National Security Agency/Central Security Service

Information Assurance Directorate

Reducing the Effectiveness of Pass-the-Hash

November 19, 2013
Revision 1

A product of the Network Components and Applications Division

TSA-13-1005-SG

Contents

List of Figures

Disclaimer

This Guide is provided "as is." Any express or implied warranties, including but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall the United States Government be liable for any direct, indirect, incidental, special, exemplary or consequential damages (including, but not limited to, procurement of substitute goods or services, loss of use, data or profits, or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this Guide, even if advised of the possibility of such damage.

The User of this Guide agrees to hold harmless and indemnify the United States Government, its agents and employees from every claim or liability (whether in tort or in contract), including attorneys' fees, court costs, and expenses, arising in direct consequence of Recipient's use of the item, including, but not limited to, claims or liabilities made for injury to or death of personnel of User or third parties, damage to or destruction of property of User or third parties, and infringement or other violations of intellectual property or technical data rights.

Nothing in this Guide is intended to constitute an endorsement, explicit or implied, by the U.S. Government of any particular manufacturer's product or service.

Trademark Information

This publication has not been authorized, sponsored, or otherwise approved by Microsoft Corporation.

Microsoft®, Windows®, Windows Server®, Windows Vista®, Active Directory®, Windows PowerShell ™, and Windows® Firewall with Advanced Security are either registered trademarks or trademarks of Microsoft Corporation in the United States and other countries.

1 Introduction

Pass-the-Hash (PtH) is a technique for compromising additional machines on a network, after one is initially exploited. In December 2012, Microsoft released a whitepaper[1] which discusses PtH in-depth, identifies numerous risk factors that make an organization highly vulnerable to PtH, and describes several mitigations. The purpose of this document is to expand on the ideas presented in [1] and provide guidance to DoD administrators.

PtH is not a new technique. PtH was publicly released in 1997[2] and reuses login credentials on one computer to access another computer with equivalent credentials. PtH enables attackers to spread laterally across a target network with ease. Even networks that require smartcard authentication are susceptible to PtH attacks.

The success of a PtH attack is dependent on many factors including the network configuration, firewall settings, and account settings. To defeat PtH attacks, administrators must differentiate authorized and unauthorized uses of Single Sign On (SSO), which is extremely difficult or impractical within any contemporary SSO architecture. SSO allows users to authenticate with the operating system *once* (generally once per login) and caches the user's credentials in memory. Each subsequent time the system requires the user's credentials to perform an action, instead of re-prompting the user, the cached credentials are used. The basic mechanisms on which PtH exploits cannot be immediately fixed with a patch and will require a fundamental SSO architecture design change. This document discusses mitigations administrators can deploy, in the interim, to reduce PtH's effectiveness by addressing some of the properties it depends upon.

2 Background

Secure computer systems implement authentication mechanisms to ensure authorized access. The authentication process requires a user to prove their identity by providing at least one authenticator (credentials). The most common authenticator continues to be passwords, but the following discussion, while explicitly referring to passwords, is generic to other authentication mechanisms, e.g., smart cards or biometrics.

For each user account, the system stores a username (identity) and a representation of the authenticator. The operating system generally uses cryptographic hash functions to store the hash of the authenticator, instead of storing it in cleartext. A hash value, or simply a hash, is the output of a hash function. Hash functions are one-way mathematical functions that take an input string and produce some fixed-size and deterministic output string. For example, when a user is prompted for a password during authentication, the system takes the password, computes the hash value, and compares the computed hash against the stored hash. If the hashes are equivalent, then the user is authenticated and is allowed access to the system; otherwise, the authentication fails.

Different systems and services allow users to authenticate in different ways. PtH attacks harvest hashed user credentials and reuse them to authenticate with services supporting a hash as an authenticator. An

attacker with administrative privileges can retrieve all of the hashed user credentials from disk by reading the Security Account Manager (SAM) file or by reading the Local Security Authority Subsystem Service (LSASS) process's memory. These credentials, if identical to credentials accepted on another machine, can be used to authenticate with remote Windows services and gain access to those systems.

Security policies continue to advocate stronger passwords that combine upper and lower case letters, special characters, and numbers. Longer, more complex passwords typically increase the difficulty of password cracking. However, password strength does not affect the success of PtH, which increases PtH's potential value in environments implementing strong password requirements.

The following attack describes a portion of the PtH methodology to gain domain administrator privileges. First, the attacker compromises an initial machine via a remote exploit or a client-side attack. Depending on the exploit, the attacker may only have user privileges and must escalate privileges to an administrator or system account, which grants access to all credentials on the local system. Next, the attacker attempts to use each set of credentials (preferably administrator credentials) to authenticate with other systems on the network via PtH. If any credentials grant access to a new machine, then more credentials are harvested. The attacker continues to spread across the network by employing this methodology and potentially gaining more credentials on each newly exploited machine until the domain administrator account is obtained or until all credentials have been exhausted. At which point, the attacker may have to use more difficult and noisier methods to expand control increasing the likelihood of being discovered.

3 Mitigations

The availability of free tools for obtaining user credentials continues to increase, e.g., Metasploit[8], Windows Credential Editor[3] (WCE), gsecdump[7], and Mimikatz[4], giving attackers a trivial way to increase their presence on a network. Successfully launching a PtH attack depends on the following conditions:

- The credentials used in the PtH attack must be valid credentials on the target system.
- The associated account name from the compromised credentials must have the Network Logon right on the target.
- The source of the attack must be able to communicate with the target over the network and the target must be configured to accept network connections.

These are the fundamental properties this paper's mitigations address.

Section 3.1 discusses the complications arising from duplicating credentials across machines. Section 3.2 discusses restricting local administrator accounts from remotely accessing systems. Section 3.3 provides Windows Firewall rules for restricting lateral movement between workstations. Deploying at least one of these mitigations is recommended, but implementing more will increase overall network security.

3.1 Creating unique local account passwords

A common practice for deploying an enterprise network is creating an image for a base machine and using this image as the baseline for other systems on the network. A side effect of this deployment strategy is that all of the built-in administrator accounts, across all machines, will have the same password. Recall the attack from Section 2. The adversary, on each machine on the network, wants to harvest all user credentials and reuse them to potentially authenticate to other systems. Under this scenario, once the attacker has presence on one machine they are capable of compromising each system on the network via PtH because all local administrator accounts share the same credentials. Even worse, when gaining unauthorized access to the new machine, the attacker already possesses administrator privileges allowing them to immediately harvest more credentials. This common setup is the ideal environment for PtH.

Enforcing unique local administrator credentials on each machine forces the adversary to perform additional steps, after they gain initial access, to achieve their goal. For example, the attacker may be forced to crack the administrator's password offline, use PtH with a standard user's account, or exploit the machine (which may make detecting an ongoing attack easier).

Part of the problem on Windows systems is that the hash of local user credentials is not computed with a salt. So, if a credential's hash is obtained, then any local user with the same credentials will also have the same hash. This section introduces a Powershell script for ensuring unique local account passwords and password hashes across a network (see Appendix A). The script creates unique passwords through two different password generation schemas. The first schema changes the password for a user-specified local account on each machine to a randomly generated password. The second schema changes a local account's password to the hostname prepended or appended to a user-defined string. For example, for password "password1" on hostname "win7-client" this schema will generate either the password "win7-clientpassword1" or "password1win7-client." Under the assumption that all hostnames on the network are unique, all resulting passwords and their corresponding hashes will also be unique. Thus, providing administrators a more manageable way to administer a network (compared to remembering or writing down many random passwords), while reducing the damage caused by an attacker replaying duplicate credentials. The viability of this schema relies on the fact that the administrator, with a high likelihood, will know the machine name a priori, which aids the convenience of the password strategy. In actuality, the string used to provide uniqueness can be anything, but to be usable it should be easily identifiable by an administrator, unique to each system, and not trivially discovered by an attacker.

The script outputs two text files. The first file contains a summary of successful password changes with the new password for each computer. The second file lists the machines with unsuccessful password changes, e.g., a machine is powered off. By default, the script forces the password file to be saved to a USB drive rather than the local file system, where it can be removed and not accessed by a remote adversary. Using an encrypted removable USB drive to store this data is recommended.

3.2 Denying local accounts from network logons

The mitigation in Section 3.1 curbs an adversary from performing PtH attacks using local administrator accounts in networks with duplicated passwords. This section presents a mitigation for restricting remote access to local administrator accounts.

Windows login architecture uses many different logon types. Each logon type is associated with a set of connection methods, which store credentials differently. Common logon types include: **interactive**, **remote interactive,** and **network**. **Interactive** logons, generally associated with a console logon or the RUNAS command, leave hashes in memory. **Remote interactive** logons also leave credentials in memory and are associated with Remote Desktop connections. N**etwork** logons, through commands such as **psexec** (without the –u option), **net use**, and **MMC snap-ins,** however, do not leave hashes in memory. See Table 6 in [1] for a list of common remote connection methods and where hashes are stored, if anywhere.

Local, non-service accounts do not generally require remote login privileges in a domain setting to perform their required tasks. Therefore, removing the **network** and **remote interactive** logon privileges from these accounts, especially local administrator accounts, will harden the system and prevent an attacker from using PtH with local accounts to obtain unauthorized access to other machines. Denying local administrators remote access forces machines to be physically administered or remotely administered through a domain account. Physically administering a machine is the most secure method, but may be an unrealistic administration method for many networks.

Remotely administering machines with a domain account provides convenience, but, if not done securely, can leave systems vulnerable to PtH attacks. First, when possible, machines should be administered with tools and methods that do not leave reusable credentials in memory (see Table 6 in [1] for a list of common methods). Second, domain accounts should be used in a way that conforms to the principles of least privilege and system isolation. That is, systems should be administered with the least privileges possible to perform the necessary task and highly trusted accounts should not administer lower trusted workstations. For example, the domain administrator account should be used to administer the domain controller, but not user workstations. This way, if a lower trusted system is compromised, then the attacker is limited to only compromising other lower trusted systems.

Appendix B presents two methods to remove the **network** logon right from local accounts. The first method runs locally on each system in a domain as a startup script and puts all local administrator accounts into a well-defined local group. The domain administrator then configures the **Deny access to this computer from the network** and the **Deny log on through Remote Desktop Services** (for Windows Server 2008R2 and later) policies to include this local group, which blocks local administrators from remotely logging in and also creates a tripwire[5]. The second method provides similar functionality through Group Policy Preferences without requiring a logon script. With one of these controls in place, if a local administrator account tries to remotely access a machine, e.g., an adversary in a PtH attack, then the system can be configured to log the event (see [5]) and trigger an alert for further investigation.

3.3 Restricting lateral movement on the network with firewall rules

To successfully perform a PtH attack, the source of the attack must be able to communicate with the target. Therefore, restrict workstations from communicating directly with other workstations using Windows Firewall rules. This should have little impact on the user since workstations generally have no need to communicate with each other. If a workstation has services which require other workstations to communicate with it, the firewall can be configured to only allow that specific traffic through. Implementing this mitigation correctly requires an administrator to thoroughly understand the entire network and provided services, but also provides a larger improvement to a network's security posture. It also reduces the number of potential targets for a PtH attack more than the mitigations presented in Sections 3.1 and 3.2. After implementing these firewall rules, an adversary on a workstation may only attack systems not protected by the firewall, find an exploit to evade the firewall, or attack servers which should have a smaller attack surface and are monitored more closely.

Appendix C provides step-by-step instructions for setting up Windows firewall rules through Group Policy Object (GPO) settings, which disallow workstation to workstation traffic.

4 Windows 8.1 Features

Windows 8.1 and Windows 2012 R2 introduce many new features designed specifically to combat PtH attacks. These features can be broken down into two categories: client-side and domain functional level (DFL) protections. The client-side features increase security on a host by host basis and require Windows 8.1 or Windows 2012 R2. The DFL protections require a Windows 2012 R2 functional level. Since it is not a guarantee that these features will be backported, assume that upgrading to Windows 8.1 will be required to use these protections. See [9] for more information on new Windows 8.1 features.

4.1 Deny local accounts from network logons in Windows 8.1

Section 3.2 discusses denying local administrator accounts access to network resources through group policy settings. In particular, the administrator is required to do this manually or through one of the scripts in Appendix B. Windows 8.1, however, by default, automatically places each local administrator account into a well-known group by default named **Local account and member of Administrators** group, which removes the need to run the provided scripts. So, the only action required by an administrator is to add the **Local account and member of Administrators** group to the **Deny access to this computer from the network** and **Deny log on through Remote Desktop Services** group policy settings.

4.2 New Remote Desktop feature in Windows 8.1

Windows 8.1 and Windows Server 2012 R2 added a new RDP option, **restrictedadmin**, which improves the security of using RDP to administer machines. (This option is only available through the command line program mstsc.exe and not the RDP GUI application.) Normally, RDP leaves credentials in memory as long as a user is logged in. With the **restrictedadmin** option enabled, reusable credentials will not be sent in plaintext during authentication and the target machine will not cache any reusable credentials. The **restrictedadmin** option requires both the client and the target to be running Windows 8.1 or Windows Server 2012 R2 and for the user to be an administrator on the target machine. Using

restrictedadmin drastically increases the security and viability of using RDP to remotely administer machines.

4.3 Protecting LSASS

Current attacker tools, such as WCE, gsecdump, and Mimikatz, retrieve credentials from LSASS's memory via injecting themselves into the process or simply reading a process's memory. Windows 8.1 introduces a new security feature that allows the user to mark LSASS as a protected process. Protected processes enforce greater access control and limit the available interactions non-protected processes can have with a protected process. For example, process injection becomes much harder because only code signed by Microsoft can execute inside of a protected process. Also, protected processes disallow any non-protected process from reading its memory (even if the user is running as an administrator or system). This breaks current attacker tools.

Despite these great advances, the problem is still not solved. Without UEFI or SecureBoot, removing LSASS as a protected process is trivial. The only action required is changing a registry key and rebooting the system. Disabling LSASS as a protected process with UEFI or SecureBoot requires a bit more work and interactions with the BIOS, but still achievable. Administrators can detect the disabling of LSASS as a protected process by monitoring and auditing the Windows Event Log. If LSASS is a protected process, then each time the system is booted an event is triggered denoting this fact. Looking for the absence of this event could show that LSASS has been tampered with.

Before deploying this feature, Microsoft recommends that enterprises thoroughly test it within their environment to ensure that all software behaves as intended. For example, smartcards require their drivers to be signed by Microsoft in order to work.

4.4 Clearing credentials

Throughout this guidance a recurring theme is the presence of credentials in memory and the non-determinism concerning the length of time these credentials would stay present. In Windows 8.1, Microsoft set an upper bound of five minutes for the length of time that credentials will stay in memory after a logout. This shrinks the window an attacker will have to grab credentials after a user logs off. Note that credentials cached from executing commands using **runas** will stay in memory until the current user logs out. So, if a regular user elevates privileges to an administrator using **runas**, then the administrator credentials will be present in memory until the regular user logs off.

4.5 Protected Users group

The Protected Users group is a new domain global group (requires domain functional level of Windows Server 2012 R2) that provides standard non-configurable protections to all of its members. The following are several restrictions placed on members of the Protected Users group:

- NTLM is forbidden. Must use Kerberos or a third party SSP.
- Windows Digest (reversibly encrypted credentials) is not cached.
- Kerberos TGTs have a shortened lifetime (4 hours vs. 10 hours).

Putting high valued accounts in this group will help protect their credentials.

5 Conclusion

This document expands on the ideas presented by [1], discusses new PtH security features in the latest Windows operating systems, and provides additional guidance to DoD administrators, including scripts and walkthroughs, for implementing three PtH mitigations. To reduce the likelihood of adversaries easily spreading across a network, upgrading to the latest operating systems, and deploying at least one of the discussed PtH mitigations is recommended. Deploying multiple mitigations will further increase overall network security.

6 References

[1] Mitigating Pass-the-Hash (PtH) Attacks and Other Credential Theft Techniques. December 2012. http://www.microsoft.com/en-us/download/details.aspx?id=36036.

[2] Bugtraq mailing list. NT "Pass the Hash" with modified SMB client vulnerability. April 8, 1997. http://www.securityfocus.com/bind/233.

[3] Windows Credential Editor website. http://www.ampliasecurity.com/research.html.

[4] Mimikatz website. http://blog.gentilkiwi.com/mimikatz.

[5] Spotting the Adversary with Windows Event Log Monitoring. February 2013. http://www.nsa.gov/ia/_files/app/Spotting_the_Adversary_with_Windows_Event_Log_Monitoring.pdf

[6] Microsoft network port requirements website. http://support.microsoft.com/kb/832017

[7] gsecdump website. http://www.truesec.se/sakerhet/verktyg/saakerhet/gsecdump_v2.0b5

[8] Metasploit website. http://www.metasploit.com

[9] Pass the Hash and Other Credential Theft and Reuse: Mitigating the Risk of Lateral Movement and Privilege Escalation. Blackhat presentation August 2013. https://media.blackhat.com/us-13/us-13-Jungles-Pass-the-Hash-and-Other-Credential-Theft-and-Reuse-Mitigating-the-risk-of-Lateral-Movement-and-Privilege-Escalation.pdf

Appendix A: Creating unique local passwords

PowerShell scripts associated with this guide can be on found on the IAD GitHub site at https://github.com/iadgov

The **PthTools** library in the provided source contains a cmdlet **Edit-AllLocalAccountPasswords**, which is a Powershell script (to be run from a machine with version 3 or later) that changes the local account password for all specified machines. The length of the password is configurable by the **minPasswordLength** and **maxPasswordLength** parameters. The machine names to enumerate is defined by the **machinesFilePath** variable or, by default, all of the systems registered on the domain are used, except for domain controllers, which do not contain local accounts. If the **machinesFilePath** switch is used, then the program expects a file listing the hostnames with one hostname on each line.

Passwords are generated using the RNGCryptoServiceProvider .Net object. The local account (denoted by **localAccountName**) password is changed via the IADsUser interface. If for any reason the password change is unsuccessful, then the program will notify the administrator which machine names failed. Upon completion, the script will have written a tab delimited file to the path of **outFilePath** with each line consisting of: **machineName localAccountName newPassword** and a separate file containing a list of each machine, one on each line, which did not have the password changed successfully.

By default, the output is saved to a USB drive, but this can be disabled by setting **forceUSBKeyUsage** to **$FALSE**. Running this script with the least privileges possible is recommended. Use the Get-Help cmdlet in Powershell to get more information on the parameters.

Appendix B: Denying local administrators network access

Section 3.2 discusses how to restrict local administrators from remotely logging into a machine. The provided source contains two methods for doing this. The **Invoke-DenyNetworkAccess** cmdlet is the Powershell option (located in the **PtHTools** library) and DenyNetworkAccess.vbs is the VBScript option. Each method scans the local computer for any local administrator accounts. (Domain accounts with local admin privileges are ignored.) Next, it creates a local group with a well-known name, which is **DenyNetworkAccess** by default, and adds the local administrator accounts to the local group. The domain administrator manually adds the well-known group name to the domain policy, so that it is part of the **Deny access to this computer from the network** and **Deny log on through Remote Desktop Services** (for Windows Server 2008R2 and later) policies under **Computer Configuration** > **Windows Settings** > **Security Settings** > **Local Policies** > **User Rights Assignment**. The script should be deployed as a Group Policy Computer Startup or Shutdown script.

It is very important to note that the group name created by this script should not be added to the GPO policy for denying network access until the group has actually been created by the script. Otherwise, a Group Policy error will occur because the group does not exist on the system. Before configuring the policy, deploy the script as a startup script and force all systems to reboot before adding the group name to the policy.

In environments with a uniform set of local account names, the same result may be achieved through Group Policy Preferences (GPP) without a logon script. A local group can be created under **Computer Configuration** > **Preferences** > **Control Panel Settings** > **Local Users and Groups**. Create a new local

group with the **Create** action and specify the well-known group name. Create a new local group action and select the **Update** action. Add each local username to be added to the group. Each username **must** be a valid username on every system or the GPP will not be applied properly and an event will be written to the event log. The GPP solution does not scale well in environments with systems that contain many local administrator accounts. For a summary of all local accounts on the domain, including local administrator accounts, run the cmdlet **Get-LocalAccountSummaryOnDomain i**n Appendix E.

Appendix C: Configuring Windows Firewall rules

This section explains how to use Active Directory to deploy Windows Firewall rules to help mitigate PtH. By creating a set of firewall rules that disallow incoming connections to workstation computers on a Windows domain, attackers will not be able to spread laterally using PtH to workstations which may be less monitored and have a higher attack surface than servers.

The following guidance explains how to define and deploy Windows Firewall rules that block workstation to workstation communication. The domain controller must be running Windows 2008 R2 and later and the workstations must be running Windows Vista and later.

1. Create a new GPO called **pthGPO.**
2. Right-click the **pthGPO** GPO and select **Edit**. The Group Policy Management Editor dialog box will appear.
3. Navigate to **Computer Configuration > Policies > Windows Settings > Security Settings > Windows Firewall with Advanced Security > Windows Firewall with Advanced Security – LDAP://CN=xxxx**.
4. Right click on **Firewall with Advanced Security – LDAP://CN=xxxx** and click on **Properties**.
5. In this dialog box there are three profiles: domain, private, and public. For each profile, in the **State** section:
    a. Set the **Firewall state** to **On**, the **Inbound connections** to **Block (default)**, and **Outbound Connections** to **Not Configured**.
    b. In the **Settings** box, click the **Customize** button.
    c. In the **Rule merging** section, set the **Apply local firewall rules** drop down to **No**.
    d. In the **Rule merging** section, set the **Apply local connection security rules** drop down to **No**.
6. Click **OK** to finish the configuration.
7. Within the Group Policy Management Editor, navigate to **Computer Configuration > Policies > Windows Settings > Security Settings > Windows Firewall with Advanced Security > Windows Firewall with Advanced Security – LDAP://CN=xxxx**,right-click on **Inbound Rules**, and then click **New Rule**.
8. Select the **Custom** radio button and click **Next**.
9. Select the **All programs** radio button and click **Next** three times.
10. In the **Which remote IP addresses does this rule apply to?** section, select the **These IP address or subnet** radio button.
11. Click the **Add** button and specify the IP address of the system(s) allowed to make inbound connections to the client workstations. The values entered in Figure 1 are specifically tailored to the requirements of each network's infrastructure and services. [6] lists the port requirements for the common Microsoft services. If a service's port requirements are not listed, then see its documentation. In Figure 1, the IP address is the domain controller.

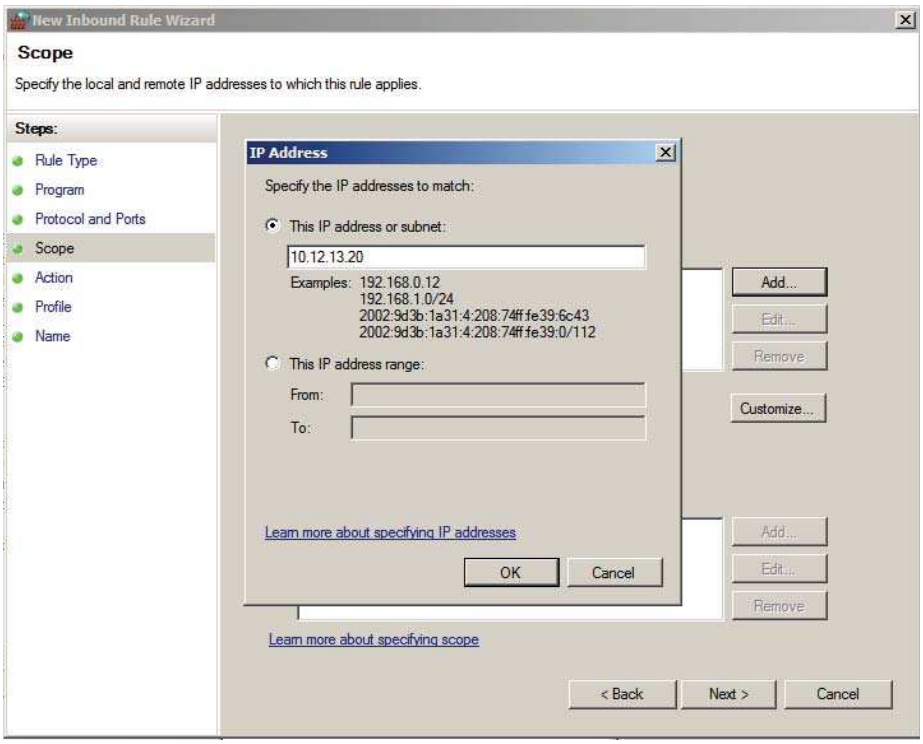

Figure 1: Scope dialog box.

12. Click **OK** and then **Next**.
13. Select the **Allow this connection** radio button and click **Next**.
14. Make sure **Domain**, **Private**, and **Public** boxes are all checked and click **Next**.
15. Click **Finish.**
16. Within the **Group Policy Management** console navigate to **Forest > Domains**, right-click the domain name, and select **New Organizational Unit**. A dialog box will appear.
17. Type in the name of the new OU. In this case, the new OU is named **pthOU**. Click **OK.**
18. Right-click **pthOU** and select **Link an Existing GPO**. The **Select GPO** dialog will appear.
19. Select **pthOU** from the list of GPOs and click **OK**.
20. In the **Security Filtering** section of the pthGPO shown in Figure 2, click the **Add** button.

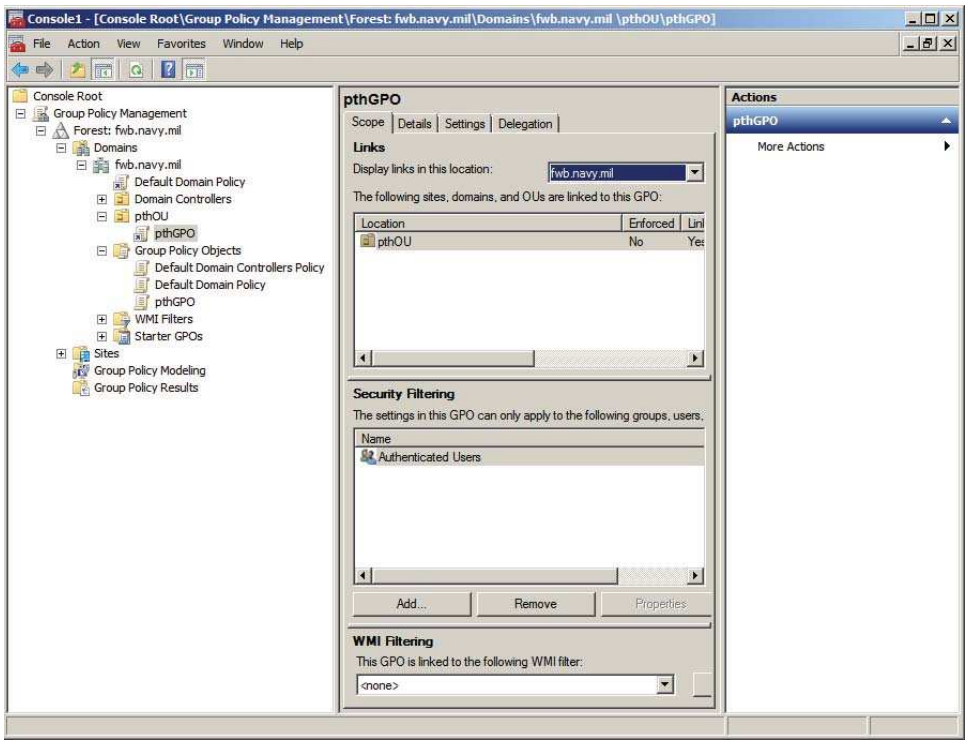

Figure 2: The new GPO linked to the PtH Organization Unit.

21. Click the **Object Types** button and make sure that the **Computer** option is selected as shown in Figure 3. Click **OK** when finished.

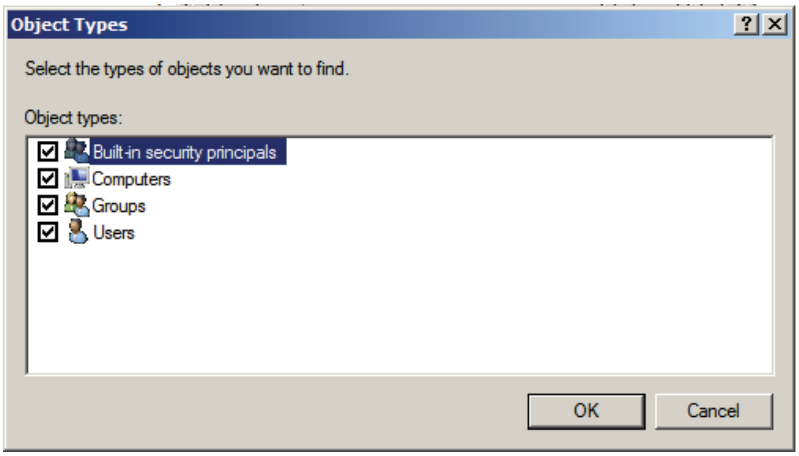

Figure 3: Object types dialog box settings.

22. In the **Enter the object name to select box,** type in the name of the system to be subject to the GPO. Click **Check Names** to verify the name. Click **OK**.
23. Repeat the two previous steps for each computer subject to the GPO.
24. Reboot each of the clients or issue the **gpupdate /force** command from each client.

Appendix D: Looking for possible PtH activity by examining Windows Event Logs

Windows is capable of logging a significant number of operating system interactions. From logon successes and failures to new user creation, Windows is capable of recording these events for auditing and future in-depth inspection, where necessary. For more information on setting up event collection and how to use event collection to spot potential PtH activity, see [5].

This appendix requires the user to have set the Local Security Policy **Security Settings > Local Policies > Audit Policy > Audit account logon events** policy to audit successes and failures.

Stopping PtH reduces to the problem of determining authorized uses of SSO from unauthorized ones. Under the existing SSO architecture, this is infeasible. However auditing local network logon events in conjunction with denying network logons for local accounts (Section 3.2) hardens the network's security posture and leaves a trail for later investigation. The cmdlet **Find-PotentialPtHEvents** located in the **PtHTools** library of the provided source analyzes each system on the network running Windows Vista and later and queries the Windows Event log for network logons using local administrator accounts. Since local administrator accounts have been explicitly denied access, any attempts to perform a network logon will likely be caused by user error, a misconfigured network service, or a PtH attempt. Regardless of the reason, network logons fitting this criteria are worth investigating and should occur rarely (making the auditing manageable).

Appendix E: Summary of Local Accounts

Keeping track of the local administrator accounts may be tough on large networks. The cmdlet **Get-LocalAccountSummaryOnDomain** in the provided source summarizes the local account names and local admin account names for each host on a domain. It can help the deployment of several of the mitigations listed in this guidance document.

Appendix F: Windows smartcard credentials

Traditionally, authentication is performed by asking the user for one or more of the following: something you know, something you have, or something you are. The most common authentication is still the password, but smartcards that require a pin are increasingly common in the enterprise and provide multifactor authentication. This section briefly discusses the differences between how Windows manages password credentials and smartcard credentials, how this affects SSO, and how this applies to PtH.

From the user's perspective, SSO should behave the same regardless of the authentication mechanism. The user should be able to input their credentials once and let the operating system continue to manage any new requests for credentials without prompting the user. This means that the operating system must cache the credentials for the smartcard just like it does for a password, and when credentials are

required, the cached credentials are used instead. When using NTLM, this means using hashes for authentication, which makes smartcard credentials susceptible to PtH, like password credentials.

The hash of smartcard credentials is independent of the pin (unlike passwords). Recall Section 3.1 discusses creating unique passwords to ensure hashes are unique. Users are typically forced to change their password on some regular interval causing the hash to also change. This is not the case with smartcards meaning that smartcard hashes have a very long lifetime. However, by toggling the SmartcardLogonRequired option in Active Directory (see below) associated with the user's account, the NTLM hash is changed. Periodically changing the smartcard hash will disrupt any active PtH attacks using a smartcard's credentials. Refreshing the hash will not generally impact users unless they are currently logged in. In which case, the user would have stale credentials and SSO will stop working (perhaps even causing an account lockout because of too many authentication failures).

```
Import-Module ActiveDirectory
$users = Get-ADUser -Filter {SmartcardLogonRequired -eq $true}
ForEach ($user in $users) {
        Set-ADUser $user -ChangePasswordAtLogon 1
        Set-ADUser $user -ChangePasswordAtLogon 0
        Set-ADUser $user –SmartcardLogonRequired 1
        Set-ADUser $user –SmartcardLogonRequired 1
        Write-Host "Reset password and hash for account $($user.name)"
}
```