Are there novel ways to mitigate credential theft attacks in Windows?

GIAC (GCIH) Gold Certification

Author: James Foster, j@jefjef.com
Advisor: Antonios Atlasis

Accepted: July 25, 2014

Abstract

Once a single system is compromised by a determined attacker in a Windows environment, the attacker often tries to move laterally through the environment and escalate his privileges, potentially resulting in compromise of additional systems, up to the entire domain or forest. A common way this happens is by the attacker stealing credentials from the compromised box and using them against other systems. Published guidance exists on how to mitigate risk from credential theft attacks, but many organizations find the known techniques difficult to implement, if they are aware of them at all. This paper first gives background on the issue and an overview of existing known mitigations in order to raise awareness. Then, it explores whether there are other, potentially more novel, ways to address this problem.

1. Introduction

In the IT world, the term "credentials" can usually include anything used to identify and authenticate a user or device. From a layman's perspective, this is most commonly a username and a password. An attacker who is able to obtain a valid set of credentials (like a username and its associated password) can simply use them to gain access to systems, unless other measures are in place to prevent it. For example, two-factor authentication prevents the successful use of a stolen password, assuming the attacker does not possess the second authentication factor. Users have been shown to be bad at choosing hard to guess passwords (SplashData, Inc., 2013), but even a conscientious user's strong password is at risk if the attacker can just steal it. While it may seem too simple for a skilled attacker to just login like an authorized user, there is little reason for an attacker to expend more effort than is necessary to achieve a goal. A burglar is not likely to climb in through a window when he can steal the front door key from under the welcome mat. An attacker using existing credentials is also often harder to detect than if he were exploiting the latest buffer overflow vulnerability. Many monitoring systems and most IT staffers are not tuned to notice seemingly legitimate user activity that might not be authorized.

The theft and subsequent use of stolen credentials is a significant problem in information security. The leading threat action in 2013 was the use of stolen credentials (Verizon Enterprise Solutions, 2014). Two steps in the Advanced Persistent Threat attack lifecycle, "escalate privileges" and "move laterally", involve the theft and subsequent use of stolen credentials (Mandiant Corporation, 2013). Although other platforms are not immune, Microsoft Windows and Active Directory are common places to steal credentials from and are common targets to use stolen credentials against. Windows is at increased risk for a few reasons. First, it is ubiquitous so significant research has been put into methods of attacking it. Second, Windows stored password hashes are functionally equivalent to passwords in many types of network-based authentication, reducing the need to crack stolen hashes before use -- called "pass-the-hash" attacks (Ashton, 1997). Third, it is often possible to recover passwords from memory on Windows systems (Delpy, 2012b). Contrast this with most modern Linux systems or web applications, where a stolen password hash is valuable only if it can be cracked, and passwords are not usually intentionally stored in memory.

In Windows, there are various types of credentials that can be stolen, such as password hashes, cached credentials, and passwords themselves (Microsoft Corporation, 2013). Depending on the credential being targeted, there are multiple methods to steal it. Various tools exist to enable easy theft, but they all use a finite list of underlying methods, such as LSASS injection, reading encrypted values from the registry, or decoding the Active Directory domain password database (NTDS.DIT). Once stolen, the credentials can be used in multiple ways – hashes and cached credentials can be cracked, hashes can be passed (depending on the protocol), and of course passwords can be used directly.

Advice is available from various sources, including Microsoft (Jungles, Margosis, Simos, Robinson, & Grimes, 2013), on how to reduce risk from credential theft attacks. This includes both reducing the attacker's ability to steal credentials and reducing the attacker's ability to use stolen credentials. Even so, professional penetration testers often find a lack of awareness of this problem among IT professionals, and a corresponding lack of implementation of existing mitigating controls (Duckwall & Campbell, 2014). Additional awareness around this topic, especially for information security staff and Windows administrators, is essential.

Aside from the existing "well-known" guidance on mitigating credential theft risk, there may be other, more novel ways of reducing an attacker's ability to steal credentials. Some of these may use simple built-in functionality of Windows (like making permission changes). Even if not perfectly effective, additional controls can slow attackers down and increase the amount of attacker effort required. In the cybercrime game, this can make attacks unprofitable. Also, some of these other controls can be used as canaries, alerting IT staff of possible malicious activity. If an attacker's normal steps are disrupted, it is more likely he will make a noticeable mistake.

2. Types of credentials to steal

In Microsoft Windows and Active Directory, various types of credentials exist that are of varying value to an attacker. A number of factors make up this value – how easy the credential is to steal, what can be done with it, and most importantly, the privilege level of the account the credential belongs to. Once some access to an environment is gained, like the compromise of a single user account in an Active Directory domain, attackers often target highly privileged accounts, like members of the Domain Admins group (Mandiant Corporation, 2013). If one of these accounts is compromised, the entire domain has fallen. Before discussing these topics, though, one should understand the various types of credentials that can be stolen in a Windows environment.

2.1. Password hashes for local accounts

Like the /etc/shadow file in many Unix/Linux systems, or a field in the "users" table in a web application's database, Windows stores a password hash for each account. For local accounts (as opposed to domain accounts), this hash must be present on the local system's disk, so that when a user attempts to login, the system can determine whether the user has entered the correct password. This is done by hashing the entered password and comparing the result to the stored value. If the values match, the user has presented the proper password and has been authenticated.

Windows uses either the LM or NT (sometimes also called NTLM) hash methods – LM is the weaker of the two, and has been usurped by the stronger NT method since Windows NT 4.0. However, all versions of Windows maintain some compatibility with LM hashes, and an LM hash is still seen in some environments alongside the NT hash even though the storage of LM hashes has been disabled by default since Windows Server 2008 and Vista (Microsoft Corporation, 2012a). Figure 1 shows lines from an fgdump (fizzgig, 2008) output file run on a Windows XP test system containing three local accounts. The first line of headers was added manually, for clarity. The "helpdesk" user has only the NT hash present, while the others have both the LM and NT hashes. The passwords set on each account are "administrator", "user", and "helpdeskhelpdesk", respectively.

```
Username:RID:LM Hash:NT Hash:::
Administrator:500:6A98EB0FB88A449CBE6FABFD825BCA61:A4141712F19E9DD5ADF16919BB38A95C:::
user:1003:22124EA690B83BFBAAD3B435B51404EE:57D583AA46D571502AAD4BB7AEA09C70:::
helpdesk:1004:NO PASSWORD*********************:9A953DD6EE75B0B00252E2B3EAEDB2EB:::
```

Figure 1 – Sample fgdump output with header line added for clarity

The LM and NT hashing methods do not use a salt or other type of per-user uniqueness, so the hash value for a particular password is always the same, regardless of the account that has that password. For example, the NT hash for the password "password" is always "8846F7EAEE8FB117AD06BDD830B7586C" regardless of account, and regardless of whether the account is a local account or domain account. Because of this, it is easy to determine which accounts are sharing the same password once hashes have been obtained. It should be noted that some tools represent LM and NT hash values using uppercase characters, while others use lowercase. There is no difference; hash values are not case sensitive.

2.2. Password hashes for domain accounts

Password hashes for domain accounts are just like password hashes for local accounts, except that they are stored on an Active Directory domain controller rather than on a local system. They still consist of an NT hash and optionally an LM hash, depending on configuration. When a domain account is used to login to a domain member system that is on the network, a domain controller is contacted in order to authenticate the user.

2.3. Cached credentials (cached password verifiers)

Consider the scenario of a domain member laptop that belongs to a normal user with an account in the domain. When connected to the corporate network, the user logs in with their domain account, and the laptop communicates with the domain controller to decide whether the user should be authenticated. This is necessary because the LM and NT hashes for a domain account are not stored on the laptop – they are stored on the domain controller, as previously mentioned. However, there are times when the laptop is not on the corporate network (or on any network at all), and the user still wants to login. Allowing domain accounts to logon to off-network systems is the reason for cached credentials. It should be noted that although the term "cached credential" is commonly used, Microsoft uses the term “Windows logon cached password verifier” (Microsoft Corporation, 2013) in modern documentation.

A cached credential is a salted and hashed version of the account’s NT hash. The salting and hashing protects the NT hash better than if it was stored in clear text, yet allows the system to easily verify if a user has entered the correct password. The cached credential approach was taken rather than just storing a copy of the domain account’s raw NT hash in order to better protect the domain account in the event the domain member system is stolen or otherwise compromised. Typically, a limited number of users’ cached credentials are stored on a domain member – by default, most versions of Windows cache only the last ten successful logons (Microsoft Corporation, 2011b).

Two versions of cached credentials exist, known in the password cracking world as “MSCash” (supernothing, 2010) and “MSCash2” (JimF, 2011). The former was the original implementation, while the latter was introduced with Windows Vista. MSCash2 was created in order to make cracking attacks against stolen cached credentials much harder. Figure 2 shows an example cached credential, in MSCash format, as output by cachedump (Pilon, Marechal, & Devine, 2005) from a Windows Server 2003 system. This is for the domain Administrator account in a domain with short name DOMAIN and long name domain.example.local.

```
Administrator:9923F95267970CD3A4C65E06C5A5A5D9:DOMAIN:domain.example.local
```

Figure 2 – Example cached credential from Windows Server 2003 system

2.4. Access tokens

Access tokens are not really credentials themselves, but they do contain credential material, and stealing or abusing these contents is often referred to as “token theft”. A simplified way of looking at access tokens is that each account logged in to a Windows system has one in memory. Each access token contains the account’s LM and NT hashes, the account’s password, and other elements outside the scope of this paper. It should be noted that sensitive elements like hashes and passwords are encrypted in memory, but the encryption key used is also stored in memory. Modern tools that steal hashes and passwords from access tokens retrieve this key and decrypt the hashes and password automatically. Access tokens exist in memory only, they are never written to disk. When access tokens are created in memory, they persist either until the associated account has logged out or until the next system reboot.

Looking in a bit more detail, access tokens are only created in memory for certain types of logons (Jungles, Margosis, Simos, Robinson, & Grimes, 2013). The most common types of logons that create access tokens are those at the physical console, most logons via Remote Desktop - except for Restricted Admin mode (Russinovich & Ide, 2014), RunAs, scheduled tasks, and remote logons where the user's identity must be impersonated by the server (often seen on servers running Outlook Web App and Sharepoint, for example). Common types of logons that do not create access tokens are mapping network drives, remote management with Microsoft Management Console (MMC) snap-ins, remote registry, and Remote Desktop Restricted Admin mode (Russinovich & Ide, 2014).

It has been known for some time that access tokens contain the account's LM and/or NT hashes. Tools were publicly available to obtain hashes from access tokens in at least 2007 (Ochoa, 2007). It was not until 2011, however, that widespread knowledge became available about the ability to recover the account's clear text password from access tokens (Delpy, 2012b). The first widely known tool to accomplish this was mimikatz (Delpy, 2012a).

3. Methods to steal credentials

Of course, all of these credential types are intended to remain secret, and Windows has measures in place to protect them. Nonetheless, researchers and attackers have come up with various methods to steal them. The focus of this section is on some of the underlying methods used in their theft, rather than on the various tools used. This is because mitigation techniques are best applied to a limited set of underlying methods rather than to a potentially unlimited number of tools using those methods. Unless noted otherwise, administrative rights are required to the target computer in order to perform any of these credential theft methods. This is not intended to be an exhaustive review of all such methods, but rather a focus on those that are most popular.

3.1. Injecting into LSASS

The Local Security Authority Subsystem Service (LSASS) is the service in Windows that manages authentication and security. Among other things, it performs password hashing and handles access tokens (Microsoft Corporation, n.d.d.). Only certain special processes, like LSASS, have the privileges to call Windows functions for retrieving credential data. Therefore, some tools inject their code into the existing LSASS process, so the evil code is able to call the necessary functions. For example, this is the approach taken by pwdump6 (also used in fgdump) to obtain local and domain account LM/NT hashes, according to its README file (fizzgig, 2009). This is also one of the methods used by Windows Credential Editor (WCE) to steal credentials from in-memory access tokens (Ochoa, 2011). A common method of code injection relies on the CreateRemoteThread function call, which in turn requires enabling the SeDebugPrivilege. One drawback of this method is that it can sometimes cause instability on the target. If the attack tool crashes or has a bug, it can affect the entire LSASS process, sometimes crashing the system. The risk of problems is not insignificant, due largely to the fact that the functions being called are not publicly documented and some guesswork was required in writing the code to exploit them.

3.2. Reading LSASS's memory (live or in a RAM dump)

The main reason many tools took the approach of injecting into the LSASS process was that credential related memory structures are not publicly documented. Given the choice between parsing undocumented memory and calling undocumented functions, authors of credential theft tools initially took the latter approach, which requires process injection. Over time, however, as more research was done on LSASS's memory structures, it has become possible for a non-LSASS process to read credentials directly from LSASS's memory area. This requires only the ability to read this memory, and does not rely on process injection. This is the newer "safe mode" method supported by WCE since version 1.1 (Ochoa, 2011).

A similar idea applies to recovering credentials from memory dumps. Since more is now known about LSASS's internal memory structures, an offline memory dump file can be parsed like memory on an active system. Windows Task Manager can make dump files in Vista and later, or the Sysinternals tool ProcDump (Russinovich, 2014) can be used on XP and later. Once obtained, the memory dump can be moved to an attacker's system and credentials recovered from it there. For example, recovering credentials from a memory dump file is supported in mimikatz version 2 alpha (Delpy, 2013).

3.3. Reading registry hives

LM and NT password hashes for local accounts are stored in the Security Accounts Manager (SAM) database file, exposed as the part of the registry at HKEY_LOCAL_MACHINE\SAM. They are encrypted with a key (often called the "bootkey" or "syskey") which is stored in the System portion of the registry, under HKEY_LOCAL_MACHINE\SYSTEM. There are various ways of getting access to these files on a live system, such as making a shadow copy using the Volume Shadow Copy Service (VSS). A more straightforward method, however, is using the built-in command line reg.exe tool to export their contents to new files and then moving the new files to an attacker's system to extract the credentials. A simple batch file like the one shown in Figure 3 can do this remotely if the attacker is already authenticated to the target (note that the file path of c:\ passed to reg.exe refers to the target's local c: drive, not the attacker's system). Once the files are obtained, a tool like samdump2 version 3.0.0 (Cuomo & Tissieres, 2012) can extract the syskey from the SYSTEM file and use it to extract and decrypt the LM and NT hashes from the SAM file.

```
net use z: \\target.example.local\c$
reg.exe save \\target.example.local\HKLM\SAM c:\SAM
reg.exe save \\target.example.local\HKLM\SYSTEM c:\SYSTEM
move z:\SAM .
move z:\SYSTEM .
net use z: /d
```

Figure 3 - Batch file to export SAM and SYSTEM registry hives using built-in commands

Cached credentials are also exposed in the registry (under HKEY_LOCAL_MACHINE\SECURITY\CACHE), so a similar method of reading their values (and the necessary keys to decrypt them) from the registry is possible. This is the approach the creddump tool (Dolan-Gavitt, 2012) uses.

3.4. Decoding NTDS.DIT

LM and NT hashes for Active Directory domain accounts are stored in the Active Directory database file, NTDS.DIT, an Extensible Storage Engine (ESE) database. This file resides on all domain controllers. The file cannot simply be opened or copied, since it is an active database and is always in use. To get around this restriction, a shadow copy of the drive can be made with the Volume Shadow Copy Service (VSS) and the file can be obtained from the shadow copy (haxrbyte, 2013). Since VSS is built-in legitimate functionality, stealing a copy of NTDS.DIT in this way is very low risk for an attacker. Note that Read-Only Domain Controllers (RODCs) do not locally store password hashes in their NTDS.DIT database by default (Microsoft Corporation, 2011a), so they are not good targets for attackers.

Once the file has been obtained and moved to the attacker's system, it can be parsed to retrieve the LM and NT hashes for all domain accounts. One way of doing this is to use esedbexport (Metz, 2013) to export tables from the NTDS.DIT database file, dsusers.py from the NTDSXtract framework (Barta, 2011) to extract and decrypt users' hashes, and ntdstopwdump.py (Damele, 2011) to convert the hashes to pwdump format (the most commonly used format for LM and NT hashes). A post exploitation module is also available for Metasploit that automates the shadow copy process (Rapid7, 2014).

4. What to do with stolen credentials

Once credentials are stolen, attackers will endeavor to use them. How they can be used depends on the type of credential and the attack surface or protocols available to the attacker. Also important is the level of privilege granted to the account the credential belongs to. Following are some of the main things that attackers do with credentials once they have been stolen.

4.1. Crack password hashes

Once LM and/or NT password hashes have been obtained, an attacker can attempt to determine the passwords that they represent. Because hashing functions are designed to be one-way, the hashes cannot simply be "reversed" or decrypted. Instead, one of two approaches is taken. The first approach is basically looking up the hash in large, precomputed tables which require little computational power to use. This is often referred to as a rainbow table style attack (Kuliukas, n.d.). Rainbow table style attacks are generally preferred for LM hashes because of LM's limited key space (a maximum of seven characters for each half of the LM hash) which makes the sizes of LM rainbow tables manageable.

The second approach to determining passwords from hashes is what most people think of as traditional "cracking". This is where a candidate password is selected, its hash is generated, and the candidate password's hash is compared to the stolen hash. If they match, the stolen hash's original password is the candidate password. If not, another candidate password is selected and the process repeats. This method requires a lot of computational power but little storage space, and it is the most common method used against NT hashes today. A modern purpose-built graphics processor (GPU) based cracking system with eight graphics cards can achieve over 174 billion NT hash operations per second (Steube, 2014). At this speed, a brute-force attack can cover all eight character (printable US-ASCII) NT passwords (sondre, 2012) in less than 11 hours. Far more efficient attacks can be constructed using good masks, wordlists with rules, and other ways of generating the candidate passwords more intelligently. For example, knowing that users often choose passwords based on dictionary words, with the first letter capitalized, followed by one or two digits, is a powerful way to increase cracking success and efficiency.

4.2. Pass password hashes

In many computer systems, stolen stored password hashes have value to an attacker for only one thing – cracking. In Windows, however, stolen LM and/or NT password hashes can themselves be used to authenticate to some types of network services (Ashton, 1997). This means that an attacker who has stolen LM and/or NT hashes can simply use them directly to connect to a remote target, rather than having to crack them first. This use of stolen hashes is often referred to as "passing the hash".

In Windows, the LM/NTLM suite of network authentication protocols (Glass, 2006) handle network authentication for many types of communications. While these authentication protocols employ the LM and/or NT password hashes, the network authentication protocols and the password hashing methods are different things, and the similar naming can be confusing. Many types of communications with Windows systems use LM/NTLM authentication. These include file/print sharing and other protocols that rely on SMB, as well as non-SMB protocols that have LM/NTLM authentication support bolted on (HTTP, SMTP, etc.) A notable exception is Remote Desktop, which does not use LM/NTLM authentication; although an exception to this exception is Remote Desktop's new Restricted Admin mode, introduced in 2013 (Be'ery, 2014).

Like many other challenge-response authentication protocols that operate in clear text (without a lower layer of encryption like TLS), the client must be able to prove knowledge of the password (or authenticator) without actually passing it over the network, and the server must be able to verify the client's claim based on authentication data that it has stored. It is important to note that in Windows, the "server" side usually does not possess the user's password, but rather only their LM and/or NT hash. Given these restrictions, the first step of the "client" side of the LM/NTLM authentication protocols is to turn the user's password into its LM and/or NT hash before carrying on with authentication (Glass, 2006). The hash, not the password, is what the client must prove to the server that it has, without actually sending it to the server. This means that an attacker already possessing a stolen LM and/or NT hash can just skip the step where the hash is generated, and carry on with authentication normally. Of course, special tools with modified code have to be used, but they are widely available.

The result of all this is that any protocol that relies on LM/NTLM authentication is susceptible to pass the hash style attacks. An attacker can use non-Windows tools that support pass the hash such as medusa (JoMo-Kun, 2012) or modified winexe (JoMo-Kun, 2003) if they are attacking from a *nix system. If attacking from a Windows system, a tool like WCE (Ochoa, 2014) allows the attacker to "insert" stolen LM and NT hashes into their current logon session, essentially becoming the compromised account. In this way, any normal Windows action is performed with the compromised account's identity, including actions taken across the network (provided they support the LM/NTLM authentication protocols).

4.3. Crack cached credentials

Stolen cached credentials have value to an attacker only as something to attempt to be cracked. However, cracking cached credentials is very slow. MSCash (supernothing, 2010), the algorithm used to protect cached credentials in Windows versions before Vista, cracks about 100 times slower than NT hashes on the author's cracking system. MSCash2 (JimF, 2011), used in Windows Vista and later, cracks about 40,000 times slower than NT hashes on the same system. In most cases, cached credentials are rarely cracked in a reasonable time unless a very weak password was used.

4.4. Abuse passwords

Oftentimes, the easiest way to compromise systems is to learn passwords of legitimate accounts and just login with them. Not only can the password be used against the system it is known to work on, but login attempts can be made with this password to other systems. Oftentimes, users use the same password across different systems (Danchev, 2011); compromising it on one may allow an attacker to use it against another.

Stolen passwords can also be tried against other accounts. This is especially useful to an attacker when the other accounts are of a higher privilege level or are on systems which have not yet been breached. It is not unusual to find the same password set on various accounts in an organization – especially across personal and generic accounts belonging to IT staff. If an IT person's user account password is obtained, it can be tried against privileged accounts (like domain admins). If a domain admin's password is obtained for one domain, it can be tried against domain admins of other domains in the organization, or even against non-Windows systems which might be administered by the same person. These techniques are surprisingly effective during security assessments at turning a small compromise into a large one.

5. Well known mitigation techniques

A number of techniques to mitigate the risk of credential theft attacks have been documented (Jungles, Margosis, Simos, Robinson, & Grimes, 2013). These methods tend to be focused on reducing the instances of credentials to steal, reducing the ability for attackers to gain administrative rights, reducing the ability for attackers to utilize stolen credentials, or some combination of the three. It is important to understand some of these "well known" techniques, as they are still not consistently practiced by many organizations (Duckwall & Campbell, 2014).

The first method is to minimize administrative rights. This means to limit the number of users with administrative rights to clients and servers, limit the number of other local administrative accounts, as well as limit the number of accounts with domain admin (or similarly privileged) access. This reduces the number of accounts that can be used by attackers to steal credentials from systems across the organization, and reduces the number of highly privileged accounts that attackers often target. Not only should the number of accounts be reduced, but accounts granted increased privileges should be given only the minimum necessary.

The next method is to restrict local accounts on clients from being able to access the client over the network. For example, if a local Administrator or Helpdesk account must exist on desktops, it should be limited so that it can be used from the console only, for recovery purposes. This can be done by adding the account(s) to the Local Security Policy settings "Deny access to this computer from the network" (Microsoft Corporation, 2005a) and "Deny log on through Remote Desktop Services" (or "Deny log on through Terminal Services" depending on the version of Windows) (Microsoft Corporation, 2005b) under Local Policies, User Rights Assignment. Network access for support purposes should be performed using a domain account.

The third method is to minimize the number of shared passwords across multiple systems and accounts. In the event that a local administrative account must exist across many systems and allow access over the network, the password for the account on each system should be different. For systems of differing trust levels or accounts across privilege boundaries, passwords should never be the same.

Another important control is separation of privilege. This means that privileged accounts (such as domain admins) should be used only when necessary; regular activities such as reading email to writing reports should be done with a regular user account. Also, when privileged accounts are used, they should be used only on more secure systems. A highly privileged account should not be used on a less trusted system (like an end-user's laptop). Systems used to administer privileged servers (like IT staff's workstations) need to be treated as sensitively as the servers themselves.

The fifth technique is to minimize the creation of access tokens, especially for privileged accounts, since they are attractive targets for credential theft. An access token that does not exist in memory cannot be stolen. Since access tokens are created only by certain types of logons, administrators are encouraged wherever possible to use the types of logons that do not create access tokens.

Next, privileged accounts should not be allowed to be delegated, by setting the "Account is sensitive and cannot be delegated" option on the account. This helps to prevent attackers from using these accounts on one system to gain increased privileges to other systems. For example, if a domain admin logs on to a compromised server that is normally trusted for delegation, this setting can help prevent the access token from being used directly against other systems (Pilkington, 2012).

An additional important control against credential theft attacks is network isolation. This can be as simple as setting the Windows firewall on clients to deny incoming connections (except from known management servers, potentially). It could be as advanced as putting all clients in private VLANs (Lapukhov, 2008) and requiring that traffic pass through an upstream security device configured with explicit ACLs, permitting only allowed communications. In any case, the idea is to limit the number of systems an attacker can access to steal credentials from, and if credentials have already been stolen, to limit the number of systems an attacker can use them against.

Lastly, the ideal control is not allowing systems to get compromised in the first place, and not allowing untrusted users to gain local administrative rights to any system. This involves all of the usual security measures to keep Windows systems well secured – good patching, strong passphrases, etc. Of course, this is easier said than done, which is why all of the other controls are necessary.

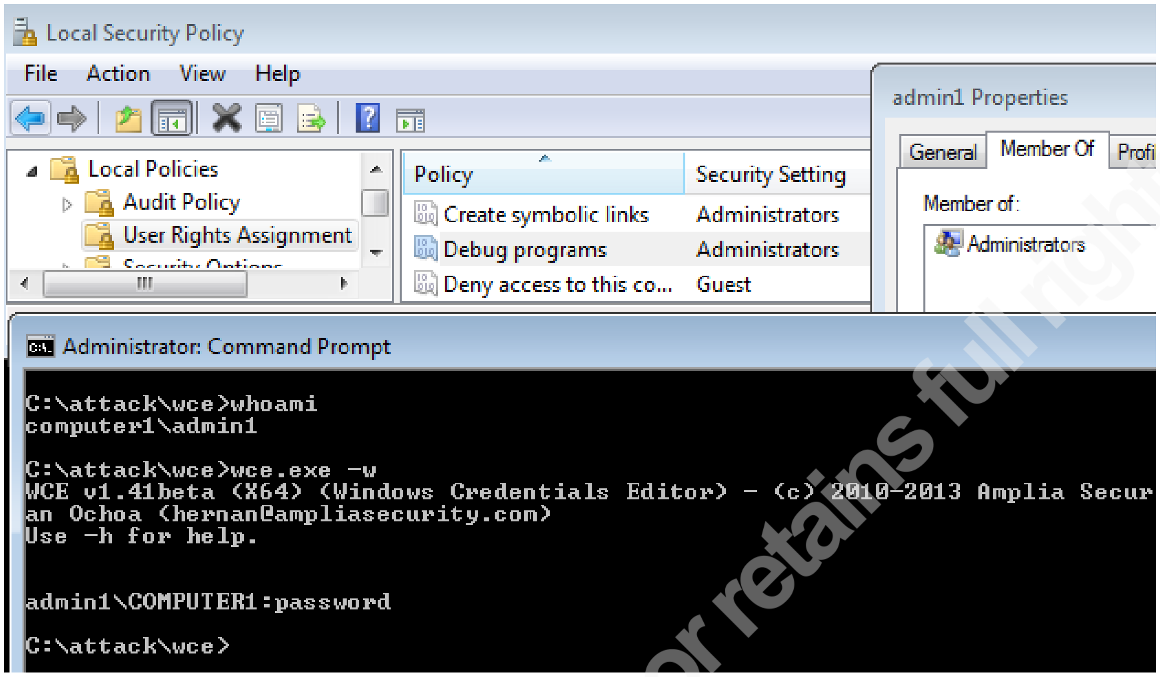
6. Novel mitigation techniques

Since local administrative access is required by all of the credential theft methods discussed so far, all of the techniques discussed in this section assume that the attacker has already gained local administrative access. In other words, these are potential methods to stop local administrators from stealing credentials. As the attacker has local administrative access, however, it is very difficult or impossible to stop all credential theft. After all, an attacker will often have sufficient rights to un-do these settings. The intent of investigating different techniques is to make attacks more difficult (slower and more expensive) and cause attackers to make mistakes which can be detected. An attacker is not likely to know that an unusual mitigation is in place until it stops them at least once, which should be sufficient to alert IT staff that something is amiss. Of course, this assumes that the appropriate log events are handled in a way that would generate an alert, for example if they are sent to a specifically configured SIEM. For this reason, the detective qualities of these controls are just as (if not more) important than their preventive qualities.

6.1. Remove debug privilege

The SeDebugPrivilege is required in order to inject code into the LSASS process (Microsoft Corporation, 2008b), so it seems logical that removing this privilege would stop this method of credential theft. To test this idea, a Windows 7 test system was setup. Under Local Security Policy, Local Policies, User Rights Assignment, there is a right called “Debug programs”. By default, this right is assigned only to the local Administrators group. It was suggested by Microsoft many years ago that to meet the Windows 2000 Security Target, this right be removed from everyone, including administrators (Microsoft Corporation, n.d.a), which supported the idea that this might be a good mitiation. For the test, the Administrators group was removed, and the WCE tool executed. As shown in Figure 4 and Figure 5, this setting change made no difference – WCE worked either way.

James Foster, j@jefjef.com



Policy	Security Setting
Create symbolic links	Administrators
Debug programs	Administrators
Deny access to this co...	Guest

Administrator: Command Prompt

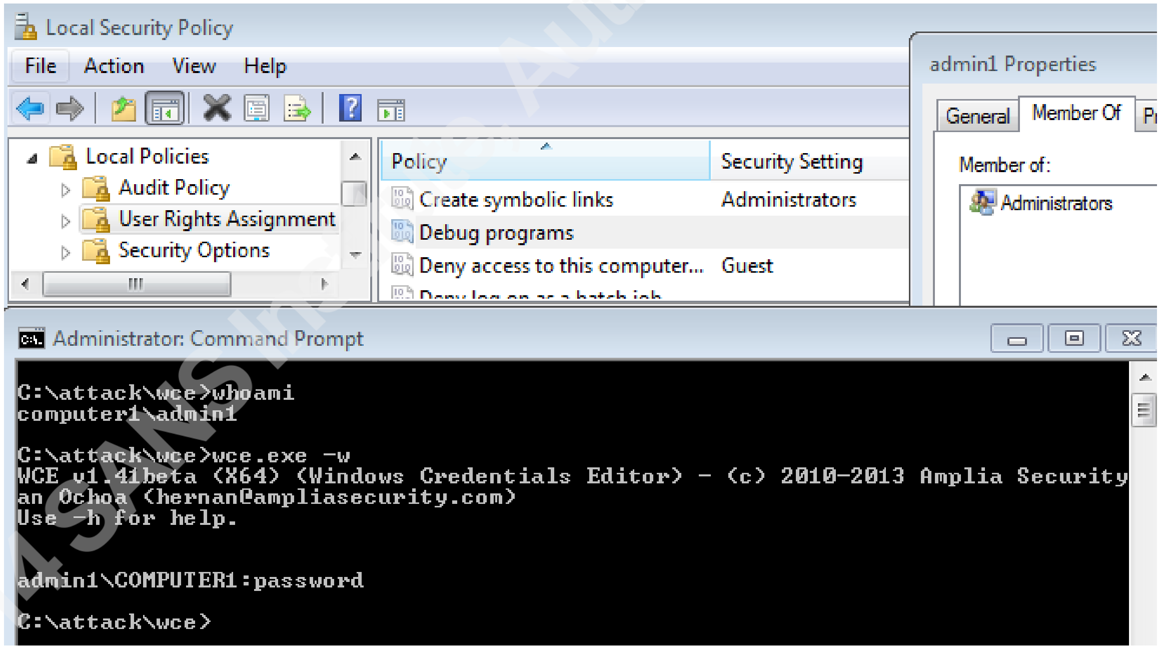
```
C:\attack\wce>whoami
computer1\admin1

C:\attack\wce>wce.exe -w
WCE v1.41beta (X64) (Windows Credentials Editor) - (c) 2010-2013 Amplia Secur
an Ochoa (hernan@ampliasecurity.com)
Use -h for help.


admin1\COMPUTER1:password

C:\attack\wce>
```

Figure 4 - WCE working as expected with default Debug programs setting



Policy	Security Setting
Create symbolic links	Administrators
Debug programs	
Deny access to this computer...	Guest
Deny log on as a batch job	

Administrator: Command Prompt

```
C:\attack\wce>whoami
computer1\admin1

C:\attack\wce>wce.exe -w
WCE v1.41beta (X64) (Windows Credentials Editor) - (c) 2010-2013 Amplia Security
an Ochoa (hernan@ampliasecurity.com)
Use -h for help.


admin1\COMPUTER1:password

C:\attack\wce>
```

Figure 5 - WCE worked even with the user not possessing Debug programs right

This somewhat surprising result shows that removing the SeDebugPrivilege did not stop the WCE tool. The same test was performed with pwdump6, with the same result. Source code is not publicly available for WCE, so the source code for pwdump6 (fizzgig, 2009) was consulted, and the answer uncovered with some help from fizzgig (the author of pwdump6). PwDump6.cpp, which is the main code that first runs, does not itself request the SeDebugPrivilege to inject into LSASS. Rather, it extracts a separate executable file from itself, writes it to the target system, and then starts that separate executable file as a service. The code in that running service is what actually requests the SeDebugPrivilege and injects itself into LSASS (this can be seen in the file pwservice.cpp). It is able to do this because, as a service, it is running as the local SYSTEM account. Using Process Monitor (Russinovich & Cogswell, 2014) to watch the execution of WCE revealed a similar operating method – a new service was created in order to perform the credential theft. This shows that simply removing the "Debug programs" right from an administrative account does not stop these common credential theft tools (at least as long as the account retains the right to create new services). Note that this control would likely be effective against tools that do not create a new service in which to run their code.

6.2. Remove right to create new services

Since the ability to create a new service and have it run as the local SYSTEM account seems to be one way to get around account-based restrictions, it was next considered whether it would be possible to remove this right. Of course, this may not be realistic in real-world environments, or at least would have to be implemented carefully (possibly where only a more limited set of accounts retained this right). The interface used to manage services is called the Service Control Manager (SCM, or scmanager) (Microsoft Corporation, 2010b). Changing permissions on the SCM object controls permissions for actions like creating new services or enumerating what services are installed, in a similar way that changing permissions to a particular service alters the ability to control that individual service. It should be noted that only since Windows Server 2003 Service Pack 1 has it been possible to change the security descriptor on the SCM (Microsoft Corporation, 2012b). The permission needed to call CreateService (which is what pwdump6 does) is SC_MANAGER_CREATE_SERVICE, which local Administrators have by default (Microsoft Corporation, 2012b).

On the Windows 7 test system, the command "sc sdshow "scmanager"" was run to view the current access control list on the SCM object. The output of this command, which is in Security Descriptor Definition Language (SDDL) (Microsoft Corporation, 2010a), is shown in Figure 6 with the portion of focus in bold. The sddlparse tool (Microsoft Corporation, 2008a) was helpful to parse this output into a more readable format, which is listed in Appendix A – Output of sddlparse.exe on page 29. A new SDDL line was constructed which keeps all of the permissions the same, except with the "DC" code (which aligns with the SC_MANAGER_CREATE_SERVICE right) omitted. The command used to apply this new permissions set to the Windows 7 test system is shown in Figure 7, with the focus area that was changed in bold. While the details of the manipulation of SCM permissions are outside the scope of this paper, it is reasonable to summarize by saying that this command removed the ability for administrators to create new services.

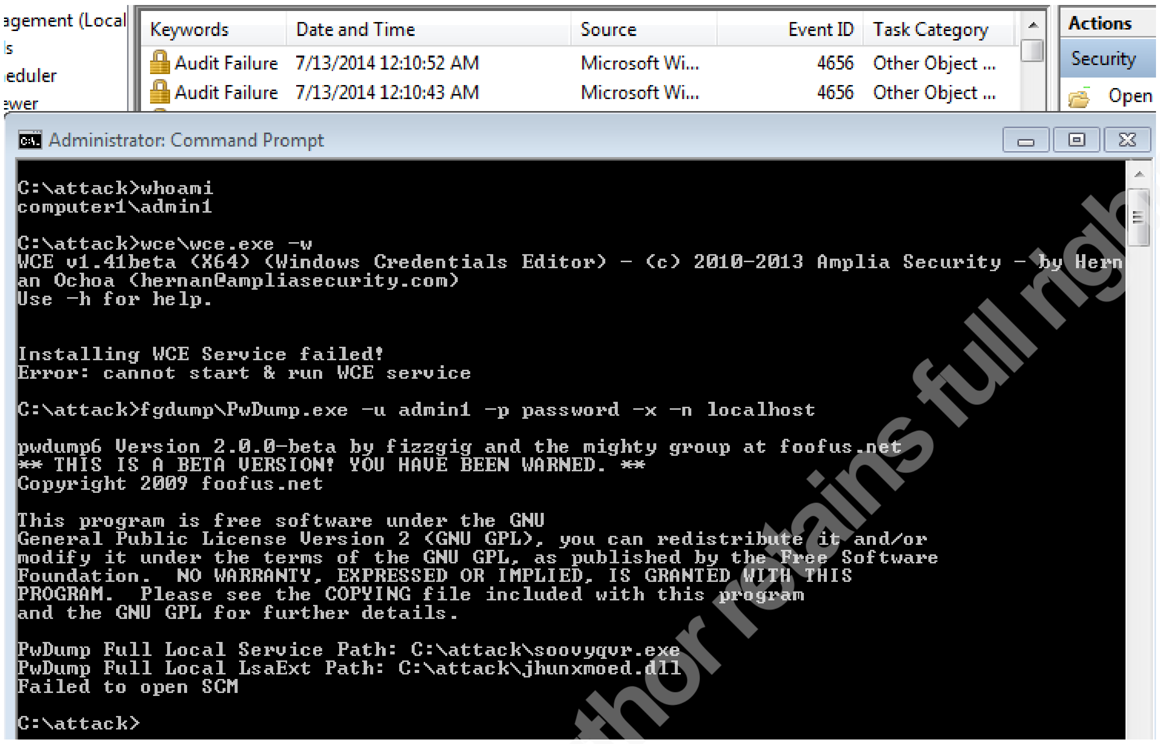
```
D:(A;;CC;;;AU)(A;;CCLCRPRC;;;IU)(A;;CCLCRPRC;;;SU)(A;;CCLCRPWPRC;;;SY)
(A;;KA;;;BA)S:(AU;FA;KA;;;WD)(AU;OIIOFA;GA;;;WD)
```

Figure 6 - Output of "sc sdshow "scmanager"" showing original SCM permissions

```
sc sdset "sdmanager"
D:(A;;CC;;;AU)(A;;CCLCRPRC;;;IU)(A;;CCLCRPRC;;;SU)(A;;CCLCRPWPRC;;;SY)
(A;;CCLCSWRPWPDTLOCRRC;;;BA)S:(AU;FA;KA;;;WD)(AU;OIIOFA;GA;;;WD)
```

Figure 7 – Command used to apply new SCM permissions to test system

During testing, the "Audit object access" policy in Local Security Policy, Local Policies, Audit Policy was set to log failures. The idea was to see log evidence of this permission change in action, and to determine whether an actionable event would be generated that could indicate when an attacker was trying to create an unauthorized service. Indeed, when both WCE and pwdump6 were tested after these changes, they were no longer able to steal credentials. Preventing the creation of new services successfully stopped these tools, and generated failure audit events as shown in Figure 8. The detail of one of these events is shown in Appendix B – Security event log detail on page 30. Unfortunately, the permission change also prevented the ability to undo the change, and an alternate group or account had not been assigned the rights to do so, so the test system had to be restored to a previous snapshot. Possibly there is a better way to implement a control like this in a more limited or thoughtful way.



Keywords	Date and Time	Source	Event ID	Task Category
Audit Failure	7/13/2014 12:10:52 AM	Microsoft Wi...	4656	Other Object ...
Audit Failure	7/13/2014 12:10:43 AM	Microsoft Wi...	4656	Other Object ...

```
C:\attack>whoami
computer1\admin1

C:\attack>wce\wce.exe -w
WCE v1.41beta (X64) (Windows Credentials Editor) - (c) 2010-2013 Amplia Security - by Hern
an Ochoa (hernan@ampliasecurity.com)
Use -h for help.


Installing WCE Service failed!
Error: cannot start & run WCE service

C:\attack>fgdump\PwDump.exe -u admin1 -p password -x -n localhost

pwdump6 Version 2.0.0-beta by fizzgig and the mighty group at foofus.net
** THIS IS A BETA VERSION! YOU HAVE BEEN WARNED. **
Copyright 2009 foofus.net

This program is free software under the GNU
General Public License Version 2 (GNU GPL), you can redistribute it and/or
modify it under the terms of the GNU GPL, as published by the Free Software
Foundation.  NO WARRANTY, EXPRESSED OR IMPLIED, IS GRANTED WITH THIS
PROGRAM.  Please see the COPYING file included with this program
and the GNU GPL for further details.

PwDump Full Local Service Path: C:\attack\soovyqvr.exe
PwDump Full Local LsaExt Path: C:\attack\jhunxmoed.dll
Failed to open SCM

C:\attack>
```

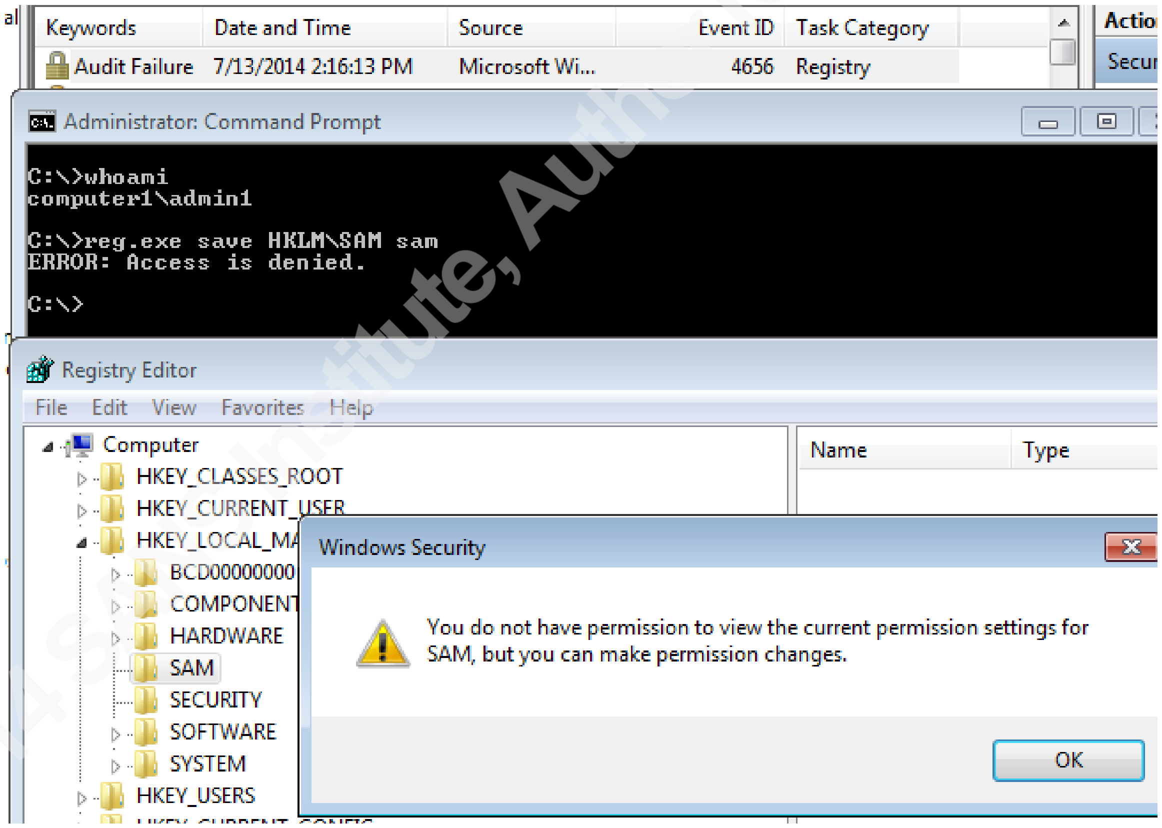
Figure 8 – Removing right to create new services stopped these tools and generated log events

6.3. Restrict registry permissions

Depending on the type of credentials being targeted, various areas in the registry need to be read. For LM and NT hashes of local accounts, data from HKLM\SYSTEM and HKLM\SAM needs to be read. As shown in Figure 3 on page 9, one way to do this is to use the built-in reg.exe utility to save the contents of these keys to files and then extract the hashes offline from those files. Removing access to the HKLM\SAM hive seemed a possible way to stop this attack. On the Windows 7 test system, the command “reg.exe save HKLM\SAM sam” was executed to first verify that the attack was possible by default, and a valid copy of this hive was indeed saved to the “sam” file.

Looking at the default permissions, both the Users and Administrators groups were granted some rights to HKLM\SAM, as well as the CREATOR OWNER. Since the Administrators group was the Owner, removing these other rights would not be sufficient. Therefore, a second local administrative account was created, called “admin2”, and ownership of HKLM\SAM was given to “admin2”. All rights for Users and Administrators were then removed. At this point, the “admin1” account did not have the necessary rights to read values from his hive, even though it was a member of the

Administrators group. In order to be able to detect attacker attempts to read these keys, auditing entries were also created on HKLM\SAM to audit all access failures made by Everyone. Figure 9 shows an attempt as "admin1" to execute the previous reg.exe command; access was denied, and an audit entry was made in the security log (the detailed log entry is shown in Appendix C – Security event log detail (HKLM\SAM) on page 32). Also note the registry editor showing that no access to the hive is available, even though the account still has sufficient rights to change the permissions. So again, an attacker could get around this protection by changing the permissions, but it is likely that they would have been slowed down and IT staff could have detected the activity. Lastly, a few minutes of cursory testing revealed no negative effects of this permission change.


```
Keywords        Date and Time          Source         Event ID   Task Category
Audit Failure   7/13/2014 2:16:13 PM   Microsoft Wi...    4656   Registry
```

```
C:\>whoami
computer1\admin1

C:\>reg.exe save HKLM\SAM sam
ERROR: Access is denied.

C:\>
```

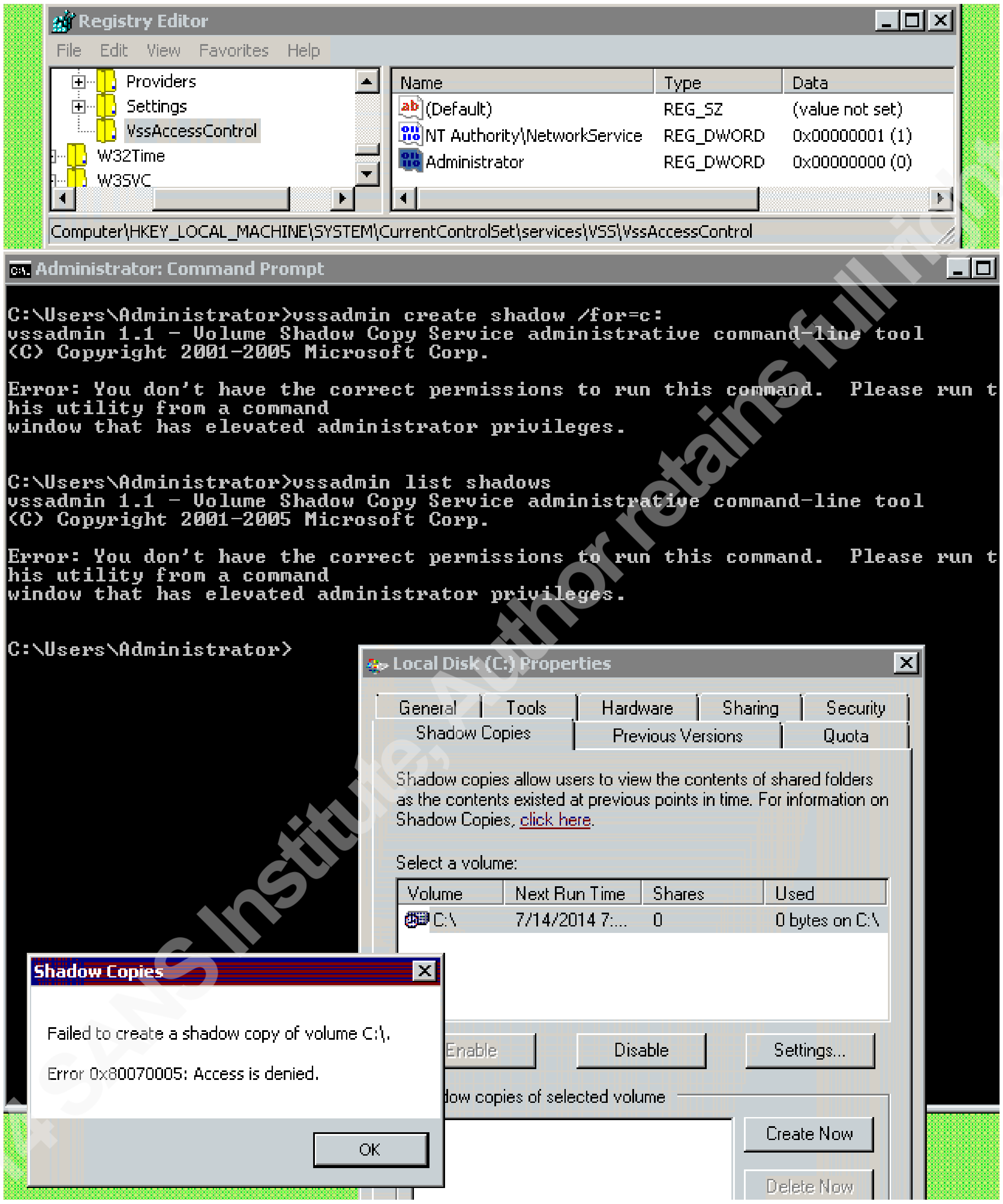
You do not have permission to view the current permission settings for SAM, but you can make permission changes.

Figure 9 - Local administrator unable to save copy of HKLM\SAM after permissions removed

6.4. Restrict shadow copying ability

The Volume Shadow Copy Service (VSS) (Microsoft Corporation, n.d.c) can be used to create a shadow copy of NTDS.DIT on a domain controller, useful for attackers to copy offline for extracting the hashes of domain accounts (haxrbyte, 2013). VSS is built in to Windows, is reliable, and its use is unlikely to be noticed in most environments, so it is a good way for an attacker who gains local administrative access to a domain controller to steal all domain account hashes. Therefore, restricting access to the VSS could be a way to mitigate this avenue of attack.

The VSS has a built-in way to restrict users from interacting with the service (Microsoft Corporation, n.d.b). This involves creating a registry subkey named as the account to be restricted, and setting the value to zero to deny access. To test this idea, a Windows Server 2008 R2 Datacenter test system was used. First, normal VSS access was verified using both the "vssadmin" command line utility and the Shadow Copies tab on the C: drive Properties dialog. Then, the registry subkey to restrict access was created, and the tests were repeated. As shown in Figure 10, the ability to use the "Create Now" button in the GUI was indeed restricted, as were some "vssadmin" commands. An application log error was also generated showing the failed attempt to create a shadow copy; the detail is shown in Appendix D – Application error log detail (VSS) on page 33.



Name	Type	Data
(Default)	REG_SZ	(value not set)
NT Authority\NetworkService	REG_DWORD	0x00000001 (1)
Administrator	REG_DWORD	0x00000000 (0)

Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\VSS\VssAccessControl

```
C:\Users\Administrator>vssadmin create shadow /for=c:
vssadmin 1.1 - Volume Shadow Copy Service administrative command-line tool
(C) Copyright 2001-2005 Microsoft Corp.

Error: You don't have the correct permissions to run this command.  Please run t
his utility from a command
window that has elevated administrator privileges.


C:\Users\Administrator>vssadmin list shadows
vssadmin 1.1 - Volume Shadow Copy Service administrative command-line tool
(C) Copyright 2001-2005 Microsoft Corp.

Error: You don't have the correct permissions to run this command.  Please run t
his utility from a command
window that has elevated administrator privileges.


C:\Users\Administrator>
```

Shadow copies allow users to view the contents of shared folders as the contents existed at previous points in time. For information on Shadow Copies, click here.

Select a volume:

Volume	Next Run Time	Shares	Used
C:\	7/14/2014 7:...	0	0 bytes on C:\

Failed to create a shadow copy of volume C:\.

Error 0x80070005: Access is denied.

Figure 10 - Restricting access to create or list shadow copies

Unfortunately, this restriction was particularly easy to get around. Through the GUI, it was still possible to alter the settings for the scheduled shadow copy, setting it to create a new copy one minute in the future. Once it was created, it was still possible to access files in it. Although it was not possible to view the volume name, these names are somewhat predictable. In this case, the full path to the SAM file was guessed, as shown in Figure 11.

```
C:\Users\Administrator>copy
\\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy5\windows\system32\config
\SAM .
     1 file(s) copied.
```

Figure 11 - Predictable shadow copy path and still able to access files

As with these other novel techniques, the attacker could also just change the permission setting in the registry to allow normal access to VSS functions unless further steps were taken to revoke registry rights. Once again, the value of a control like this may be more detective than preventive, since it is very difficult to completely restrict the activities of an attacker who has gained administrative rights.

7. Conclusion

The theft and subsequent unauthorized use of credentials is a significant risk in Windows environments today. While some of these issues, such as pass-the-hash (Ashton, 1997), have been known for over 17 years, increased attention is now being paid to them. This is evidenced by the availability of more straightforward and high profile information from Microsoft (Jungles, Margosis, Simos, Robinson, & Grimes, 2013), (Russinovich & Ide, 2014) as well as changes in recent versions of Windows to mitigate some of the issues (Russinovich & Ide, 2014).

While many mitigation techniques are already known, a number of novel techniques can be thought up when the underlying credential theft techniques are understood. Some of these include removing rights to create new services, restricting registry permissions, and restricting shadow copy rights. In all cases, a skilled attacker with administrative rights can get around these novel protections – for example, by giving himself the rights that were taken away. However, the attacker likely will not be expecting these techniques to be in place, so he will be slowed down, and more opportunities will exist for him to be detected. Of course, a downside of these techniques is that they may break some legitimate functionality, so they are best used in a well understood IT environment by a well-informed IT staff. It is likely these techniques would be most valuable as defense-in-depth measures in environments with high security requirements, where traditional mitigations are already in place.

These novel techniques also have value against automated attacks where a skilled attacker is not manually involved. For example, most malware that steals credentials will likely not be written to handle an unexpected restriction of rights, and so such malware is likely to simply fail. This also shows the detective value of these controls – if IT staff is watching for failure audit events in the right way, failed attempts to steal credentials can be detected. Using these controls for detection is most likely to be realistic in an already well secured environment with mature monitoring, detection and response capabilities. These ideas should be considered additional tools in the skilled defender's toolbox.

8. References

Ashton, P. (1997, April 8). *NT "Pass the Hash" with Modified SMB Client Vulnerability*. Retrieved from SecurityFocus: http://www.securityfocus.com/bid/233/info

Barta, C. (2011). *NTDSXtract*. Retrieved from NTDSXtract: http://www.ntdsxtract.com/

Be'ery, T. (2014, April 1). *Remote Desktop's Restricted Admin: Is the Cure Worse Than the Disease?* Retrieved from Aorato Blog: http://www.aorato.com/blog/remote-desktops-restricted-admin-cure-worse-disease/

Cuomo, N., & Tissieres, C. (2012, April 20). *samdump2 3.0.0*. Retrieved from Sourceforge - ophcrack: http://sourceforge.net/projects/ophcrack/files/samdump2/3.0.0/

Damele, B. (2011, December 27). *miscellaneous / ntdstopwdump.py*. Retrieved from inquisb (Bernardo Damele A. G.) - GitHub: https://github.com/inquisb/miscellaneous/blob/master/ntdstopwdump.py

Danchev, D. (2011, September 30). *Survey: 60 percent of users use the same password across more than one of their online accounts*. Retrieved from ZDNet: http://www.zdnet.com/blog/security/survey-60-percent-of-users-use-the-same-password-across-more-than-one-of-their-online-accounts/9489

Delpy, B. (2012a, August). *mimikatz*. Retrieved from Blog de Gentil Kiwi: http://blog.gentilkiwi.com/mimikatz

Delpy, B. (2012b, June 3). *Présentations - PhDays 2012.* Retrieved from Blog de Gentil Kimi: http://blog.gentilkiwi.com/downloads/mimikatz@phdays.pptx

Delpy, B. (2013, April 2). *minidump*. Retrieved from Blog de Gentil Kiwi: http://blog.gentilkiwi.com/securite/mimikatz/minidump

Dolan-Gavitt, B. (2012, August 1). *creddump*. Retrieved from Google Project Hosting: https://code.google.com/p/creddump/

James Foster, j@jefjef.com

Duckwall, A. S., & Campbell, C. (2014, March 3). *Still Passing the Hash 15 Years Later*. Retrieved from Still Passing the Hash 15 Years Later: http://passing-the-hash.blogspot.com/

fizzgig. (2008, September 18). *fgdump: Take *THAT* LSASS!* Retrieved from foofus.net: http://foofus.net/goons/fizzgig/fgdump/

fizzgig. (2009, January 29). *pwdump6.* Retrieved from foofus.net: http://www.foofus.net/fizzgig/pwdump/pwdump6-2.0.0-beta.zip

Glass, E. (2006). *The NTLM Authentication Protocol and Security Support Provider*. Retrieved from Davenport WebDAV-SMB Gateway - Sourceforge: http://davenport.sourceforge.net/ntlm.html

haxrbyte. (2013, November 21). *Tutorial for NTDS goodness (VSSADMIN, WMIS, NTDS.dit, SYSTEM)*. Retrieved from Trustwave SpiderLabs: http://blog.spiderlabs.com/2013/11/tutorial-for-ntds-goodness-vssadmin-wmis-ntdsdit-system-.html

JimF. (2011, June 24). *MSCash2 Algorithm*. Retrieved from Openwall Community Wiki: http://openwall.info/wiki/john/MSCash2

JoMo-Kun. (2003, September 24). *Passing the Hash*. Retrieved from foofus.net: http://foofus.net/goons/jmk/passhash.html

JoMo-Kun. (2012). *Medusa Parallel Network Login Auditor*. Retrieved from foofus.net: http://foofus.net/goons/jmk/medusa/medusa.html

Jorge. (2008, March 26). *Parsing SDDL Strings*. Retrieved from Jorge's Quest For Knowledge!: http://jorgequestforknowledge.wordpress.com/2008/03/26/parsing-sddl-strings/

Jungles, P., Margosis, A., Simos, M., Robinson, L., & Grimes, R. (2013, June 12). *Mitigating Pass-the-Hash (PtH) Attacks and Other Credential Theft Techniques (v1.1).* Retrieved from Microsoft Download Center: http://www.microsoft.com/en-us/download/details.aspx?id=36036

Kuliukas, K. (n.d.). *How Rainbow Tables work*. Retrieved from Kestas' home page: http://kestas.kuliukas.com/RainbowTables/

Lapukhov, P. (2008, July 14). *Private VLANs Revisited*. Retrieved from INE: http://blog.ine.com/2008/07/14/private-vlans-revisited/

Mandiant Corporation. (2013, February 18). *APT1: Exposing One of China's Cyber Espionage Units.* Retrieved from Mandiant Intelligence Center Report: http://intelreport.mandiant.com/Mandiant_APT1_Report.pdf

Metz, J. (2013, June 13). *libesedb*. Retrieved from Google Project Hosting: https://code.google.com/p/libesedb/

Microsoft Corporation. (2005a, January 21). *Deny access to this computer from the network*. Retrieved from Microsoft TechNet: http://technet.microsoft.com/en-us/library/cc758316%28v=ws.10%29.aspx

Microsoft Corporation. (2005b, January 21). *Deny log on through Terminal Services*. Retrieved from Microsoft TechNet: http://technet.microsoft.com/en-us/library/cc737453%28v=ws.10%29.aspx

Microsoft Corporation. (2008a). *sddlparse download.* Retrieved from Israel's Blogging Community: http://blogs.microsoft.co.il/files/folders/guyt/entry70399.aspx

Microsoft Corporation. (2008b, March 14). *If you grant somebody SeDebugPrivilege, you gave away the farm*. Retrieved from MSDN Blogs: http://blogs.msdn.com/b/oldnewthing/archive/2008/03/14/8080140.aspx

Microsoft Corporation. (2010a). *ACE Strings*. Retrieved from Microsoft Developer Network: http://msdn.microsoft.com/en-us/library/aa374928%28VS.85%29.aspx

Microsoft Corporation. (2010b). *Service Control Manager*. Retrieved from Microsoft Developer Network: http://msdn.microsoft.com/en-us/library/ms685150%28v=vs.85%29.aspx

Microsoft Corporation. (2011a, January 13). *AD DS: Read-Only Domain Controllers*. Retrieved from Microsoft TechNet: http://technet.microsoft.com/en-us/library/cc732801%28v=ws.10%29.aspx

Microsoft Corporation. (2011b, September 11). *Cached domain logon information*. Retrieved from Microsoft Support: http://support.microsoft.com/kb/172931

Microsoft Corporation. (2012a, November 15). *Network security: Do not store LAN Manager hash value on next password change*. Retrieved from Microosft TechNet: http://technet.microsoft.com/en-us/library/jj852276%28v=ws.10%29.aspx

Microsoft Corporation. (2012b). *Service Security and Access Rights*. Retrieved from Microsoft Developer Network: http://msdn.microsoft.com/en-us/library/ms685981%28v=vs.85%29.aspx

Microsoft Corporation. (2013, June 5). *Cached and Stored Credentials Technical Overview*. Retrieved from Microsoft TechNet: http://technet.microsoft.com/en-us/library/hh994565%28v=ws.10%29.aspx

Microsoft Corporation. (n.d.a). *Microsoft Windows 2000 Security Configuration Guide Appendix C - User Rights and Privileges*. Retrieved from Microsoft TechNet: http://technet.microsoft.com/en-us/library/dd277460.aspx

Microsoft Corporation. (n.d.b). *Security Considerations for Requestors (Windows)*. Retrieved from Microsoft Developer Network: http://msdn.microsoft.com/en-us/library/aa384604.aspx

Microsoft Corporation. (n.d.c). *Volume Shadow Copy Service Overview*. Retrieved from Microsoft Developer Network: http://msdn.microsoft.com/en-us/library/aa384649%28v=vs.85%29.aspx

Microsoft Corporation. (n.d.d.). *Security Subsystem Architecture*. Retrieved from Microsoft TechNet: http://technet.microsoft.com/en-us/library/cc961760.aspx

NetworkAdminKB.com. (2009, May 13). *Understanding the SDDL permissions in the ACE_String*. Retrieved from NetworkAdminKB: http://networkadminkb.com/KB/a6/understanding-the-sddl-permissions-in-the-ace-string.aspx

Ochoa, H. (2007). *What is Pass-The-Hash Toolkit?* Retrieved from CoreLabs: http://corelabs.coresecurity.com/index.php?module=Wiki&action=view&type=tool&name=Pass-The-Hash_Toolkit

Ochoa, H. (2011, March 5). *Amplia Security - Research - WCE Internals.* Retrieved from Amplia Security: http://www.ampliasecurity.com/research/WCE_Internals_RootedCon2011_ampliasecurity.pdf

Ochoa, H. (2014). *Windows Credentials Editor (WCE) F.A.Q.* Retrieved from Amplia Security: http://www.ampliasecurity.com/research/wcefaq.html

Pilkington, M. (2012, March 21). *Protecting Privileged Domain Accounts: Safeguarding Access Tokens*. Retrieved from Blog: SANS Digital Forensics and Incident Response: http://digital-forensics.sans.org/blog/2012/03/21/protecting-privileged-domain-accounts-access-tokens

Pilon, A., Marechal, S., & Devine, C. (2005, December 28). *cachedump*. Retrieved from Openwall file archive: http://download.openwall.net/pub/projects/john/contrib/cachedump/

Rapid7. (2014, May 22). *Module: Msf::Post::Windows::ShadowCopy*. Retrieved from Metasploit Framework: https://dev.metasploit.com/api/Msf/Post/Windows/ShadowCopy.html

Russinovich, M. (2014, May 13). *ProcDump*. Retrieved from Windows Sysinternals: http://technet.microsoft.com/en-us/sysinternals/dd996900.aspx

Russinovich, M., & Cogswell, B. (2014, March 7). *Process Monitor*. Retrieved from Windows Sysinternals: http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx

Russinovich, M., & Ide, N. (2014, February 26). *USA 2014: Pass-the-Hash: How Attackers Spread and How to Stop Them (HTA-W03).* Retrieved from RSA Conference: http://www.rsaconference.com/writable/presentations/file_upload/hta-w03-pass-the-hash-how-attackers-spread-and-how-to-stop-them.pdf

sondre. (2012, August 24). *Effect of password policies on keyspace reduction*. Retrieved from Openwall Community Wiki: http://openwall.info/wiki/john/policy

SplashData, Inc. (2013). *"Password" unseated by "123456" on SplashData's annual "Worst Passwords" list*. Retrieved from SplashData News: http://splashdata.com/press/worstpasswords2013.htm

Steube, J. A. (2014). *oclHashcat - advanced password recovery*. Retrieved from hashcat: http://hashcat.net/oclhashcat/

supernothing. (2010, March 03). *MSCash Algorithm*. Retrieved from Openwall Community Wiki: http://openwall.info/wiki/john/MSCash

U47. (2012, August 27). *What has changed with Win 2008 Server that Samba rpc control no longer works?* Retrieved from Serverfault: http://serverfault.com/questions/398056/what-has-changed-with-win-2008-server-that-samba-rpc-control-no-longer-works

Verizon Enterprise Solutions. (2014, April 23). *2014 Data Breach Investigations Report.* Retrieved from 2014 Verizon Data Breach Investigations Report

(DBIR): http://www.verizonenterprise.com/DBIR/2014/reports/rp_Verizon-DBIR-2014_en_xg.pdf

9. Appendix A – Output of sddlparse.exe

The default SDDL on the SCM object is listed in Figure 6 on page 19. When this SDDL is fed to the sddlparse.exe tool, it is parsed to the output listed in Figure 12 (line breaks have been added for readability).

```
Ace count: 5

**** ACE 1 of 5 ****
ACE Type: ACCESS_ALLOWED_ACE_TYPE
Trustee: NT AUTHORITY\Authenticated Users
AccessMask:
  ADS_RIGHT_DS_CREATE_CHILD
Inheritance flags: 0

**** ACE 2 of 5 ****
ACE Type: ACCESS_ALLOWED_ACE_TYPE
Trustee: NT AUTHORITY\INTERACTIVE
AccessMask:
  ADS_RIGHT_READ_CONTROL
  ADS_RIGHT_DS_CREATE_CHILD
  ADS_RIGHT_ACTRL_DS_LIST
  ADS_RIGHT_DS_READ_PROP
Inheritance flags: 0

**** ACE 3 of 5 ****
ACE Type: ACCESS_ALLOWED_ACE_TYPE
Trustee: NT AUTHORITY\SERVICE
AccessMask:
  ADS_RIGHT_READ_CONTROL
  ADS_RIGHT_DS_CREATE_CHILD
  ADS_RIGHT_ACTRL_DS_LIST
  ADS_RIGHT_DS_READ_PROP
Inheritance flags: 0

**** ACE 4 of 5 ****
ACE Type: ACCESS_ALLOWED_ACE_TYPE
Trustee: NT AUTHORITY\SYSTEM
AccessMask:
  ADS_RIGHT_READ_CONTROL
  ADS_RIGHT_DS_CREATE_CHILD
  ADS_RIGHT_ACTRL_DS_LIST
  ADS_RIGHT_DS_READ_PROP
  ADS_RIGHT_DS_WRITE_PROP
Inheritance flags: 0
```

```
**** ACE 5 of 5 ****
ACE Type: ACCESS_ALLOWED_ACE_TYPE
Trustee: BUILTIN\Administrators
AccessMask:
  ADS_RIGHT_DELETE
  ADS_RIGHT_READ_CONTROL
  ADS_RIGHT_WRITE_DAC
  ADS_RIGHT_WRITE_OWNER
  ADS_RIGHT_DS_CREATE_CHILD
  ADS_RIGHT_DS_DELETE_CHILD
  ADS_RIGHT_ACTRL_DS_LIST
  ADS_RIGHT_DS_SELF
  ADS_RIGHT_DS_READ_PROP
  ADS_RIGHT_DS_WRITE_PROP
Inheritance flags: 0
```

Figure 12 - Default SCM permissions parsed by sddlparse.exe

The final ACE block (ACE 5 of 5) indicates the rights in question. The BUILTIN\Administrators group DACL (A;;KA;;;BA) basically means "allow all rights". Some effort is needed to translate the listed ADS_ rights to the unique permissions applicable to the SCM. The detail of ACE DACL translation is outside the scope of this paper, but some helpful references are provided (U47, 2012), (Jorge, 2008), (NetworkAdminKB.com, 2009).

10. Appendix B – Security event log detail (scmanager)

The full detail of one of the security event log audit failures shown in Figure 8 on page 20 is listed in Figure 13.

```
Log Name:      Security
Source:        Microsoft-Windows-Security-Auditing
Date:          7/13/2014 12:03:22 AM
Event ID:      4656
Task Category: Other Object Access Events
Level:         Information
Keywords:      Audit Failure
User:          N/A
Computer:      COMPUTER1
Description:
A handle to an object was requested.

Subject:
      Security ID:            COMPUTER1\admin1
      Account Name:           admin1
      Account Domain:         COMPUTER1
      Logon ID:         0x1e7cc

Object:
      Object Server:          SC Manager
      Object Type:            SC_MANAGER OBJECT
      Object Name:            ServicesActive
      Handle ID:        0x0

Process Information:
      Process ID:       0x1e4
      Process Name:           C:\Windows\System32\services.exe

Access Request Information:
      Transaction ID:         {00000000-0000-0000-0000-000000000000}
      Accesses:         Connect to service controller
                        Create a new service
                        Enumerate services

      Access Reasons:         -
      Access Mask:            0x7
      Privileges Used for Access Check:   -
      Restricted SID Count:   0

Event Xml:
<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
  <System>
    <Provider Name="Microsoft-Windows-Security-Auditing"
      Guid="{54849625-5478-4994-A5BA-3E3B0328C30D}" />
    <EventID>4656</EventID>
    <Version>1</Version>
    <Level>0</Level>
    <Task>12804</Task>
    <Opcode>0</Opcode>
    <Keywords>0x8010000000000000</Keywords>
    <TimeCreated SystemTime="2014-07-13T04:03:22.035384800Z" />
    <EventRecordID>16327</EventRecordID>
    <Correlation />
    <Execution ProcessID="492" ThreadID="516" />
    <Channel>Security</Channel>
    <Computer>COMPUTER1</Computer>
    <Security />
  </System>
  <EventData>
    <Data Name="SubjectUserSid">S-1-5-21-3577214157-4138306266-
      1988903399-1000</Data>
    <Data Name="SubjectUserName">admin1</Data>
    <Data Name="SubjectDomainName">COMPUTER1</Data>
    <Data Name="SubjectLogonId">0x1e7cc</Data>
    <Data Name="ObjectServer">SC Manager</Data>
    <Data Name="ObjectType">SC_MANAGER OBJECT</Data>
    <Data Name="ObjectName">ServicesActive</Data>
    <Data Name="HandleId">0x0</Data>
    <Data Name="TransactionId">{00000000-0000-0000-0000-
      000000000000}</Data>
    <Data Name="AccessList">%%7168
                        %%7169
                        %%7170
                        </Data>
    <Data Name="AccessReason">-</Data>
    <Data Name="AccessMask">0x7</Data>
    <Data Name="PrivilegeList">-</Data>
    <Data Name="RestrictedSidCount">0</Data>
    <Data Name="ProcessId">0x1e4</Data>
    <Data Name="ProcessName">C:\Windows\System32\services.exe</Data>
  </EventData>
</Event>
```

Figure 13 - Security event log detail of audit failure for access to scmanager

11. Appendix C – Security event log detail (HKLM\SAM)

The full detail of one of the security event log audit failures shown in Figure 9 on page 21 is shown in Figure 14.

```
Log Name:      Security
Source:        Microsoft-Windows-Security-Auditing
Date:          7/13/2014 2:16:13 PM
Event ID:      4656
Task Category: Registry
Level:         Information
Keywords:      Audit Failure
User:          N/A
Computer:      COMPUTER1
Description:
A handle to an object was requested.

Subject:
      Security ID:            COMPUTER1\admin1
      Account Name:           admin1
      Account Domain:         COMPUTER1
      Logon ID:         0x222ab

Object:
      Object Server:          Security
      Object Type:            Key
      Object Name:            \REGISTRY\MACHINE\SAM
      Handle ID:        0x0

Process Information:
      Process ID:       0x9b0
      Process Name:           C:\Windows\System32\reg.exe

Access Request Information:
      Transaction ID:         {00000000-0000-0000-0000-000000000000}
      Accesses:         READ_CONTROL

      Access Reasons:         -
      Access Mask:            0x20000
      Privileges Used for Access Check:   -
      Restricted SID Count:   0
Event Xml:
<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
  <System>
    <Provider Name="Microsoft-Windows-Security-Auditing"
Guid="{54849625-5478-4994-A5BA-3E3B0328C30D}" />
    <EventID>4656</EventID>
    <Version>1</Version>
    <Level>0</Level>
    <Task>12801</Task>
    <Opcode>0</Opcode>
    <Keywords>0x8010000000000000</Keywords>
    <TimeCreated SystemTime="2014-07-13T18:16:13.302216900Z" />
    <EventRecordID>13305</EventRecordID>
    <Correlation />
    <Execution ProcessID="480" ThreadID="504" />
    <Channel>Security</Channel>
    <Computer>COMPUTER1</Computer>
    <Security />
  </System>
  <EventData>
    <Data Name="SubjectUserSid">S-1-5-21-3577214157-4138306266-
1988903399-1000</Data>
    <Data Name="SubjectUserName">admin1</Data>
    <Data Name="SubjectDomainName">COMPUTER1</Data>
    <Data Name="SubjectLogonId">0x222ab</Data>
    <Data Name="ObjectServer">Security</Data>
    <Data Name="ObjectType">Key</Data>
    <Data Name="ObjectName">\REGISTRY\MACHINE\SAM</Data>
    <Data Name="HandleId">0x0</Data>
    <Data Name="TransactionId">{00000000-0000-0000-0000-
000000000000}</Data>
    <Data Name="AccessList">%%1538
                        </Data>
    <Data Name="AccessReason">-</Data>
    <Data Name="AccessMask">0x20000</Data>
    <Data Name="PrivilegeList">-</Data>
    <Data Name="RestrictedSidCount">0</Data>
    <Data Name="ProcessId">0x9b0</Data>
    <Data Name="ProcessName">C:\Windows\System32\reg.exe</Data>
  </EventData>
</Event>
```

Figure 14 - Security event log detail of audit failure for access to HKLM\SAM

12. Appendix D – Application error log detail (VSS)

The full detail of the application error log entry caused by the activity shown in Figure 10 on page 23 is shown in Figure 15.

```
Log Name:      Application
Source:        VSS
Date:          7/14/2014 4:26:58 AM
Event ID:      7001
Task Category: None
```

```
Level:         Error
Keywords:      Classic
User:          N/A
Computer:      SERVER1
Description:
VssAdmin: Unable to create a shadow copy: You don't have the correct
permissions to run this command.  Please run this utility from a
command window that has elevated administrator privileges.
Command-line: 'vssadmin  create shadow /for=c:'.

Event Xml:
<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
  <System>
    <Provider Name="VSS" />
    <EventID Qualifiers="0">7001</EventID>
    <Level>2</Level>
    <Task>0</Task>
    <Keywords>0x80000000000000</Keywords>
    <TimeCreated SystemTime="2014-07-14T04:26:58.000000000Z" />
    <EventRecordID>7117</EventRecordID>
    <Channel>Application</Channel>
    <Computer>SERVER1</Computer>
    <Security />
  </System>
  <EventData>
    <Data>Unable to create a shadow copy</Data>
    <Data>You don't have the correct permissions to run this command.
Please run this utility from a command window that has elevated
administrator privileges. </Data>
    <Data>vssadmin  create shadow /for=c:</Data>
    <Data>
    </Data>

<Binary>2D20436F64653A2041444D5641444D4330303030303834332D2043616C6C3A2
041444D5641444D4330303030303738302D205049443A20203030303030343033322D2054
49443A20203030303032323236302D20434D443A2020767373616D696E2020637265617
46520736861646F77202F666F723D633A202D20557365723A204E616D653A2057494E2D
54524D35373430543255505C41646D696E6973747261746F722C205349443A532D312D3
52D32312D313137353435303036312D3233323031303235303382D323834373636383237
392D3530302020</Binary>
  </EventData>
</Event>
```

Figure 15 – Application error log detail for failed vssadmin activity

James Foster, j@jefjef.com