# Detecting shadow credentials

March 21, 2022

*A defenders perspective on msDS-KeyCredentialLink*

**TL;DR;**

This article is about my journey into tracing changes to the msDS-KeyCredentialLink attribute to verify if their origin is legitimate or a potential attack (aka. Shadow Credentials). If you just want to know the “gist” of it, scroll down to the bottom and you’ll find a mindmap.

## What’s msDS-KeyCredentialLink and why should I care?

The msDS-KeyCredentialLink (aka. “kcl”) attribute can be used to link an RSA key pair with a computer or user object in order to authenticate with said key pair against the KDC to receive a Kerberos TGT. So the kcl is in fact a set of alternate credentials that works alongside username/password. An excellent source of information around msDS-KeyCredentialLink is Michael Grafnetters talk at Black Hat 2019 and his famous powershell module DSInternals. I highly recommend you watch his talk since he also explains the relationship between msDS-KeyCredentialLink and Windows Hello for Business (WHfB).

Fast forward a couple of years, manipulating kcl has become a well-known attack technique known as “Shadow Credentials” and I especially recommend this article by Elad Shamir and this documentation by Charlie Bromberg for further details, if you are not familiar with the topic. In a nutshell: if an attacker can write the kcl of an account, he/she will be able to impersonate that account either to gain access or for persistence. Write access to the kcl could be achieved through ACL-misconfigurations, control over another, high-privilged account or through relaying (NTLM) authentication. The NTLM-relay-scenario was in fact the one that made me look into this matter for a couple of reasons:

- It is available in a default setup if we focus on computer accounts. Microsoft closed PetitPotam but there are still other authentication triggers available like good old spool sample and also LDAP Signing is not enforced by default.
- Tooling is publicly available. You can use the above mentioned original POC for coercing authentication or choose one of these triggers. Charlie Bromberg also pushed a customized version of ntlmrelayx which supports shadow credentials.
- And finally: it is not easy to reliably detect and that’s what this whole post is about.

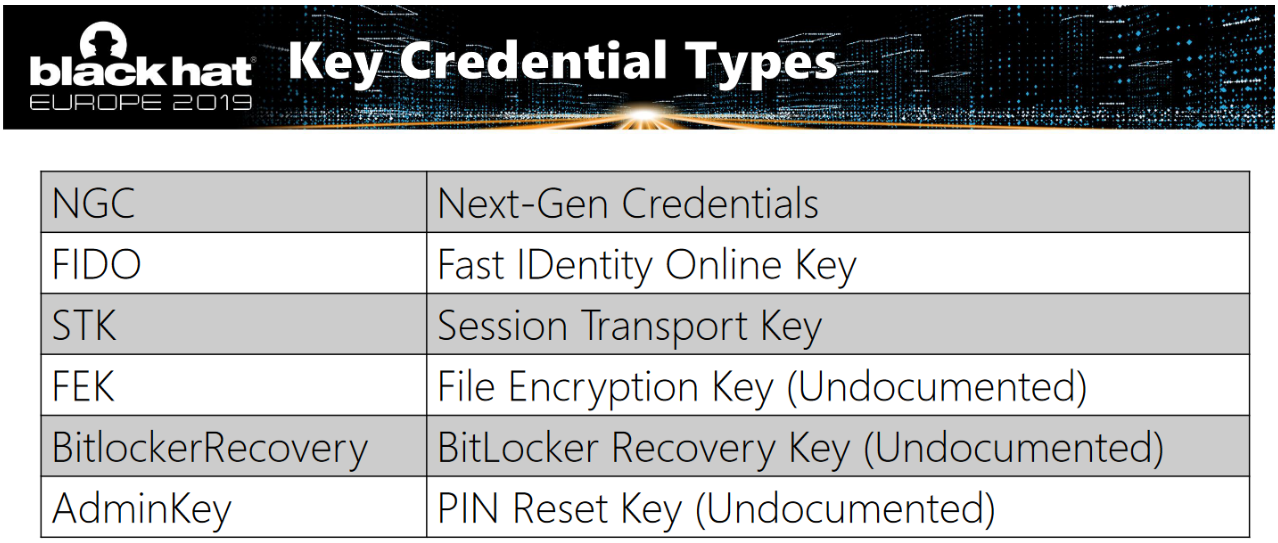
## Auditing msDS-KeyCredentialLink

Manipulation of this attribute can be traced through the common “Audit Directory Service Changes” subcategory, resulting in an event `5136: A directory service object was modified`. Have a look at this post by elastic on how to enable auditing and don’t forget to set an appropriate SACL for user and computer objects.

Here’s how such a malicious event looks like. The `Subject` refers to the actor and the `Object` is the account that was modified.

```
A directory service object was modified.

Subject:
        Security ID:            NETCORP\evil
        Account Name:           evil
        Account Domain: NETCORP
        Logon ID:               0xBCDAC

Directory Service:
        Name:   netcorp.at
        Type:   Active Directory Domain Services

Object:
        DN:     CN=pc1,OU=Client,OU=Computer,OU=company,DC=netcorp,DC=at
        GUID:   CN=pc1,OU=Client,OU=Computer,OU=company,DC=netcorp,DC=at
        Class:  computer

Attribute:
        LDAP Display Name:      msDS-KeyCredentialLink
        Syntax (OID):   2.5.5.7
        Value:  B:828:<Binary>:CN=adlab-
01,OU=Client,OU=Computer,OU=company,DC=netcorp,DC=at

Operation:
        Type:   Value Added
        Correlation ID: {5bff0e70-432c-4ae9-9081-06675dba7869}
        Application Correlation ID:     -
```

Now that we’ve started collecting these events, we have to find ways to verify the legitimazy of a single event. For example, the subject and object are different in the above event, but is this always a useful indicator? It turns out the answer highly depends on your individual environment and varies if you refer to computer or user objects. Therefore, it’s necessary to identify legitimate use-cases which invoke the modification of the kcl attribute. I will describe the two scenarios that seem the most commont to me in the following sections, however this is not a complete list. As Michael Grafnetter showed in his talk, there might be more (undocumented) use-cases.



black hat
EUROPE 2019

Key Credential Types

| | |
|---|---|
| NGC | Next-Gen Credentials |
| FIDO | Fast IDentity Online Key |
| STK | Session Transport Key |
| FEK | File Encryption Key (Undocumented) |
| BitlockerRecovery | BitLocker Recovery Key (Undocumented) |
| AdminKey | PIN Reset Key (Undocumented) |

## Use-case 1 - WHfB Hybrid Azure AD Joined Key Trust

If you search for information about kcl on the internet, you'll end up reading a lot about Windows Hello for Business (WHfB). There is a wealth of really good documentation about WHfB available, which explains the details far more better than I ever could (including the talk by Michael Grafnetter mentioned earlier), so I won't cover it in this post.

The short story is: during WHfB enrollment, the user will generate a key pair and link the public key to the users object in AAD. During the next sync cycle, Azure AD Connect will sync this public key into the msDS-KeyCredentialLink attribute of the user object in on-prem AD, alongside a couple of other information. Note that this refers to a hybrid AAD join environment with WHfB enabled.

A modification event generated in such a scenario looks like this:

```
A directory service object was modified.

Subject:
        Security ID:            netcorp\MSOL_8bee7c7b05af
        Account Name:           MSOL_8bee7c7b05af
        Account Domain: netcorp
        Logon ID:               0xAFEC9F

Directory Service:
        Name:   netcorp.at
        Type:   Active Directory Domain Services

Object:
        DN:
CN=whfbuser,OU=WHFB,OU=DomainUser,OU=User,OU=company,DC=netcorp,DC=at
        GUID:
CN=whfbuser,OU=WHFB,OU=DomainUser,OU=User,OU=company,DC=netcorp,DC=at
        Class:  user

Attribute:
        LDAP Display Name:      msDS-KeyCredentialLink
        Syntax (OID):   2.5.5.7
        Value:  B:854:
<Binary>:CN=whfbuser,OU=WHFB,OU=DomainUser,OU=User,OU=company,DC=netcorp,DC=at

Operation:
        Type:   Value Added
        Correlation ID: {10148d86-8374-4197-84d6-586a201dfa4b}
        Application Correlation ID:     -
```

If you see an event like this in an environment without WHfB or even hybrid AAD join, then this is obviously a red flag. So for the following detection and verification workflow, we’ll asume that we live in such an environment and want to make sure that it is in fact a legitimate event.

Let’s try to go through the information we have. Focusing on the event data above, we see that subject/actor is the AAD Connect sync account. Usually, this should be the only account that modifies this attribute. Also, note that the object class is `user`. If a user enrolls for WHfB, the key is written to the user object and not to the computer object on which the user went through the enrollment. The event also contains a Logon ID, which is a unique identifier to link the modification event 5136 to a logon event 4624. Here’s the corresponding example.

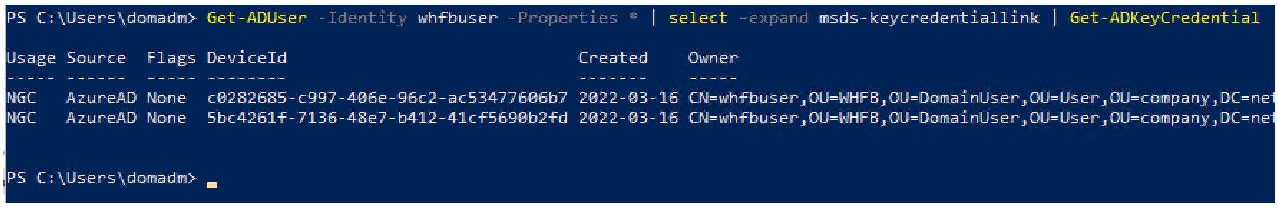
```
An account was successfully logged on.

Subject:
        Security ID:            NULL SID
        Account Name:           -
        Account Domain:         -
        Logon ID:               0x0

Logon Information:
        Logon Type:             3
        Restricted Admin Mode:  -
        Virtual Account:                No
        Elevated Token:         Yes

Impersonation Level:            Impersonation

New Logon:
        Security ID:            netcorp\MSOL_8bee7c7b05af
        Account Name:           MSOL_8bee7c7b05af
        Account Domain:         NETCORP.AT
        Logon ID:               0xAFEC9F
        Linked Logon ID:                0x0
        Network Account Name:   -
        Network Account Domain: -
        Logon GUID:             {a095cd77-3ac7-c998-61d3-9995308dc76d}

Process Information:
        Process ID:             0x0
        Process Name:           -

Network Information:
        Workstation Name:       -
        Source Network Address: fe80::2d5f:1c5a:ede5:89c8
        Source Port:            54856

Detailed Authentication Information:
        Logon Process:          Kerberos
        Authentication Package: Kerberos
        Transited Services:     -
        Package Name (NTLM only):       -
        Key Length:             0
```

Keep an eye on the source network address in that event. This allows you to verify if the authentication actually came from the server running AAD Connect, which already gives you three indicators to verify:

- Is the subject the AAD Connect sync service account?
- Is the object class “user”?
- Is the corresponding authentication originating from the correct server?

If you answer all three questions with “yes”, then that’s already a strong argument I’d say but we can go further. Let’s have a look at the kcl attribute in AD using DSInternals.

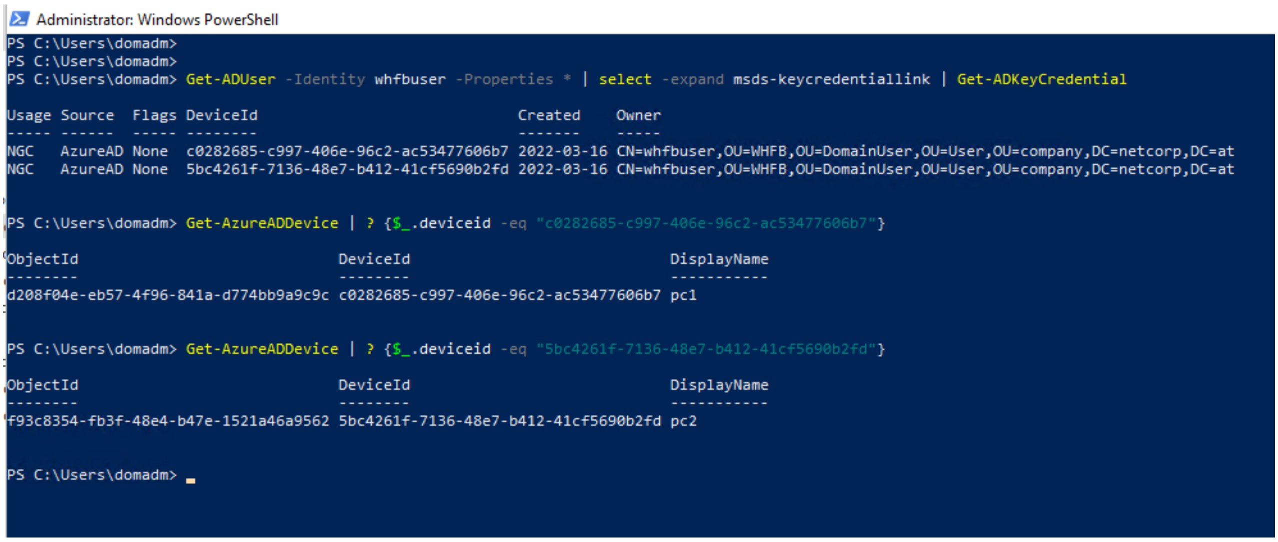
```
Get-ADUser -Identity whfbuser -Properties * | select -expand msds-keycredentiallink | Get-ADKeyCredential
```



```
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>
```

The user “whfbuser” got two credentials registered. Each credential is associated with a device ID, which maps to the device ID attribute of the computer object in AAD. At the time of this writing, public attack tools like Whisker just generate a random GUID since there is no real device to assoicate with. That’s something we can check from a defensive point of view, using AzureAD powershell.

```
Get-AzureADDevice | ? {$_.deviceid -eq "<device-id>"}
```

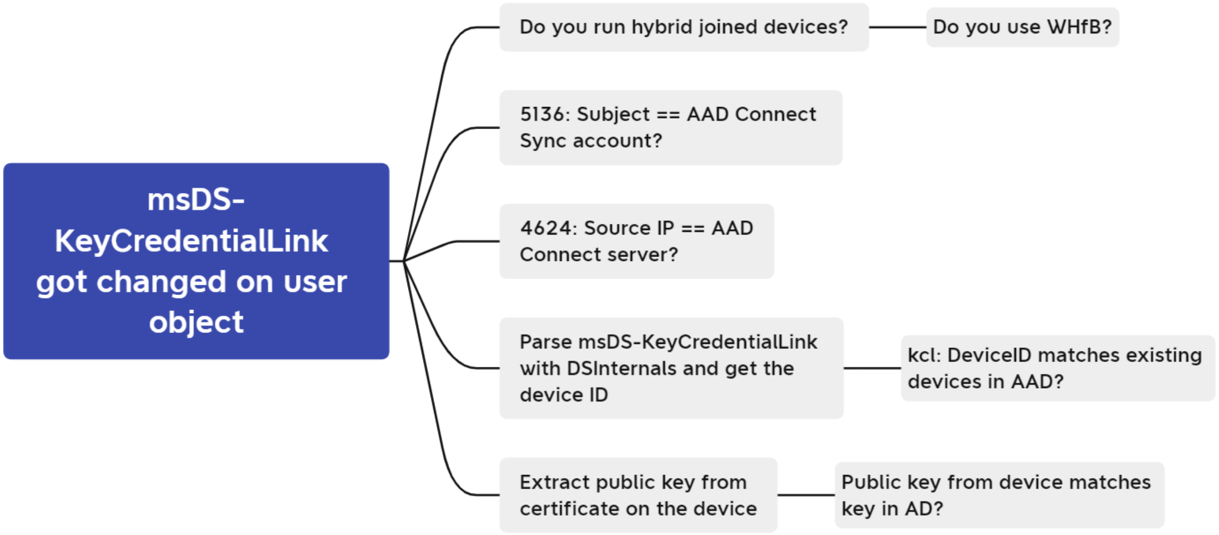


Administrator: Windows PowerShell

```
PS C:\Users\domadm>
PS C:\Users\domadm>
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=netcorp,DC=at
NGC   AzureAD None  5bc4261f-7136-48e7-b412-41cf5690b2fd 2022-03-16 CN=whfbuser,OU=WHFB,OU=DomainUser,OU=User,OU=company,DC=netcorp,DC=at


PS C:\Users\domadm> Get-AzureADDevice | ? {$_.deviceid -eq "c0282685-c997-406e-96c2-ac53477606b7"}

ObjectId                             DeviceId                             DisplayName
--------                             --------                             -----------
d208f04e-eb57-4f96-841a-d774bb9a9c9c c0282685-c997-406e-96c2-ac53477606b7 pc1


PS C:\Users\domadm> Get-AzureADDevice | ? {$_.deviceid -eq "5bc4261f-7136-48e7-b412-41cf5690b2fd"}

ObjectId                             DeviceId                             DisplayName
--------                             --------                             -----------
f93c8354-fb3f-48e4-b47e-1521a46a9562 5bc4261f-7136-48e7-b412-41cf5690b2fd pc2


PS C:\Users\domadm>
```

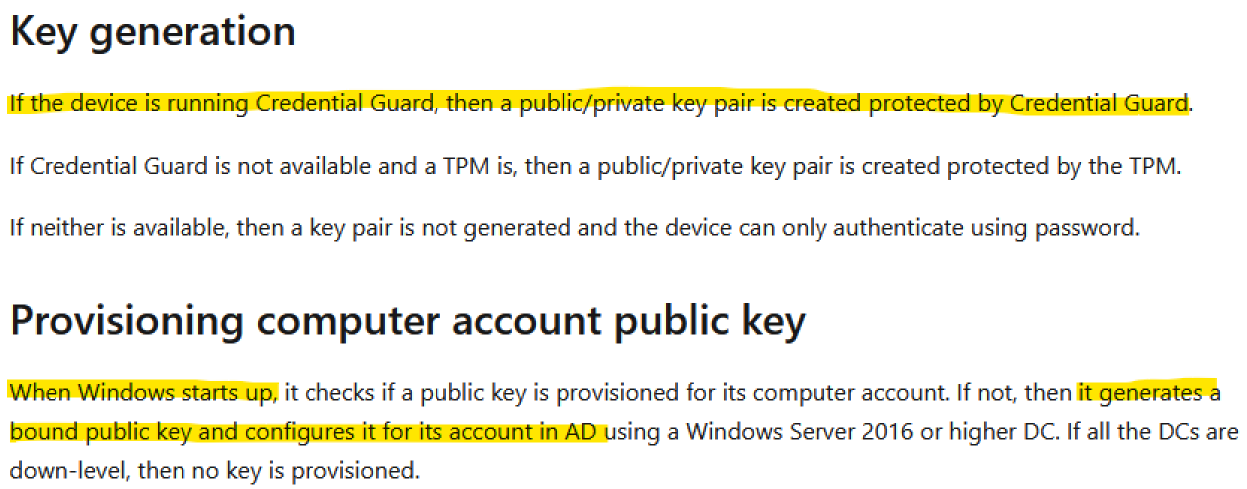
We can see that both credentials map to an existing device and we could now use information from a third-party database like a CMDB to check if these devices truely belong to the user. Again, this is not bulletproof depending on the access the attacker already has but still not that bad. As a final verification step, we could reach out to the client to grab the generated public key and verify if this is really the same public key stored in the kcl attribute. Stephan Waelde describes how to follow a key from the device to Azure AD and back to on-prem AD in his blog post, so I’ll refer you to his guide.

Finally, we can put everything we’ve learned up until now in a nice mindmap.


msDS-KeyCredentialLink got changed on user object
Do you run hybrid joined devices?
Do you use WHfB?
5136: Subject == AAD Connect Sync account?
4624: Source IP == AAD Connect server?
Parse msDS-KeyCredentialLink with DSInternals and get the device ID
kcl: DeviceID matches existing devices in AAD?
Extract public key from certificate on the device
Public key from device matches key in AD?

## Use-case 2 - Credential Guard

Usage of Credential Guard is one of the lesser known (at least as far as I know) triggers for kcl population. It is just briefly mentioned in this article from Microsoft docs.



### Key generation

If the device is running Credential Guard, then a public/private key pair is created protected by Credential Guard.

If Credential Guard is not available and a TPM is, then a public/private key pair is created protected by the TPM.

If neither is available, then a key pair is not generated and the device can only authenticate using password.

### Provisioning computer account public key

When Windows starts up, it checks if a public key is provisioned for its computer account. If not, then it generates a bound public key and configures it for its account in AD using a Windows Server 2016 or higher DC. If all the DCs are down-level, then no key is provisioned.

The corresponding event 5136 for this action looks like this.

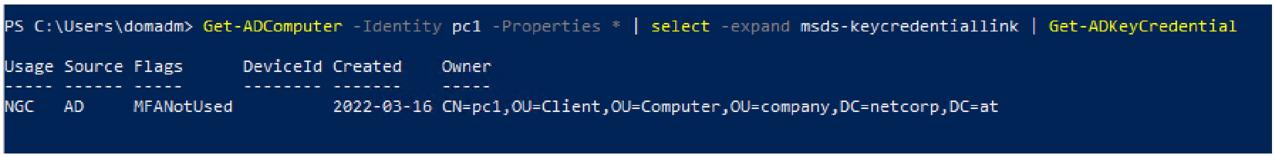
```
A directory service object was modified.

Subject:
        Security ID:            NETCORP\PC1$
        Account Name:           PC1$
        Account Domain:         NETCORP
        Logon ID:               0xA170D

Directory Service:
        Name:   netcorp.at
        Type:   Active Directory Domain Services

Object:
        DN:     CN=pc1,OU=Client,OU=Computer,OU=company,DC=netcorp,DC=at
        GUID:   CN=pc1,OU=Client,OU=Computer,OU=company,DC=netcorp,DC=at
        Class:  computer

Attribute:
        LDAP Display Name:      msDS-KeyCredentialLink
        Syntax (OID):   2.5.5.7
        Value:  B:754:
<Binary>:CN=pc1,OU=Client,OU=Computer,OU=company,DC=netcorp,DC=at

Operation:
        Type:   Value Added
        Correlation ID: {cb1839a0-32c4-4843-bf57-a5b5308dbd73}
        Application Correlation ID:     -
```

You can see that the subject and the object are the same, which makes it harder for us to draw a useful conclusion just from the event data. This becomes a bigger problem if we think about a malicious change that was introduced through NTLM-relay, as mentioned in the beginning. Here’s how this event looks after an NTLM-relay attack using ntlmrelayx.

```
A directory service object was modified.

Subject:
        Security ID:            netcorp\PC1$
        Account Name:           PC1$
        Account Domain:         netcorp
        Logon ID:               0x125BAF

Directory Service:
        Name:   netcorp.at
        Type:   Active Directory Domain Services

Object:
        DN:     CN=pc1,OU=Client,OU=Computer,OU=company,DC=netcorp,DC=at
        GUID:   CN=pc1,OU=Client,OU=Computer,OU=company,DC=netcorp,DC=at
        Class:  computer

Attribute:
        LDAP Display Name:      msDS-KeyCredentialLink
        Syntax (OID):   2.5.5.7
        Value:  B:828:
<Binary>:CN=pc1,OU=Client,OU=Computer,OU=company,DC=netcorp,DC=at

Operation:
        Type:   Value Added
        Correlation ID: {971bae70-6821-4b8c-bdbf-dbb14101b21c}
        Application Correlation ID:     -
```

You don't see much difference? Me neither. You could go for the corresponding event 4624 again and check IP addresses but this is usually a less reliable source of information for clients and would also imply that you have things like DHCP logs. So let's have a look at msDS-KeyCredentialLink itself again. Note, that we're analysing the legitimate credential and not the one added by ntlmrelayx.

```
Get-ADComputer -Identity pc1 -Properties * | select -expand msds-keycredentiallink
| Get-ADKeyCredential
```



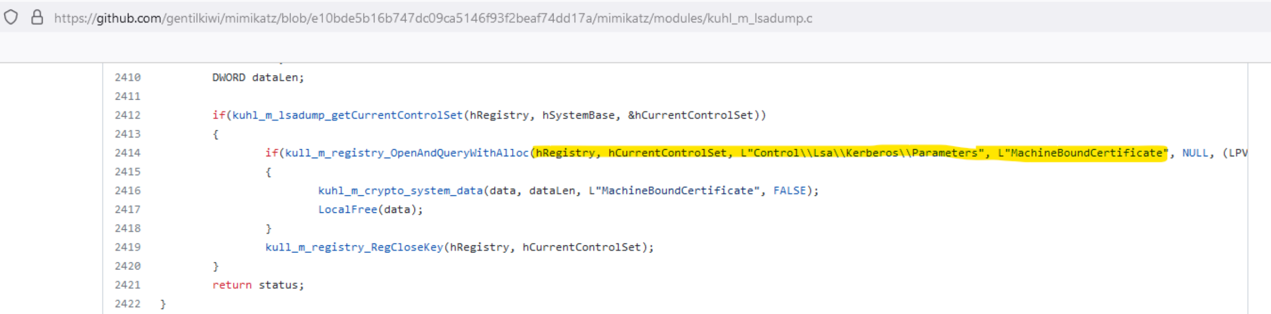
```
PS C:\Users\domadm> Get-ADComputer -Identity pc1 -Properties * | select -expand msds-keycredentiallink | Get-ADKeyCredential

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

If you compare this key-credential with the one from the previous use-case, you can see two of differences.

- The key source is instead of **AzureAD**.
- There is no device ID since the key is already stored in the kcl of a device so that wouldn't make sense.

Remember that tools like whisker and also ntlmrelayx automatically set a device ID by default, so they mimic the WHfB use-case discussed previously but actually raise a red-flag when it comes to the credentialguard use-case. That's good for starters, however you could obviously change the tools without much effort and then we're back at the beginning.

So why don't we compare the key on the device with the key in AD? It turns out, that the location and the structure of this key is not publicly documented by Microsoft. However, there's one man who already got us covered and it shouldn't be a surprise ;)

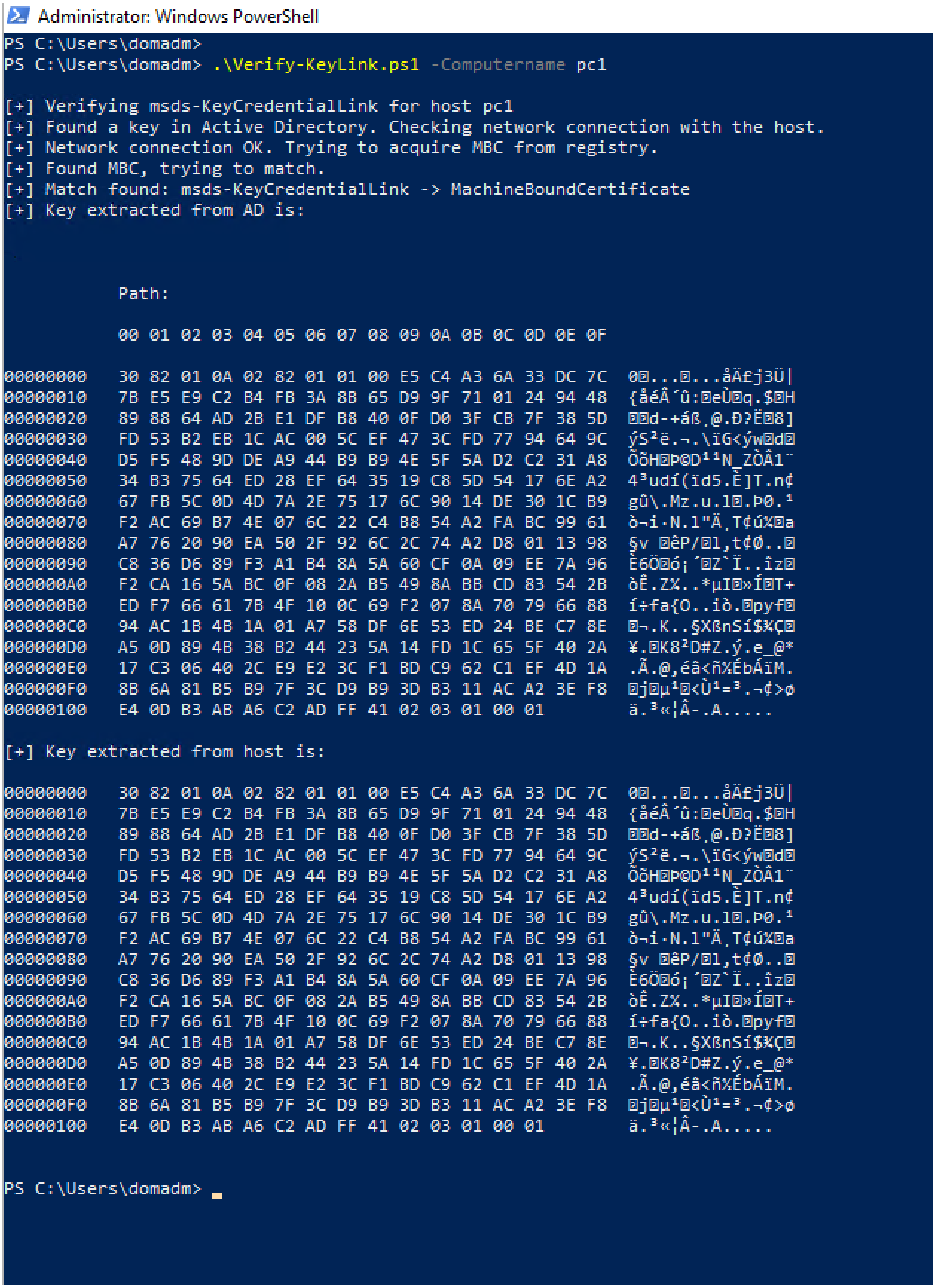


https://github.com/gentilkiwi/mimikatz/blob/e10bde5b16b747dc09ca5146f93f2beaf74dd17a/mimikatz/modules/kuhl_m_lsadump.c

```
	DWORD dataLen;

	if(kuhl_m_lsadump_getCurrentControlSet(hRegistry, hSystemBase, &hCurrentControlSet))
	{
		if(kull_m_registry_OpenAndQueryWithAlloc(hRegistry, hCurrentControlSet, L"Control\\Lsa\\Kerberos\\Parameters", L"MachineBoundCertificate", NULL, (LPV
		{
			kuhl_m_crypto_system_data(data, dataLen, L"MachineBoundCertificate", FALSE);
			LocalFree(data);
		}
		kull_m_registry_RegCloseKey(hRegistry, hCurrentControlSet);
	}
	return status;
}
```

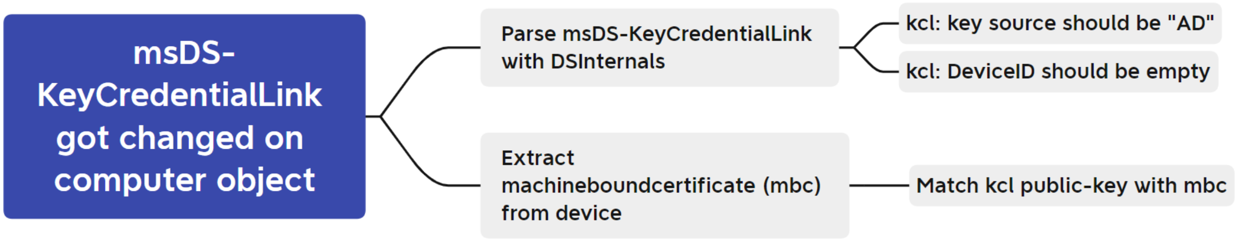
Microsoft calls the structure this key is stored in a "machine-bound certificate" and Benjamin Delpy not only knows where to find it but also how to extract it. Nice. Unfortunately, defenders can't usually just go around and randomly execute Mimikatz for various reasons, so I chose a less elegant way to "reimplement" the functionality. This script uses PSRemoting to grab the content of the `machineboundcertificate` registry value of the host and runs a byte-wise comparison with the content of the public key from the kcl attribute in AD (very quick and dirty - I know ^^).

Here's how the output of this script looks like.



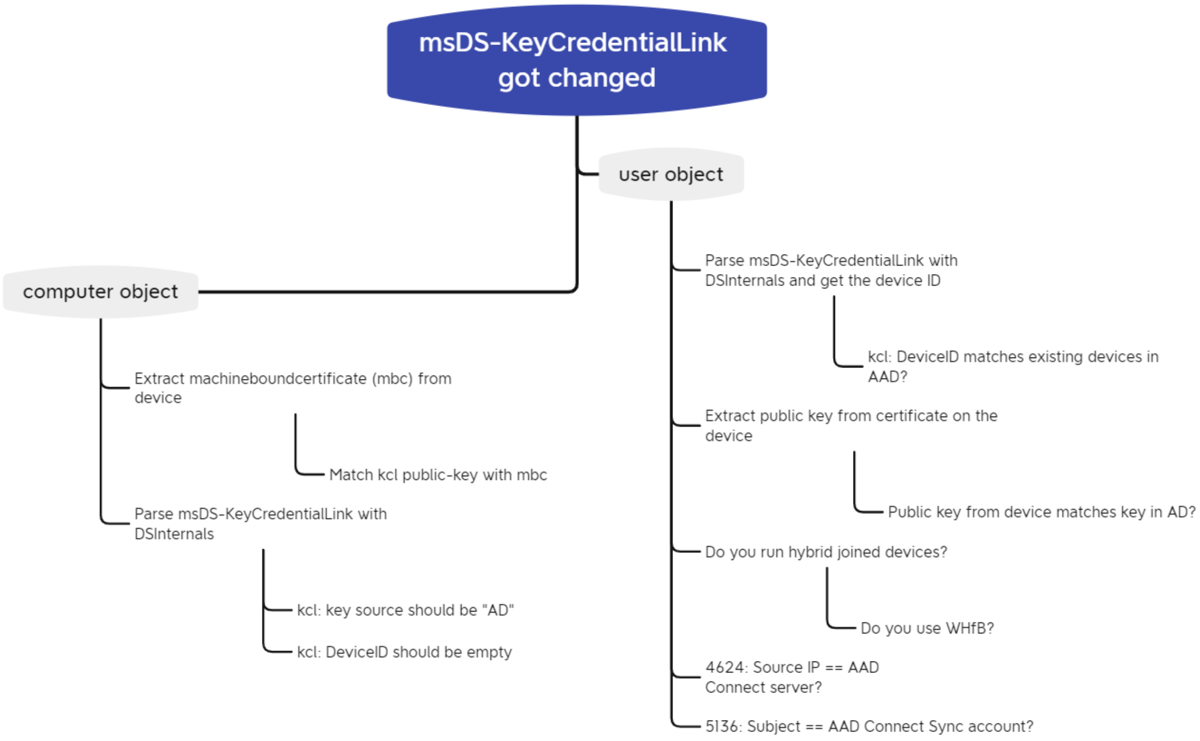
```
PS C:\Users\domadm>
PS C:\Users\domadm> .\Verify-KeyLink.ps1 -Computername pc1

[+] Verifying msds-KeyCredentialLink for host pc1
[+] Found a key in Active Directory. Checking network connection with the host.
[+] Network connection OK. Trying to acquire MBC from registry.
[+] Found MBC, trying to match.
[+] Match found: msds-KeyCredentialLink -> MachineBoundCertificate
[+] Key extracted from AD is:



           Path:

           00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000   30 82 01 0A 02 82 01 01 00 E5 C4 A3 6A 33 DC 7C   0⍰...⍰...åÄ£j3Ü|
00000010   7B E5 E9 C2 B4 FB 3A 8B 65 D9 9F 71 01 24 94 48   {åéÂ´û:⍰eÙ⍰q.$⍰H
00000020   89 88 64 AD 2B E1 DF B8 40 0F D0 3F CB 7F 38 5D   ⍰⍰d-+áß¸@.Ð?Ë⍰8]
00000030   FD 53 B2 EB 1C AC 00 5C EF 47 3C FD 77 94 64 9C   ýS²ë.¬.\ïG<ýw⍰d⍰
00000040   D5 F5 48 9D DE A9 44 B9 B9 4E 5F 5A D2 C2 31 A8   ÕõH⍰Þ©D¹¹N_ZÒÂ1¨
00000050   34 B3 75 64 ED 28 EF 64 35 19 C8 5D 54 17 6E A2   4³udí(ïd5.È]T.n¢
00000060   67 FB 5C 0D 4D 7A 2E 75 17 6C 90 14 DE 30 1C B9   gû\.Mz.u.l⍰.Þ0.¹
00000070   F2 AC 69 B7 4E 07 6C 22 C4 B8 54 A2 FA BC 99 61   ò¬i·N.l"Ä¸T¢ú¼⍰a
00000080   A7 76 20 90 EA 50 2F 92 6C 2C 74 A2 D8 01 13 98   §v ⍰êP/⍰l,t¢Ø..⍰
00000090   C8 36 D6 89 F3 A1 B4 8A 5A 60 CF 0A 09 EE 7A 96   È6Ö⍰ó¡´⍰Z`Ï..îz⍰
000000A0   F2 CA 16 5A BC 0F 08 2A B5 49 8A BB CD 83 54 2B   òÊ.Z¼..*µI⍰»Í⍰T+
000000B0   ED F7 66 61 7B 4F 10 0C 69 F2 07 8A 70 79 66 88   í÷fa{O..iò.⍰pyf⍰
000000C0   94 AC 1B 4B 1A 01 A7 58 DF 6E 53 ED 24 BE C7 8E   ⍰¬.K..§XßnSí$¾Ç⍰
000000D0   A5 0D 89 4B 38 B2 44 23 5A 14 FD 1C 65 5F 40 2A   ¥.⍰K8²D#Z.ý.e_@*
000000E0   17 C3 06 40 2C E9 E2 3C F1 BD C9 62 C1 EF 4D 1A   .Ã.@,éâ<ñ½ÉbÁïM.
000000F0   8B 6A 81 B5 B9 7F 3C D9 B9 3D B3 11 AC A2 3E F8   ⍰j⍰µ¹⍰<Ù¹=³.¬¢>ø
00000100   E4 0D B3 AB A6 C2 AD FF 41 02 03 01 00 01         ä.³«¦Â-.A.....

[+] Key extracted from host is:

00000000   30 82 01 0A 02 82 01 01 00 E5 C4 A3 6A 33 DC 7C   0⍰...⍰...åÄ£j3Ü|
00000010   7B E5 E9 C2 B4 FB 3A 8B 65 D9 9F 71 01 24 94 48   {åéÂ´û:⍰eÙ⍰q.$⍰H
00000020   89 88 64 AD 2B E1 DF B8 40 0F D0 3F CB 7F 38 5D   ⍰⍰d-+áß¸@.Ð?Ë⍰8]
00000030   FD 53 B2 EB 1C AC 00 5C EF 47 3C FD 77 94 64 9C   ýS²ë.¬.\ïG<ýw⍰d⍰
00000040   D5 F5 48 9D DE A9 44 B9 B9 4E 5F 5A D2 C2 31 A8   ÕõH⍰Þ©D¹¹N_ZÒÂ1¨
00000050   34 B3 75 64 ED 28 EF 64 35 19 C8 5D 54 17 6E A2   4³udí(ïd5.È]T.n¢
00000060   67 FB 5C 0D 4D 7A 2E 75 17 6C 90 14 DE 30 1C B9   gû\.Mz.u.l⍰.Þ0.¹
00000070   F2 AC 69 B7 4E 07 6C 22 C4 B8 54 A2 FA BC 99 61   ò¬i·N.l"Ä¸T¢ú¼⍰a
00000080   A7 76 20 90 EA 50 2F 92 6C 2C 74 A2 D8 01 13 98   §v ⍰êP/⍰l,t¢Ø..⍰
00000090   C8 36 D6 89 F3 A1 B4 8A 5A 60 CF 0A 09 EE 7A 96   È6Ö⍰ó¡´⍰Z`Ï..îz⍰
000000A0   F2 CA 16 5A BC 0F 08 2A B5 49 8A BB CD 83 54 2B   òÊ.Z¼..*µI⍰»Í⍰T+
000000B0   ED F7 66 61 7B 4F 10 0C 69 F2 07 8A 70 79 66 88   í÷fa{O..iò.⍰pyf⍰
000000C0   94 AC 1B 4B 1A 01 A7 58 DF 6E 53 ED 24 BE C7 8E   ⍰¬.K..§XßnSí$¾Ç⍰
000000D0   A5 0D 89 4B 38 B2 44 23 5A 14 FD 1C 65 5F 40 2A   ¥.⍰K8²D#Z.ý.e_@*
000000E0   17 C3 06 40 2C E9 E2 3C F1 BD C9 62 C1 EF 4D 1A   .Ã.@,éâ<ñ½ÉbÁïM.
000000F0   8B 6A 81 B5 B9 7F 3C D9 B9 3D B3 11 AC A2 3E F8   ⍰j⍰µ¹⍰<Ù¹=³.¬¢>ø
00000100   E4 0D B3 AB A6 C2 AD FF 41 02 03 01 00 01         ä.³«¦Â-.A.....


PS C:\Users\domadm>
```

A very opsec-concerned attacker could still work around this by also setting the content of the registry value correctly, but I think it’s still an indicator worth looking at. Finally, it’s mindmap time again.


msDS-KeyCredentialLink got changed on computer object
Parse msDS-KeyCredentialLink with DSInternals
kcl: key source should be "AD"
kcl: DeviceID should be empty
Extract machineboundcertificate (mbc) from device
Match kcl public-key with mbc

## Conclusion

Auditing msDS-KeyCredentialLink changes is easy, verifying a change is not and heavily depends on a solid knowledge of what's "normal" in your environment. I hope this post allows you to save some time during the process. It took me quite a bit to get this sorted out so far, but I still have the strong feeling that I'm missing something (or a lot). So if you read this and know more than I do, I'd happy if you could let me know. That said, here's another mindmap because #MindMapsAreFun :)


msDS-KeyCredentialLink got changed
user object
Parse msDS-KeyCredentialLink with DSInternals and get the device ID
kcl: DeviceID matches existing devices in AAD?
Extract public key from certificate on the device
Public key from device matches key in AD?
Do you run hybrid joined devices?
Do you use WHfB?
4624: Source IP == AAD Connect server?
5136: Subject == AAD Connect Sync account?
computer object
Extract machineboundcertificate (mbc) from device
Match kcl public-key with mbc
Parse msDS-KeyCredentialLink with DSInternals
kcl: key source should be "AD"
kcl: DeviceID should be empty