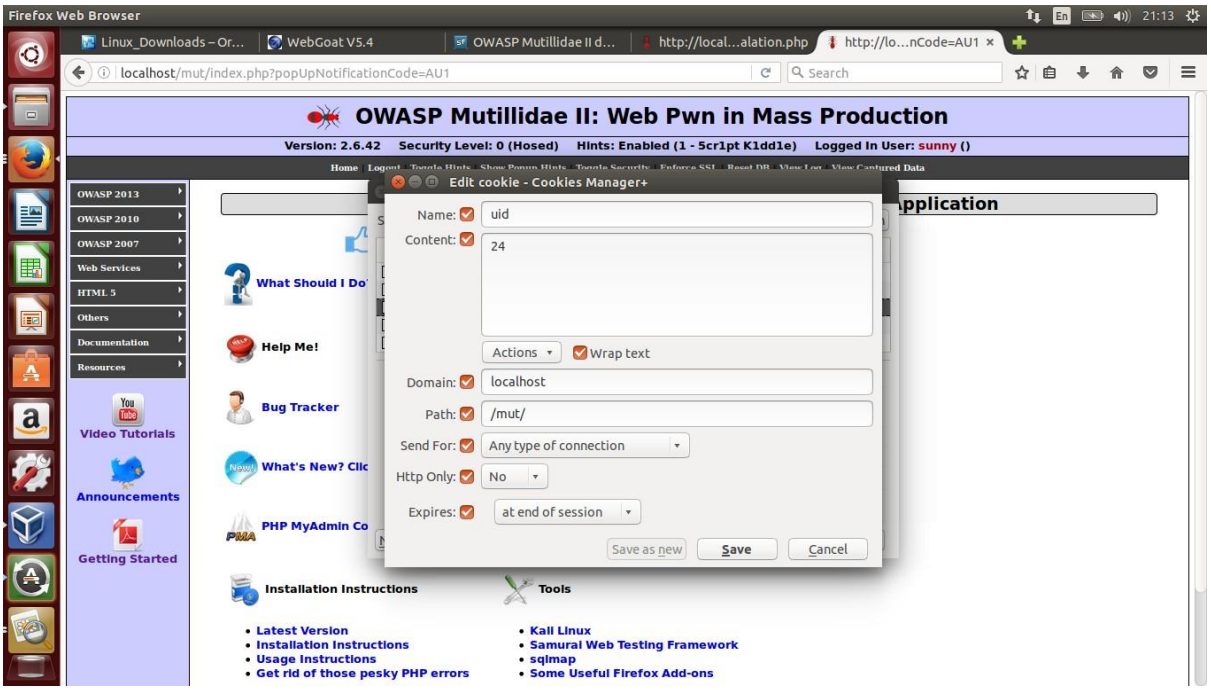

Figure 6

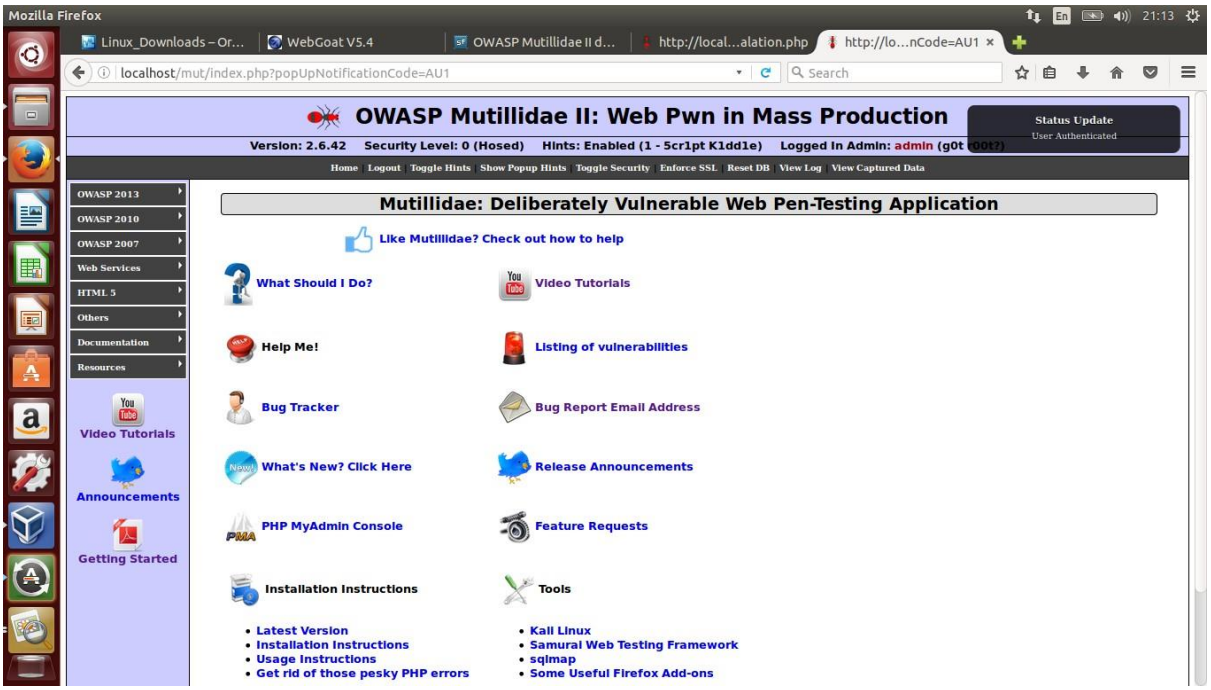

Figure 7

4. Broken Authentication and Session management - Authentication Bypass – Via SQL Injection

In this vulnerability , authentication bypass is exploited via SQL Injection as follows :

This is done with tampering data. First username is entered as admin and password is typed as any random number and in this case we chose ‘123’ and then from tools we selected tamper data and we click on start tamper. The next step is to login as admin in the html page and the tamper info tab appears as shown in figure 8 then we edit the right side of the tab via SQL injection by typing ‘ or ‘1’ = ‘1 as shown in figure 9 and is saved. Then the html page is refreshed and we find that admin is logged into mutillidae as shown in figure 10.

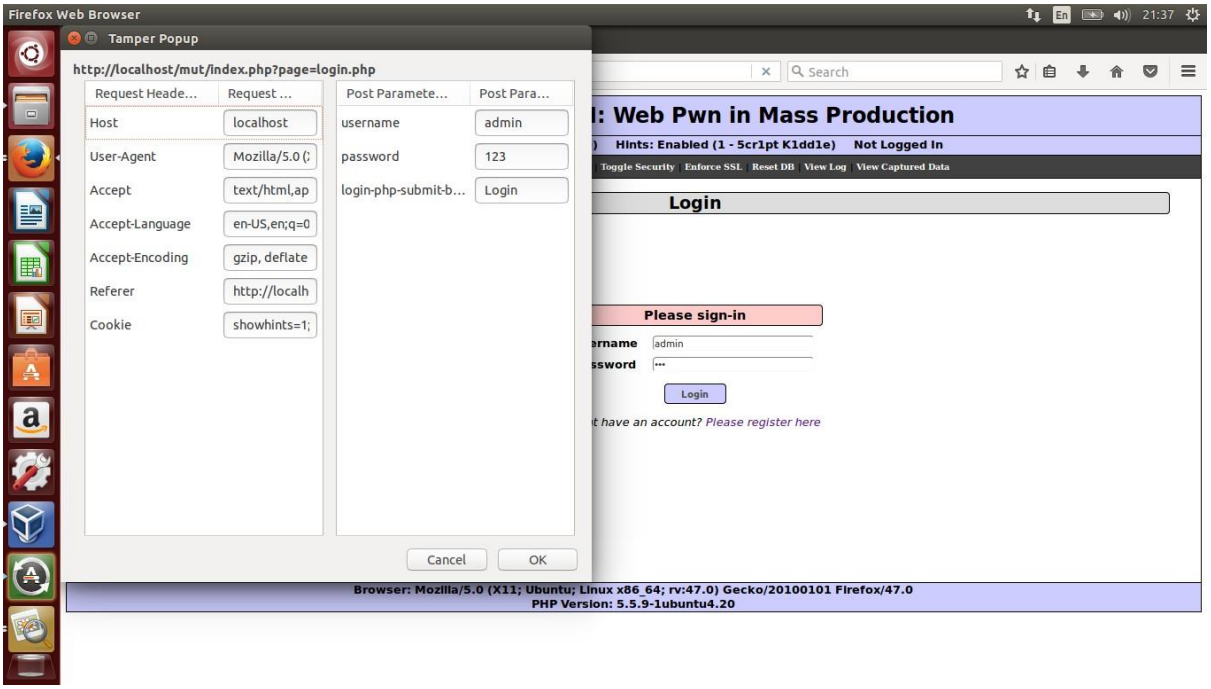

Figure 8

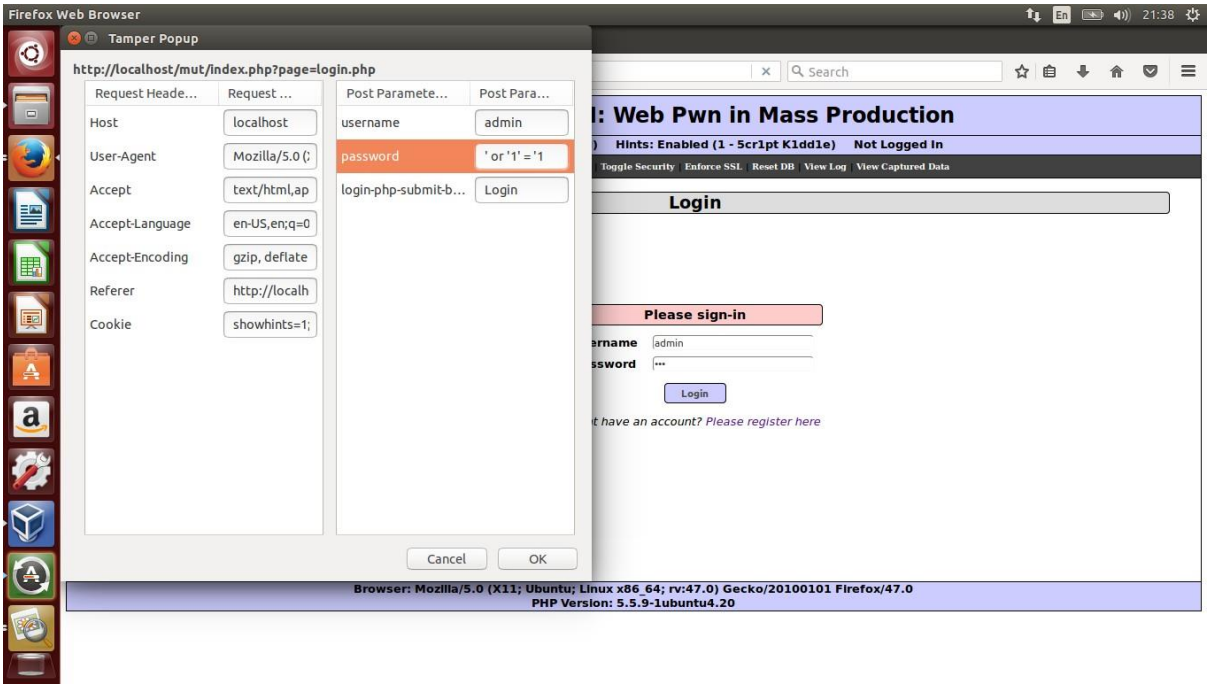

Figure 9

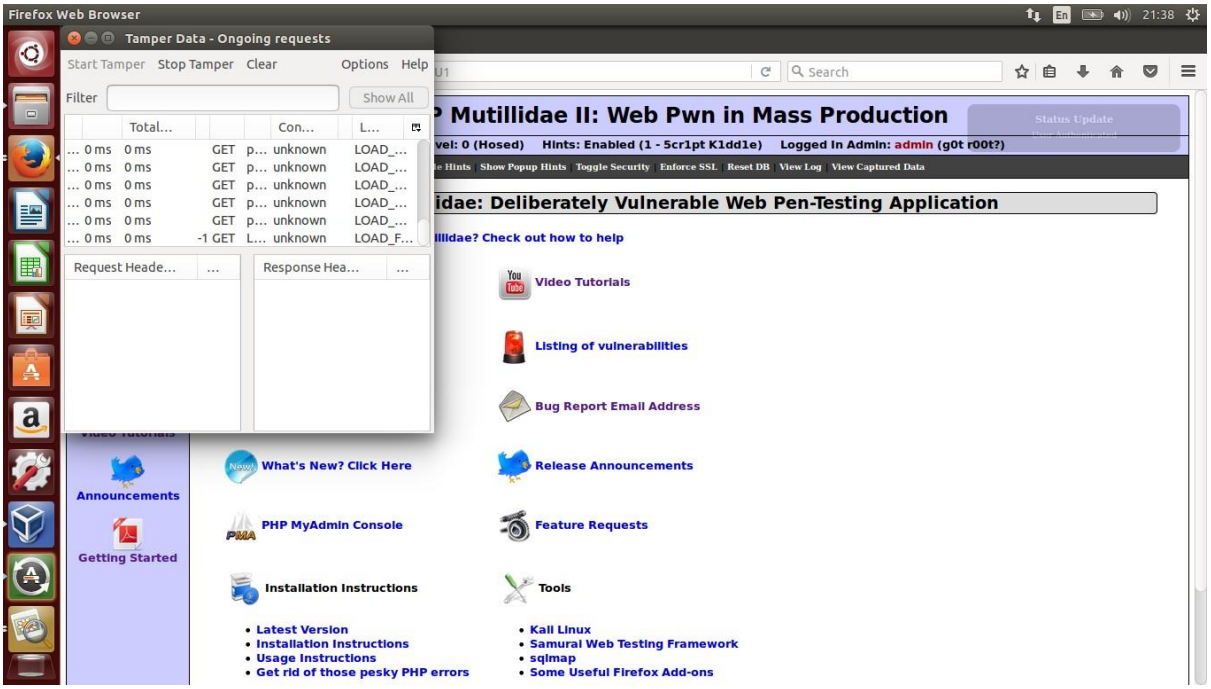

Figure 10

Countermeasures for 3 & 4 :[8]

1. Enabling encryption on request that has sensitive content
2. Actions should be taken to avoid XSS flaws
3. Invalidation of session ID after the desired pre-determined time for logging out the web application

5. Cross Site Scripting (XSS) – via Cookie Injection – Capture Data Page

In this vulnerability we exploited cross site scripting (XSS) via Cookie Injection by capture data page and we done data capture by page capture-data.php and the captured records are shown in the html page . Now the exploitation is done via cookie manager + and here we edit PHPSESSID cookie with code “<script>alert(0)</script>” as shown in figure 11 to figure 12. The cookie is saved and the page is refreshed , the results are shown in figure 13 , 14 and 14.a. The exploitation is shown in the captured page record with no PHPSESSID.

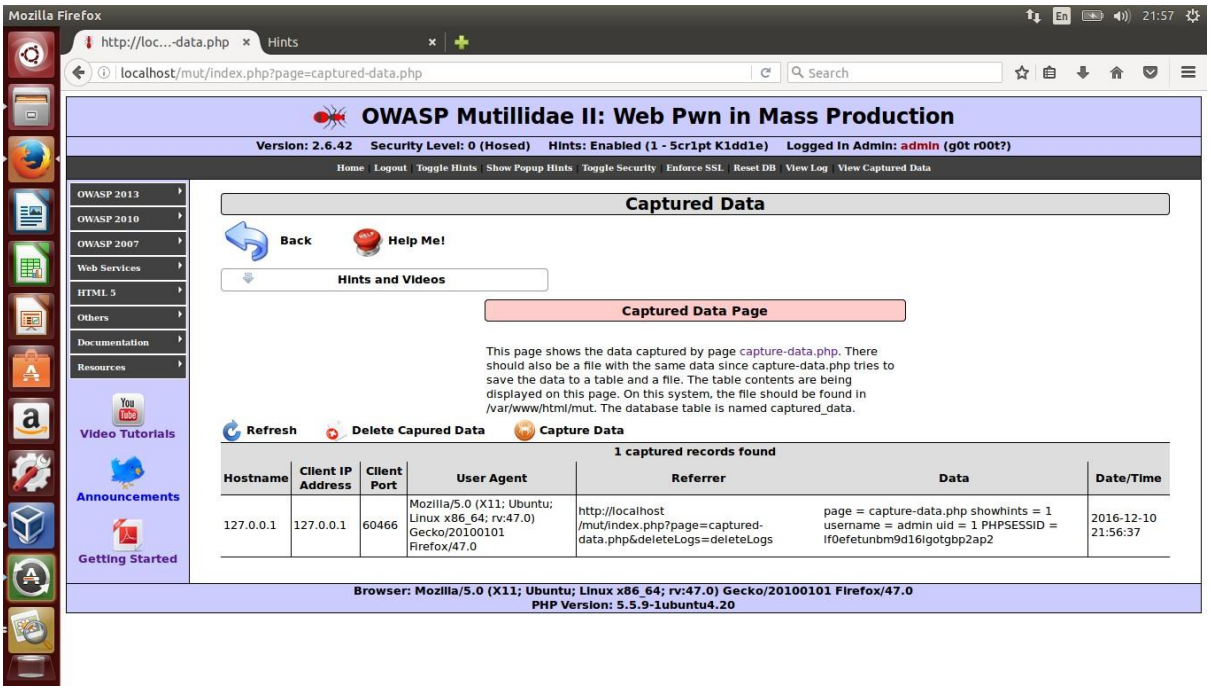

Figure 11

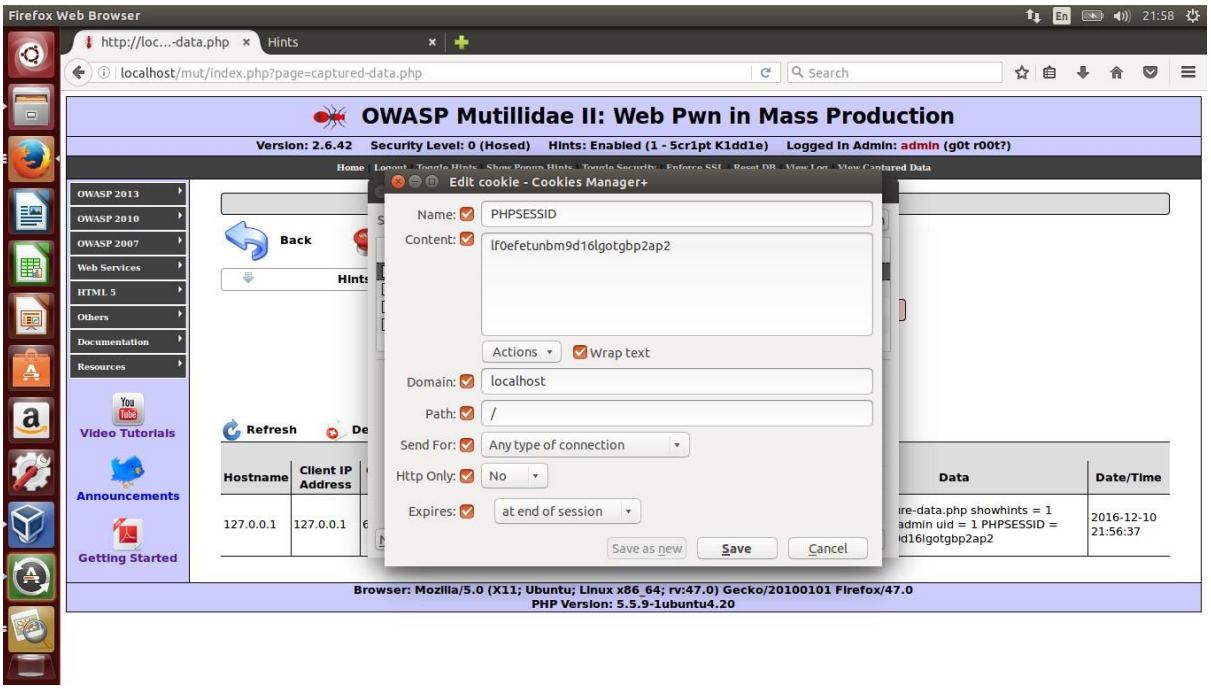

Figure 12

F

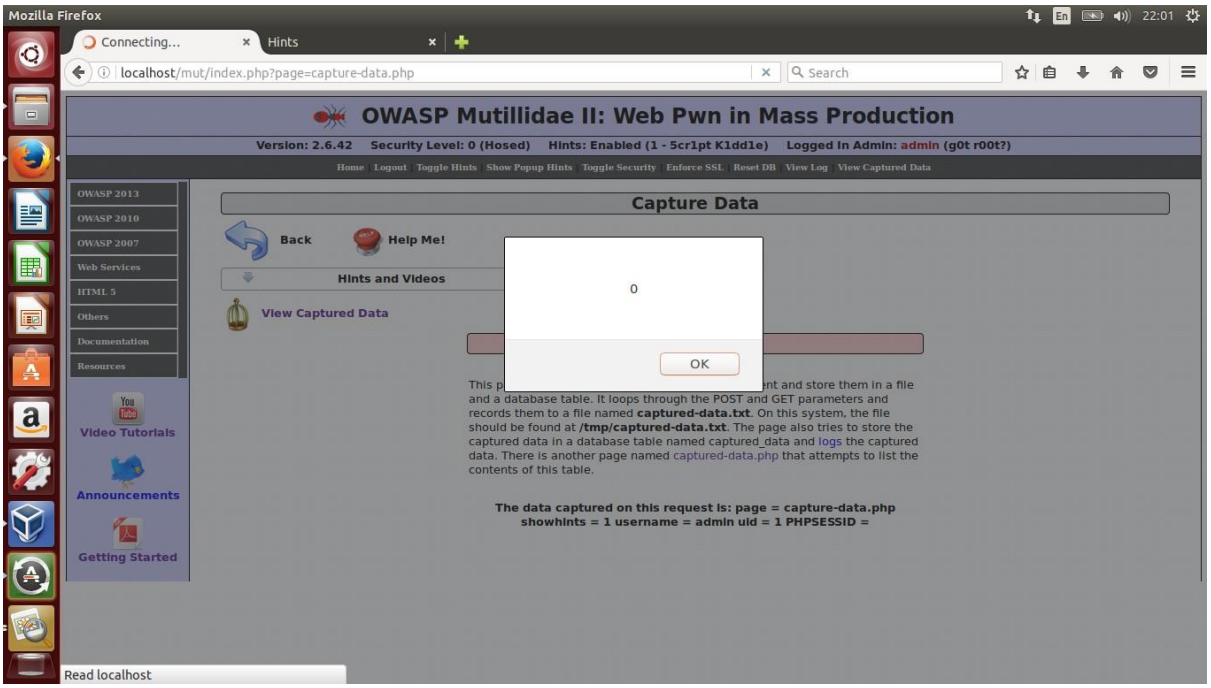

Figure 13

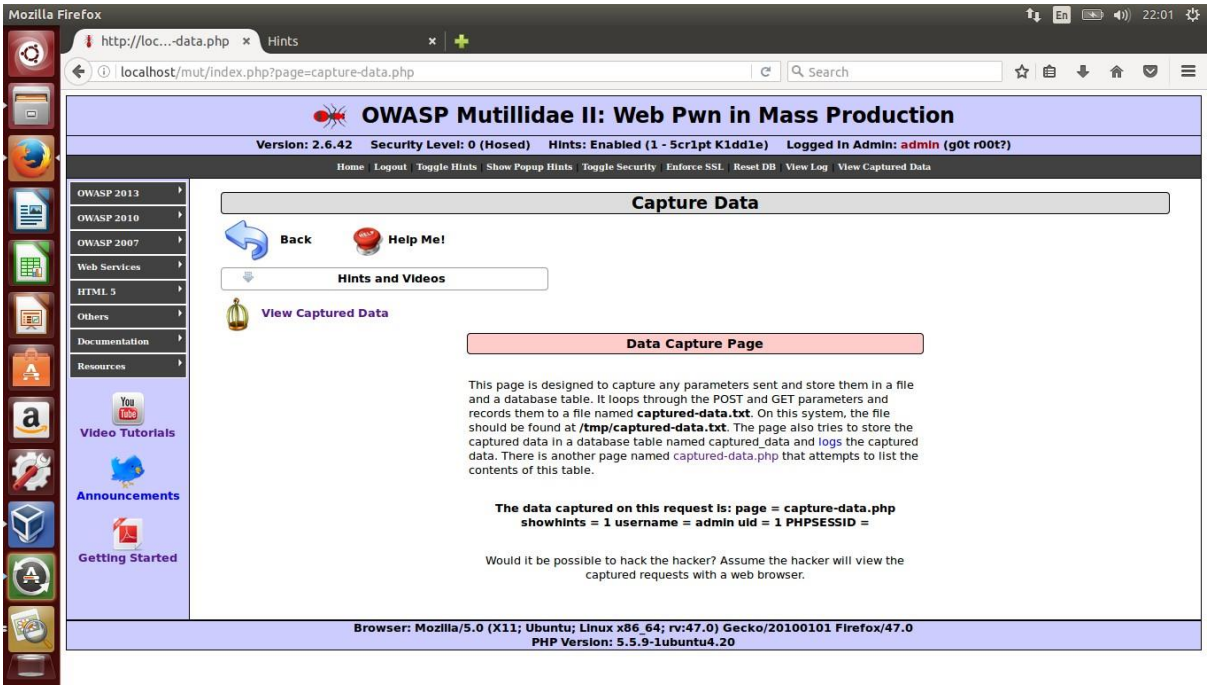

Figure 14

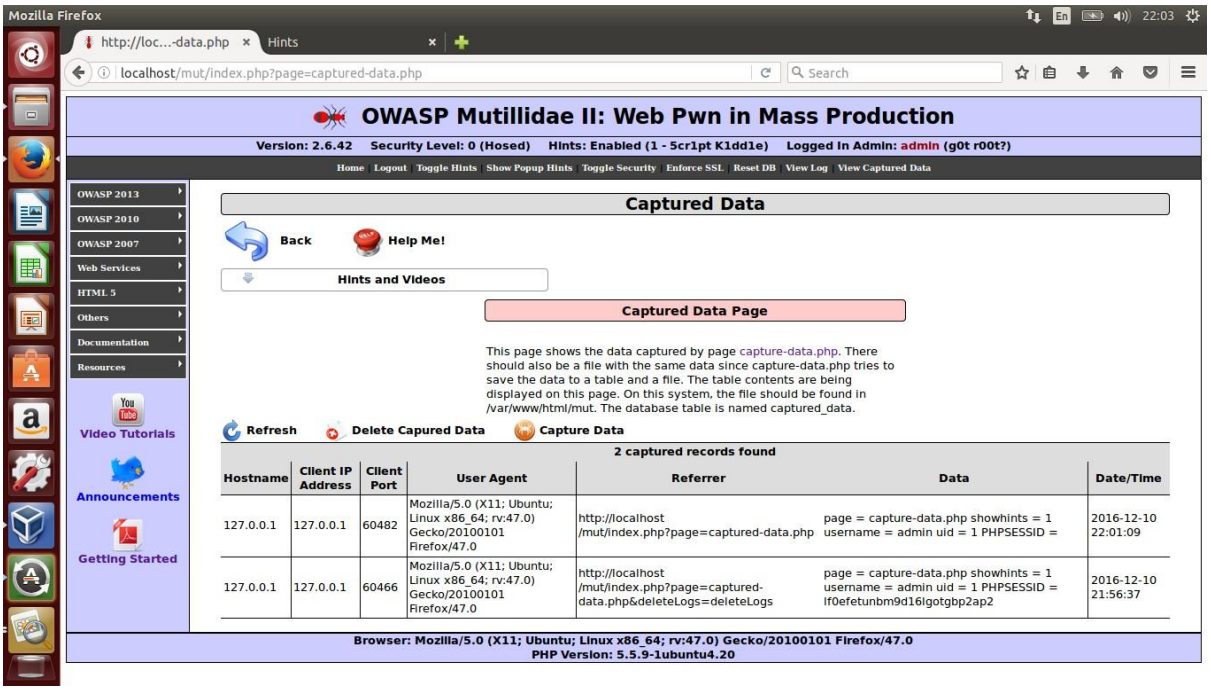

Figure 14.a

6. Cross Site Scripting (XSS) – Via "Input" (GET/POST)

In this vulnerability XSS is exploited via input through adding to the blog. While adding it was noticed that certain tags shouldn't be allowed from the note section. The exploitation is done by adding the code "<script>alert(0)</script>" and the blog entry is saved. Then the exploitation is shown in the blog entries as shown in figure 15.a,b,c.

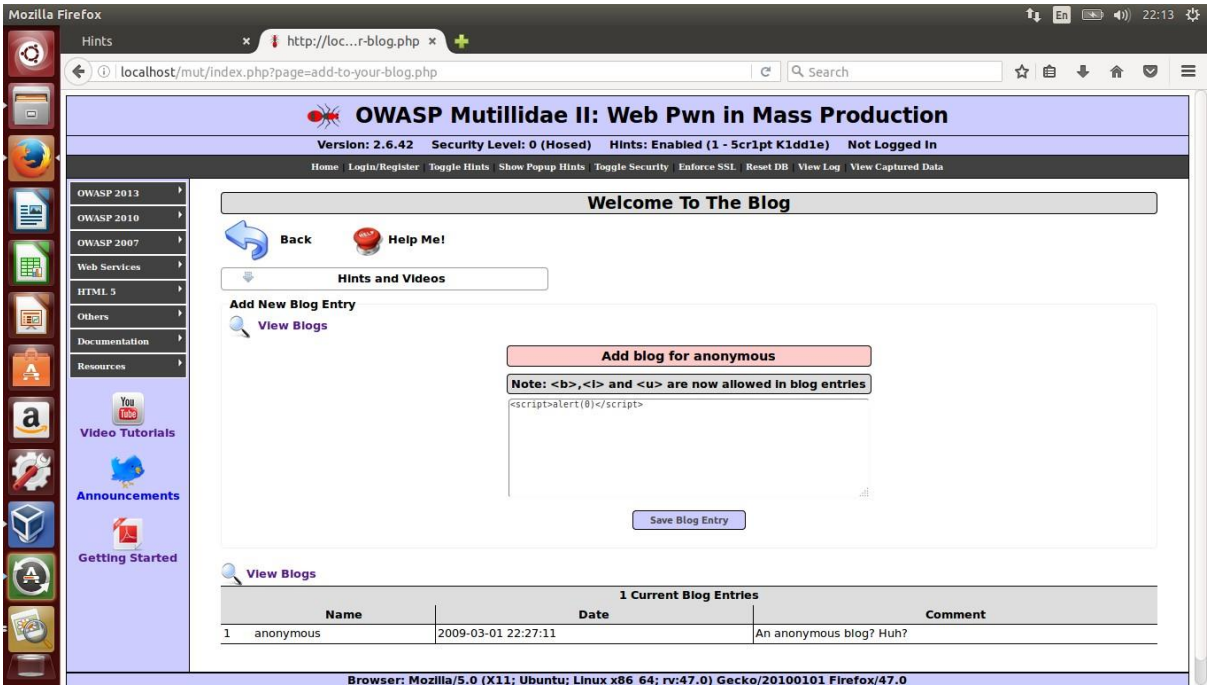

Figure 15.a

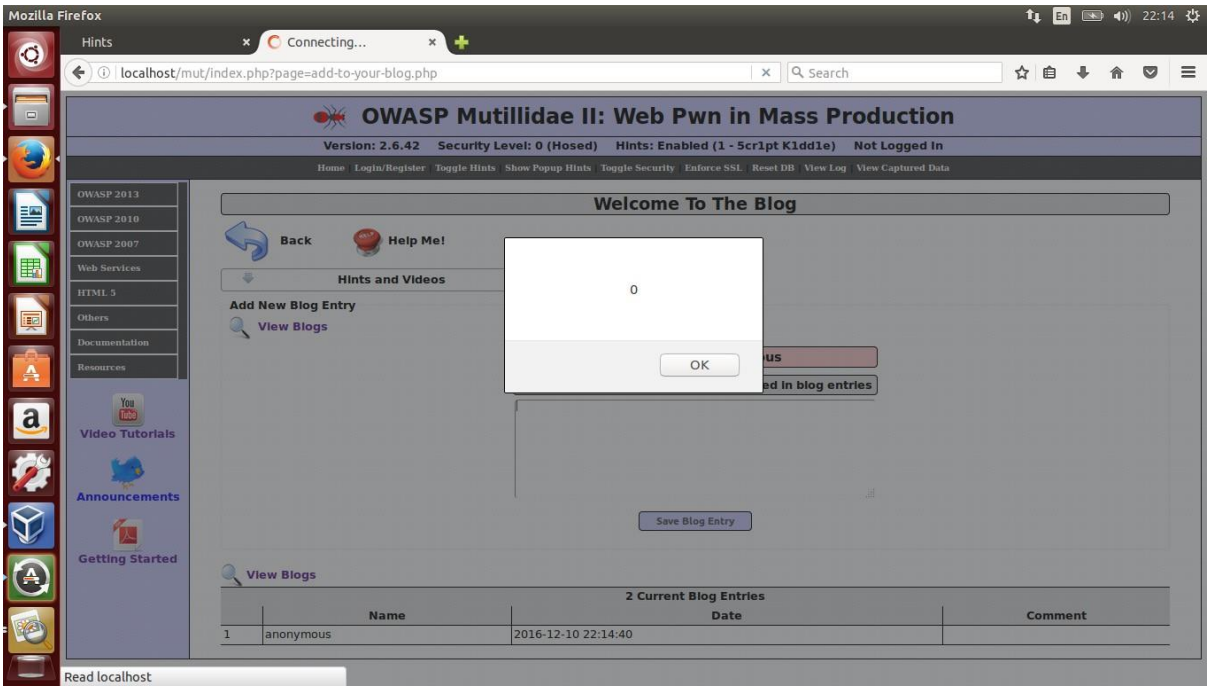

Figure 15.b

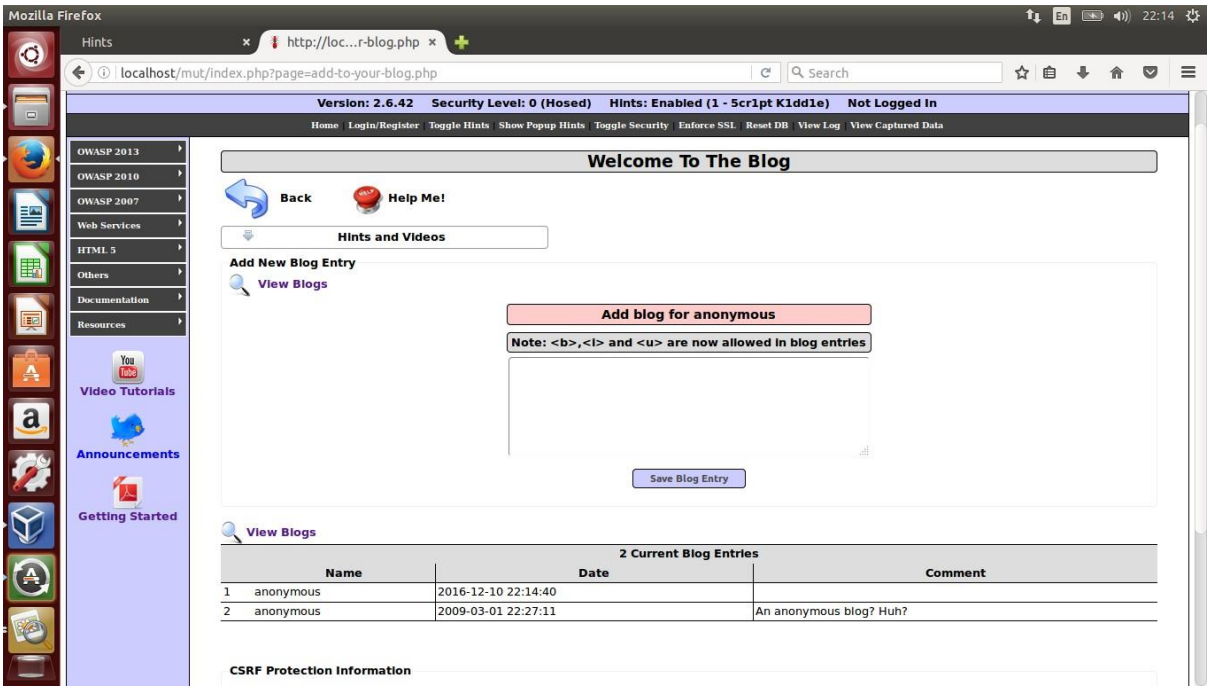

Figure 15.c

Countermeasures for 5 & 6 :[9]

1. Positive or white listing input validation protects against XSS
2. Output Encoding should be done to protect application from XSS
3. Untrusted Data shouldn't be allowed in the page

Figure 17

7. Insufficient Transport Layer Protection – Login

In this Vulnerability transport layer protection is exploited via login. It is done via cookie manager. Here mutillidae is logged by random member with username “user” and password “user” and by cookie manager uid of the current cookie was changed to ‘1’ and saved. By refreshing the website we find that admin is logged into mutillidae as shown in figure 16 and 17

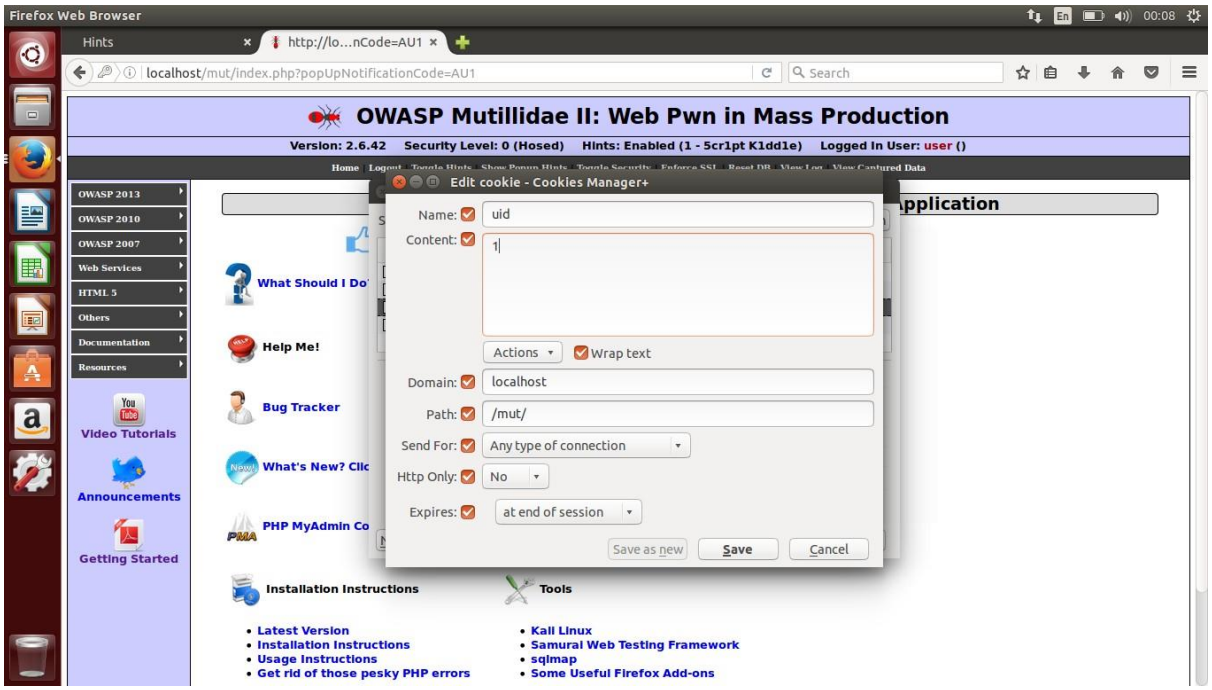

Figure 16

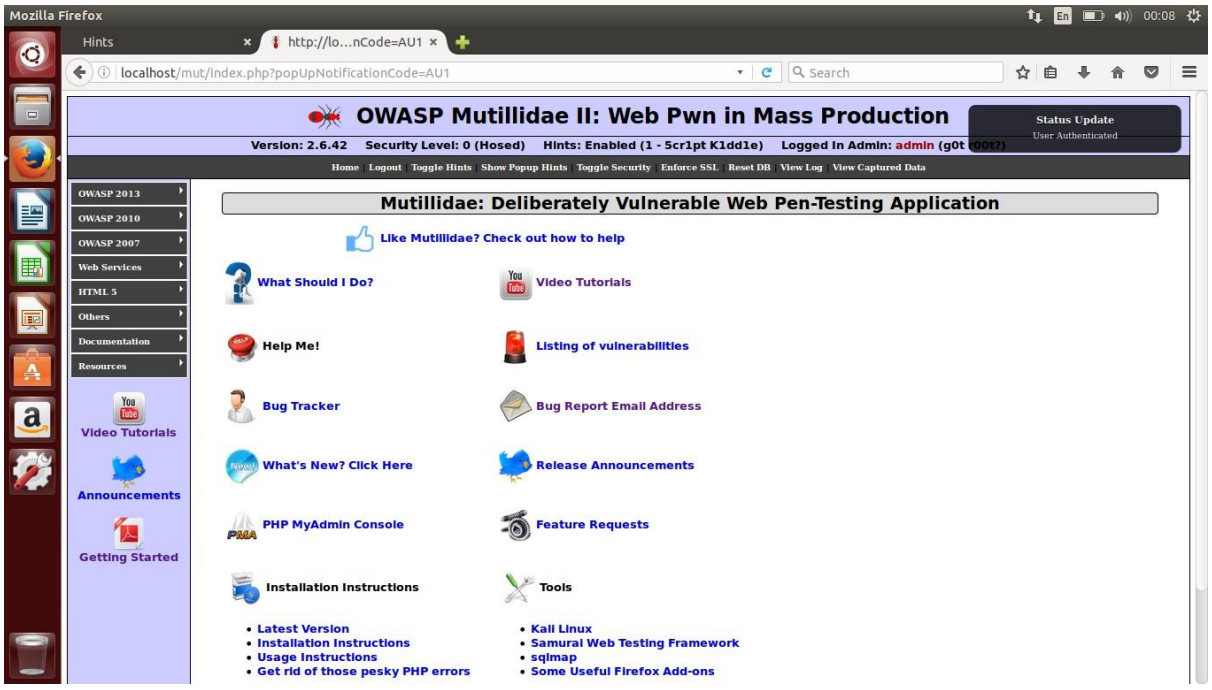

Figure 17

7. Insufficient Transport Layer Protection – Login

In this Vulnerability transport layer protection is exploited via login. It is done via cookie manager. Here mutillidae is logged by random member with username "user" and password "user" and by cookie manager uid of the current cookie was changed to 1 and saved. By refreshing the website we find that admin is logged into mutillidae as shown in figure 16 and 17

Countermeasures :[10]

1. Setting the secure flag on all the cookies
2. Ensuring that the certificates are valid and matches all domains of the given html page or website

8. Security Misconfiguration – Directory Browsing

In this vulnerability attack is done via directory.We found it that mutillidae has sensitive information in the form of directories such as passwords, phpinfo.php . This can be done by editing URL as shown in figure 18 . In the same way passwords, documentation etc can also be seen in the same way.

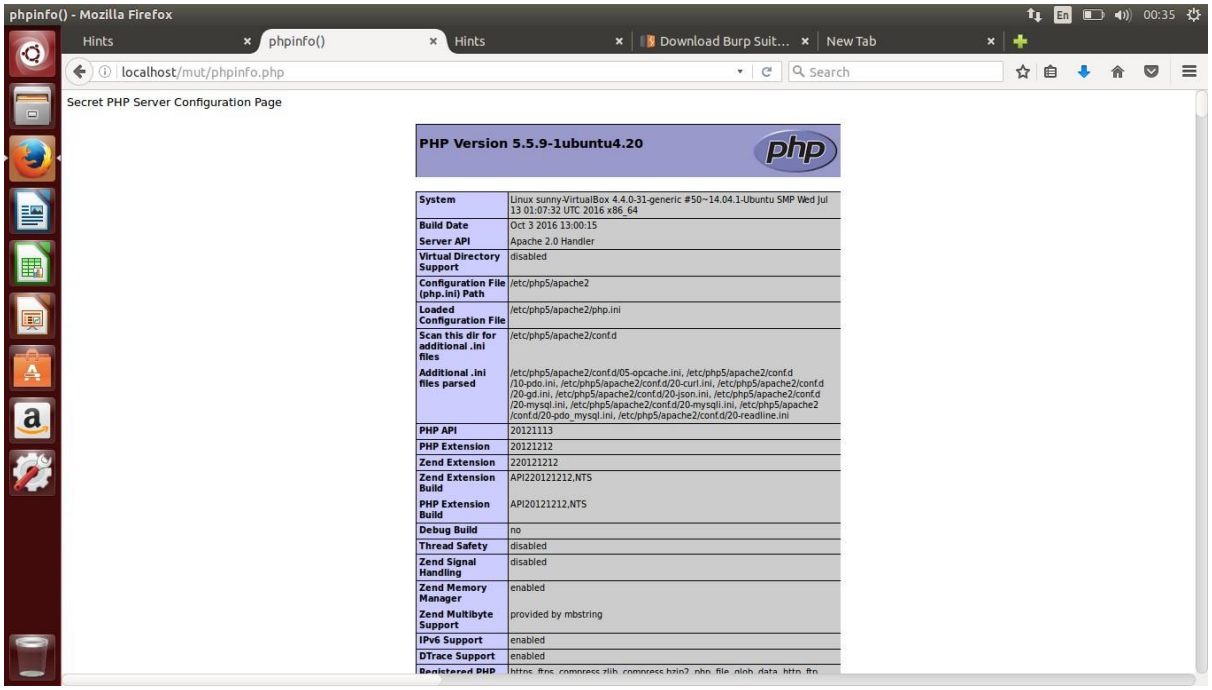

Figure 18

Countermeasures :[11]

1. Running scans during audits in periodic time intervals to prevent any future misconfiguration
2. A strong application architecture should be implemented
3. Configure server to prevent any unauthorized access

9. Sensitive Data Exposure – Information Disclosure- robots.txt

In this vulnerability sensitive information can be exploited via Robots.txt files. Webservers metafiles are stores in robots.txt files. Any attackers can use this information to identify valid directories from applications. The robots file provides a simple method in finding the folders on the web server without linked directly from the main as shown in figure 19. Attackers can steal assets from robots.txt which causes exploitation of sensitive info

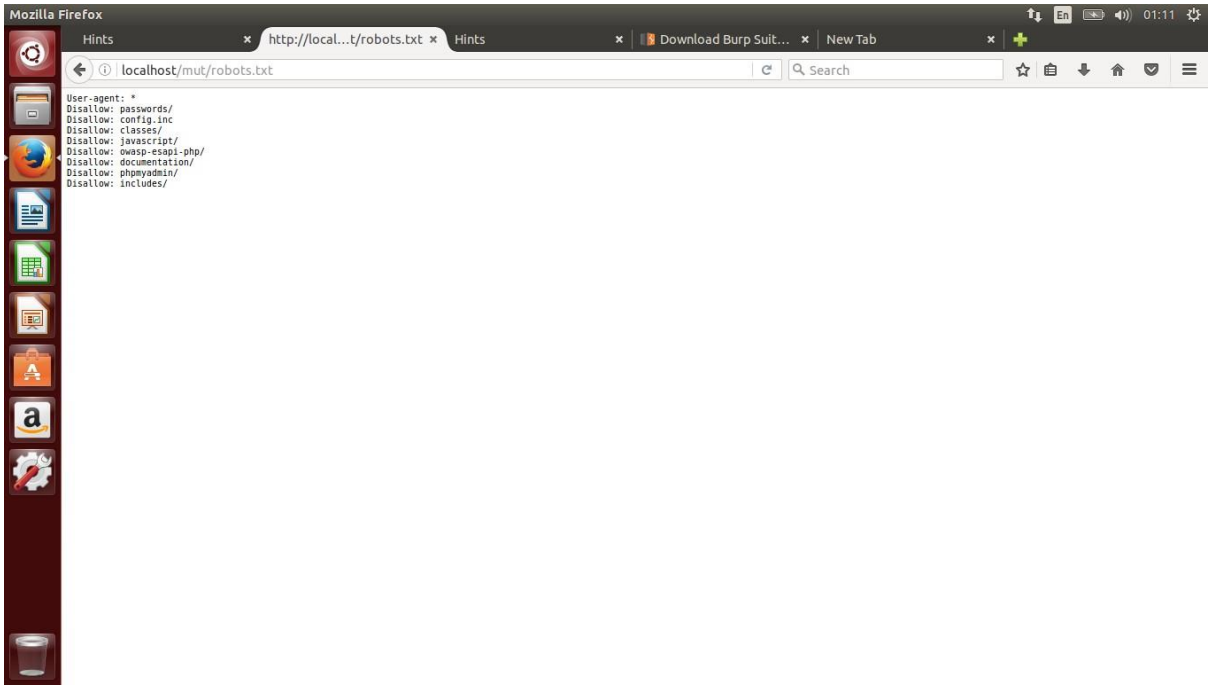

Figure 19

Countermeasures : [13]Sensitive information shouldn't be put on these files and Attackers who attempt to hack this can be discovered through Google Honeypot.

10. Sensitive Data Exposure – Information Disclosure – PHP Info page

In this vulnerability sensitive information can be exploited directly by viewing PHP info page. PHP developers usually use PHPinfo() to het them know the php version and it's features. This makes the hacker easily get access to the web server information. Phpinfo file shows the valuable information for RECON State of hardcore hacking as the features are shown in figure 20. Attackers can easily bypass the possible IDS detection

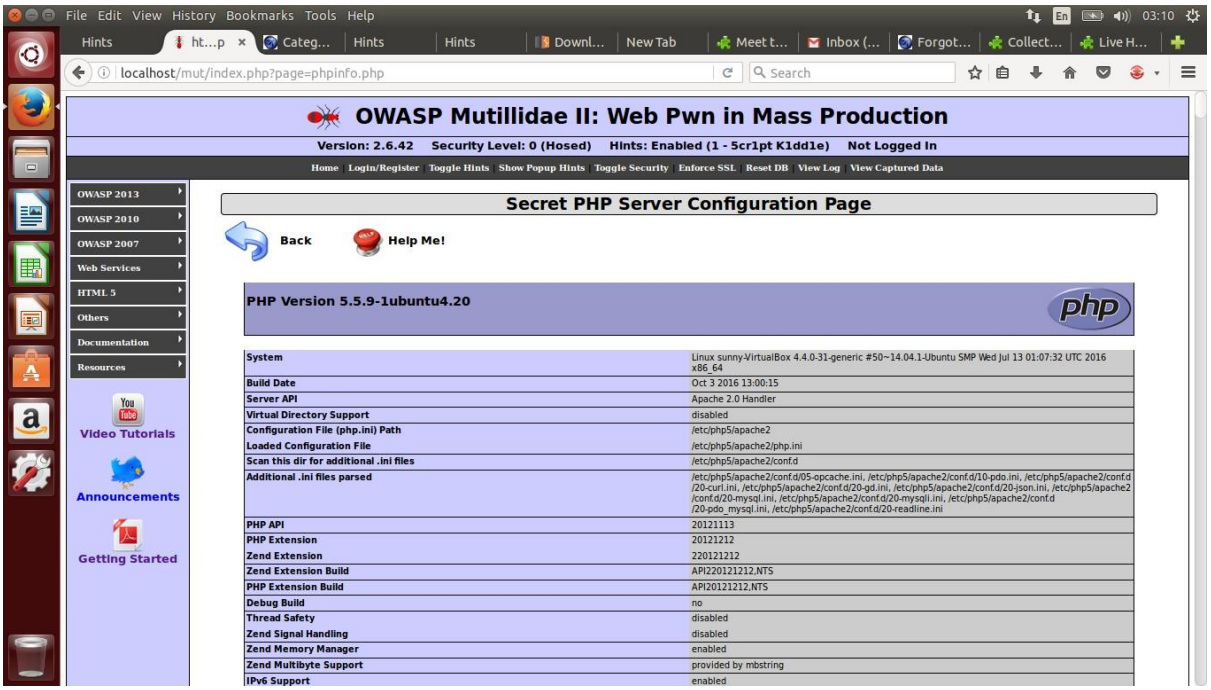

Figure 20

Countermeasures :[14] phpinfo shoudnt be enclosed in the server. For checking purpose we can scan sensitive files in certain time intervals to prevent such threat.

REFERENCES :

[1] “NVD - Detail.” [Online]. Available: https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2016-2873#VulnChangeHistoryDiv. [Accessed: 09-Dec-2016].

[2] “IBM QRadar Security Intelligence Platform,” 01-Jan-2016. [Online]. Available: http://www.ibm.com/software/products/en/qradar, http://www.ibm.com/software/products/en/qradar. [Accessed: 09-Dec-2016].

[3] “SQL Injection - OWASP.” [Online]. Available: https://www.owasp.org/index.php/SQL_Ihttps://www.owasp.org/index.php/Top_10_2010-A6-Security_Misconfigurationnjection. [Accessed: 09-Dec-2016].

[4] “SQL injection,” *Wikipedia*. 09-Dec-2016.

[5] “SQL Injection Prevention Cheat Sheet - OWASP.” [Online]. Available: https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet. [Accessed: 09- Dec-2016].

[6] https://www.owasp.org/index.php/Main_Page

[7] https://www.owasp.org/index.php/Top_10_2010-A1-Injection

[8] https://www.owasp.org/index.php/Top_10_2010-A3

[9] https://www.owasp.org/index.php/Top_10_2010-A2

[10] https://www.owasp.org/index.php/Top_10_2010-A9

[11] https://www.owasp.org/index.php/Top_10_2010-A6-Security_Misconfiguration

[12] https://www.ibm.com/developerworks/library/se-owasptop10/

[13] http://yehg.net/lab/pr0js/papers/Disclosure%20Vulnerability%20Robots.txt.pdf?1481499876

[14] http://yehg.net/lab/pr0js/papers/Disclosure%20Vulnerability%20PHPINFO.pdf