

## LAB 1 PART 1: INFORMATION GATHERING AND EXTRACTING UNENCRYPTED DATA

### TASK 1: SNIFF LOGIN DETAILS FROM UNENCRYPTED HTTP

- Data transmitted through HTTP- vulnerable to interception, enabling attackers to eavesdrop on unencrypted packets + extract sensitive info
- Explore the extraction of data transmitted over unencrypted HTTP connections

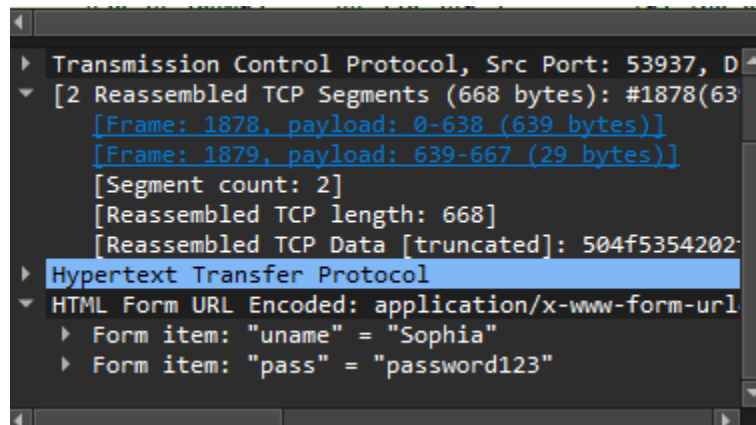
#### STEP 1

- a. Open Wireshark, ensure promiscuous mode is enabled on all interfaces
- b. Start capturing files
- c. Go to <http://testphp.vulnweb.com/login.php>
- d. Enter name on the user, enter "password123"

#### STEP 2

- a. Utilise search bar to locate the packet with the login info (packet will be using HTTP protocol and will start with POST)
- b. In Wireshark, locate login details which are the username and the password used (write "frame contains "POST"" in search bar)

### SCREENSHOT OF USERNAME + PASSWORD FROM WIRESHARK



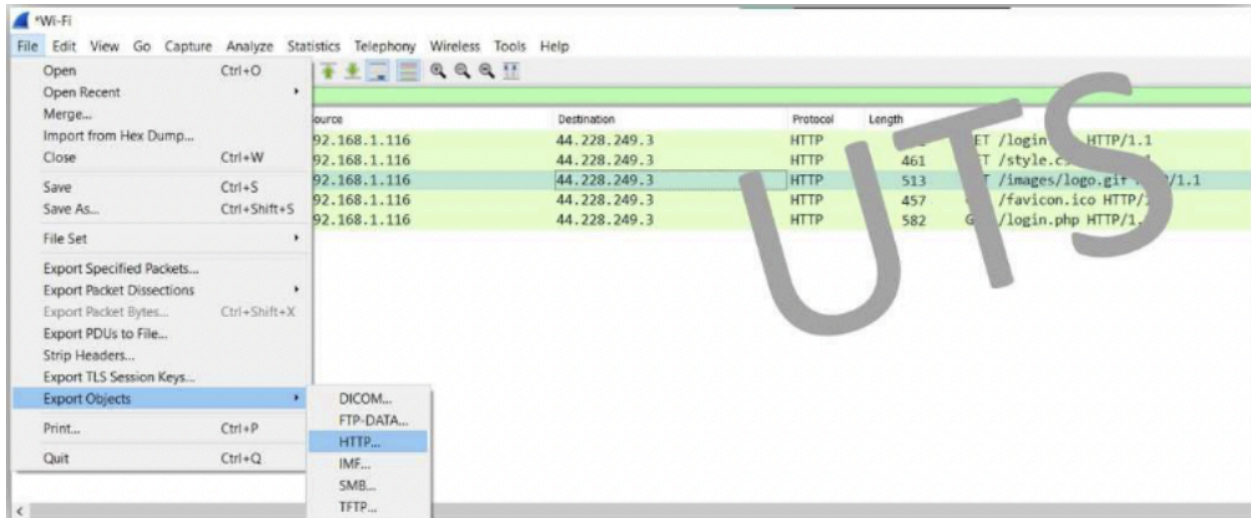
- Filtered under HTTP, looked under HTML form URL encoded section

### TASK 2: EXTRACT AN IMAGE FROM UNENCRYPTED HTTP PACKET

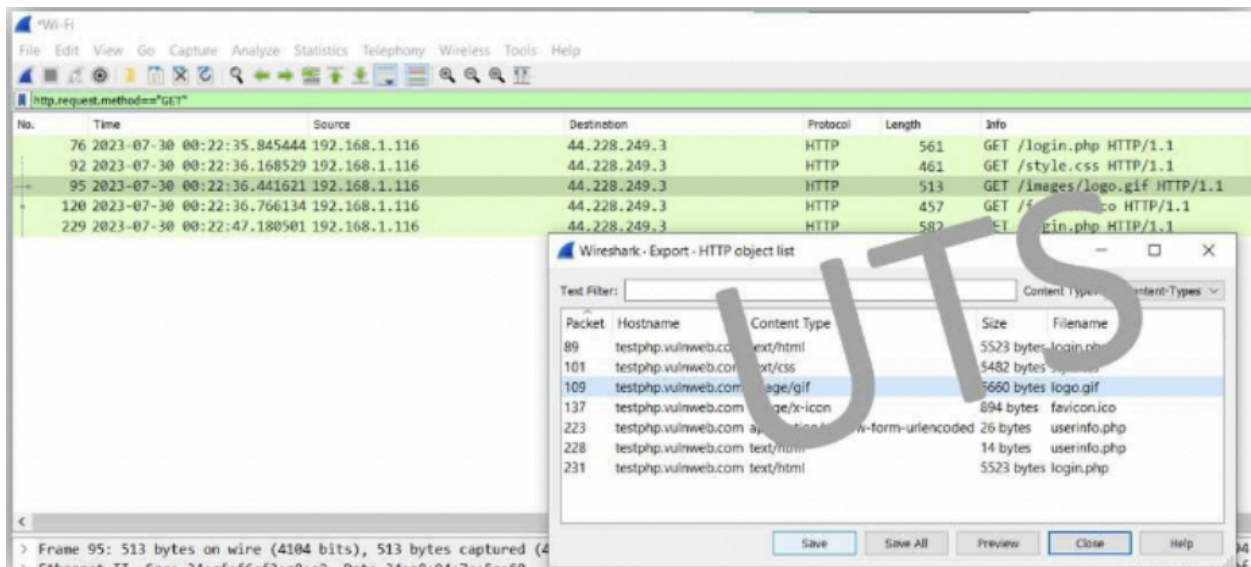
- Wireshark can extract images from unencrypted packets transmitted through websites that use HTTP protocol/unencrypted protocols

#### STEPS:

- Extract logo image of the same website^^
  - Use the same packets captured from task 1
- a. Locate the GET packet that contains the logo.gif raw data
  - b. Click on the packet with the image info and go to File > Export Objects > HTTP



c. Download the object that contains the logo.gif data



**SCREENSHOT OF THE IMAGE LOGO.GIF IN THE HTTP OBJECT LIST**

```

59876 681.11000    192.168.86.29    192.168.86.31    HTTP      98 HTTP/1.1 404 Not Found
263 14.468824     192.168.86.31    192.168.86.31    HTTP      81 POST /userinfo.php HTTP/1.1 (application/x-www-form-urlencoded)
453 23.937266     192.168.86.31    192.168.86.31    HTTP      81 POST /userinfo.php HTTP/1.1 (application/x-www-form-urlencoded)
179 103.696100    192.168.86.31    44.228.249.3     HTTP      83 POST /userinfo.php HTTP/1.1 (application/x-www-form-urlencoded)
192 800.123035     192.168.86.31    192.168.86.29    HTTP      239 GET /ssdp/device-desc.xml HTTP/1.1
6421 800.839773   192.168.86.29    192.168.86.31    HTTP/X..   1242 HTTP/1.1 200 OK
6430 800.971575    192.168.86.31    192.168.86.29    HTTP      248 GET /apps/com.spotify.Spotify.TVv2 HTTP/1.1
6432 800.977882    192.168.86.29    192.168.86.31    HTTP      98 HTTP/1.1 404 Not Found
2619 1048.865625   192.168.86.31    192.168.86.29    HTTP      239 GET /ssdp/device-desc.xml HTTP/1.1
2620 1048.865625   192.168.86.29    192.168.86.31    HTTP/X..   1242 HTTP/1.1 200 OK
2826 1041.005250    192.168.86.31    192.168.86.29    HTTP      248 GET /apps/com.spotify.Spotify.TVv2 HTTP/1.1
2826 1041.011225   192.168.86.29    192.168.86.31    HTTP      98 HTTP/1.1 404 Not Found
2915 1160.868945   192.168.86.31    192.168.86.29    HTTP      239 GET /ssdp/device-desc.xml HTTP/1.1
2915 1160.868945   192.168.86.29    192.168.86.31    HTTP/X..   1242 HTTP/1.1 200 OK
Internet Protocol Version 4, Src: 192.168.86.31, Dest: 44.228.249.3
Transmission Control Protocol, Src Port: 53937, Dst Port: 80, Seq: 640, Ack: 1, Len: 29
[ 2 Reassembled TCP Segments (668 bytes): #1878(639), #1879(29) ]
[Frame: 1878, payload: 0-638 (639 bytes)]
[Frame: 1879, payload: 639-667 (49 bytes)]
[Segment count: 2]
[Reassembled TCP length: 668]
[Reassembled TCP Data: [truncated]: 504f354202f75736572696e666f2e70687020485454502f312e310d0a486f73743a20746573747068702e7656c6
Hypertext Transfer Protocol
POST /userinfo.php HTTP/1.1\r\n
Host: testphp.vulnweb.com\r\n
Connection: keep-alive\r\n
Content-Length: 29\r\n
Cache-Control: max-age=0\r\n
Upgrade-Insecure-Requests: 1\r\n
Origin: http://testphp.vulnweb.com\r\n
Content-Type: application/x-www-form-urlencoded\r\n
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36 OPR/\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exch\r\n
Referer: http://testphp.vulnweb.com/login.php\r\n
Accept-Encoding: gzip, deflate\r\n
Accept-Language: en-us,en;q=0.9\r\n
\r\n
[Full request URI: http://testphp.vulnweb.com/userinfo.php]
[HTTP request 1/2]
[Response in frame: 1884]
[Next request in frame: 1885]
File Data: 29 bytes

```

- Tried clicking on the logo on the web page, but got redirected to another site

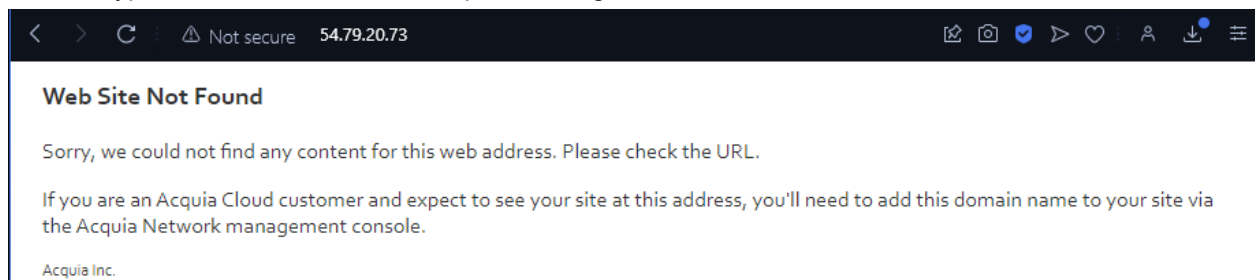
### TASK 3: INFORMATION GATHERING

- Passive information gathering: can be used in a direct attack/reinforce other attacks- risks the security of an organisation
- Active information gathering: collects current data (location, ISP, network constraints)- investigates current state of target
- Zenmap: network discovery, security auditing- useful for network inventory, managing service upgrade schedules and monitoring host or service uptime
- Netcraft: internet services company that provides internet security services (antifraud, anti-phishing services, application testing, code reviews, automated penetration testing)

### QUESTIONS (using zenmap/netcraft to scan www.uts.edu.au)

<https://sitereport.netcraft.com/?url=http://uts.edu.au>

1. IP address: 54.79.20.73
2. Type IP address in browser, explain findings:



- Website can't be found: since the web server doesn't know the domain name, and it can't be specified by only putting the IP address
3. IP owner: Amazon Corporate Services Pty Ltd
  4. Server's operating system: Linux
  5. Type of web server being used: Apache

6. Server-side scripting technology: PHP
7. Email for the domain admin of the website (phishing attack): unavailable due to security reasons?
8. Reverse DNS for the website: ec2-54-79-20-73.ap-southeast-2.compute.amazonaws.com (uts.edu.au)
9. The domain registrar: audns.net.au
10. Nameserver organisation: whois.audns.net.au
11. Company hosting the website: Amazon - Asia Pacific (Sydney) Datacenter
12. Location of the hosting company: Sydney, NSW

## LAB 1 PART 2: PASSWORD CRACKING USING JOHN THE RIPPER AND SQL INJECTION

- John the Ripper: auto detects the encryption on the hashed data, compares it against a large plain-text file containing popular passwords, hashing each password then stopping when it matches

### TASK 4: USING JOHN THE RIPPER

#### STEP 1

- a. Change to documents directory
- b. Check contents of "mypasswd" file

#### STEP 2- RECOVERING PASSWORDS

- a. Type john -show mypasswd
- b. Enter john mypasswd to enter crack mode
- c. Use control c to abort the program
- d. Type in john -show mypasswd again to check the file
- e. Type in john mypasswd -wordlist="password.lst" to retrieve password hashes

### **PASSWORD FOR USER ERIC**

- Student!
- The password cracking process involves decrypting hashed data and comparing it against a text file that contains popular passwords. To create a secure password is to ensure that the attacker would have difficulty deciphering the hashed data.

```
SophiaPineda - cybersec-server@ubuntu: ~/Documents
0g 0:00:03:09 56% 2/3 0g/s 458.7p/s 458.7c/s 458.7C/s 4juhani..4xanth
0g 0:00:03:10 56% 2/3 0g/s 458.7p/s 458.7c/s 458.7C/s Sydney2..Pizza2
0g 0:00:03:11 56% 2/3 0g/s 458.7p/s 458.7c/s 458.7C/s Creative2..Kristen2
0g 0:00:03:12 56% 2/3 0g/s 458.7p/s 458.7c/s 458.7C/s Dodgers2..Highland2
0g 0:00:03:13 57% 2/3 0g/s 458.7p/s 458.7c/s 458.7C/s Whitney2..Tigre2
0g 0:00:03:14 57% 2/3 0g/s 458.7p/s 458.7c/s 458.7C/s Fish2..Ace2
0g 0:00:03:15 57% 2/3 0g/s 458.7p/s 458.7c/s 458.7C/s Dipper2..Liz2
0g 0:00:03:16 57% 2/3 0g/s 458.7p/s 458.7c/s 458.7C/s Morecats2..Master!
Student! (Eric)
1g 0:00:03:17 100% 2/3 0.005073g/s 458.7p/s 458.7c/s 458.7C/s Minnie!..Casper!
Use the "--show" option to display all of the cracked passwords reliably
Session completed
cybersec-server@ubuntu:~/Documents$ john mypasswd
Loaded 5 password hashes with 5 different salts (crypt, generic crypt(3) [?/64])
No password hashes left to crack (see FAQ)
cybersec-server@ubuntu:~/Documents$ john --show mypasswd
cybersec-server:cybersec:1000:1000:1000:1000:/home/cybersec-server:/bin/bash
Alice:password:1001:1001::/home/Alice:
Bob:12345:1002:1002::/home/Bob:
Eve:Pa$$w0rd:1003:1003::/home/Eve:
Eric:Student!:1004:1004::/home/Eric:

5 password hashes cracked, 0 left
cybersec-server@ubuntu:~/Documents$
```

## SQL INJECTION

- Vulnerability in SQL query exploited (stems from interaction between user and application)
  - User inputting additional data
- Web application lacks proper input validation

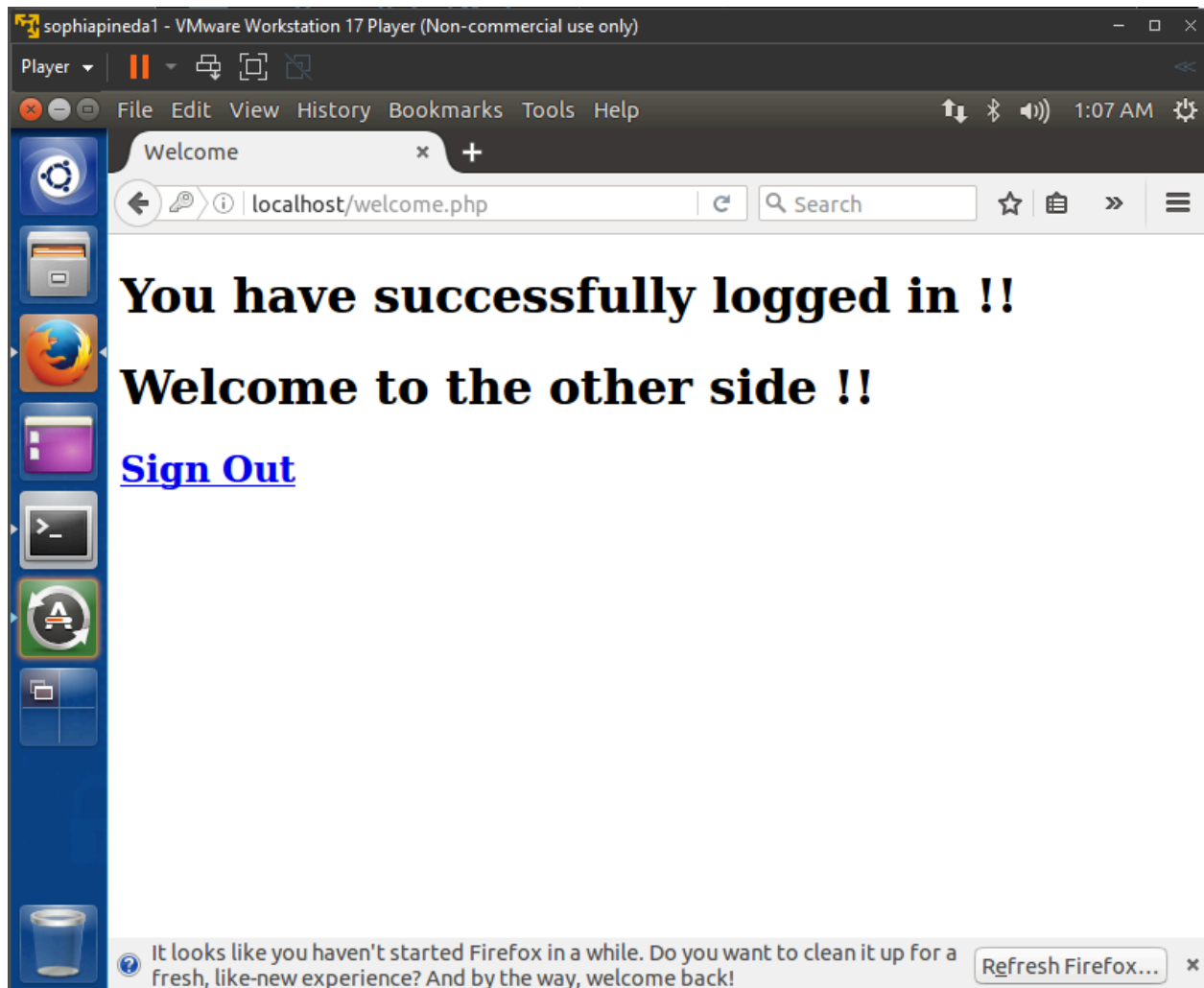
## TASK 5: SQL INJECTION

### STEPS:

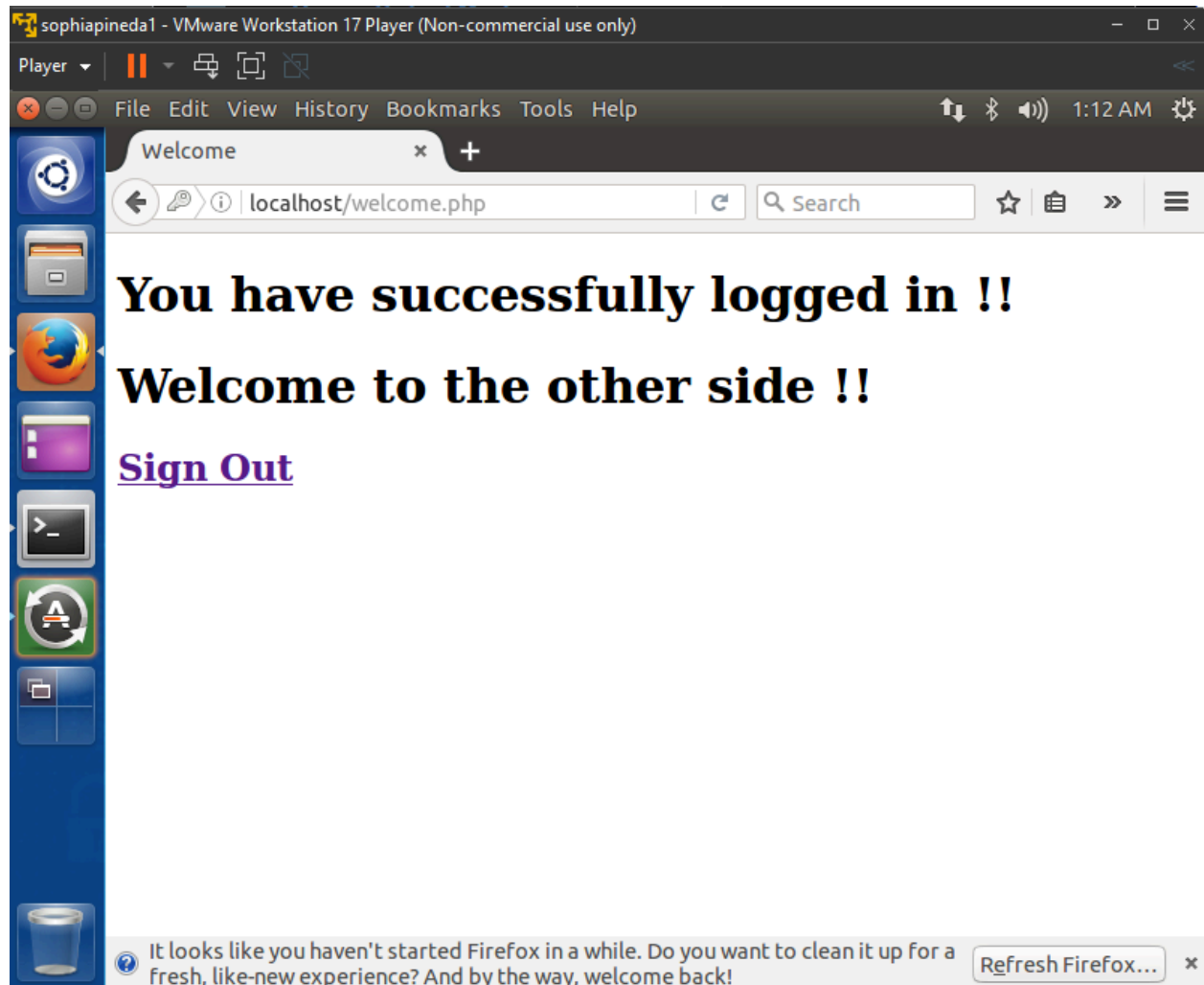
- a. Open localhost
- b. Type in ' in the user
- c. Type in nano /var/www/sqlinjection/index.php

**QUESTIONS** <https://www.invicti.com/blog/web-security/sql-injection-cheat-sheet/>

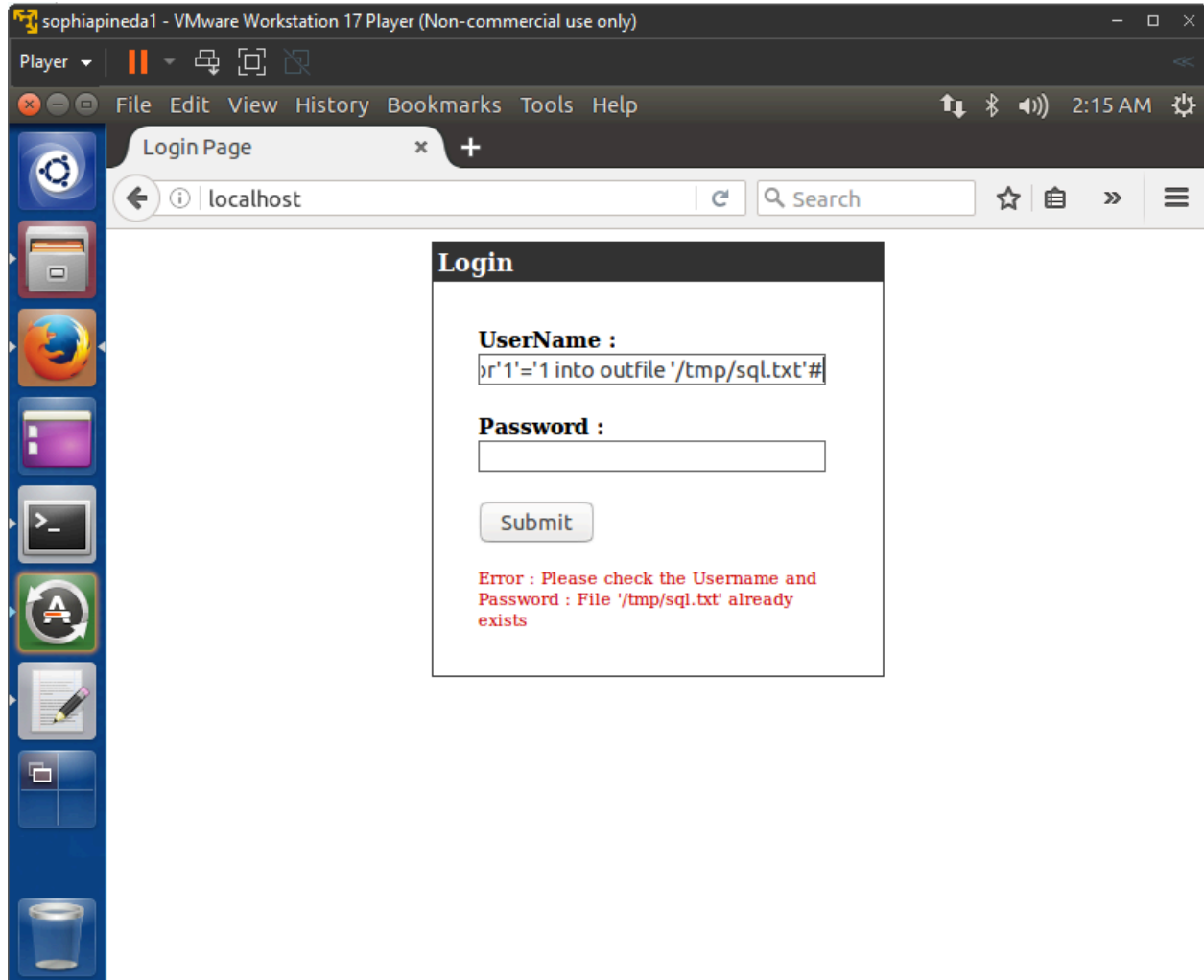
1. Login user "123456789" password as an SQL command to gain unauthorised access



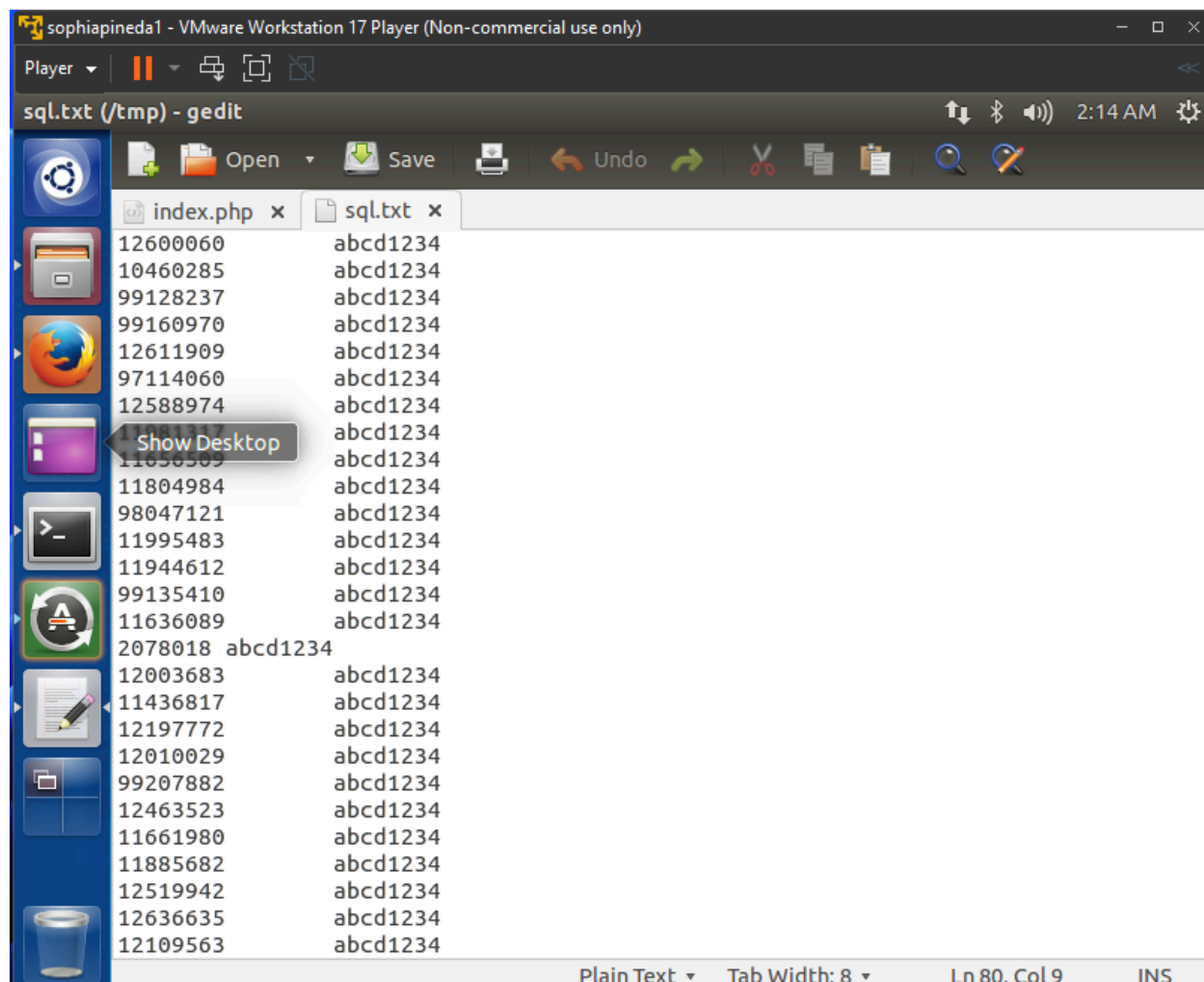
- Typing in 'or' 1'=1 in the password field- means that values are always true, bypassing the validation of the password field, enabling access
2. Login with both username and password as SQL commands



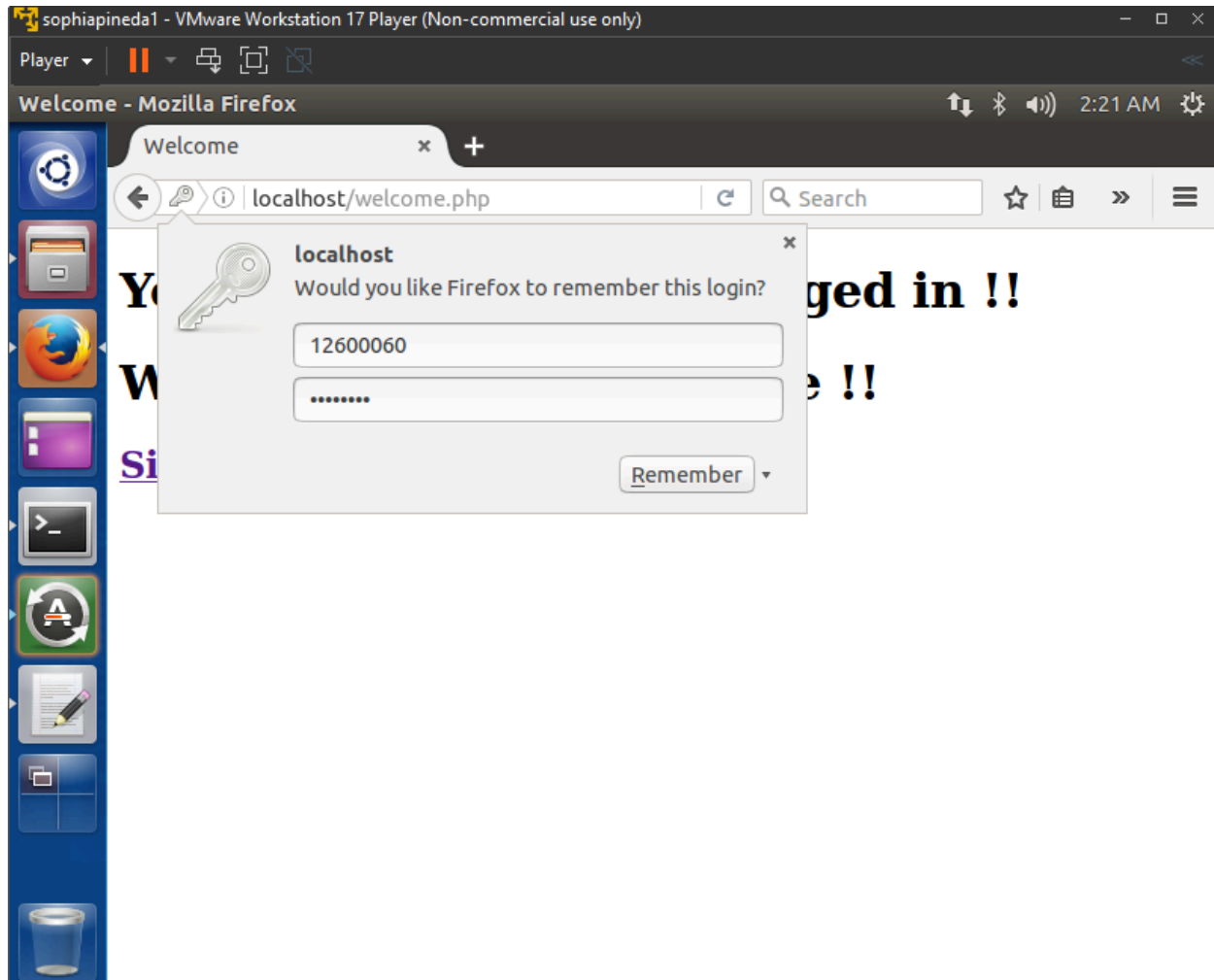
- Typing in 'or' 1'=1 in both fields (password + user)- means the values are always true, bypassing the validation
3. Find table details containing all the usernames and passwords through SQL injection







- Open terminal, type in gedit /var/www/sqlinjection/index.php
  - Typing in 'or' 1'=1 into outfile '/tmp/sql.txt'# in the user, while leaving the password empty
  - Going in files and opening the text to see the usernames and passwords
4. Login into a specific user account by extracting the username and password from the table



- Left hand column: password
- Right hand column: user

#### SUGGESTED DEFENCE (MITIGATION)

- Add input validation: clearly defining the data requirements
- Parameterized queries: separates user input from the query
- Developer of the web page must sanitise all input: removing any unsafe characters

'or'1='1' into outfile '/tmp/sql.txt'#

Cat /tmp/sql.txt