Investigating Credential Stealing Attacks on Microsoft Windows Platforms

Lujo Bauer Nicolas Christin Tushar Goyal
Carnegie Mellon CyLab

Abstract
Microsoft Windows Active Directory is commonly used to centrally manage user accounts and security controls over entire networks of machines. However, Active Directory might lead to security vulnerabilities when Windows services are configured to run with the privileges of so-called Active Directory users. Indeed, while Active Directory user credentials are stored in encrypted form on a central server, these secrets can be potentially decrypted by *any user* having administrative access to *any machine* that is part of that Windows domain. A malicious user who can capture these credentials may then gain access to any machine that is part of the same Windows domain. In this report, we first provide a practical demonstration of the attack, using an open source tool ("mimikatz"). We then examine in more detail which Windows deployments are vulnerable; we highlight that a large number of Windows deployments can be exploited, which in turn can result in large-scale network compromise. We conclude by outlining possible mitigations.

1. Introduction

Organizations commonly use databases to centrally manage user accounts and security policies. This centralization allows systems administrators to set up user accounts and policies only once for the whole network, instead of creating user accounts on every single computer belonging to their organization.

One implementation of such centralized systems, called "directory servers," rely on the Lightweight Directory Access Protocol (LDAP) to provide an endpoint for its member machines to authenticate users and check for security policies. Microsoft Windows servers use a similar implementation, called "Active Directory." The Active Directory server allows administrators to implement fine-grained access policies across the network, e.g., based on client type, state, or on specific operations.

Active Directory (AD) is also useful to configure the execution context in which Windows services (similar to UNIX daemons) operate. More precisely, Active Directory supports the creation of Active Directory user accounts. AD user accounts define the execution context of the services that use them. For example, if a Windows service is configured to run under the account "DOMAIN\UserName", then the privileges of the service at runtime are those that were assigned to that account. When a service on a local machine is configured to run under a domain account (e.g., as one would do with a print service), these account credentials are cached in the registry of that machine. Such configuration of services is often done automatically (e.g., by scripts) and on multiple machines at once (e.g., a script is automatically deployed to multiple machines and enters the same credentials into each).

Unfortunately, as we discuss in this report, these secrets that are stored in the registry of a machine can be decrypted by any user who has administrative access to that machine; they can then be used to gain access to any other machine in the same domain, regardless of whether that machine had been configured to use the same service. These other machines could contain sensitive data or support critical functions that can thus become compromised, or could simply be exploited to stage further attacks (e.g., denial-of-service).

In the next sections, we first describe the terminology used in the Windows Active Directory architecture. We then present our empirical setup and demonstrate how to decrypt LSA secrets, using an open-source tool called mimikatz. We generalize our results by explaining how vulnerable various combinations of client-servers with disparate Windows versions might be. Finally, we turn to discussing mitigations and conclude.

2. LSA Secrets in Windows Registry

In Windows, the Local Security Authority (LSA) is the sub-system responsible for managing the local security policy. In particular, LSA in charge of auditing, authenticating, and logging users onto the system, as well as storing private data. The LSA sub-system uses a secure store, called Local Security Authority Secrets or LSA Secrets to keep important pieces of information safe. All data stored in LSA Secrets is encrypted and obfuscated [1]. These secrets are only accessible to System users (i.e., users in the System group).

Examples of data usually stored in LSA Secrets include:
- *DefaultPassword*: Password used to automatically log in to the Windows operating system;
- *NL$KML*: A secret used to encrypt cached domain passwords;
- *L$RTMTIMEBOMB_[...]*: A structure which marks the expiration date of an unactivated Windows instance.

More precisely, LSA Secrets are located in the SECURITY hive of the Windows registry, under the *SECURITY\Policy\Secrets* key. Under this registry key, each secret has its own entry (sub-key), under which the secret and its metadata are stored (e.g., the password for a domain service account might be stored under SECURITY\Policy\Secrets_SC_SQL). Under each key corresponding to a secret (e.g., a password to a domain service account), one can find values named *SecDesc*, *CurrVal*, *OldVal*, *CupdTime*, and *OupdTime*. The data associated with these values contain the security descriptor (which defines the access rights of that account), the current and old values of the secret (i.e., current and old password), and the time when the current and old values were created (which can be used to enforce password expiration requirements, for example). We will elaborate on these when we describe the process through which these secrets can be decrypted.

Credentials (e.g., encryption keys) that can be used to decrypt secrets (and metadata that describes the credentials) are stored under the *SECURITY\Policy\* registry key. In particular, this registry key contains the version of LSA Secrets encryption used (SECURITY\Policy *PolRevision*), along with the list of encryption keys (SECURITY\Policy\PolEKList) used to encrypt secrets.

Under each registry key that represents a secret such as a password (e.g., SECURITY\Policy \Secrets_SC_SQL), *CurrVal* and *OldVal* use identical data structures to store information about the current and previous versions of the secret. In this data structure, the first 4 bytes denote a version, and are followed by a 16-byte (64-bit) encryption key identifier used to select among possible entries in *PolEKList*. The 4 bytes following the identifier define the encryption and hashing algorithm being used, and the next 4 bytes define the flags used in the encryption. Subsequent bytes correspond to the encrypted secret. More information about this data structure and format variation between different version of Windows operating systems can be found in related technical documents [1], [2].

When Windows services are configured with domain account credentials, these credentials are stored in encrypted form in LSA Secrets, as described above. The encryption keys that can be used to decrypt them, however, are also stored in LSA Secrets, unencrypted and accessible to anyone with administrative access to the computer (and hence registry). Thus, any local administrator can decrypt these credentials. When these credentials are to a domain account, whoever has gained access to the credentials can use them to log into any machine in that domain, regardless of whether that domain account has been used to access that machine previously.

3. Validation experiment

We next describe the experiment we use to validate that such network-level compromises are possible. We start by describing our network testbed, before turning to specific instructions on each machine.

3.1 Prerequisites

Conducting the verification test requires to set up three networked machines which can contact each other. In the scenario depicted in Figure 1, the attacker has full local administrative control on machine B, but does not have any specific access to machine A nor to machine C.

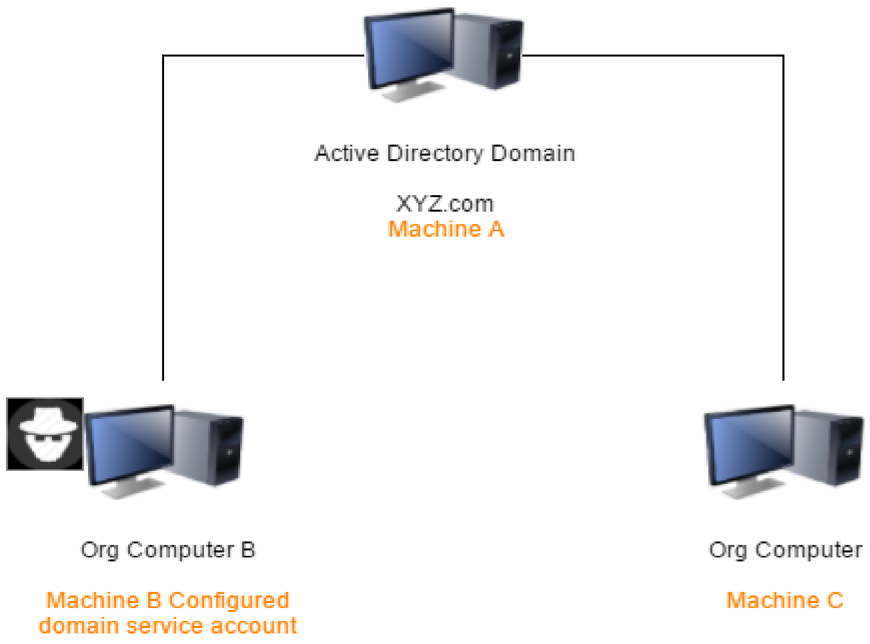


Figure 1. Organisation setup for the testing procedure

In the figure, Machine A acts as a Domain Controller (Directory Server). The machine is the central point for all user accounts and associated security policies. The directory server needs to have an account configured as logon account (i.e., an account that allows a user to log on) for the Windows service it wants to run on Machine B. Functional details and the specifics of the active directory configuration can be found in Appendix A.

Machine B is the machine on which credential theft will occur. Machine B runs a Windows service configured with a domain user account. This allows the service to run in the execution context of the domain account defined on Machine A.

Machine C is our target: namely, it is used in our experimental setup to verify that credentials obtained on Machine B can be used to gain access to another machine (Machine C). Gaining access to Machine C is a side effect of the fact that Machine C is also connected to the domain; if successful, the attack will allow us to log on to Machine C with the credentials recovered on Machine B.

3.2 Configuring a Windows service to run as domain account [optional]

The following steps configure a Windows service so that its execution context is a domain user account.

1. Press WinKey + R, type *services.msc* inside the textbox, and hit Enter; or, search for *Services* in the Windows Start menu, which gives the location as "*C:\Windows\system32*"

2. Search for the service that you will associate with a domain (logon) account. (In a real attack, such a service account would already need to be present; in practice, this is a relatively common scenario.)
3. Right-click the selected service, select “Properties” and then move to the “Log On” tab.
4. Select the radio button “This Account,” enter the qualified credentials as “domain name/username,” and input the corresponding password. For instance, the account *abc* in domain *XYZ.com* and NetBIOS name *XYZ* should be entered as “*XYZ/abc.*” This step will configure the execution context of the service to be linked to the account domain_name/username.
5. Click Apply to apply the changes and OK to close the Properties windows.

3.3 Checking if the account credentials were stored as LSA Secrets

As noted before, the LSA sub-system (LSA, or LSASS) is a Windows system process which handles critical security tasks such as security policy enforcement and user logon. LSA stores some of the data in a protected storage called LSA Secrets. The location for the LSA Secrets is *HKLM\\Security\\Policy\\Secrets* in the registry hive [1].

Configuring a service as described in Section 3.2 results in a registry folder (sub-key) to be created in LSA Secrets. This folder is named *_SC_servicename*. For instance, the MSSQL server (service) would yield an LSA Secrets sub-key called *_SC_MSSQLServer*; the *gupdate* service would produce a sub-key called *_SC_gupdate*.

The following steps describe how the LSA Secrets store can be accessed with the help of internal system tools (“Sysinternal tools”) from Microsoft. We use here a tool called *PsExec* to create a process in the execution context of a different user. The whole suite of tools can be downloaded freely [3]. We use PsExec to launch a terminal (command prompt) under System privileges, which then allows us to read the registry location for LSA Secrets and verify that the service credentials are being stored there.

1. Launch a command prompt with administrative privileges on the host computer.
2. Navigate to the directory folder where tools have been downloaded on the command prompt.
3. Run command “psexec -i -s cmd.exe”. The command launches another command prompt under System-level privileges. This enables us to see the LSA Secrets location in a registry editor; this is not typically accessible by regular user or even power users with administrative privileges [4].
4. In the launched command prompt, type “*regedit.exe*” and hit enter. The Registry Editor Browser will start, and can next be used to view the registry entries for the local system.
5. Navigate to HKEY_LOCAL_MACHINE, and then open the Security\\Policy\\Secrets folder. There should be entry(ies) starting with _SC_servicename. Select and expand the entry. As discussed in Section 2, the folder contains the following values [2]:
    a. *CurrVal*: Current encrypted value of the secret

b. *CupdTime*: Time when the value was last updated
c. *OldVal*: The previously encrypted value for the secret
d. *OupdTime*: Time when previous value was updated

At this point, all these secrets are still encrypted. The next step is to decrypt them.

3.4 Decrypting LSA Secrets from the Registry hive

The primary objective of the activity is to decrypt the secrets stored in the Registry hive so that the decrypted credentials can be used to log on to Machine C.

The decryption requires the *SeDebugPrivilege* privilege. The privilege is used for system-level debugging purposes. Crucially, a process running with the *SeDebugPrivilege* privilege gains the ability to debug processes owned by *other* users; in particular, one can debug processes owned by the System user [5]. Since the LSA secrets are owned by the system-level LSA process, the SeDebugPrivilege privilege will be instrumental in decrypting them. A local (or domain) administrator can always cause the SeDebugPrivilege privilege to be added to a process.

To accomplish our decryption task, we use the open source utility *mimikatz* [6]. Mimikatz has various uses such as retrieving password hashes from the Security Access Manager (SAM) and generating Kerberos tokens. The latest release can be downloaded freely [7]. For the purpose of this experiment, we use mimikatz to (1) gain SeDebugPrivilege, (2) escalate the mimikatz process token to system-level, and (3) to decrypt the service credentials stored in LSA secrets.

We decrypt and dump LSA secrets using the following procedure:

1. Launch a Windows command prompt with administrative privilege on the host computer.
2. Navigate to the directory where mimikatz was downloaded. Navigate to folder “win32” or “x64” according to the architecture of the operating system.
3. Type mimikatz.exe and hit enter. A new command prompt will appear with “*mimikatz* >” as the terminal prompt. This instantiates mimikatz utility with a welcome message.
4. The first step is to gain the SeDebugPrivilege. We use the command “*privilege::debug*” for this. The prompt will display a success message. The successful execution gives us SeDebugPrivilege for the current mimikatz process.
5. We elevate the process token to system-level privilege with “*token::elevate*.” Once again, the prompt will display a success message confirming the process token elevation [8].
6. We can now decrypt the LSA Secrets with the command “*lsadump::secrets*”. This command leads mimikatz to attempt to decrypt all the secrets, including password entries, stored under LSA Secrets [8]. Checking the entries corresponding to _*SC_servicename* gives the username and the password displayed in plain text.

3.5 Checking the access to the system using the stolen credentials

Since the credentials acquired as described in Section 3.4 are Active Directory credentials, these can be used to log in to any machine in the network linked to the directory server.

This can be checked by physically accessing Machine C and using the credentials to log in; the login will be successful. If physical access is not possible, one can use the Remote Desktop Protocol (RDP) to remotely attempt to log in to Machine C.

The following steps use RDP to log in remotely to Machine C:

1. From a Windows machine, search for “Remote Desktop Connection” in the Start Menu, usually found under “Accessories”. The remote desktop connection application can also be launched by pressing “WinKey + R” and entering “mstsc.exe”.
2. In the textbox, enter the hostname or IP address of the computer (Machine C). Press Connect.
3. When prompted, enter the credentials extracted from the earlier section from Machine B. The application will try to authenticate through the credentials supplied and open the interactive user session on Machine C.

4. Results

Our test setup used three virtual machines (VMs) for Machine A, B and C, all running on a single desktop computer with Intel(R) Core(TM) i7-4770 CPU @ 3.40 GHz, 32.0 GB RAM, AMD Radeon HD 8570 graphics card, 1GB DDR3 memory, and running Windows 10 Professional. We used VMWare Workstation 12 Pro for running VMs on the machine. This setup allowed us to test multiple combinations of operating systems for the hosts and domain controllers.

Machine A. Machine A, i.e. the domain controller directory server, was installed using Windows Server 2008 R2. The virtual machine was granted 2GB of RAM and 40GB of operating system space. We used XYZ.com as a domain name, with NetBIOS name of XYZ, and the machine acted as primary DNS server for other client machines. We first configured the domain controller to be running Windows Server 2008 R2. In a subsequent variation, we used a different virtual machine with the same parameters, but running Windows Server 2012 R2 for the domain controller.

We configured the test account under which the target service would be executed as “XYZ/servicelogin” with a password of “P@ssword123”.

We used mimikatz version 2.1.0-20161126, which was released on November 25, 2016 [9].

We downloaded the most recent version of the sysinternals suite (November 18, 2016 update) from the official website [3].

Machine B. For Machine B, we tested several variants of Windows, each with a Windows services configured to run as domain account XYZ\servicelogin. The service which was used was *gupdate.* Gupdate is a service offered by Google that periodically checks with Google servers for updates to its Chrome browser. (In practice, one would not associate this particular service with domain credentials; we are using it just for illustration.) The hard drive and RAM configuration for several types of client configurations of Machine B can be found in Appendix B.

Experimental results. Table A shows a summary of the results. Columns specify the clients used; rows specify the domain controller server OS. Tick marks indicate compromise success. That is, when the service was configured to be running with "XYZ/servicelogin," the credentials were stored in LSA Secrets and we were able to recover the plaintext password "P@ssword123" using mimikatz.

	Windows 7	Windows 8	Windows Server 2008	Windows Server 2008 R2	Windows Server 2012	Windows Server 2012 R2
Windows Server 2008 R2	✔	✔	✔	✔	✔	✔
Windows Server 2012 R2	✔	✔	✔	✔	✔	✔

The tests showed uniform behavior across the clients: we were able to consistently decrypt the LSA Secrets using the technique outlined above.

Gaining access to Machine C with the obtained credentials was verified both in physical console mode and with a Remote Desktop Connection, as described in Section 3.5. This demonstrates that the credentials allow us to log on to any machine belonging to the domain network.

5. Mitigation Measures

The experiments in Section 3 demonstrate that Windows stores service credentials in the LSA secrets registry entry location. The ability for a system-level account to access this location, and decrypts its content, yields the vulnerability.

To prevent this from happening, the solution is to either 1) avoid storing passwords in LSA Secrets (or in any location readable by a system account) or 2) improve the security of LSA Secrets.

This section first describes new managed service accounts (sMSA) and group managed service accounts (gMSA) available in recent versions of Windows Active Directory. These accounts do not store passwords in LSA Secrets and instead obtain credentials in dynamically as needed. Second, we discuss CredentialGuard, which takes a different approach of combining hardware and virtualization-based security to harden the security of the LSA Secrets store.

5.1 Managed Service Accounts

Windows provides two security principals, namely, managed service accounts and group managed service accounts, which can be used to avoid the vulnerabilities linked with storing passwords in LSA Secrets.

Managed Service Accounts (sMSA) [10] were first introduced in Windows Server 2008 R2. Their main focus was to introduce management for the service accounts at a domain level. In particular, sMSA provide automatic password management, with automatic expiration every 30 days by default. Unfortunately, a single sMSA cannot be used by multiple clients in the domain and, hence, requires different accounts to be created for each client [11]. This limits the usefulness of these accounts in the context of Active Directory, whose main objective is to provide a centralized interface to account management.

Group Managed Service Accounts (gMSA) are a recent extension to sMSA, and appeared in Windows 2012. Most interestingly to us, they remove the limitations of one service account per client. Hence, one account can be shared by multiple clients.

gMSA work with the help of Microsoft’s key distribution service. That is, rather than storing passwords, the service only stores the account name, and gets the password on demand from the domain when required [11], [12]. Our experiments confirm that when a service is configured with a group managed service account, the registry entries are created at configuration time [13]. However, when the LSA Secrets are decrypted, the only value we are able to capture is the account name of the security principal.

5.2 Credential Guard

A different approach to improve the security of LSA Secrets is pursued by Credential Guard [14]. Credential Guard was introduced in new versions of Windows: Windows 10 Enterprise Edition and Windows Server 2016. The prerequisites for using Credential Guard is the availability of TPM hardware and the Intel Processor based virtualization support (Intel VT-x).

Credential Guard protects LSA Secrets by combining virtualization with hardware based security. To be more precise, Credential Guard stores secrets in a different virtualized minimal operating system, which is accessible only to privileged processes. The secrets are stored and protected by an isolated virtualized LSA Process running in the Windows HyperVisor over the native processor virtualization extension. This virtualized LSA Process is hosted on a minimal operating system with no driver, and interfaces with the local LSA process through remote procedure calls. Credential Guard relies on the TPM to ensure that only the correct isolated process is running on the appropriate virtual machine.

Since the credentials are shielded in a virtual machine, the technique we described in this report would not work; requests to retrieve the password would be blocked. Along the same lines, malware that manages to commandeer administrative privileges would not be able to perform credential extraction.

Credential guard requires (at least) Windows 10 Enterprise Edition, which supports Windows Hyper-V, a 64-bit CPU that supports virtualization extensions and extended page tables, and a Trusted Platform Module (TPM) of version 1.2 or greater [15].

Appendix C lists the relevant documentation from Microsoft for configuring the Managed Service accounts and Credential Guards and other details.

6. Conclusions

Deployments of Windows often rely on Windows domain servers to ease management of clients. Common methods of deploying and configuring services on these clients cause domain-managed credentials to be cached on individual machines, where they are vulnerable to theft by anyone who has administrative access to any of those administrative machines. Once compromised, the credentials can be used to gain access to other machines on the network.

Our experiments confirm that this vulnerability is easy to take advantage of, and that a wide range of commonly deployed versions of Windows are affected. Fortunately, new versions of Windows offer multiple options for completely mitigating this vulnerability.

7. References

[1] B. Dolan-Gavitt, “Decrypting LSA Secrets.” [Online]. Available: http://moyix.blogspot.com/2008/02/decrypting-lsa-secrets.html. [Accessed: 25-Feb-2017]
[2] Passcape Software, “LSA Secrets in Windows.” [Online]. Available: https://www.passcape.com/index.php?section=docsys&cmd=details&id=23. [Accessed: 25-Feb-2017]
[3] Microsoft Sysinternals, “Sysinternals Suite.” [Online]. Available: https://technet.microsoft.com/en-us/bb842062. [Accessed: 25-Feb-2017]
[4] J. Davis, “Running a CMD prompt as System (XP/Vista/Win7/Win8),” *The Realm of the Verbal Processor*, 06-Dec-2007. [Online]. Available: https://verbalprocessor.com/2007/12/05/running-a-cmd-prompt-as-local-system/. [Accessed: 25-Feb-2017]
[5] Microsoft Technet, “Privilege Constants (Windows).” [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/bb530716(v=vs.85).aspx. [Accessed: 25-Feb-2017]
[6] B. Delpy, “GitHub - gentilkiwi/mimikatz: A little tool to play with Windows security.” [Online].

Available: https://github.com/gentilkiwi/mimikatz. [Accessed: 25-Feb-2017]
[7] B. Delpy, “Releases · gentilkiwi/mimikatz · GitHub.” [Online]. Available: https://github.com/gentilkiwi/mimikatz/releases. [Accessed: 25-Feb-2017]
[8] B. Delpy, “module ~ lsadump · gentilkiwi/mimikatz Wiki · GitHub.” [Online]. Available: https://github.com/gentilkiwi/mimikatz/wiki/module-~-lsadump#secrets. [Accessed: 25-Feb-2017]
[9] mimikatz GitHub Repository, “Releases gentilkiwi/mimikatz GitHub” [Online]. Available: https://github.com/gentilkiwi/mimikatz/releases/download/2.1.0-20161126/mimikatz_trunk.zip. [Accessed: 25-Feb-2017]
[10] Microsoft Technet, “Introducing Managed Service Accounts.” [Online]. Available: https://technet.microsoft.com/en-us/library/dd560633(v=ws.10).aspx. [Accessed: 25-Feb-2017]
[11] N. Pyle, “Managed Service Accounts: Understanding, Implementing, Best Practices, and Troubleshooting.” [Online]. Available: https://blogs.technet.microsoft.com/askds/2009/09/10/managed-service-accounts-understanding-implementing-best-practices-and-troubleshooting/. [Accessed: 25-Feb-2017]
[12] Microsoft Technet, “Create the Key Distribution Services KDS Root Key.” [Online]. Available: https://technet.microsoft.com/en-us/library/jj128430(v=ws.11).aspx. [Accessed: 25-Feb-2017]
[13] D. Symalla, “Windows Server 2012: Group Managed Service Accounts.” [Online]. Available: https://blogs.technet.microsoft.com/askpfeplat/2012/12/16/windows-server-2012-group-managed-service-accounts/. [Accessed: 25-Feb-2017]
[14] B. Lich, “Protect derived domain credentials with Credential Guard (Windows 10).” [Online]. Available: https://technet.microsoft.com/en-us/itpro/windows/keep-secure/credential-guard. [Accessed: 25-Feb-2017]
[15] A. de Zylva, “Windows 10 Device Guard and Credential Guard Demystified,” *Ash’s Blog*. [Online]. Available: https://blogs.technet.microsoft.com/ash/2016/03/02/windows-10-device-guard-and-credential-guard-demystified/. [Accessed: 25-Feb-2017]

Appendix A: Active Directory Configuration

The steps to configure active directory were followed from the Microsoft Active Directory configuration documentation for MSDN (Microsoft Developer Network).

https://msdn.microsoft.com/en-us/library/ee799699(v=cs.20).aspx

The Active Directory domain functional level and forest functional level were chosen to support the current OS functional level. For instance, on a Windows 2008R2 server, we configured Windows 2008R2 as domain and forest functional level.

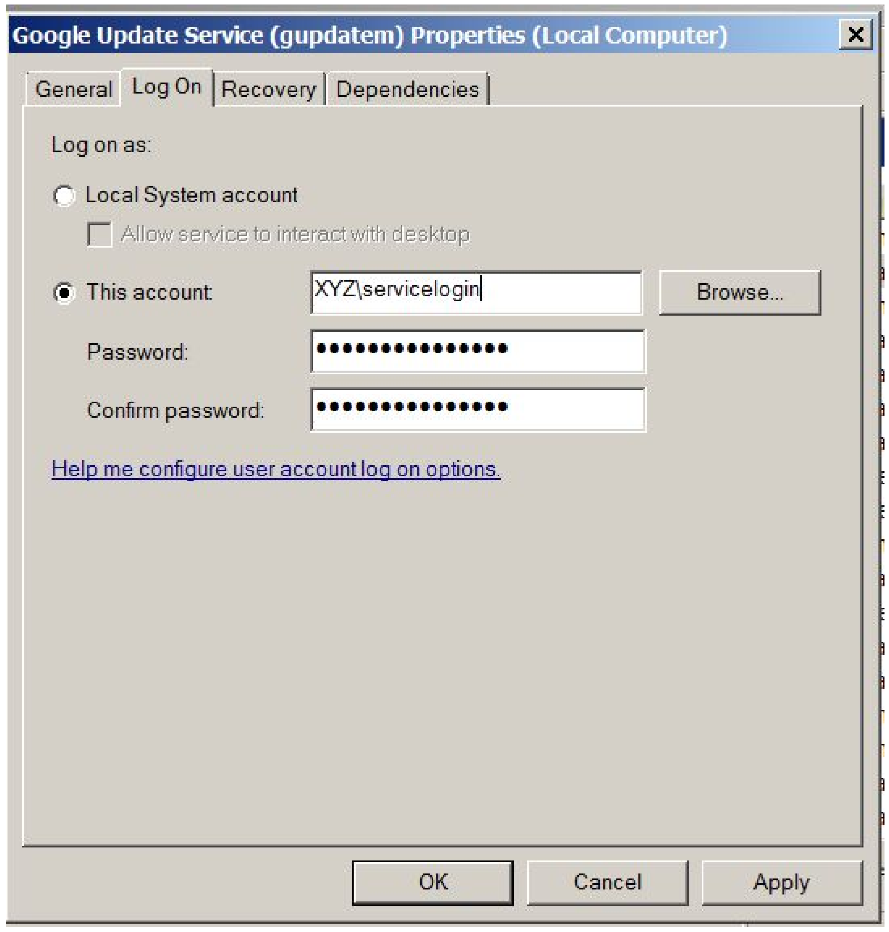
Appendix B: Machine B Client machine VM hardware details

Client Operating System	**RAM**	**Hard Drive Space**
Windows 2008	1 GB	40 GB
Windows 2008 R2	2 GB	40 GB
Windows 2012	2 GB	60 GB
Windows 2012 R2	2 GB	60 GB
Windows 7 Professional	1 GB	60 GB
Windows 8	2 GB	60 GB

Appendix C: Managed Service Accounts and Credential Guard

1. Official Documentation: Protect derived domain credentials with Credential Guard
2. Group Managed Service Account: Overview
   Getting started with gMSA
3. Windows Managed Service Account:
   Official Documentation: Introducing Managed Service Accounts
   Service Accounts Step-by-Step Guide
   Managed Service Accounts: Understanding, Implementing, Best Practices, and Troubleshooting

Appendix C: Screencapture for the configured Machine B


Google Update Service (gupdatem) Properties (Local Computer)

Log on as:

Local System account

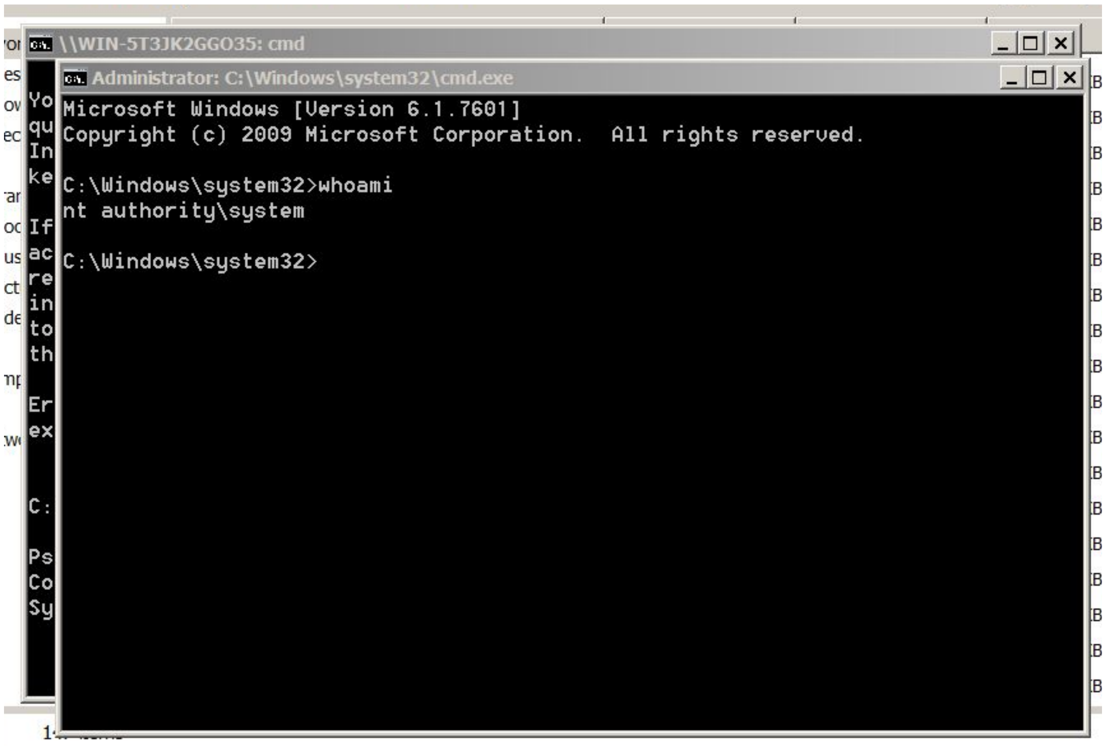
Allow service to interact with desktop

[✓ This account:] XYZ\servicelogin

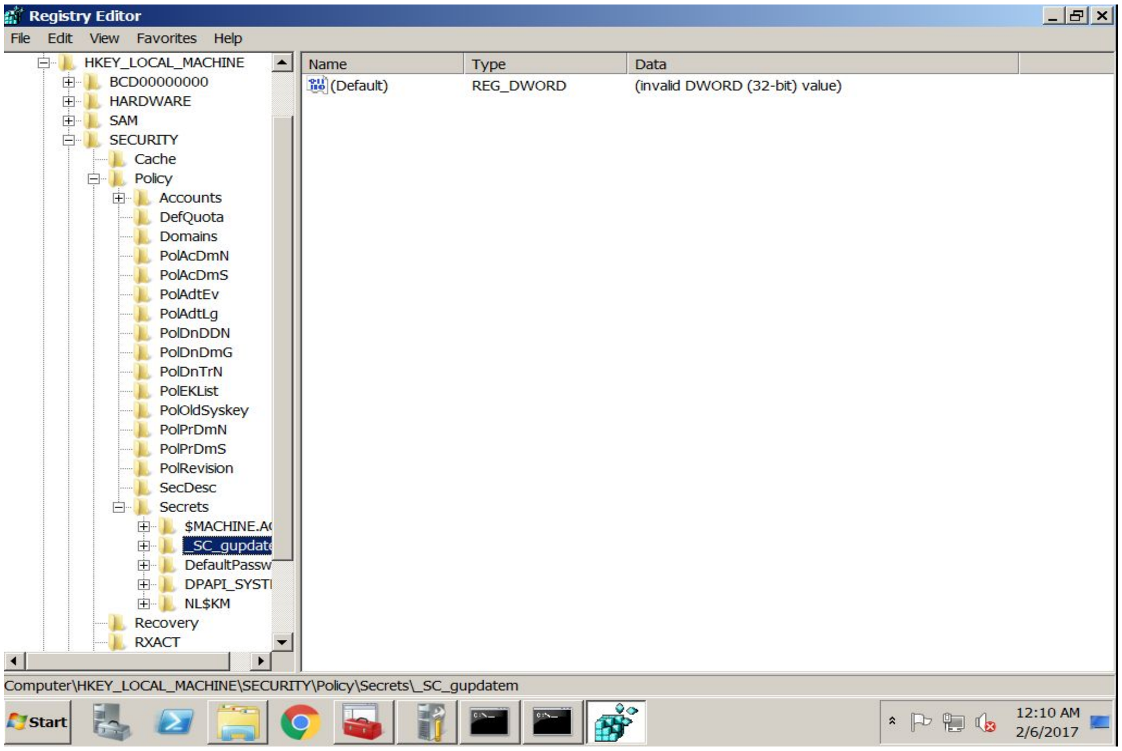
Password: •••••••••••••••

Confirm password: •••••••••••••••

Figure 1: Service configured with XYZ\servicelogin as log on account

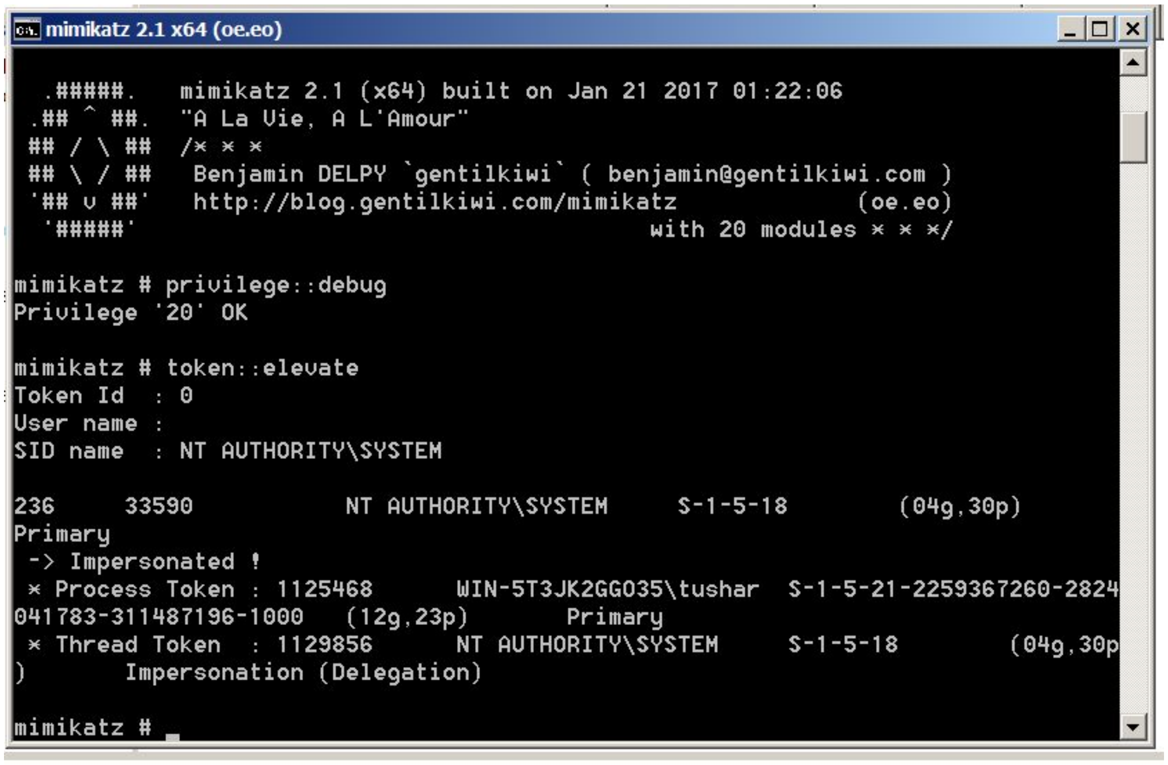


```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>whoami
nt authority\system

C:\Windows\system32>
```



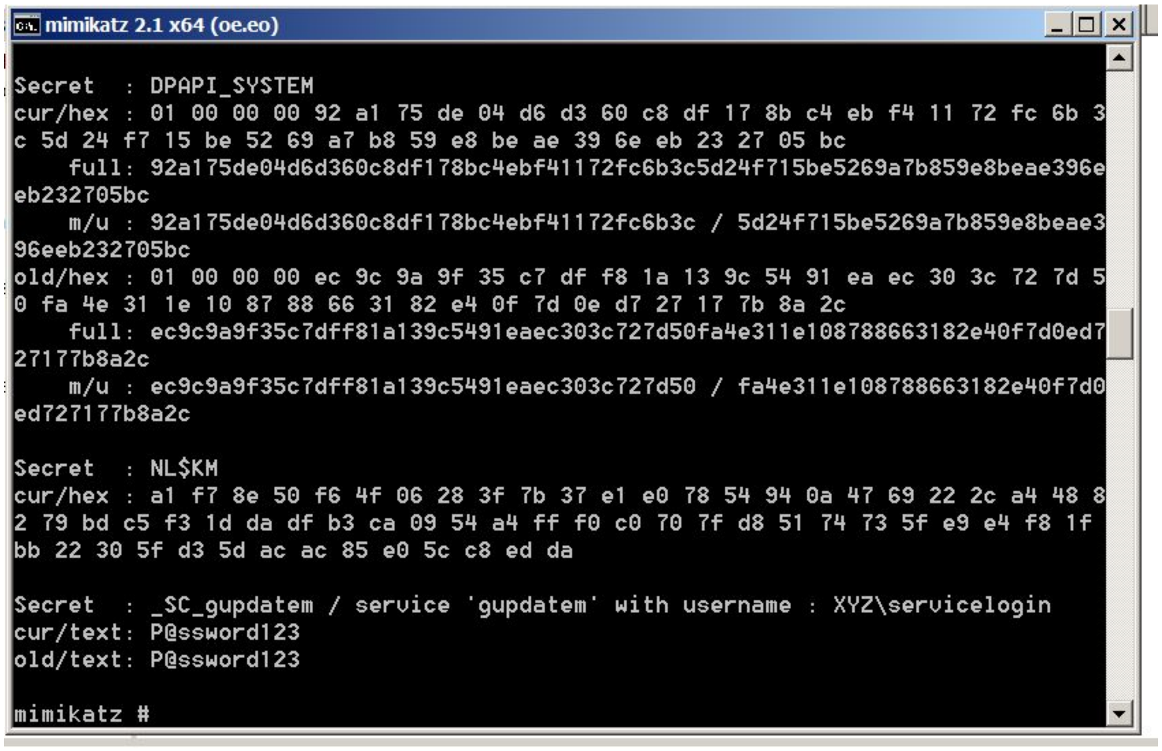
Name	Type	Data
(Default)	REG_DWORD	(invalid DWORD (32-bit) value)

Computer\HKEY_LOCAL_MACHINE\SECURITY\Policy\Secrets_SC_gupdatem



mimikatz 2.1 x64 (oe.eo)

```
  .#####.   mimikatz 2.1 (x64) built on Jan 21 2017 01:22:06
 .## ^ ##.  "A La Vie, A L'Amour"
 ## / \ ##  /* * *
 ## \ / ##   Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 '## v ##'   http://blog.gentilkiwi.com/mimikatz             (oe.eo)
  '#####'                                   with 20 modules * * */

mimikatz # privilege::debug
Privilege '20' OK

mimikatz # token::elevate
Token Id  : 0
User name :
SID name  : NT AUTHORITY\SYSTEM

236     33590           NT AUTHORITY\SYSTEM     S-1-5-18        (04g,30p)
Primary
 -> Impersonated !
 * Process Token : 1125468      WIN-5T3JK2GG035\tushar  S-1-5-21-2259367260-2824
041783-311487196-1000   (12g,23p)       Primary
 * Thread Token  : 1129856      NT AUTHORITY\SYSTEM     S-1-5-18        (04g,30p
)       Impersonation (Delegation)

mimikatz # _
```



mimikatz 2.1 x64 (oe.eo)

```
Secret  : DPAPI_SYSTEM
cur/hex : 01 00 00 00 92 a1 75 de 04 d6 d3 60 c8 df 17 8b c4 eb f4 11 72 fc 6b 3
c 5d 24 f7 15 be 52 69 a7 b8 59 e8 be ae 39 6e eb 23 27 05 bc
    full: 92a175de04d6d360c8df178bc4ebf41172fc6b3c5d24f715be5269a7b859e8beae396e
eb232705bc
    m/u : 92a175de04d6d360c8df178bc4ebf41172fc6b3c / 5d24f715be5269a7b859e8beae3
96eeb232705bc
old/hex : 01 00 00 00 ec 9c 9a 9f 35 c7 df f8 1a 13 9c 54 91 ea ec 30 3c 72 7d 5
0 fa 4e 31 1e 10 87 88 66 31 82 e4 0f 7d 0e d7 27 17 7b 8a 2c
    full: ec9c9a9f35c7dff81a139c5491eaec303c727d50fa4e311e108788663182e40f7d0ed7
27177b8a2c
    m/u : ec9c9a9f35c7dff81a139c5491eaec303c727d50 / fa4e311e108788663182e40f7d0
ed727177b8a2c

Secret  : NL$KM
cur/hex : a1 f7 8e 50 f6 4f 06 28 3f 7b 37 e1 e0 78 54 94 0a 47 69 22 2c a4 48 8
2 79 bd c5 f3 1d da df b3 ca 09 54 a4 ff f0 c0 70 7f d8 51 74 73 5f e9 e4 f8 1f
bb 22 30 5f d3 5d ac ac 85 e0 5c c8 ed da

Secret  : _SC_gupdatem / service 'gupdatem' with username : XYZ\servicelogin
cur/text: P@ssword123
old/text: P@ssword123

mimikatz #
```