# Hello
## My name is :
# MICROSOFT
## and I have a credential problem

By Skip Duckwall and Chris Campbell

for Blackhat USA 2013

# Introduction

User credentials are one of the most powerful items an attacker can obtain. Single-factor, or password-based credentials are dangerous to lose because of their extreme portability. From almost anywhere inside a prototypical business network, a username and password can and will get the attacker into almost any application from email to databases. Thus, protecting user credentials from the prying eyes of both attackers and malicious insiders should be a priority on the network. Unfortunately for the network defenders, Microsoft does not make this an easy task. Invoking imagery from the 1980 movie "Airplane", Microsoft has a credential problem.

Microsoft's problem with credentials starts with the implementation of the Single Sign on Solution (SSO) built into many Microsoft products, including Windows. Meant to improve user experience, Windows goes to extensive lengths behind the scenes to prevent the user from having to type in their username and password more than once. If this feature didn't exist, every time a user would try to access a network resource, such as email for a fileshare, they would be prompted for their password. In the average corporate network environment, this could end up being 10 or more times upon initial logon. During a recent talk entitled "Pass the Hash and Other Credential Theft and Reuse: Preventing Lateral Movement and Privilege Escalation" at the Microsoft Bluehat Security Conference, Mark Russinovich, Technical Fellow at Microsoft, called this phenomenon "Credential Fatigue[1]". The fact there is an internal term for overly bothering the user to type in their password illustrates how pervasive SSO is in the MS current and future development.

Behind the scenes, Windows authentication can interact with multiple authentication schemes. In order to provide the correct authenticator to the authentication service, Windows caches the user's credentials in memory in whatever format is required. In the case of NTLM, all that is saved is the user's password hash. For other authentication providers, including Kerberos, however, the plaintext username and password are saved. This means with the proper tools and SYSTEM level access, an attacker can recover the username and password for all users that are logged in. On servers with a large number of user logins, such as a file server or Exchange server, this could be devastating. This leaves attackers asking "Why 'Pass the Hash' when you can get the actual username and password?"

Even if an organization has implemented two-factor authentication, its use is only for "interactive sessions". This means that the second authentication factor (smart card, passcode token, soft cert, etc) is only used when physically logging into a server at the console or via Remote Desktop Protocol /terminal services. It is also possible for websites to require the security certificate of a user before allowing access. Unfortunately, most interactions that a user has with the work environment aren't considered "interactive". Access to file shares, email and database logins, to name a few, cannot use

[1] http://channel9.msdn.com/Events/Blue-Hat-Security-Briefings/BlueHat-Security-Briefings-Fall-2012-Sessions/BH1208

the second factor for authentication. What does this mean? Windows still relies on the passwords and/or password hashes behind the scenes to authenticate to network resources that are incapable of utilizing two-factor authentication.

Outside of the SSO world it isn't exactly rainbows and unicorns either. User level credentials are potentially lying all over the place. Examples are domain cached credentials, plaintext passwords for saved accounts in the LSA secrets, local SAM (Security Account Manager) accounts, application-saved passwords in the registry, application saved passwords in application configuration files, password vaults, keylogging, and the list goes on. This presents a veritable virtual smorgasbord of choices for an attacker to go after in order to gain access to other systems. Not all of these possibilities yield immediate gold, but that doesn't mean they aren't useful to an attacker with time and resources to burn.

If for some reason an attacker isn't interested in hunting for credentials, they have other options they can use. For example, Windows provides that ability for an attacker to create a thread in an already existing process to run their own code. This thread will run with all the security privileges of the user running the process. This code can then interact with the network while impersonating the account in question. It's also possible to export Kerberos tickets and reintroduce them onto a different machine, thus granting the ability to literally transplant users from one system to another without any knowledge of the user's credentials.

# Single Sign-On: The Great and Terrible

A typical day at the office for the average person involves showing up, sitting down at their workstation, logging in to the computer, bringing up their email, browsing updates on the corporate internet, and then going on about their daily routine. Behind the scenes, when the user logs in, scripts run that mount all the required file shares dependent on the user's level of access. All of these interactions with network resources only required the user to input their password one time. Through the magic of SSO, Windows makes it so that as long as all the applications are configured correctly, the user rarely has to type their passwords in again. How does this work? What's behind the curtain?

Somewhat simplified, when the user initially logs in interactively (via the computer's console) they interact with the "Winlogon[2]" process. "Winlogon" accepts the user's credentials and passes them onto the "LSASS" (Local Security Authority Subsystem Service) process for validation. "LSASS" uses "authentication packages" to validate that the credentials are valid, for example by validating against Active Directory or the local SAM database. Assuming the credentials are validated, a shell is created for the user and the logon process continues on. However, the valid credentials are cached for future

---

[2] http://msdn.microsoft.com/en-us/library/windows/desktop/aa380543(v=vs.85).aspx

use by the LSASS process.  In fact, LSASS is a true polyglot, speaking several different authentication methods.

Authentication dialects are supported by “Security Support Providers”, or SSPs for short.  There are SSPs for Kerberos, NTLM, Digest-MD5, and others.  Many of the SSPs might not be used at all during the average user’s session.  However, each of the SSPs keeps what it needs in order to properly process requests for that particular dialect.  You know, just in case...

# Parlez-Vous [Kerberos|NTLM|Digest|...]?

As previously mentioned, each SSP handles different types of authentication mechanisms as well as keeps whatever information it needs to act on behalf of the user around just in case it is needed.  We will briefly discuss the various methods of authentication and what (apparent) information is kept hanging around in the memory of the LSASS process.   All an attacker needs is System level privilege to be able to access the memory of the LSASS process.

- Kerberos

  Kerberos is the default authentication mechanism for Active Directory.  Kerberos originated  at MIT and is based on a user being granted a time-limited ticket.  By providing this ticket to network services, the authentication exchanges are streamlined resulting in much lower overhead during the period the ticket is active.

  Certain criteria have to be met before Kerberos can be used.  For example, both ends of the connection (client and server) have to be in the same domain (or trusted domain).  The client and/or server must refer to each other by their DNS names.  If one end uses an IP address, then Kerberos cannot be used.

  More details on security issues with Kerberos will be discussed later in this paper.

  What is kept in memory in LSASS?

  Based on the research of Mr. Benjamin  Delpy[3], AKA gentilkiwi, it appears that LSASS keeps username and the unencrypted password for the Kerberos SSP.  Below is the output of his tool Mimikatz:

```
kerberos
 * Utilisateur   : blackhat2013
 * Domaine       : DEMO.LOCAL
 * Mot de passe  : P@$$W0rd!!
```

Figure 1- Output from Mimikatz for the Kerberos SSP

[3] http://blog.gentilkiwi.com/mimikatz

- NTLM (MSV1_0)

  Prior to the introduction of Kerberos, NTLM was the authentication method used by Microsoft Windows Domains. However, NTLM is still used today, even in Active Directory environments. If Kerberos cannot be used, for example if one endpoint is referred to by IP address, then NTLM is used to authenticate instead. NTLM can also be used to authenticate with virtually any protocol that Microsoft supports. For example, it is not unusual to see NTLM traffic used for accessing SharePoint. Exchange can configure NTLM authentication for most email protocols, to include SMTP, IMAP, and POP3. MSSQL servers can be configured to use NTLM authentication as well.

  There are 2 different hash types that get calculated: the older LanManager (LM) hash and the NT hash. LM hashes suffer from serious cryptographic problems, including a 14 character limit on the length of passwords. Because of this, the passwords of 14 characters or less can be easily recovered from the LM hash via rainbow tables. **Note: In the LSASS process, a LM hash is calculated REGARDLESS for any password of 14 characters or less. Even if the registry settings exist to prevent the LM passwords from being written to disk.** The NT hash doesn't suffer the same cryptographic issues as the LM hash, however, it is not a cryptographically strong hash, and modern hardware can crack these hashes at a very high rate of speed.

  Additional security risks of NTLM usage will be discussed later in this paper.

  What is kept in memory in LSASS?

  Mimikatz shows us that both the LM hash and the NT hash are kept in the LSASS process memory.

```
msv1_0
 * Utilisateur   : blackhat2013
 * Domaine       : DEMO
 * Hash LM       : b0109442b77b46c7a67a448822b50c99
 * Hash NTLM     : 564644a3eef2263fbd5642e1dd898154
```

Figure 2 - Mimikatz Output for the MSV1 (NTLM) SSP

- Digest MD5

  Digest authentication was primarily introduced as a method of authentication to use with web servers to replace the credentials being sent in cleartext via so called "basic authentication". The username, password, a time-based nonce (number used once) and other information are concatenated and passed through the MD5 hashing algorithm.

  What is kept in memory in LSASS?

  Mimikatz was the first tool to introduce the world to the fact that plaintext credentials were being cached in LSASS, and the Digest-MD5 SSP was the first place they were found.

```
ssp
wdigest
 * Utilisateur   : blackhat2013
 * Domaine       : DEMO
 * Mot de passe  : P@$$W0rd!!
```

Figure 3 - Output of Mimikatz for the MD5 Digest SSP

Beyond these three major authentication schemes, there is also support for other, less used SSPs, like LiveSSP, used to authenticate your computer systems to the Microsoft Live-ID authentication. Who doesn't want their user's accounts tied in with their XBOX account at home? There is also support for custom SSPs for those who have some sort of custom authentication scheme.

# Tit for Tat - Your Move [Microsoft | Researcher]

Microsoft is in a constant arms race with security researchers. For example, in the most recent preview release of Windows Blue (8.1) Microsoft made efforts to counter the research that initially brought us Mimikatz and subsequently added to WCE[4]. However, within a few hours, Mimikatz was updated[5] and the plaintext passwords were still found. Microsoft will more than likely fix the problems and the fight will return to the researchers. Lather. Rinse. Repeat.

However, even if Microsoft does find a way to prevent Mimikatz or tools like it from working in current versions of Windows, there is no guarantee any such features will be backported into previous versions as this will be terribly expensive for Microsoft. The time to backport the fix(es) into the older codebases, build patches, update documentation, test, etc could simply be infeasible for Microsoft to undertake. This leaves most organizations vulnerable to these attacks. The typical upgrade cycle for large organizations usually takes several years and often corresponds with hardware upgrades. Without a backport Mimikatz and tools like it will be useful for the foreseeable future.

What about simply removing the offending features? What about making the features optional? What about just prompting users for a password the first time one of these SSPs gets called rather than simply enabling everything? These are all interesting questions that would need to be discussed with the designers of Windows authentication. Perhaps there are well-reasoned arguments why this is infeasible. Microsoft, please, enlighten the masses!

In the next section we explore what an attacker has to play with in the event that there's nothing available to use in LSASS.

[4] http://hexale.blogspot.com/
[5] http://blog.gentilkiwi.com/securite/oups-compte-microsoft-windows-8-1-preview-en-clair

# The Delusion Starts, LSASS Memory is Empty

Even if Microsoft were to fix the issues with SSPs and plaintext credentials, attackers still have the venerable "Pass-the-hash Attacks" (PTH) to fall back on as well as possible attacks against Kerberos, such as "Pass-the-ticket".

As alluded to in the SSP section of this paper, NTLM suffers from several weaknesses. The first, which allows the so called PTH attacks relies on the fact that the NTLM protocols acts on the unsalted and hashed form of the password instead of the actual password. If the password hashes are compromised, then an attacker can use the hashes themselves to authenticate to almost anything on the network and access any and all data. **It is worth noting that this is exactly what the NTLMSSP does in Windows systems, it passes the hash! This is evidenced by the fact that only the hashes are kept by the NTLMSSP.**

There are two primary versions of NTLM in use today. The older version, sometimes referred to as NTLMv1 (also known as MS-CHAPv2) has been completely broken. Please refer to Moxie Marlinspike and David Hutton's talk last year at Defcon 20 for more details[6]. Their attack allows the initial password hash to be recovered, thus enabling PTH attacks. While their talk specifically talks about MS-CHAPv2, the authentication process is the exactly same for NTLMv1. A third security researcher, Mark Gamache[7], released a tool to pull out the necessary parts and run the NTLMv1 hash against Moxie's Cloudcrack[8] service. The other NTLM protocol in use, NTLMv2 has corrected many of the problems with NTLMv1, however it still affords an opportunity for the plaintext password to be cracked directly if captured on the wire.

In addition, both NTLMv1 and NTLMv2 are vulnerable to a relay attack[9]. In the relay scenario, an attacker somehow gets a user to authenticate to a service they control using NTLM. This could be an icon in a word document that references a network share on the attackers computer, or possibly a website requiring NTLM authentication. When the victim's computer automatically tries to authenticate (thanks again SSO), the attacker then attempts to authenticate to a service on a different server, simply forwarding on the victim's authentication to that service. When the service responds back with the challenge, the attacker forwards the challenge back to the victim's computer. This continues throughout the rest of the authentication exchange. After the authentication is complete, the attacker has an authenticated session to do with what they please.

It is worth noting that by default, domain controllers, require SMB signing in order to authenticate using NTLM. This effectively prevents the relay attack from working against a domain controller. However, the relay attack will work just fine against any other member servers which typically do not have SMB signing enabled.

---

[6] https://www.cloudcracker.com/blog/2012/07/29/cracking-ms-chap-v2/
[7] http://markgamache.blogspot.com/2013/01/ntlm-challenge-response-is-100-broken.html
[8] https://github.com/moxie0/chapcrack
[9] http://www.room362.com/blog/2012/7/10/cross-protocol-chained-pass-the-hash-for-metasploit.html

The Microsoft implementation of Kerberos also has its issues. First off, WCE allows the attacker to dump the user's TGT from one computer and reload it on a different computer, effectively transplanting the user from machine to machine. If the attacker gets ahold of a Domain Admin's TGT, they can effectively replicate the user on as many domain-connected computers as they want. Also, while technically the ticket granting tickets (TGTs) expire, it is relatively trivial to get the ticket's timeout extended, since the Microsoft implementation will automatically do it for you if you present an expired TGT when requesting access to services.

In addition, from our Blackhat USA 2012 whitepaper, the long term secret keys between the domain and the Kerberos entities is their password hash. In addition, the secret key used to sign all Kerberos TGTs is the KRBTGT hash. It's theoretically possible for somebody to take a regular users TGT, alter the server only portion to grant access as a Domain Admin, repackage it all together and then introduce it back onto the Windows system. Talk about a golden ticket.

People might think that these attacks against Kerberos are some sort of recent or newly discovered problems. However, many of the essential problems with Kerberos have been described in the 1991 paper "Limitations of the Kerberos Authentication System[10]" by Bellovin and Merrit. While the paper primarily relates to Kerberos 4 and spends most of its time talking about specific message interactions between Kerberos clients and servers, there are still some eerie security discussions that ring just as true today. Things like the previously discussed "pass-the-ticket" attack, compromise of the Kerberos Key Distribution Center can allow an attacker the ability to emulate any user, and the fact that the long term secret keys are the user passwords are all discussed. As George Santayana said "those who cannot remember the past are condemned to repeat it."

Hypothetically, what would happen if NTLM were eliminated and Kerberos didn't suffer from these weaknesses?

# The Descent into Madness - Tokens

If for whatever reason NTLM / Kerberos were unavailable, the next rung down the ladder relates to "security tokens". Security tokens are associated with every Windows process and detail information regarding the security context of the process. The seminal paper on the subject was entitled "Security Implications of Windows Access Tokens - A Penetration Tester's Guide[11]" by Luke Jennings from 2008. Even 5 years later this paper is worth the read.

As a quick summary, there are essentially 2 privilege levels of security tokens that are useful for attackers, impersonation and delegation. Impersonation allows an attacker to impersonate another user on the compromised system. Impersonation tokens are typically created when the computer is accessed non-interactively. This is primarily used as a privilege escalation technique in order to get to System level privileges on the compromised computer.

---

[10] http://academiccommons.columbia.edu/catalog/ac%3A127106
[11] http://labs.mwrinfosecurity.com/assets/142/mwri_security-implications-of-windows-access-tokens_2008-04-14.pdf

Delegation allows the user to leverage the level of access this token has on this system onto another system. Delegation tokens are typically created when the user establishes an interactive session. This is truly what an attacker wants in that it allows an attacker who finds a token for a user with higher privileges than what is currently available to move laterally and upwards towards the ultimate prize, a domain admin account. There could be several intermediate hops between a regular user and domain administrator. Usually from an attacker's standpoint it's simply a matter of time and jumping from account to account to discover the path to the top. One of our favorite techniques is to try to entice an administrator to log onto the system by causing some sort of issue on the computer. The admin logs in, we steal their token, life is good. Barring trying to invoke a login by an administrator user, attackers will often target network file shares as they have a large number of users who have file shares mounted through login scripts, and thus often have usable tokens.

The biggest problem with tokens is that in many cases, security tokens can remain available long after the process has been terminated. This is especially true with the way that many admin tools interact with the system. If the program being used properly logs out, the tokens generally disappear properly. However, in many cases, the only way to prevent the token from being used is to reboot the system!

For example, if a server administrator used RDP to log into a server and didn't explicitly log out of his session, the delegate tokens will remain. Other tools such as Sysinternals PsExec often leave tokens as well, depending on how it was used. Microsoft did some testing as part of their revision of their PTH whitepaper and has a section listing under what circumstances a limited set of tools leaves usable tokens.

Note: We have personally seen other tools leave tokens lying about. For example, recently we saw a McAfee AV scan kicked off by an IT department leave its security token after the scan was completed. An attacker now has instant access to the account the McAfee scanner used, often a highly privileged domain account!

The risk associated with delegate tokens can be somewhat mitigated on newer versions of Windows. Windows 2008 domains and later have the ability to designate accounts as "sensitive and cannot be delegated". However, in many cases this will prevent things from working properly. For example, the McAfee virus scanner previously mentioned needs to log on interactively in order to execute the scan that was initiated by the IT staff. Without delegation this account couldn't be used to execute the scanner on the remote machine.

So, what else is there to worry about? In the next section we'll talk about some items that deserve attention, but we really want to focus on GPP, or Group Policy Preferences, a commonly used feature that doesn't really help the problem.

# Other Flotsam and Jetsam

- Domain Computer Accounts / Domain Service Accounts

When a computer joins Active Directory, a computer account is created. The plaintext password (a long random unicode string) is kept in an area called the LSA (Local Security Authority) secrets. While using this account isn't anything terribly sexy, it's still a member of the 'Domain Users' group, which by default can gather all sorts of information regarding the Active Directory, such as a list of users, list of groups, group memberships, etc. This could also mean the ability to mount a file share. Useless Trivia: When a computer account is pre-created (or the password for the computer account is reset) the password is automatically set to the be the same as the computer account name. When the account is used to join the domain, the password is automatically reset to something random. Domain service accounts also save their passwords in LSA secrets.

- Domain Cached Credentials

  In the event that a computer loses touch with the Active Directory, by default up to 10 domain logon credentials in a salted and hashed format are stored. This allows the computer to authenticate an account while the connection between the computer and Active Directory is being fixed. While these credentials cannot be passed in a PTH attack, they can be dumped with tools like cachedump and they can be attacked offline with John the Ripper. These hashes are also stored in the LSA secrets.

- Plaintext Credentials in unattend.xml/autounattend.xml

  If an organization uses the Windows unattended installer to install new workstations, it's possible to recover credentials (possibly Domain Administrator!) from the master configuration file for the installation. There are several locations on a workstation where the file could be found after the installation. Read this cited article for more information[12]. Please note, oftentimes the unattended installer is installed on a network share and executed remotely leaving the configuration files readable by anybody on the domain.

- Plaintext credentials in configuration files / registry

  Some applications store credentials in configuration files or hide them somewhere in the registry. Metasploit has a large number of post-exploitation scripts that will hunt for these credentials since oftentimes they will be used in other places across the network.

- Local credentials

  Every Windows box has local accounts that are stored in a special registry hive called the SAM (Security Accounts Manager). By default Windows has a local administrator account (sometimes called the 500 account, because of its SID) and a guest account (default sid is 501). On a fresh install of Vista+, the local administrator account and the guest account are both disabled and have a blank password. When the machine finishes installation, a new account is created and

[12] http://technet.microsoft.com/en-us/library/cc749415(v=ws.10).aspx

added to the local administrators group on the computer.  One notable exception to this rule is with Windows 2012.  By default the local administrator account is not disabled.

UAC, or User Account Control, is enabled by default as well.  UAC affects the way that the local accounts can access the computer across the network.  The idea is that a local account should be used for administrative tasks locally.  Hence, when a local account attempts to authenticate to a workstation remotely, the OS prevents it from working properly.  However, there are a couple of very large caveats.

First, the capability that prevents the members of the workstations "administrators" group from accessing the workstation remotely can be overridden with a registry setting.  It is not unusual for an organization to enable this particular setting when transitioning from older versions of Windows to newer versions to get around workflow issues[13].

Secondly, by default the local 500 account is unaffected by the UAC settings, since the account is assumed to be disabled.  There is an explicit option to enable enforcement of UAC on the local 500 account.  So in the event that the local 500 has been enabled, unless that other setting is enabled, UAC isn't enforced.  This provides a popular method for lateral movement on the network in the event that the local administrator passwords are the same across multiple machines using PTH.

This leads us to discuss the previously mentioned Group Policy Preferences.

# Making a Bad Problem Worse: Group Policy Preferences

As if the Pass the Hash vulnerabilities aren't bad enough, Microsoft has introduced two features that make the problems dramatically worse. Group Policy Preference settings can easily ensure that every machine in your enterprise is vulnerable to lateral movement attacks by setting an identical password on every host or server. In addition, it suffers from a design flaw that allows for trivial password recovery.

# Group Policy Preferences

Group Policy Preferences were introduced with Windows Server 2008 and allow for more granular and flexible policy enforcement[14]. One of the features is the ability to configure local accounts, service accounts and scheduled tasks with credentials.  Administrators often use this feature to uniformly rename the built-in administrator account and ensure that password is consistent. A consistent password is exactly what enables Pass the Hash and why it is so successful.

---

[13] http://msdn.microsoft.com/en-us/library/windows/desktop/aa826699(v=vs.85).aspx
[14] http://technet.microsoft.com/en-us/library/cc731892(v=ws.10).aspx

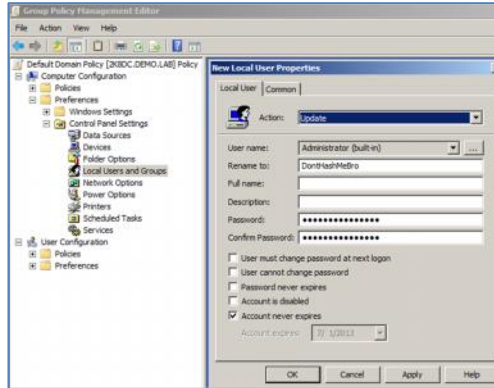


Figure 4 - Configuring GP Preference object to update username/pw

Newer versions of Windows Server actually warn the user against setting passwords in Group Policy preferences and this paragraph was added to a popular Technet blog post on the subject from 2008:

> "Because passwords in preference items are not secured, we recommend that you carefully consider the security ramifications when deciding whether to store passwords in preference items. If you choose to use this feature, we recommend that you consider creating dedicated accounts for use with it and that you do not store administrative passwords in preference items.[15]"

Although Microsoft never hid this vulnerability, adoption of the feature by administrators was slow and it didn't pique security researcher Emilien Giraul's interest until 2012[16]. Utilizing a network capture, Emilien discovered that passwords were enforced utilizing XML files that obscured the passwords by encrypting them with a static key. Microsoft also published the key[17]:

All passwords are encrypted using a derived Advanced Encryption Standard (AES) key.<2>

The 32-byte AES key is as follows:

```
4e 99 06 e8  fc b6 6c c9  fa f4 93 10  62 0f fe e8
f4 96 e8 06  cc 05 79 90  20 9b 09 a4  33 b6 6c 1b
```

Figure 5 - AES key published by Microsoft

With the key, it is trivial to decrypt the encoded and encrypted passwords with a PowerShell function[18]:

[15] http://blogs.technet.com/b/grouppolicy/archive/2008/08/04/passwords-in-group-policy-preferences.aspx
[16] http://esec-pentest.sogeti.com/exploiting-windows-2008-group-policy-preferences
[17] http://msdn.microsoft.com/en-us/library/2c15cbf0-f086-4c74-8b70-1f2fa45dd4be.aspx#endNote2
[18] Get-DecryptedCpassword is a tool to be released with this talk

```powershell
function Get-DecryptedCpassword {
    Param (
        [Parameter(Position = 0, Mandatory = $True)]
        [ValidateNotNullOrEmpty()]
        [String]
        $Cpassword
    )

    try {

        #Append appropriate padding based on string length
        $Mod = ($Cpassword.length % 4)
        if ($Mod -ne 0) {$Cpassword += ('=' * (4 - $Mod))}

        $Base64Decoded = [Convert]::FromBase64String($Cpassword)

        #Create a new AES .NET Crypto Object
        $AesObject = New-Object System.Security.Cryptography.AesCryptoServiceProvider
        [Byte[]] $AesKey = @(0x4e,0x99,0x06,0xe8,0xfc,0xb6,0x6c,0xc9,0xfa,0xf4,0x93,0x10,0x62,0x0f,0xfe,0xe8,
                             0xf4,0x96,0xe8,0x06,0xcc,0x05,0x79,0x90,0x20,0x9b,0x09,0xa4,0x33,0xb6,0x6c,0x1b)

        #Set IV to all nulls to prevent dynamic generation of IV value
        $AesIV = New-Object Byte[]($AesObject.IV.Length)
        $AesObject.IV = $AesIV
        $AesObject.Key = $AesKey
        $DecryptorObject = $AesObject.CreateDecryptor()
        [Byte[]] $OutBlock = $DecryptorObject.TransformFinalBlock($Base64Decoded, 0, $Base64Decoded.length)

        return [System.Text.UnicodeEncoding]::Unicode.GetString($OutBlock)

    } catch {Write-Error "$Error[0]"}
}
```

Figure 6 - Get -DecryptedCpassword PowerShell function to decrypt GPP passwords

With this feature, any member of the “authenticated users” group can simply browse to the policies folder on their domain controller:

```
\\127.0.0.1\sysvol\demo.lab\Policies\{31B2F34      \\127.0.0.1\sysvol\demo.lab...

  <?xml version="1.0" encoding="UTF-8"?>
- <Groups clsid="{3125E937-EB16-4b4c-9934-544FC6D24D26}">
  - <User clsid="{DF5F1855-51E5-4d24-8B1A-D9BDE98BA1D1}" removePolicy="0" userContext="0"
     uid="{C6E52168-0E48-4901-93C1-E102E2694E82}" changed="2013-07-02 05:43:21"
     image="2" name="Administrator (built-in)">
       <Properties userName="Administrator (built-in)" subAuthority="RID_ADMIN" acctDisabled="0"
          neverExpires="0" noChange="0" changeLogon="0"
          cpassword="9KQYhHxSxrrZjFo8Frt/nExdMLKsQM+ThhW0JKajaRc" description=""
          fullName="" newName="mspresenters" action="U"/>
     </User>
  </Groups>
```

Figure 7 - Groups .xml file from Domain Controller containing encrypted password string

Then easily recover passwords to administrative and service accounts providing a trivial privilege escalation path:

```
PS C:\demo> Get-DecryptedCpassword "9KQYhHxSxrrZjFo8Frt/nExdMLKsQM+ThhW0JKajaRc"
Recycling*3ftw!
PS C:\demo>
```

Figure 8 - Using  Get-DecryptedCpassword from the PowerShell console

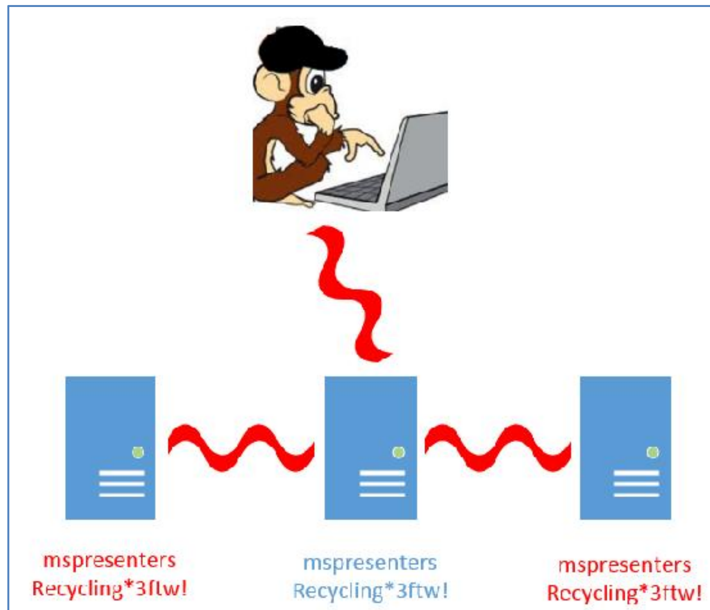

Figure 9 - GPP enables lateral movement with hashes or passwords

Instead of utilizing Group Policy Preferences to set passwords on local accounts, administrators have another option. Utilize a script to set a different password on every machine. There are tools such as Passgen[19], but it is not available for download and isn't open-source so we wrote Set-UniquePassword[20]. Administrators can use the function to set unique passwords for local accounts in two basic ways. The first is the random option which sets a 30 to 60 character random password for each account. The hostnames and their associated username/password combinations should be immediately taken offline or not stored at all. The second option is to provide a password phrase that can be appended or prepended to a unique string from the machine. Currently, hostname, MAC and serial number are the only "tokens" supported. Both the hostname and MAC address contribute to a unique password, but only the serial number would be difficult for an attacker to gain from the network and therefore add some additional protection if the phrase is discovered.

If no two local admin account passwords are the same, then no two hashes are the same effectively neutering the one of the traditional PTH attacks using local credentials.

[19] Protect Your Windows Network: From Perimeter To Data (Addison Wesley Professional, 2005)
[20] Set-UniquePassword is a tool to be released with this talk

# Dispelling Rumors: Smart cards as a possible solution

One idea that keeps being trumpeted as the solution to the entire problem is the notion of smart cards. Microsoft describes smart cards as "small, tamperproof computers" which "perform their own cryptographic operations" and are "as revolutionary to the computer industry as the introduction of the mouse or CD-ROM."[21] Ultimately, they are wallet-sized two factor authentication tokens embodying something the user possesses as well as a pin which is something the user must know. Unfortunately, Microsoft's implementation of the technology is flawed.

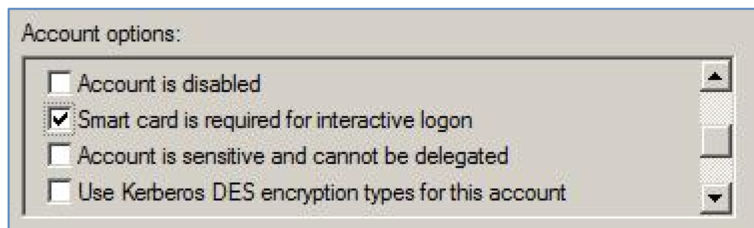


Figure 10 - AD setting requiring the use of smart cards

The easiest way to explain the concern with smart cards is with an example scenario that represents a common and secure environment. An organization requires smart cards for all interactive logons to include administrative and privileged accounts.

Traditional enterprise services such as SharePoint, Exchange and Outlook Web Access (OWA) are internet and intranet-facing. Users login into their workstations with their smart card and access an external service via SSO. They are not required to authenticate more than once in order to avoid "credential fatigue". Additionally, the organization has several internally-developed web applications that customers use. All servers are patched regularly and the organization has a robust security infrastructure with antivirus, host-based security products and third-party application whitelisting.

Despite all of security controls, an attacker is able to discover a seemingly insignificant file-upload vulnerability in a vendor receipt submission system. The attacker uploads a custom ASP file which is executed as the limited "NT AUTHORITY/Network Service" account[22]. The webshell is used to quickly discover domain credentials in database connection strings. This access is used to mount and search the domain controller for passwords in Group Policy Preferences. A local administrator name change and password are discovered and trivially reversed with a script[23]. The attacker then uses the credentials to invoke a malicious PowerShell script[24] to securely exfiltrate the database containing all password and pulls it off the network with a browser. The attack took less than half an hour from start to finish.

Fortunately, the attack is discovered within 12 hours when an alert employee discovers the strange receipt submissions within the web application. Incident handlers recover a large encrypted file and

[21] http://technet.microsoft.com/en-us/library/dd277362.aspx
[22] http://msdn.microsoft.com/en-us/library/windows/desktop/ms684272(v=vs.85).aspx
[23] https://github.com/mattifestation/PowerSploit/blob/master/Recon/Get-GPPPassword.ps1
[24] http://gallery.technet.microsoft.com/scriptcenter/Get-PasswordFile-4bee091d/

notice the access times of all the web application source code which leads them to conclude that the attacker compressed the source code and files associated with each of the websites on the server. The organization chooses to decommission the server and rush the new and more secure version into production. After careful monitoring, no evidence of continued compromise was found and the organization continued business operations after lauding their incident response prowess.

Thanks to common misconceptions, one hour and a small web vulnerability (which could easily be replaced with a spear-phishing attack), this organization will likely be compromised for years. How is the enterprise still compromised when the attacker didn't leave behind any code? Armed with the NTDS.dit and system files, the attacker can easily retrieve the NT hashes of each user account offline[25]. How are the credentials useful to the attacker without persistent internal network access? Aren't smart card passwords pseudo-random, long and nearly impossible to crack?

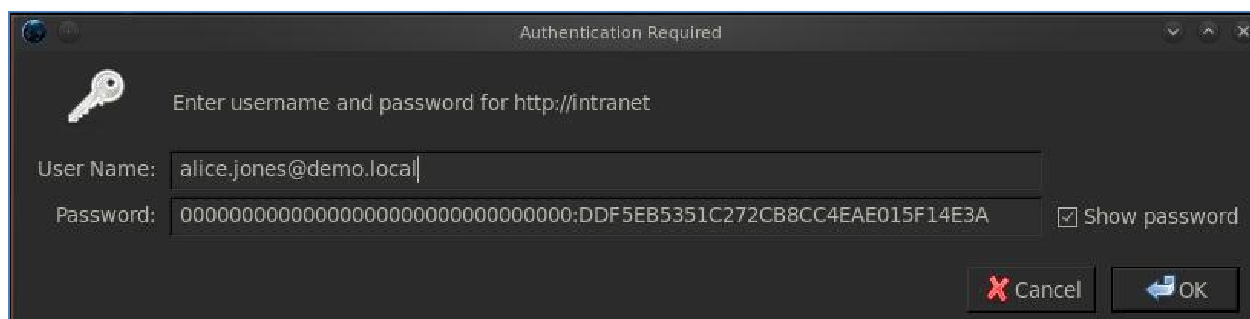


*Figure 11 - Hash being supplied instead of the password for NTLMSSP authentication*

The attacker doesn't need to crack them and the hashes can be passed to the internet services that support NTLMSSP with a version of Firefox patched[26] to Pass the Hash. In our scenario, the attacker can now authenticate to SharePoint and OWA as any user in the domain from the internet. If internal access was needed, the attacker could certainly leverage webmail to accomplish it.

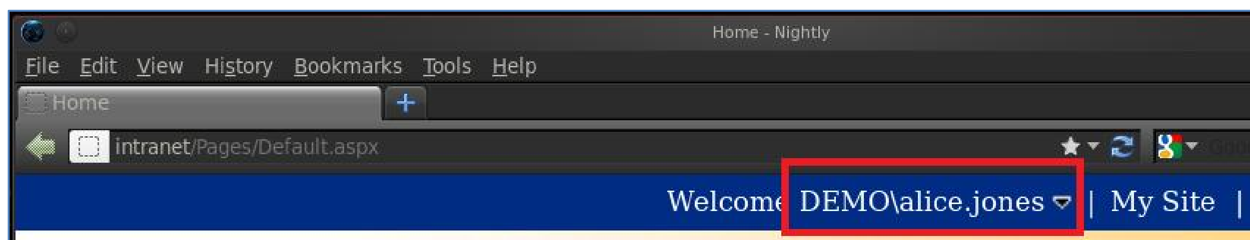


*Figure 12 - Successful pass-the-hash attack utilizing PTH-Firefox from the PTH-Suite*

It may not be immediately clear what role smart cards play in this scenario, but it is a devastating one. In a typical domain, the attacker would have to contend with semi-regular password changes that would render the stolen Active Directory database ineffective after a few months. Additionally, the attacker would be forced to repeat a similar brazen attack or leave behind code for persistence which could be

[25] http://www.ntdsxtract.com/en/ntdsxtract.html
[26] https://code.google.com/p/passing-the-hash/downloads/list

discovered. However, by enforcing smart card logons the hashes associated with each account will likely never change because the pseudo-random password is only generated once. That means that one successful attack against this enterprise will yield elevated and user access to services until the active directory is rebuilt.

We can see how this works by creating a new user, setting a password and dumping the hash of the associated account:

```
SmartCardUser:1105:aad3b435b51404eeaad3b435b51404ee:d8888d892771451f18b263c916721bed:::
```

Figure 13 - Username:ID:LM:NT::: hash format of newly created account

Next we enable the setting for "Smart card is required for interactive logon" from the account options of the new user:

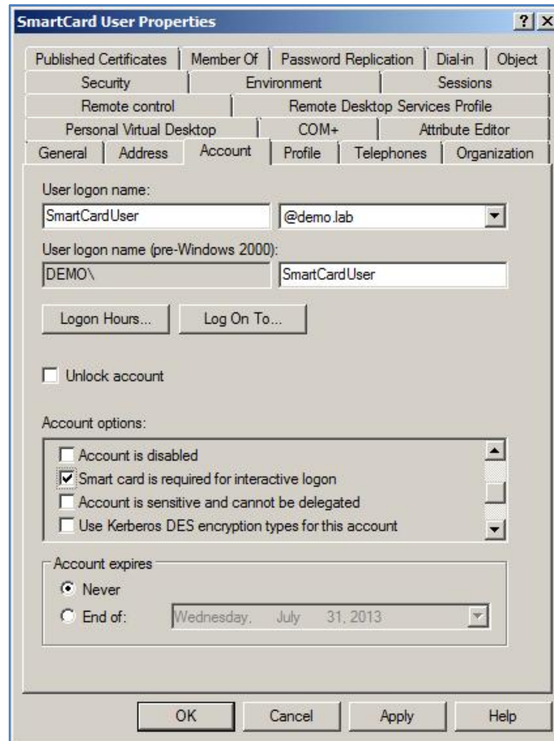


Figure 14 - Smart Card is enforced for interactive logons

The password change is confirmed by the new NT hash:

```
SmartCardUser:1105:aad3b435b51404eeaad3b435b51404ee:c4f5cb57e34555c2d19b294554187c1d:::
```

Figure 15 - The NT password hash has been altered

In order to get the plaintext, we need to login with a smart card and we can use Mimikatz[27] to dump the wdigest password from lsass:

```
66 ee b2 ba c4 5d ff 54 3d 6e cc fc 02 cc f2 8b 3b aa df 42 f9 eb 80 89 55 b5 06 7f 0c f2 f7 83
43 98 f5 fe 25 d9 74 13 7e f4 c4 67 6f 84 a4 0c 86 59 66 c5 5e d1 1a 81 6e 0a f0 07 c1 53 97 f2
1a 11 fa 49 bc 9a 51 4b 7b 86 b1 40 11 0e 22 8f 2f 9e 8b d1 28 ef d7 0c c0 79 a7 69 e5 2d 6c 98
57 b4 c2 60 88 b4 45 93 8e ce b2 7a 60 aa 7e f4 b5 f4 a4 bf 4e aa c8 25 e4 38 3f 1d a1 b8 3e 8f
3a d0 c8 e8 18 4a 5d 8a ae d4 a5 41 81 0d da 8c 2d 2f ff 21 74 0c b4 da 51 65 88 b3 f4 fe 92 4a
da 4c 9c 7d 31 5d 58 d4 ee f2 f4 4b be 5a 14 a7 01 3a a6 bd 43 9c 5c b0 2f 00 f3 0d fe 6e 1f 1a
4c 84 e0 ad b8 6f fc 5a 3c 82 c4 84 42 19 ad 9a 41 0b e9 09 f0 89 d9 e3 dc 80 a6 d2 94 00 d6 37
1f 28 43 40 ef a2 24 3a ba dc ac 0a 61 1a 8f 88
```

Figure 16 - Hex representation of account's plaintext password from Mimikatz

Even if the account's password age is allowed to expire, the hash will not change. There are two general ways of addressing this problem, but both suffer from the same major shortcoming. The first way is to toggle the account option setting thereby causing Windows to generate a new random password for the account. This can easily be automated and ran against every relevant account in the domain with PowerShell[28]:

```
#Import AD Module
Import-Module ActiveDirectory

#Create an array of all accounts that have smart card logon enforced
$ADUsers = (Get-AdUser -Filter * -Properties 'SmartcardLogonRequired' |
    Where-Object {($_.SmartcardLogonRequired)}).SamAccountName

if ($ADUsers -eq $null) {Write-Error "No Accounts Require Smart Cards for logon"}

#Iterate through each account that has the setting enabled and toggle
ForEach ($User in $ADUsers) {
    Get-AdUser -Identity $($User) | Set-AdUser -SmartcardLogonRequired $False
    Start-Sleep 1
    Get-AdUser -Identity $($User) | Set-AdUser -SmartcardLogonRequired $True
}
```

Figure 17 - Code snippet from Reset-SmartCardPassword

The other solution is to regularly generate random passwords for every account. This is generally not recommended because of the difficulty in generating truly random passwords. Pseudo-random passwords can of near random lengths can be created with .Net and PowerShell[29]:

[27] http://blog.gentilkiwi.com/mimikatz
[28] Reset-SmartCardPassword is a tool to be released with this talk
[29] Get-RandomPassword is a tool to be released with this talk as a part Set-UniquePassword

```
Function Get-RandomPassword {
    #random password length should help prevent masking attacks
    $PasswordLength = Get-Random -Maximum 60 -Minimum 30

    #Use .net to generate a random password
    [Reflection.Assembly]::LoadWithPartialName("System.Web") | Out-Null
    $GeneratedPassword = [System.Web.Security.Membership]::GeneratePassword($PasswordLength,0)

    #check complexity http://technet.microsoft.com/en-us/library/cc786468(v=ws.10).aspx
    $ComplexityTest = $GeneratedPassword -cmatch `
    "^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[\x21-\x2F\x3A-\x40\x5B-\x60\x7B-\x7F]).{30,60}$"

    if (!($ComplexityTest)) {Get-RandomPassword}
    Return $GeneratedPassword
}
```

Figure 18 - Code snippet from Set-UniquePassword

Either solution is flawed in that it renders all current interactive logon sessions invalid. This shortcoming would have to be solved with policy or a hotfix from Microsoft.

# Mitigation

We summed up mitigation in our previous whitepaper[30]:

> A common misconception is that security tools and configuration changes can prevent this type of attack from being successful. Unfortunately, most antivirus tools are looking for specific, known attack tools and their behavior. For example, several antivirus products flag Hernan Ochoa's WCE as malicious and remove it. Other products detect the use the Metasploit PSEXEC exploitation module[31] due to the service it starts. However, AV products can be trivially bypassed[32], in many cases, by encoding or otherwise obfuscating the binary in question.
>
> On the network side, a successful "network login" event is virtually indistinguishable from one created with a hash. Up until recently (Windows 2008 R2)[33] there wasn't a distinction between NTLM and Kerberos being used for logins that was noted in the event logs. The ultimate mitigation against the "Pass the Hash" attack is to protect the hashes from being disclosed to an attacker. However, there are steps that can be taken to lessen the chance of a successful attack. The first is regular password changes. NT hashes have limited value once the corresponding account's password has been changed. Frequent password changes forces an attacker to repeat their attack which could be detected. Audit log review and monitoring of privileged accounts is also critical to detecting the attack. Networks should be segmented to prevent lateral movement by attackers. Administrators should also prevent external services from being accessed from external hosts. Sensitive ports such as SMB (TCP 445) and NetBios (TCP 139)

[30] http://media.blackhat.com/bh-us-12/Briefings/Duckwall/BH_US_12_Duckwall_Campbell_Still_Passing_WP.pdf
[31] http://www.metasploit.com/modules/exploit/windows/smb/psexec
[32] http://www.obscuresecurity.blogspot.com/2012/12/finding-simple-av-signatures-with.html
[33] http://blogs.technet.com/b/askds/archive/2009/10/08/ntlm-blocking-and-you-application-analysis-and-auditing-methodologies-in-windows-7.aspx

should be blocked and integrated authentication should not be used by externally accessible websites (e.g. SharePoint).

The "Pass the Hash" attack is actually a documented part of the way that Windows protocols interact. Since the designers of the protocols didn't want to ask the user for their username/password every time network authentication takes place, the password hash is stored locally in memory and is constantly being recycled as long as the user stays logged in. Because it's an integral part of the design of Windows authentication, the "attack" does not have a traditional defense. The best way to prevent the attack from happening is to protect the Domain Controller from compromise with a comprehensive defense in depth approach. Enforce least privilege, enable UAC, limit the number of elevated accounts, etc. Once the DC is lost, all your data becomes accessible.

The best and only real mitigation to the Pass-the-Hash attack is to completely prevent credentials from ever being exposed. There are hundreds of ways to lessen the severity of the attack and Microsoft's working group on the subject published an 80-page whitepaper[34] that settles on 3 main suggestions to mitigate the attack:

1. Restrict and protect high privileged domain accounts.
2. Restrict and protect local accounts with administrative privileges.
3. Restrict inbound traffic using Windows Firewall.

Microsoft covers step-by-step how to implement each mitigation with controls that range from obvious to absurd in any realistic enterprise. Some of the absurdity lies in the fact the authors chose to only reference Microsoft products and ignore common network devices that likely exist in every enterprise. For example, creating an administrative VLAN and ensuring administrative workstations are isolated is far more realistic for most enterprises than creating complex Windows Firewall rules, especially given the lack of true centralized management capabilities for large Windows Firewall installations.

# Detection

Any mitigation strategy that focuses on preventing the "Pass the Hash" attack will likely result in failure because a well-executed attack is indistinguishable from an administrator doing their job. That isn't to say that administrators shouldn't be reviewing their logs for signs of potential compromise, because detection is probably the best second best defense to this attack.

Administrators should be tracking all privileged account usage and verifying that the usage is authorized. Reviewing logs is time expensive and log analytics tools can be spend expensive, however any review is better than none at all. There are also ways to automate the analysis without spending any money on software and there are several strategies that have proven effective in detecting this attack (not preventing it).

[34] http://www.microsoft.com/en-us/download/details.aspx?id=36036

The concept of two man integrity should be utilized when reviewing logs for any system. By reviewing the logs of a server administered by someone else, members of the team will learn and recommend improved methods in addition to adding an early detection process for insider threats. Any unusual activity should be investigated by both administrators.

Another promising technique is to have a weekly review of all elevated account usage. Modern Windows systems support a feature[35] that can be used to forward all logs to a central point for review. With the logs centrally located, they can be reviewed and analyzed for signs of compromised credentials. For example, a company's security policy prohibits the use of the popular PsExec[36] tool due to the fact that it sends credentials in plaintext. They have aggregated their logs to one server and would like to automate detection of policy violations:

```
Function Find-PSExecService {
    $ServiceStarted = (Get-Eventlog -LogName "system" | Where-Object {$_.EventID -eq 7045}`
        | Where-Object {$_.Message -like "*PSExec*"})

        $ServiceStarted | Foreach-Object {
            $UserName = $_.UserName
            $Time = $_.TimeGenerated
            $Hostname = $_.MachineName

            $ObjectProps = @{'Hostname' = $Hostname;
                             'UserName' = $UserName;
                             'Time' = $Time;}

            $Results = New-Object -TypeName PSObject -Property $ObjectProps
            Write-Output $Results
        }
}
```

Figure 19- Find-PSExecService searches the registry for evidence of the PSEXECSVC

This simple function quickly churns through the system log for evidence of services being created and returns the relevant information:

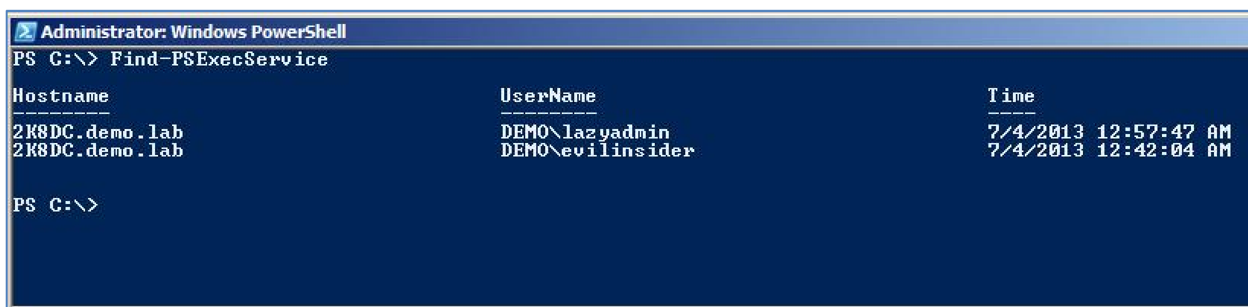


Figure 20 - Find-PSExecService used to detect suspicious credential use

[35] http://technet.microsoft.com/en-us/library/4aa6403f-d4b8-43a4-a70d-ceb7f88c524e
[36] http://technet.microsoft.com/en-us/sysinternals/bb897553.aspx

Now administrators can quickly audit for a specific banned application on regular intervals, but what if we want to be alerted sooner? All of these functions can be schedule to run hourly and combined with the built in PowerShell commandlet Send-MailMessage[37]. This enables an administrator to receive an email or text close to when the activity is originally logged.

This process can quickly become ineffective if all detection is focused on publically available attack and penetration tools such as Metasploit. In all likelihood, these will not be the exact tools that will be used against you. For example, there are a few attributes that when combined make the service that the Metasploit PsExec module creates easily identifiable. By default, the service name begins with "M" and is followed by a random number of characters[38]. Additionally, the service binary is always eight characters long and placed in the "%SYSTEMROOT%" directory. Using those two facts we can quickly locate log entries that were likely created by the Metasploit module:

```
Function Find-MsfPSExec {
    $ServiceStarted = (Get-Eventlog -LogName "system" |`
    Where-Object {$_.EventID -eq 7045} |`
    Where-Object {($_.Message -match "Service Name:  M")} |`
    Where-Object {($_.Message -like "*%SYSTEMROOT%\????????.exe*")})

        $ServiceStarted | Foreach-Object {
            $UserName = $_.UserName
            $Time = $_.TimeGenerated
            $Hostname = $_.MachineName

            $ObjectProps = @{'Hostname' = $Hostname;
                             'UserName' = $UserName;
                             'Time' = $Time;}

            $Results = New-Object -TypeName PSObject -Property $ObjectProps
            Write-Output $Results
        }
}
```

Figure 21 - Find-MsfPSExec searches the logs for activity matching the default config of a Metasploit module

[37] http://technet.microsoft.com/en-us/library/hh849925.aspx
[38] https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/windows/smb/psexec.rb#L241

```
Administrator: Windows PowerShell
PS C:\> Find-MsfPSExec

Hostname                                UserName                                Time
--------                                --------                                ----
2K8DC.demo.lab                          DEMO\evilinsider                        7/4/2013 2:02:38 AM

PS C:\>
```

Figure 22 - Find-MsfPSExec used to detect potential compromised credentials

The limitation of this type of signature matching should be obvious. The attacker could change the service name to match a legitimate service[39] or not start a service at all[40]. So instead of targeting the tools, we can target the specific activity that we need to monitor for. Service creation is definitely something you want to check for, but network logon events[41] are definitely more relevant.

By focusing on the activity that should be monitored and not the tools, administrators can potentially detect other authentication problems or attacks. We will need to search for all events with the ID of 4624 and instead of using the Get-Eventlog Commandlet in PowerShell, we can take advantage of the new filtering available in Windows Server 2008:

```
Function Find-NTLMNetworkLogon {
    $Filter = "*[EventData[Data = 'NtLmSsp ']]"
    $Events = Get-WinEvent -Logname "security" -FilterXPath $Filter |`
        Where-Object {$_.ID -eq 4624}

        if ($Events) {$Events | ForEach-Object {

            $ObjectProps = @{'Hostname' = $_.Properties[11].value;
                             'IPAddress' = $_.Properties[18].value;
                             'UserName' = $_.Properties[5].value;
                             'Domain' = $_.Properties[6].value;
                             'Time' = $_.TimeCreated;
                             'Workstation' = $_.MachineName}

            $Results = New-Object -TypeName PSObject -Property $ObjectProps
            Write-Output $Results
            }

        }
}
```

Figure 23 - Find-NTLMNetworkLogon function to parse event log quickly

[39] http://www.room362.com/blog/2012/6/25/evidence-of-compromise-metasploits-psexec.html
[40] http://www.metasploit.com/modules/auxiliary/admin/smb/psexec_command
[41] http://support.microsoft.com/kb/947226

On a busy server this is going to return a lot of results, so we should filter out objects that we don't want to see:

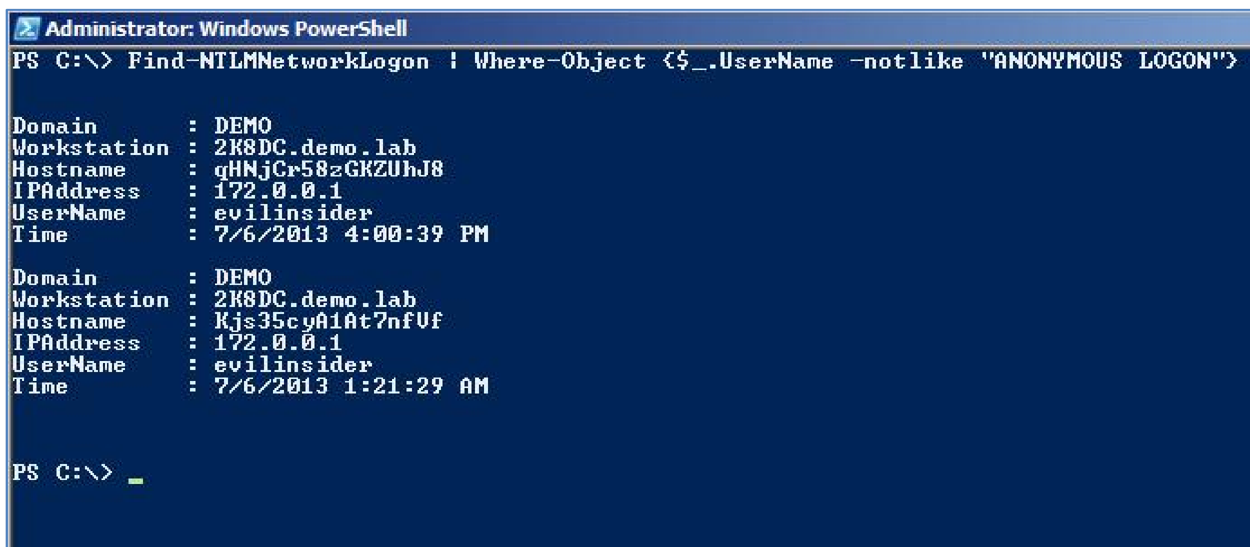


Figure 24- Filtered results detecting 2 Metasploit PSExec Attacks

This could be combined with a filter for privileged accounts and piped to Send-MailMessage all in a task that sends a report daily. Alternatively, a task could be configured with event-triggering[42] to act as an Intrusion Detection System (IDS) for nefarious credential use. Automating tedious tasks like log review is an important task if you hope to stop attacks such as Pass-the-Hash.

# Conclusion

An attacker has multiple vectors to get credentials in a Microsoft Active Directory environment, with PTH attacks being only a small part of the problem. While the term "Pass the Hash" is pretty sexy, it does a disservice to the numerous other issues that Windows has with credential leakage. Whether it's plaintext passwords found in memory, plaintext passwords found somewhere on disk, using tokens or whatever, attackers tend to take the path of least resistance. For the attacker, all roads eventually lead to Domain Admin.

The only true preventative measure that can be taken is to make the trip from regular user on a computer to SYSTEM level privileges take as long as possible. This often means basic security principles such as not allowing desktop users to be admins on their local machines, enforcing privilege separation, etc. By preventing an attacker from gaining SYSTEM, it becomes harder to extract credentials from memory since they don't have easy access to credentials in LSASS, and thus have to work a little harder for their access.

Detective measures will ultimately be the best defense. We hope that enterprises find the tools we release as ultimately useful and helpful in narrowing down possible attacks.

---

[42] http://technet.microsoft.com/en-us/library/ff935309(v=ws.10).aspx