

From Crash to Compromise: Unlocking the Potential of Windows Crash Dumps in Offensive Security

Author: Jason Mull, research@jasonmull.com

Advisor: Lee Crognale

Accepted: April 11, 2025

Abstract

Windows crash dump files, frequently overlooked in offensive contexts despite their forensic value, contain several sensitive elements that threat actors can exploit for privilege escalation, credential harvesting, lateral movement, and data exfiltration. Furthermore, the operating system already generates these crash dumps, so a threat actor does not need to risk discovery by dumping sensitive processes using noisier, more traditional methods. Offensive security education often emphasizes exploit development, network pivoting, and credential theft, but largely overlooks the value of memory forensics in post-exploitation scenarios. This research explores how offensive security practitioners can incorporate crash dump analysis into their workflows to extract sensitive data such as plaintext credentials, encryption keys, and files from memory. It also investigates methods to detect the creation of crash dump files, allowing defensive practitioners to identify and respond to their presence within an organization.

1. Introduction

On July 11th, 2023, news broke of a malicious campaign targeting the Exchange Online environments of multiple organizations worldwide. This campaign granted Storm-0558, a China-based threat actor, access to the email accounts of 22 organizations, including the U.S. Department of State, the U.S. Department of Commerce, and the U.S. House of Representatives (Cyber Safety Review Board, 2024).

During the incident investigation, Microsoft discovered that a consumer signing system crash in 2021 generated a crash dump file. While the crash dump generation process should have removed sensitive data, an error led to a sensitive signing key existing within the crash dump. Furthermore, this crash dump file was later moved from an isolated production network into a debugging environment within Microsoft's corporate network. The later compromise of a Microsoft engineer's corporate account ultimately led to Storm-0558's access to Microsoft's debugging environment and the crash dump (Microsoft Security Response Center, 2024).

As a result of this incident, Storm-0558, over six weeks, accessed the Exchange Online mailboxes of 503 individuals worldwide and exfiltrated approximately 60,000 emails from the U.S. Department of State alone (Cyber Safety Review Board, 2024). Several failures led to this compromise; however, one aspect of this incident that deserves greater attention is the content and handling of crash dump files and the sensitive information that could exist within these crash dumps.

1.1. Background & Purpose

This malicious campaign underscores what any experienced offensive practitioner understands well—privilege escalation within an environment frequently occurs due to sensitive data loosely secured within a network that is unknown or forgotten to the organization. It often comes in the form of passwords in configuration files or social security numbers on an unencrypted corporate endpoint. For Microsoft, it came as a crash dump, stored in a debugging environment within Microsoft's internet-connected corporate network, containing a signing key that was not redacted.

Multiple instances of research exist regarding the contents of memory dumps, both for offensive and forensic purposes (Ligh, Case, Levy, & Walters, 2014). Offensively, much of this research focuses on a threat actor dumping known sensitive processes, such as LSASS.exe, to obtain credential data (Delpy, 2019). These actions typically require the threat actor to initiate a process dump as part of their Tactics, Techniques, and Procedures (TTPs). As these are frequent behaviors of threat actors, these activities and associated detections and mitigations are well-documented (MITRE, 2024).

Crash dumps, such as the one stolen by Storm-0558, are generated by the operating system due to a system or application crash (Marcho, 2019). These crash dumps could contain a treasure trove of information available to an offensive practitioner without executing a process dump and potentially triggering detections.

1.2. Objective

This research will investigate the types of sensitive data present within crash dump files generated by Microsoft Windows operating systems, as well as the different methods defensive practitioners can utilize to detect the existence of these crash dump files within their networks. The aim is to better understand what data is available to a threat actor and how defensive practitioners can tune their detection capabilities to detect the creation and manipulation of these files.

2. Research Method

2.1. Infrastructure Configuration

This research will focus on Windows Server 2022 and Windows 11 23H2 Enterprise crash dump behaviors. All testing occurred within virtual machines, and the virtualization platforms used for research were Proxmox and the Ludus Cyber Range project. The usage of Ludus provides a repeatable, script-driven lab provisioning process. Mandiant's FlareVM provided a platform for memory analysis tooling. A Microsoft 365

E5 Developer subscription enabled testing of Microsoft 365 web and desktop applications.

2.2. Tested Applications

The following test cases are frequently encountered in business environments and serve as a foundation for consistent and valuable data to examine during the analysis of each crash dump type.

- **Microsoft 365 Desktop Applications** – A PowerShell script will stage mock sensitive data in multiple formats, including Word, Excel, and PDF (APPENDIX A). The script will then email these documents using Microsoft Outlook. The aim is to determine what documents and Exchange Online credentials can be obtained from the different crash dump types.
- **Browser Secrets** – A second script will automate the login process to Exchange Online and validate if offensive practitioners can obtain Exchange Online authentication information, such as cookies, authentication tokens, or credentials, from Chrome and Firefox (APPENDIX B).
- **Operating System Credentials & Encryption Keys** – The LSASS.exe process is a frequent target for threat actors, containing user credentials, password hashes, Kerberos tickets, and other operating system secrets. While several detections exist to catch threat actors interacting with this process, this research will investigate if the same process information is obtainable by examining operating system-generated crash dumps.

2.3. Memory Analysis Process

2.3.1. Memory Dump Types

Microsoft operating systems allow for generating different crash dump types, providing flexibility for disk space utilization and crash dump generation times. These crash dumps are valuable for diagnosing system failures, analyzing memory content at the time of a crash, and uncovering vulnerabilities or unwanted system behaviors.

Complete Memory Dumps capture all physical memory the operating system uses during a system crash. Complete Memory Dumps also include the contents of virtual machine memory if the endpoint is a virtual machine host (Microsoft, 2021). This format provides the most value to an offensive practitioner.

Successful generation of a Complete Memory Dump requires a page file on the boot drive large enough to hold all physical RAM plus 1 megabyte. Disk space of the same size to store the generated memory dump is also required (Microsoft, 2025).

Active Memory Dumps strike a middle ground between Complete Memory Dumps and Automatic Memory Dumps. This format includes user-mode application memory, though it filters out information that is not likely to benefit troubleshooting to create a smaller dump file (Microsoft, 2021). This format also excludes the memory contents of virtual machines running on the endpoint, another distinguishing factor from Complete Memory Dumps.

Automatic Memory Dumps are the current default crash dump format on Windows Server 2022 and Windows 11. This format captures kernel-mode memory at the time of a crash. This data can include OS code and kernel mode drivers. In addition, it does not contain memory allocated to user-mode programs (Microsoft, 2025).

This dump format contains the same information as a Kernel Memory Dump. The difference between these two formats lies in the flexibility the operating system has in adjusting the system paging file to ensure a successful crash dump capture (Microsoft, 2021).

User-mode dumps are available on and above Windows Server 2008 and Windows Vista with Service Pack 1 (SP1) to generate crash dump files when a user-mode application crashes. This feature is not enabled by default. When enabled, however, user-mode crash dumps are stored in %LOCALAPPDATA%\CrashDumps by default (Microsoft, 2024).

2.3.2. Memory Dump Analysis

Memory analysis was conducted with three primary applications:

Jason Mull, research@jasonmull.com

- **Volatility Framework** – The Volatility Framework is an open-source memory forensics tool well-known throughout the Digital Forensics and Incident Response (DFIR) community (The Volatility Foundation, n.d.). This research will be using Volatility 3 v2.11.0.
- **MemProcFS** – MemProcFS, developed by Ulf Frisk, is a newer memory analysis tool that focuses on presenting memory artifacts in a file system-like structure (Frisk, MemProcFS (Version 5.13.4) [Source code], 2025). This research will use MemProcFS v5.13.4, and all crash dump files were accessed using the command `memprocfs -f <memory file> -forensic 1`. The `-forensic 1` parameter loads various forensic modules useful for file carving.
- **Pypykatz** – Pypykatz is an up-to-date Python implementation of Mimikatz (SkelSec, 2025). While not specifically a memory analysis tool, Pypykatz can parse the `LSASS.exe` process to uncover credentials stored in memory. This research will be using Pypykatz v0.6.11.

3. Findings

Since Complete Memory Dumps contain the most information, the research will first focus on this type to establish a best-case scenario of available data. Subsequently, the research will analyze the differences in data availability between Complete Memory Dumps, Active Memory Dumps, and Automatic Memory Dumps.

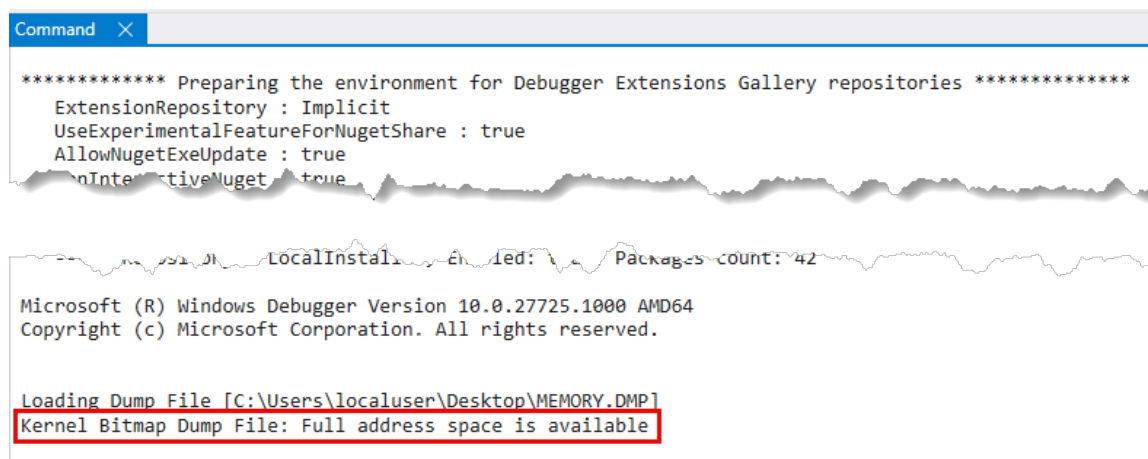
3.1. Crash Dump Similarities

Prior to examining the data contained within each crash dump type, this paper will explore the similarities among these crash dump types.

3.1.1. Identification

By default, all tested system crash dump types write to the same file, `%SystemRoot%\MEMORY.DMP`. While making assumptions about the crash dump type

based on file size is possible, there is no direct way to distinguish between the file types without examining the contents. The most direct way to obtain this information is to open the crash dump in WinDbg. Within the Command tab, the highlighted text indicates the dump type:

A screenshot of the WinDbg Command window. The window title is 'Command'. The text inside shows the process of preparing the environment for debugger extensions. At the bottom, the text 'Loading Dump File [C:\Users\localuser\Desktop\MEMORY.DMP]' is followed by 'Kernel Bitmap Dump File: Full address space is available', which is highlighted with a red rectangular box.

```
Command X
***** Preparing the environment for Debugger Extensions Gallery repositories *****
ExtensionRepository : Implicit
UseExperimentalFeatureForNugetShare : true
AllowNugetExeUpdate : true
InteractiveNuget : true

LocalInstall Enabled: Packages count: 42

Microsoft (R) Windows Debugger Version 10.0.27725.1000 AMD64
Copyright (c) Microsoft Corporation. All rights reserved.

Loading Dump File [C:\Users\localuser\Desktop\MEMORY.DMP]
Kernel Bitmap Dump File: Full address space is available
```

Figure 1: Crash Dump Identification

Commonly seen identifiers include:

- **Complete Memory Dump** – “Kernel Bitmap Dump File: Full address space is available.”
- **Active Memory Dump** – “Kernel Bitmap Dump File: Active memory is available.”
- **Automatic Memory Dump** – “Kernel Bitmap Dump File: Kernel address space is available. User address space may not be available.”
- **User-mode Memory Dump** – “User Mini Dump File: Only registers, stack, and portions of memory are available.”

3.1.2. Crash Dump Permissions

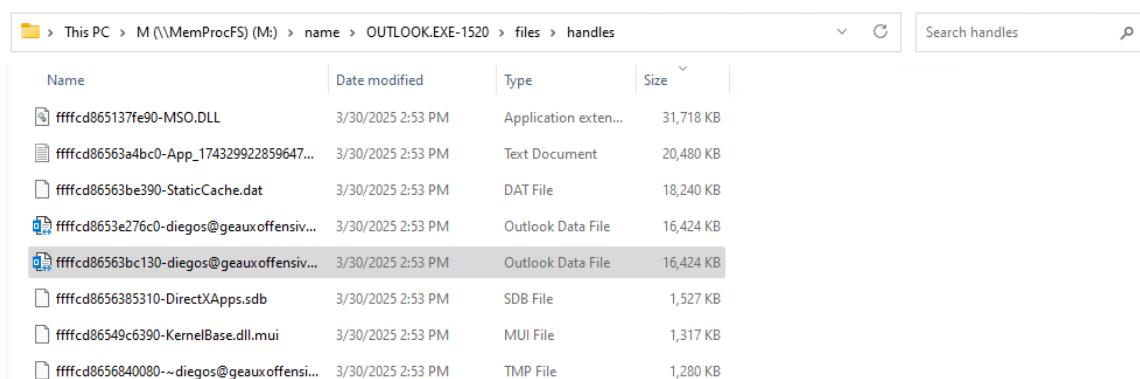
As noted previously, all system crash dumps write to %SystemRoot%\MEMORY.DMP by default. Also, by default, access to this file is limited to SYSTEM and the Local Administrators group. This is a critical point, as it establishes that a threat actor must have these permissions before obtaining this file from

its default location. It also demonstrates the importance of ensuring proper access controls on the crash dump file, especially if moving the file to a different location.

3.2. Microsoft 365 Desktop Applications

The Microsoft 365 desktop suite is ubiquitous throughout organizations and government entities worldwide. As these applications are the foundation of many business tasks, they can contain a wealth of sensitive information.

Using Volatility and MemProcFS, obtaining file system listings and recovering documents from the Complete Memory Dump was possible. File acquisition was accomplished by extracting the user's Outlook OST file and reviewing files carved from memory. MemProcFS makes it straightforward to browse through the memory dump in a file-tree format to explore data accessible within each process. When browsing the `OUTLOOK.EXE` process and reviewing accessible files, the OST file is readily available.



Name	Date modified	Type	Size
ffffcd865137fe90-MSO.DLL	3/30/2025 2:53 PM	Application exten...	31,718 KB
ffffcd86563a4bc0-App_174329922859647...	3/30/2025 2:53 PM	Text Document	20,480 KB
ffffcd86563be390-StaticCache.dat	3/30/2025 2:53 PM	DAT File	18,240 KB
ffffcd8653e276c0-diegos@geauxoffensiv...	3/30/2025 2:53 PM	Outlook Data File	16,424 KB
ffffcd86563bc130-diegos@geauxoffensiv...	3/30/2025 2:53 PM	Outlook Data File	16,424 KB
ffffcd8656385310-DirectXApps.sdb	3/30/2025 2:53 PM	SDB File	1,527 KB
ffffcd86549c6390-KernelBase.dll.mui	3/30/2025 2:53 PM	MUI File	1,317 KB
ffffcd8656840080-~diegos@geauxoffensi...	3/30/2025 2:53 PM	TMP File	1,280 KB

Figure 2: Complete Memory Dump - OUTLOOK.EXE Process Files

The OST file is readable with any OST file recovery utilities available online. When opened with Kernel OST Viewer, emails and associated attachments are available for viewing.

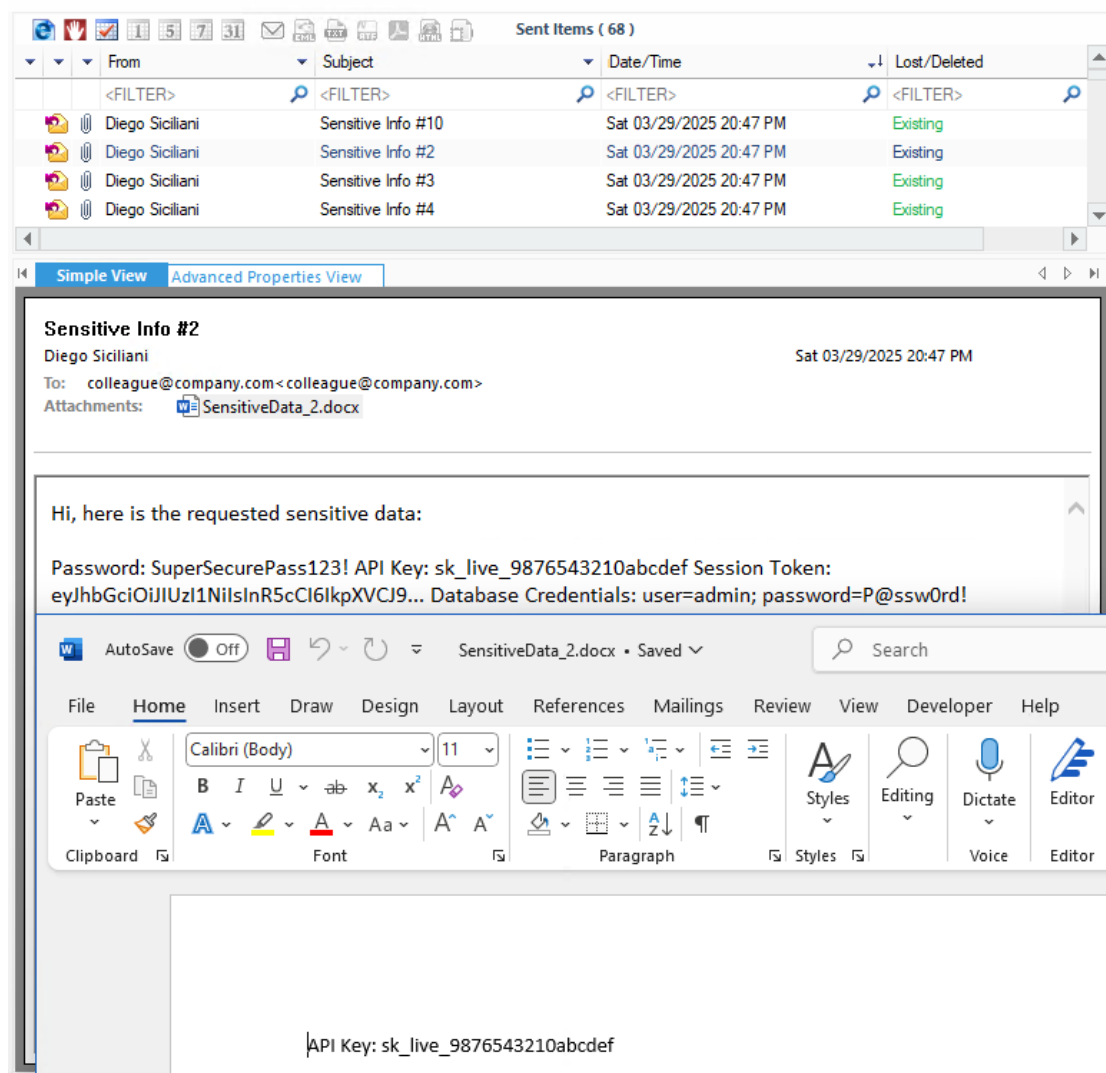


Figure 3: Complete Memory Dump - OST File Access

It is also possible to obtain documents stored in memory using the MemProcFS forensic modules. The MemProcFS mount point contains two folders, forensic/files and forensic/ntfs. These folders contain a listing of all files carved by MemProcFS and a file directory to access those files. The MemProcFS documentation states this is a best-effort reconstruction based on recoverable file objects in the kernel and the NTFS Master File Table (MFT) (Frisk, The forensic directory, 2023). As a result, it is possible that recovered files could have errors. That said, obtaining documents through these forensic modules was possible.

Jason Mull, research@jasonmull.com

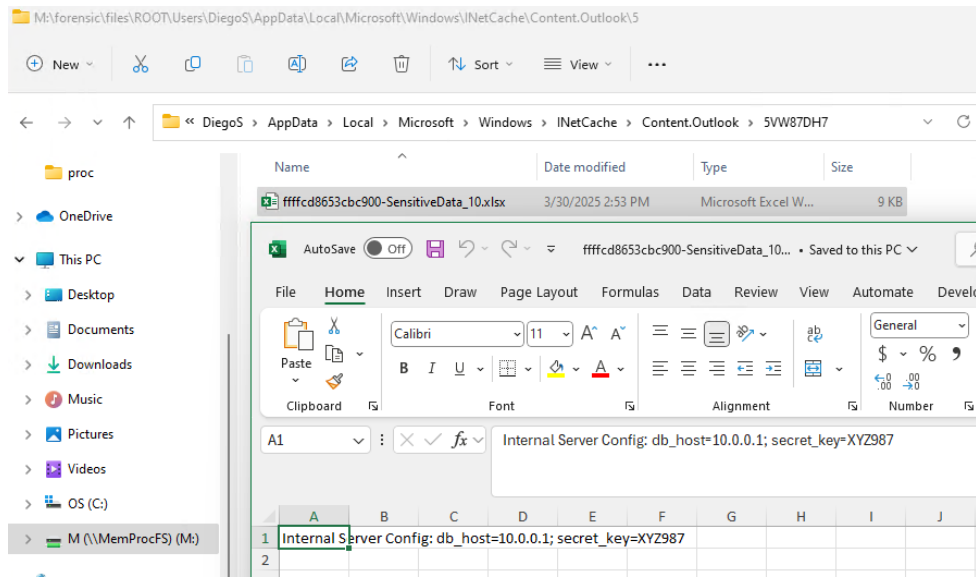


Figure 4: Complete Memory Dump - MemProcFS Forensic Module Usage

In addition, it was possible to use the Volatility Framework to search for and extract interesting files using two plugins. The first plugin, `windows.filescan.FileScan`, searches the crash dump for files. It provides the file name, path, and the file's location in memory. The second plugin, `windows.dumpfiles.DumpFiles`, extracts the file located at the defined memory address. These files can be interacted with as usual once extracted.

```
localuser@GEAUX-FLARE: $ vol -f "/mnt/c/Users/localuser/Desktop/Full Dump/MEMORY.DMP" windows.filescan.FileScan | grep "SensitiveData"
0xcd8651a843f0 0\Users\DiegoS\Documents\GeneratedAttachments\SensitiveData_6.docx
0xcd8651d46180 \Users\DiegoS\Documents\GeneratedAttachments\SensitiveData_5.xlsx
0xcd8651d461c0 \Users\DiegoS\Documents\GeneratedAttachments\SensitiveData_1.docx
0xcd8651d4d520 \Users\DiegoS\Documents\GeneratedAttachments\SensitiveData_8.xlsx
0xcd8653cbc900 \Users\DiegoS\AppData\Local\Microsoft\Windows\INetCache\Content.Outlook\5VW87DH7\SensitiveData_10.xlsx
0xcd8656461320 \Users\DiegoS\Documents\GeneratedAttachments\SensitiveData_5.txt
0xcd86564657e0 \Users\DiegoS\Documents\GeneratedAttachments\SensitiveData_1.txt
0xcd8656469980 \Users\DiegoS\Documents\GeneratedAttachments\SensitiveData_7.txt
localuser@GEAUX-FLARE: $ vol -f "/mnt/c/Users/localuser/Desktop/Full Dump/MEMORY.DMP" windows.dumpfiles.DumpFiles --virtaddr 0xcd8656461320
Volatility 3 Framework 2.11.0
Progress: 100.00 PDB scanning finished
Cache FileObject FileName Result
DataSectionObject 0xcd8656461320 SensitiveData_5.txt file.0xcd8656461320.0xcd8656a15e90.DataSectionObject.SensitiveData_5.txt-1.dat
localuser@GEAUX-FLARE: $ cat file.0xcd8656461320.0xcd8656a15e90.DataSectionObject.SensitiveData_5.txt-1.dat
Encryption Key: 0x4E5342AC3FA1994B...
```

Figure 5: Complete Memory Dump - Volatility File Extraction

While specialized software helps extract interesting files, it is not a requirement. Finding interesting data can be as simple as running `strings` against the dump file and filtering for the desired search criteria. While it might not be as clean a solution as extracting the file with Volatility or MemProcFS, it could be a quick first step in looking for interesting data.

Jason Mull, research@jasonmull.com

```
localuser@GEAUX-FLARE:~$ strings "/mnt/c/Users/localuser/Desktop/Full Dump/MEMORY.DMP" | grep -E 'Credit Card|API Key'
o #8) Credit Card Info: 4111-1111-1111-1111 Exp: 12/26 CVV: 123
/SensitiveData_8.txt (Sensitive Info #8) Credit Card Info: 4111-1111-1111-1111 Exp: 12/26 CVV: 123
Password: SuperSecurePass123! API Key: sk_live_9876543210abcdef Session Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
Database Credentials: user=admin; password=P@ssw0rd! Encryption Key: 0x4E5342AC3FA1994B... SSH Private Key: -----BEGIN
RSA PRIVATE KEY-----... OAuth Access Token: ya29.a0AfH6SMC... Credit Card Info: 4111-1111-1111-1111 Exp: 12/26 CVV: 12
3 JWT Token: eyJhbGciOiJIUzI1Ni... Internal Server Config: db_host=10.0.0.1; secret_key=XYZ987[4-1]
/SensitiveData_2.pdf (Sensitive Info #2) API Key: sk_live_9876543210abcdef
Password: SuperSecurePass123! API Key: sk_live_9876543210abcdef Session Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
Database Credentials: user=admin; password=P@ssw0rd! Encryption Key: 0x4E5342AC3FA1994B... SSH Private Key: -----BEGIN
RSA PRIVATE KEY-----... OAuth Access Token: ya29.a0AfH6SMC... Credit Card Info: 4111-1111-1111-1111 Exp: 12/26 CVV: 12
3 JWT Token: eyJhbGciOiJIUzI1Ni... Internal Server Config: db_host=10.0.0.1; secret_key=XYZ987[2-1]
ord=P@ssw0rd! Encryption Key: 0x4E5342AC3FA1994B... SSH Private Key: -----BEGIN RSA PRIVATE KEY-----... OAuth Access To
ken: ya29.a0AfH6SMC... Credit Card Info: 41
Password: SuperSecurePass123! API Key: sk_live_9876543210abcdef Session Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
Database Credentials: user=admin; password=P@ssw0rd! Encryption Key: 0x4E5342AC3FA1994B... SSH Private Key: -----BEGIN
RSA PRIVATE KEY-----... OAuth Access Token: ya29.a0AfH6SMC... Credit Card Info: 4111-1111-1111-1111 Exp: 12/26 CVV: 12
3 JWT Token: eyJhbGciOiJIUzI1Ni... Internal Server Config: db_host=10.0.0.1; secret_key=XYZ987[5-1]
```

Figure 6: Complete Memory Dump - Strings Enumeration

3.3. Browser Secrets

Session cookies are a popular target for threat actors. If obtained, threat actors can use these session cookies to gain access to web services as authenticated users without needing a username, password, and MFA token (MITRE, 2024).

3.3.1. Firefox Cookie Access - SQLite Database Files

Firefox stores cookies in a SQLite database within the user’s profile folder. When opening the database, the cookies exist in an unencrypted state. Specifically, ESTSAUTHPERSISTENT cookies can be observed. These cookies contain a user’s Entra ID session information and are used to facilitate Single Sign-On (Microsoft, 2024).

↑ << files >> ROOT > Users > Diego > AppData > Roaming > Mozilla > Firefox > Profiles > 2i8yrj7z.default

Name	Date modified	Type	Size
fffc8653e1c950-cookies.sqlite	3/30/2025 2:53 PM	SQLITE File	512 KB
fffc8653e1e890-cookies.sqlite-shm	3/30/2025 2:53 PM	SQLITE-SHM File	32 KB
fffc8653e07af0-compatibility.ini	3/30/2025 2:53 PM	Configuration sett...	1 KB
fffc8653e07e10-pkcs11.txt	3/30/2025 2:53 PM	Text Document	1 KB

DB Browser for SQLite - C:\Users\localuser\Desktop\fffc8653e1c950-cookies.sqlite

File Edit View Tools Help

New Database Open Database Write Changes Revert Changes Undo Open Project Save Project Attach Database Close Database

Database Structure Browse Data Edit Pragma Execute SQL

Table: moz_cookies

	name	value	host	path	expiry
1	ESTSAUTHPERSISTENT	1.AXwA1sRHK2B1C0ihyTmq30RxEyC_mZE_o...	.login.microsoftonline.com	/	1751075108 17

Figure 7: Complete Dump - ESTSAUTHPERSISTENT Cookie Visibility in Firefox

3.3.2. Chrome Cookie Access – Process Memory

Attempts to access Chrome's Cookies and Login Data SQLite databases led to the discovery that Chrome encrypts the Login Data file with DPAPI (Harris, Detecting Browser Data Theft Using Windows Event Logs, 2024) and the Cookies file with an App-Bound Encryption Key (Harris, Improving the Security of Chrome Cookies on Windows, 2024). Understanding DPAPI and Windows credential storage will help offensive practitioners gain access to this data.

3.4. Operating System Credentials & Encryption Keys

Credentials are a favorite target of offensive practitioners. Credential acquisition frequently comes in the form of dumping either registry hives or the LSASS .EXE process on an endpoint. While both are effective acquisition methods, their popularity has led to well-crafted detection rules. Crash dumps triggered by a faulty process can provide a stealthier method of credential acquisition where a threat actor can perform all credential harvesting work away from the target infrastructure.

3.4.1. Credential Access Artifacts

Multiple artifacts must be collected to successfully extract credential and encryption information from crash dumps.

The SAM (Security Account Manager) registry hive contains hashed versions of passwords for all local user accounts on a Windows endpoint (Vidas, Kaplan, & Geiger, 2014). Acquisition of this file can provide password hashes that can be cracked or forwarded to other endpoints using a pass-the-hash attack, enabling lateral movement throughout the target infrastructure. However, accessing this registry hive's content is not as simple as opening the file. A bootkey from the SYSTEM registry hive is required. The bootkey is a value stored across multiple registry keys under `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa\` (Vidas, Kaplan, & Geiger, 2014). Given access to both hives, tools such as Mimikatz and Pypykatz can expose the hashes stored in the SAM hive. These hive files are available

M:\registry\hive files.

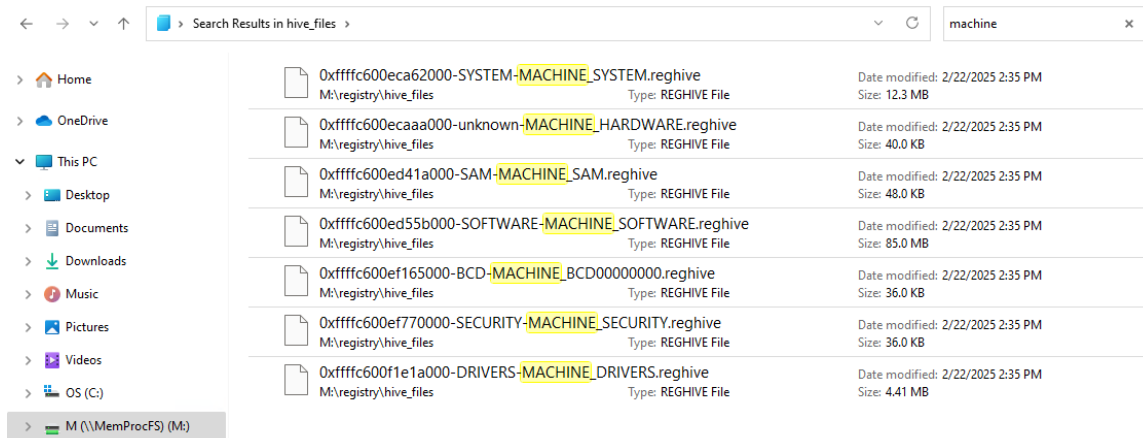


Figure 8: Complete Memory Dump - Registry Hive Location Within MemProcFS

While the SAM hive contains local user hashes, these hashes do not provide access to Active Directory. A different registry hive, SECURITY, holds cached Active Directory domain credential hashes. As with the SAM hive, the credential data within SECURITY is encrypted and requires the bootkey contained within the SYSTEM hive to become readable (Vidas, Kaplan, & Geiger, 2014).

[illegible]

Figure 9: Complete Memory Dump - Credential Acquisition with Pypykatz

3.4.2. Chrome Saved Credentials

DPAPI (Data Protection Application Programming Interface) is a programming interface that allows third-party applications to encrypt data on the endpoint. To decrypt data encrypted with DPAPI, the following three components should be of concern to offensive practitioners (Bursztein & Picod, 2010):

- DPAPI Blob – This blob contains the encrypted data and a blob key to encrypt the data held within it.
- Master Key – The master key is responsible for managing the encryption of the blob key. Per-user master keys exist in %AppData%\Microsoft\Protect\<SID>. It is common to see multiple master key files in this directory, as keys rotate every three months. The Preferred file acts as a pointer to the latest master key iteration. These files exist within M:\forensic\files or M:\forensic\ntfs.
- Pre-Key – This key is responsible for decrypting the master key and is derived from the user’s password. As such, the strength of encryption is dependent on the strength of the user’s password.

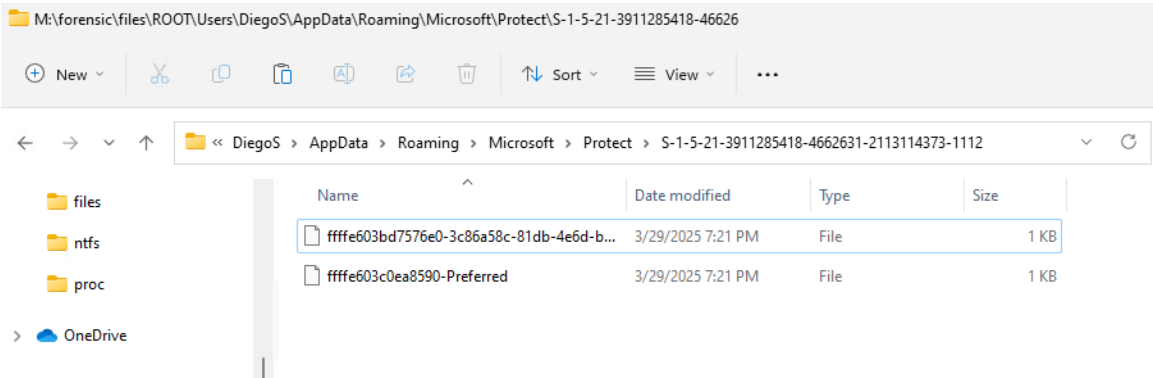


Figure 10: Complete Memory Dump - User Master Key File Location

Chromium browsers use DPAPI to encrypt sensitive browser data, including saved credentials (Harris, Detecting Browser Data Theft Using Windows Event Logs, 2024). Where this information was previously accessed in Firefox without needing to decrypt data, additional work is required to obtain logins and cookies from a Chromium

browser. To begin, three files are needed from the Complete Memory Dump. The forensic file carving capabilities of MemProcFS provide the most straightforward way to access this data, and the files exist within `M:\forensic\files` or `M:\forensic\ntfs`.

- **Cookies** – Browser cookies are encrypted and stored within the file `%LocalAppData%\Google\Chrome\User Data\Default\Network\Cookies`.
- **Login Data** – Saved credentials within the browser are encrypted and stored within the file `%LocalAppData%\Google\Chrome\User Data\Default>Login Data`.
- **Local State** -This file contains a key that encrypts Cookies and Login Data. The user's DPAPI master key encrypts the file itself.

Pypykatz can obtain most of the information needed for Chrome data decryption. The first step is to execute `pypykatz prekey password <sid> <password> -o <prekey output file>` while passing the user's SID and password as a parameter. The user's password is obtained by cracking the user's password hash, which was previously extracted from the registry. Outputting the pre-key to a file will simplify feeding the key into the following steps.

The next step is discovering the active master key using `pypykatz dpapi preferredkey <preferred file>`. The output of this command will point us to the correct master key file for the next steps.

With the pre-key and correct master key identified, the master key can now be decrypted. `Pypykatz dpapi masterkey <masterkey file> <prekey output file> -o <decrypted masterkey output file>` will output a decrypted master key that can decrypt the Chrome Local State file.

To decrypt the Login Data file, the decrypted master key is used to decrypt the Chrome encryption key held within Local State. The Local State file

contains an encryption key to decrypt the Login Data file. The command to execute this process is `pypykatz dpapi chrome -logindata <login data file> <decrypted masterkey output file> <local state file>`. With this command, saved browser usernames and passwords are output. Note that the password is represented as `b'<password>'`.

```
PS C:\analysis\chromedata > pypykatz dpapi prekey password S-1-5-21-3911285418-4662631-2113114373-1112 Spring2025 -o prekey
PS C:\analysis\chromedata > pypykatz dpapi preferredkey fffffe603c0ea8590-Preferred
[GUID] 3c86a58c-81db-4e6d-b539-3c0db035fdc4
PS C:\analysis\chromedata > pypykatz dpapi masterkey fffffe603bd7576e0-3c86a58c-81db-4e6d-b539-3c0db035fdc4 prekey -o mkf
PS C:\analysis\chromedata > pypykatz dpapi chrome --logindata "ffffcd8656805bb0-Login Data" mkf "Local State"
file: fffffcd8656805bb0-Login Data user: DiegoS@geauxoffensive.com pass: b'G)215990061298uw' url:
```

Figure 11: Complete Memory Dump - Chrome Password Acquisition

While passwords saved within Chrome are accessible using this method, Google has strengthened the protection around Cookie storage, starting with Chrome Version 127 (Harris, Improving the Security of Chrome Cookies on Windows, 2024). While tools such as ChromeKatz (Meckazin, 2024) can extract cookies from the memory of a dumped Chrome process, this research could not duplicate the same results for a Chrome process dump extracted from a system crash dump. Further investigation into why this issue occurs is a recommendation for further research.

3.5. Additional Crash Dump Types

Now that items of interest obtained from a Complete Memory Dump have been identified, the focus will shift to the smaller crash dump types to document the variations in artifacts present within each type.

3.5.1. Active Memory Dump

Microsoft describes the Active Memory Dump as “similar to a Complete Memory Dump, but it filters out pages that are not likely to be relevant to troubleshooting problems on the host machine” (Microsoft, 2021). When comparing the size of the crash dump files, the Complete Memory Dump was roughly 8,382 MB, which matches up with the memory assigned to the virtual machine. In contrast, the size of the Active Memory Dump is roughly 5,117 MB. Therefore, artifact loss is expected.

The first visible sign of artifact loss occurs when attempting to access files with the MemProcFS forensic module. While files do appear in `M:\forensic\files`, most are empty or corrupt. Furthermore, the list of files within `M:\forensic\ntfs` contains roughly 57,000 files in the Complete Memory Dump, compared to just over 900 files listed in the Active Memory Dump.

There is similar data loss when accessing the registry hive data contained with the Active Memory Dump. Running `pypykatz` against the `SYSTEM`, `SAM`, and `SECURITY` hives produces errors referencing missing registry values.

```
PS C:\analysis\active_dump > pypykatz registry --sam SAM.reg hive --security SECURITY.reg hive SYSTEM.reg hive
WARNING:pypykatz:SOFTWARE hive path not supplied! Parsing SOFTWARE will not work
Traceback (most recent call last):
  File "<frozen runpy>", line 198, in _run_module_as_main
  File "<frozen runpy>", line 88, in _run_code
  File "C:\Python313\Scripts\pypykatz.exe\__main__.py", line 7, in <module>
    __init__(main())
  File "C:\Python313\Scripts\pypykatz.exe\__main__.py", line 7, in __init__
    key = self.find_key(key_path, throw)
  File "C:\Python313\Lib\site-packages\aiowinreg\hive.py", line 132, in find_key
    raise Exception('Could not find subkey! Full path: %s Working path: %s Missing key name: %s' % (key_path, working_path, key))
Exception: Could not find subkey! Full path: SAM\Domains\Account Working path: \SAM Missing key name: Domains
```

Figure 12: Active Memory Dump - Missing Registry Hive Data

Even with limited file access, offensive practitioners can still search for sensitive strings in memory to get around automated file carving limitations. As in Section 3.2, `Strings` is a basic, yet valuable utility that can uncover specific text strings within memory. In this case, Microsoft 365 authentication cookies can be extracted from the Active Memory Dump.

```
PS C:\users\localuser\desktop\active_dump> strings MEMORY.DMP | Select-String "ESTSAUTH"

ESTSAUTH

ESTSAUTHPERSISTENT=1.AXwAme_N-B6jSkUTS-9XhPELWg0
bY2yxFRDwldmCLPTqP4wnT6TIq; ESTSAUTH=1.AXwAme_N-B6jSkUTS5F9XhPELWgIAAAAAAPEPzgAAAA
ESTSAUTHPERSISTENT=1.AXwAme_N-B6jSkUTS5F9XhPELWgIAAAAAAPEPzgAAAAAADMAOp8AA.AgABFwQAAABVrSpeuWamRam2jAF1XRQEAwD
s_wUA9P-NldDWT0eJC_2JqA-yd5kA7j7f965gdIH3kkZ2-29k5958IsPD29HbaLJ_M5mywc52oyf4k7eD9SfZYfjPM33dxwW4Hm98LwBwGrmZHf
aKhL0NNCaaZsGGmW3FuVh4tcnZ74QAp-bM5cXnRjEm4QrU_9L18AYoakqjk249rGtdPavETojcT2ZBwDE6KhoK2PtjIKosms24yELq1Qp0J5rTZ
BTMJIMdysD7u0hbZbj6MyOqZo_3651rJ6_t2hyz0tkhXJc8B-9BcUcFjdXE0s2CieZ9F5L05L6xRJD5qJDqPMXztEZZ-5VkiId4kyheHGFHS-Vfy
tqBEnOgEupuFRz6Dxrng-R7lfm8cBZMrjc7qMXJZw1xIB4NqmeH32stJF-G3ZsJV_0xyUNaS; domain=.login.microsoftonline.com;
expires=Wed, 04-Jun-2025 02:01:20 GMT; path=/; secure; HttpOnly; SameSite=None
```

Figure 13: Active Memory Dump - Searching for Useful Strings

Individual process minidumps are also still accessible within the Active Memory Dump. By default, MemProcFS does not extract the `LSASS.exe` process memory for

security reasons, but a forked version of the project makes this possible (0xNemo, 2025). Offensive practitioners can extract the `LSASS.exe` minidump file from within the crash dump, interact with it like any other `LSASS.exe` process dump, and obtain local and cached Active Directory hashes.

```
PS C:\analysis\active_dump > pypykatz lsa minidump minidump.dmp
INFO:pypykatz:Parsing file minidump.dmp
FILE: ===== minidump.dmp =====
== LogonSession ==
authentication_id 1723807 (1a4d9f)
session_id 0
username geaux_admin
domainname lab
logon_server GEAX-DC01
logon_time 2025-03-30T03:29:53.840001+00:00
sid S-1-5-21-3911285418-4662631-2113114373-1105
luid 1723807
== MSV ==
  Username: geaux_admin
  Domain: lab
  LM: NA
  NT: 5c5aefbcab1053c010bc9c1cfcc6f95d
  SHA1: b64bf7fc887121c5998c14d532b12eb1b0cc31b5
  DPAPI: 9eadda3bf791b3b2887f6531bc5ffa900000000
== Kerberos ==
  Username: geaux_admin
  Domain: LAB.GEAXOFFENSIVE.COM
  AES128 Key: 5c5aefbcab1053c010bc9c1cfcc6f95d
  AES256 Key: 6bf12011b327e5be8599f8b79d583a05ea6e006e43b9c6768a86cf75fc5f51c2
== DPAPI [1a4d9f]==
  luid 1723807
  key_guid 445a491c-3ba2-4399-b5ce-5c09d4e7547e
  masterkey 16f46f1b87ec48ee1dcd5d8812e4d7e95290d43e2a71fc51663cd62ef2b69145aacd105f1b7b9a8e418ec0c6f1f6ad39bb4358cfbe3d4
177185d5da76ada5a08
  sha1_masterkey bc3b8a7bc96d4e6357f7957043133a44cf9ff521
== LogonSession ==
authentication_id 866651 (d395b)
session_id 2
username DiegoS
domainname lab
logon_server GEAX-DC01
logon_time 2025-03-30T03:28:37.231013+00:00
sid S-1-5-21-3911285418-4662631-2113114373-1112
luid 866651
== MSV ==
  Username: DiegoS
  Domain: lab
  LM: NA
  NT: 045d0f10ff8f7211af43cd83c57dfbb1
  SHA1: 2c4e9ef651957956310d06cac656b3ba57c337b0
  DPAPI: b90c7bc25281de3227737afe935a162d00000000
== Kerberos ==
  Username: DiegoS
  Domain: LAB.GEAXOFFENSIVE.COM
  AES128 Key: 045d0f10ff8f7211af43cd83c57dfbb1
  AES256 Key: 00c0a6d4ac2a7f3aa085461d176fd32c6d11464aa84c5bf9651ad87a1134f017
== DPAPI [d395b]==
  luid 866651
  key_guid 3c86a58c-81db-4e6d-b539-3c0db035fcdc4
  masterkey 2f928fc2de78a57b9669bf9c739fdbclac664ce0842f4d6b41fc3c517110921d8b528548e3dcd1dfafae6a4a800353e9f39b69f2c05
72762e28e1bc2b3091e
  sha1_masterkey e3703fd596e831bfe0a1f823fa75f94481cf420c
```

Figure 14: Active Memory Dump - LSASS Process Enumeration

3.5.2. Automatic Memory Dump

The Automatic Memory Dump provides significantly less data to work with, coming in at roughly 700 MB within the testing environment, compared to 5 GB for the Active Memory Dump and 8 GB for the Complete Memory Dump. As with the Active

Memory Dump, the MemProcFS forensic module cannot carve the registry hive files successfully. In addition, process memory is no longer successfully carved.

Searching for sensitive data with `Strings` does not provide much, compared to the other crash dump types. However, valuable data, including a URL-encoded Microsoft 365 username and password and some sensitive documentation staged on the endpoint, still exists.

```
PS C:\Users\localuser\Desktop\Auto Dump > strings MEMORY.DMP | Select-String "geauxoffensive.com"

diegos@geauxoffensive.com
https://odc.officeapps.live.com/odc/v2.1/federationprovider?domain=ad.geauxoffensive.com
https://odc.officeapps.live.com/odc/v2.1/federationprovider?domain=ad.geauxoffensive.com

ad.geauxoffensive.com
ad.geauxoffensive.com
x<-DiegoS@GeauxOffensive.com - Default Outlook Profile.ost.tmp
os%40geauxoffensive.com&loginfmt=diegos%40geauxoffensive.com&type=11&LoginOptions=3&lrt=&lrtPartition=&hisR
egion=&hisScaleUnit=&passwd=G%29215990061298uw&ps=2&psRNGCDefaultType=&psRNGCEntropy=&psRNGCSLK=&canary=9xn
3Hg2AcWsnYrmQHUC0UcNWGCe%2FHVITJ5EIN1ddtI%3D3%3A1%3ACANARY%3ANmbEbIc2ZU0thzrVL2USKfeyUfrw3KKf05mFo2xvSsk%3
D&ctx=rQQIARAAjZE9bBjHAIbvOHq2tVrSxcBjNE6NB_fHfYDpAAeUAwQKXpkxLcDHnZzw8eHd8dt0d-zcwcGxiYlxMtXBzartJxObGBu7N
DVRy2LSUYiLV5_hybu9yfuuUbyXD93j_iGwM70cYfAs1GfpP6yVRU_obLPsUtnC4-Tq1--3AmcHJNNynJ4d8vLw3-lg3PziwzChLsp-L8TI
h4d131uSPCBjC5I8cAFZDIAPPC-IsigHJA54A02ZawSDMitynMhKksSZAQglVvJ1aEARCH5R0nEtZ8N9pyXMhC1zov92LRjYQrUetp196pC
MVuxgVFKH8bCSrrV0cYwGBh4ForAYHkwzSjgZwUJYbNdUsKV3EQM9LIA5NMbQLSwzXFLcwjyCh5nLEK8gKNqjEuDOWLz1VEKfsmVbUBJF
RUcqCnQfthqtIZtDVgpJ9yTh3oG-XIBHHPGpIyTTS2-makE9SeDSZVq1zTTH9GSLchcPoZUE86qTKqVSULb4Mu8PfTOX94eEBdaf43FD0dF
OHuEUXjnt41m8du8tRNXriXOC00P7_oIvaJ08SLm3w5N33qy40d7Z_xu4n34-3bryKfIKM5X3DUFNRNPhDAsFboVC20mNJjVYKa0oei-xJaQ
FpP-mJrhm01HXRDd_B5N7tH0Ib0wT3kIhLJy_DLN_qLJ59eIw4Ur3H58nTxZkhZp2KmbyFSZ22HMZs3Bbb3LhHaYebJrEM7SoN7p6zYtesR
MK5jt3d3dDzeIy5t_3n37uP_i84_ExfL9DGxEGioUqlrWaU2srbSgFVKGPtYMayimUkbWj-JYGA0Kdvn9tYc49RB_AQ2&hpgrequestid=5
d7b3bdd-46ef-4823-bc79-5d1aad42d600&FlowToken=AQABIQEAAABVrSpeuWamRam2jAF1XRQE4q7GAKJTWVmAjXbmqqz_LYVStpfwg
QEFLa58NjEIUCHU09vGoDH1MUVWrf9RnoEGt-vfdsk05-77YQenYOCZt6ZuB2DmZGC5IjkM_IJAM8dJFx5OcL2imyBnNbN_chHreFo9BFK3u
q-3V8uZcNmvpQxwa5W2cI6z2690h5yHUR7qoaqgzIjhYRDdDhJb3idZmJd1hMDv2zJN6jY24ys_V_4LxmbkzI6inDaDN0BgVdnSlkKaLC
DOPiWeednc0Yn_1wJL8c8EGa6rkjBk3hQaMa3TgbdxfMbNoLbawHQVvw2k6J0123WCtKAXnfalieqKBqft_sdSjUJJj5qoTiYj_w2okxYGL
S-JCMgN66yKDNMYvVehLzPZRLGd_IIO-IILcsMyulrQG9G0hxv_LinFUT57uGhskkOxFeVFY9I9dLKNA1ANP19ufQi6ek7cA-bqS6BK3dd7
y7akzg7h4ygtMadwvCwUnrXmQypC_Zngu-DHsEMiVkc0PyfjsGTX112IAA&PPSX=&NewUser=1&FoundMSAs=&fpost=0&si21=0&Cookie
Disclosure=0&IsFidoSupported=1&isSignupPost=0&dfpArtifact=5i19=10224
```

Figure 15: Automatic Memory Dump - URL Encoded Credentials

```
PS C:\users\localuser\Desktop\auto dump > strings MEMORY.DMP | Select-String "sensitive"

note: only register values is case sensitive, key is

SensitiveData_10.docx
4Sensitive Info #10 : SensitiveData_10.docx (/DiegoS@geauxoffensive.com/Sent Items)/
SensitiveData_10.docx
SensitiveData_10.docx (Sensitive Info #10)J
SensitiveData_10.docx (Sensitive Info #10)
/SensitiveData_10.docx (Sensitive Info #10) Internal Server Config: db_host=10.0.0.1; secret_key=XYZ987
:SensitiveData_10.docx
!/DiegoS@geauxoffensive.com/Sent Items/Sensitive Info #10 : SensitiveData_10.docx-
ISensitiveData_9.pdf (Sensitive Info #9)
ESensitiveData_9.pdf
4Sensitive Info #9 : SensitiveData_9.pdf (/DiegoS@geauxoffensive.com/Sent Items)/
SensitiveData_9.pdf
SensitiveData_9.pdf (Sensitive Info #9)0
SensitiveData_9.pdf (Sensitive Info #9)
/SensitiveData_9.pdf (Sensitive Info #9) JWT Token: eyJhbGciOiJIUzI1Ni...
(<SensitiveData_10.txt
SensitiveData_
```

Figure 16: Automatic Memory Dump - Sensitive Strings

4. Recommendations & Detection Options

This research has validated the ability to access sensitive data valuable to a threat actor during each stage of memory analysis. While some luck and good timing are likely needed to obtain sensitive data in a production scenario, the potential to obtain this data exists, especially when dealing with larger crash dump formats.

The existence of unencrypted operating system secrets on an endpoint poses an interesting question: Is the generation of crash dumps as a default practice still advisable? To answer this question, an organization should consider the following points:

- What is an organization's realistic capability to proactively review crash dumps for troubleshooting purposes? Put more directly, will an organization do anything with the generated crash dump files, or will they sit undetected on an endpoint without serving any real purpose? If there is no intention to investigate these files proactively, it would be more advisable to harden systems by disabling crash dump generation by default and enabling the functionality in specific circumstances where one is actively researching a system fault.
- Is there any forensic benefit to retaining crash dumps? Zong Fu Chua proposes offloading complete crash dumps throughout the organization to a centralized share to investigate the crash dumps for evidence of malware or other intrusion (Chua, 2015). While there is clear forensic value to these crash dumps, a couple of concerns arise:
 1. The proposal suggests storing crash dumps from endpoints throughout an organization in a centralized location. While there is an argument that this is more secure than retaining this data on each endpoint, the organization risks exposing the operating system secrets of multiple endpoints if the centralized share is not secure.
 2. Several open-source and commercial products offer purpose-built mechanisms for endpoint forensic triage capture. Velociraptor, KAPE, and Kansa can all capture forensic triage artifacts from endpoints at

scale. In addition, Sentinel One even offers the ability to perform these captures automatically based on a detected threat. These solutions offer a more straightforward and scalable approach to forensic data capture.

Industry benchmark maintainers do not offer direct guidance on this topic. CIS's only hardening advice for crash dumps is to recommend configurations that limit crash dump data transmission to Microsoft (Center for Internet Security, 2024). An organization must weigh the pros of using crash-dump data against the risk of data exposure to decide how to handle crash-dump generation. Regardless of that decision, several detective capabilities and configuration options are available to detect and address crash dump activity within the organization.

4.1. Detection Methodology

Gaining visibility into the creation of crash dumps can aid an organization, regardless of its stance on whether to permit the creation of crash dumps. For organizations that permit and actively use crash dump data for troubleshooting purposes, it can provide insight into locations where crash dump files might reside and if any endpoints are generating a significant number of crash dumps, a possible sign of system instability or intrusion. For organizations that have decided not to create crash dumps as a default behavior, searching for these crash dumps can uncover endpoints that might have fallen out of configuration scope. Defensive practitioners can search for several artifacts to discover created crash dump files.

- Presence of `MEMORY.DMP` in `C:\Windows` is an easy indicator, though there are some concerns with writing a detection based on this file: This file path is not guaranteed, and by default, Windows overwrites previous crash dump files upon creating a new crash dump. This can confirm the presence of a crash dump but offers no visibility into the timing or frequency of previous dumps.
- If Windows is configured to write an event at the time of a crash, it writes Event ID 1001 (BugCheck) to the System Event Log when a crash occurs.

This event contains the path to the crash dump and stop codes. This artifact is valuable because it specifies the crash-dump location and logs multiple crash-dump generations.

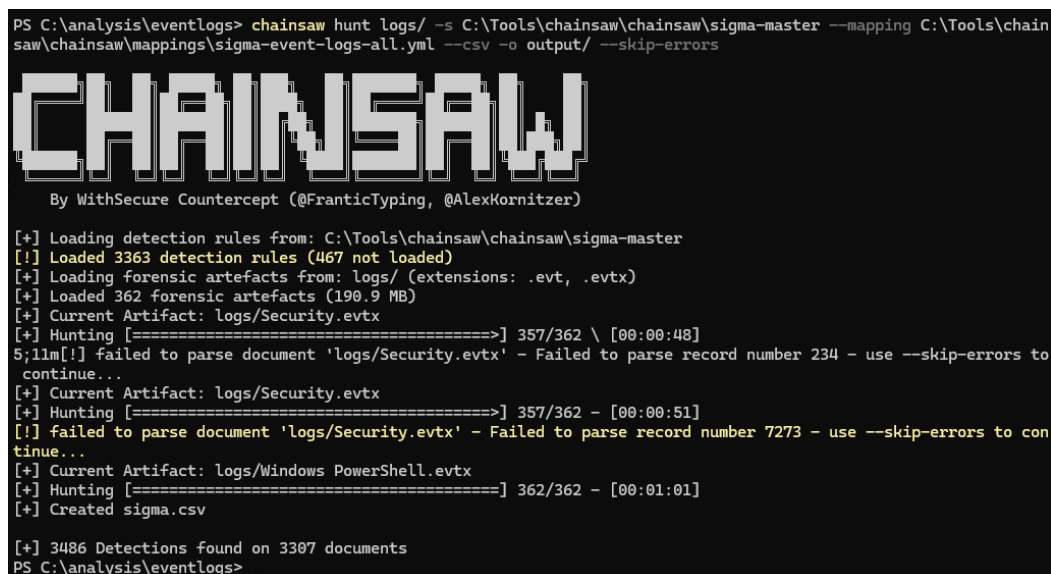
- File system auditing can detect any interaction with crash dump files. While this would log attempts by a threat actor to access the file, system processes, such as backups, could trigger a considerable number of false positives. An organization would need to exclude valid process access for this to be a helpful option.
- While viewing Endpoint Detection & Response (EDR) telemetry after triggering a crash dump, defensive practitioners can note the re-creation of a series of registry keys,
`HKLM\SYSTEM\ControlSet001\Control\CrashControl\MachineCrash\DumpFile` and
`HKLM\SYSTEM\ControlSet001\Control\CrashControl\LastCrashdump\Info.Dumpfile` provides the name of the crash dump file, while `Info` provides crash information similar to what is found within Event ID 1001, including memory addresses related to the crash. Building detections around the creation of these registry keys could provide an additional alerting mechanism.

4.1.1. Event Log Detection with Sigma

Sigma is an open-source detection rule format that can describe log events in a straightforward and vendor-agnostic manner (SigmaHQ, 2025). These rules can be imported into a SIEM platform or used on an ad-hoc basis against offline files for threat-hunting and incident-response purposes. For the first detection, a newly developed Sigma Rule (APPENDIX C) with Chainsaw (WithSecureLabs, 2024) can highlight evidence of crash dump generation. When executing Chainsaw, the program expects a path to Windows event log files, a Sigma rule directory, a Chainsaw mapping file, and an output format. The following command is recommended: `chainsaw hunt <event log`

```
path> -s <sigma rule path> --mapping <mapping file path> --  
csv -o <output path> --skip-errors.
```

```
PS C:\analysis\eventlogs> chainsaw hunt logs/ -s C:\Tools\chainsaw\chainsaw\sigma-master --mapping C:\Tools\chain  
saw\chainsaw\mappings\sigma-event-logs-all.yml --csv -o output/ --skip-errors
```



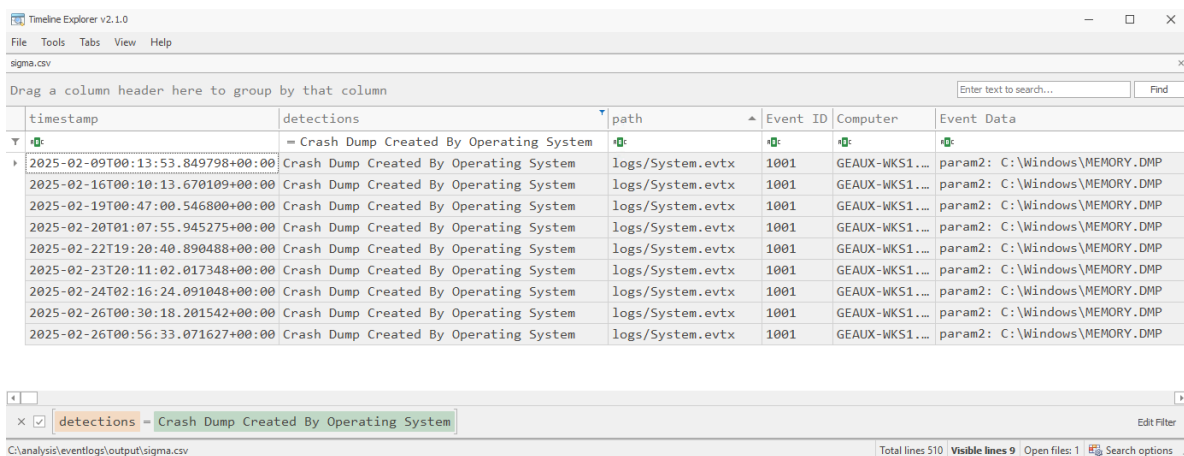
```
CHAINSaw
By WithSecure Countercept (@FranticTyping, @AlexKornitzer)

[+] Loading detection rules from: C:\Tools\chainsaw\chainsaw\sigma-master
[!] Loaded 3363 detection rules (467 not loaded)
[+] Loading forensic artefacts from: logs/ (extensions: .evt, .evtx)
[+] Loaded 362 forensic artefacts (190.9 MB)
[+] Current Artifact: logs/Security.evtx
[+] Hunting [=====] 357/362 \ [00:00:48]
5;llm[!] failed to parse document 'logs/Security.evtx' - Failed to parse record number 234 - use --skip-errors to  
continue...
[+] Current Artifact: logs/Security.evtx
[+] Hunting [=====] 357/362 - [00:00:51]
[!] failed to parse document 'logs/Security.evtx' - Failed to parse record number 7273 - use --skip-errors to con  
tinue...
[+] Current Artifact: logs/Windows PowerShell.evtx
[+] Hunting [=====] 362/362 - [00:01:01]
[+] Created sigma.csv

[+] 3486 Detections found on 3307 documents
PS C:\analysis\eventlogs>
```

Figure 17: Chainsaw Execution

Within the generated `sigma.csv` file, analysts can filter the `Detections` column for Crash Dump Created by Operating System to display relevant events, highlighting the creation of multiple crash dumps on the endpoint. This provides the flexibility to query for this information on an ad-hoc basis, and the logic of the Sigma rule can also be used in a SIEM to obtain real-time alerts if ingesting event logs into the SIEM.



timestamp	detections	path	Event ID	Computer	Event Data
2025-02-09T00:13:53.849798+00:00	Crash Dump Created By Operating System	logs/System.evtx	1001	GEAUX-WKS1...	param2: C:\Windows\MEMORY.DMP
2025-02-16T00:10:13.670109+00:00	Crash Dump Created By Operating System	logs/System.evtx	1001	GEAUX-WKS1...	param2: C:\Windows\MEMORY.DMP
2025-02-19T00:47:00.546800+00:00	Crash Dump Created By Operating System	logs/System.evtx	1001	GEAUX-WKS1...	param2: C:\Windows\MEMORY.DMP
2025-02-20T01:07:55.945275+00:00	Crash Dump Created By Operating System	logs/System.evtx	1001	GEAUX-WKS1...	param2: C:\Windows\MEMORY.DMP
2025-02-22T19:20:40.890488+00:00	Crash Dump Created By Operating System	logs/System.evtx	1001	GEAUX-WKS1...	param2: C:\Windows\MEMORY.DMP
2025-02-23T20:11:02.017348+00:00	Crash Dump Created By Operating System	logs/System.evtx	1001	GEAUX-WKS1...	param2: C:\Windows\MEMORY.DMP
2025-02-24T02:16:24.091048+00:00	Crash Dump Created By Operating System	logs/System.evtx	1001	GEAUX-WKS1...	param2: C:\Windows\MEMORY.DMP
2025-02-26T00:30:18.201542+00:00	Crash Dump Created By Operating System	logs/System.evtx	1001	GEAUX-WKS1...	param2: C:\Windows\MEMORY.DMP
2025-02-26T00:56:33.071627+00:00	Crash Dump Created By Operating System	logs/System.evtx	1001	GEAUX-WKS1...	param2: C:\Windows\MEMORY.DMP

Figure 18: Sigma Rule Output

4.1.2. EDR Detection with Sentinel One

As an alternative to event log review, EDR telemetry can provide a different view into the behaviors performed by an endpoint when a crash dump is triggered. Using Sentinel One's Deep Visibility platform, defensive practitioners can construct a query (APPENDIX D) to search for the creation of the specific registry keys tied to crash dump generation and trigger an alert when the activity occurs, if desired. Depending on the environment, it is possible to become overwhelmed with alerts based on this query. When considering implementing this query as an alerting mechanism, consider the following items:

- Determine the use case for crash dumps before developing the alerting capability. Alerting on crash dump generation is likely more valuable to an organization that has disabled crash dump generation by default. Organizations that permit automatic crash dump generation might find generating crash dump audit reports at specific intervals more useful.
- Consider treating the initial introduction of this query as a threat-hunting engagement. When performing this search in a production environment over 30 days against 4500 endpoints, 155 unique endpoints generated 291 crash events. In addition, 56 events were attributable to a single machine, highlighting an endpoint to investigate for possible system performance or malware issues.
- If configuring this query to alert, consider implementing rate limiting the alerts. For instance, consider only allowing one alert per day or week per endpoint. Having the insight of knowing that a crash dump exists on the endpoint is more valuable than receiving an alert each time an endpoint generates a crash dump.

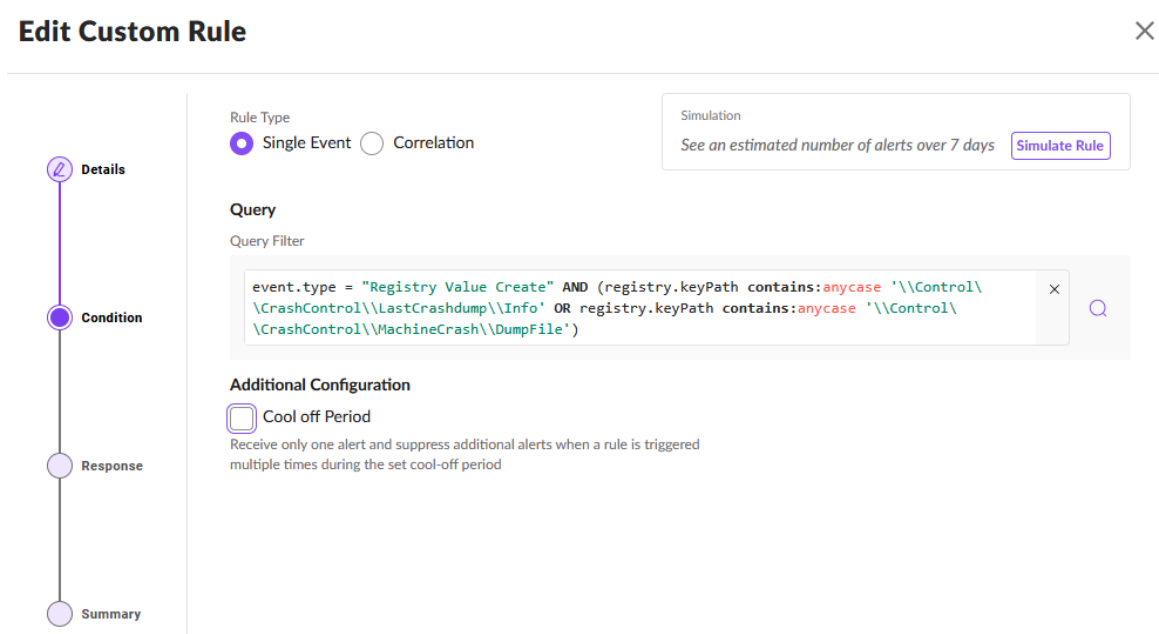


Figure 19: Sentinel One Deep Visibility Detection Rule Creation

Endpoint Name	Event type	Event Time	Source Process Unique ID	Registry Key Path
<input type="checkbox"/> GEAX-WKS1	Registry Value Create	Feb 25 2025 18:56:31	smss.exe 0E33233022605302	MACHINE\SYSTEM\ControlSet001\Control\CrashControl\MachineCrash\DumpFile
<input type="checkbox"/> GEAX-WKS1	Registry Value Create	Feb 25 2025 18:56:28	ntoskrnl.exe 0733233022605302	MACHINE\SYSTEM\ControlSet001\Control\CrashControl\LastCrashdump\Info
<input type="checkbox"/> GEAX-WKS1	Registry Value Create	Feb 25 2025 18:30:11	smss.exe 64F1223022605302	MACHINE\SYSTEM\ControlSet001\Control\CrashControl\MachineCrash\DumpFile
<input type="checkbox"/> GEAX-WKS1	Registry Value Create	Feb 25 2025 18:30:07	ntoskrnl.exe 50F1223022605302	MACHINE\SYSTEM\ControlSet001\Control\CrashControl\LastCrashdump\Info
<input type="checkbox"/> GEAX-WKS1	Registry Value Create	Feb 23 2025 20:16:22	smss.exe 67FE1E3022605302	MACHINE\SYSTEM\ControlSet001\Control\CrashControl\MachineCrash\DumpFile
<input type="checkbox"/> GEAX-WKS1	Registry Value Create	Feb 23 2025 20:16:19	ntoskrnl.exe 5FFE1E3022605302	MACHINE\SYSTEM\ControlSet001\Control\CrashControl\LastCrashdump\Info

Figure 20: Sentinel One Deep Visibility Query Results

4.1.3. File System Discovery

As an alternative to using event logs and EDR queries to detect crash dump creation behaviors, defensive practitioners can search for indicators of crash dumps within the file system. This could be a particularly useful tactic for an organization looking to find crash dumps created outside of the time constraints of event logs or EDR data. The Volatility Foundation provides some initial guidance on crash dump header structures (The Volatility Foundation, 2019). Specifically, crash dumps from 64-bit Jason Mull, research@jasonmull.com

systems contain a header signature of `PAGEDU64`, while crash dumps from 32-bit systems contain a header signature of `PAGEDUMP`. Furthermore, reviewing a user-mode crash dump from within a hex editor shows a header signature of `MDMP`.

With a file signature defined, defensive practitioners can script out the capability to search a file system for evidence of crash dumps. A proof-of-concept PowerShell script is provided (APPENDIX E) that gathers all drive letters on an endpoint and searches all files on all enumerated drives for files containing the relevant header signature.

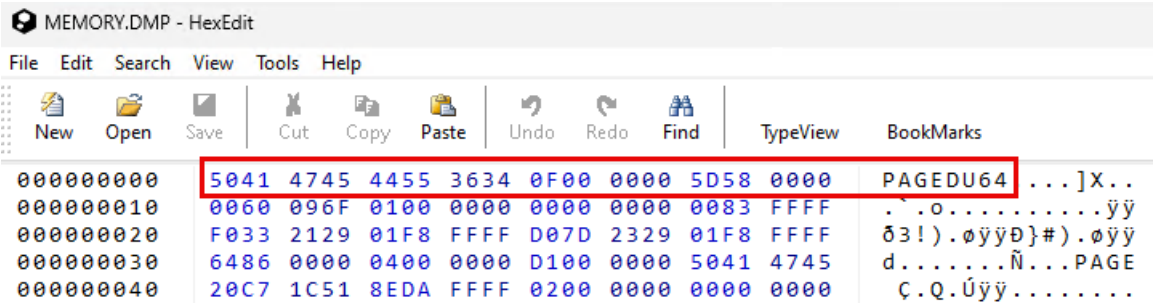


Figure 21: System Crash Dump File Header

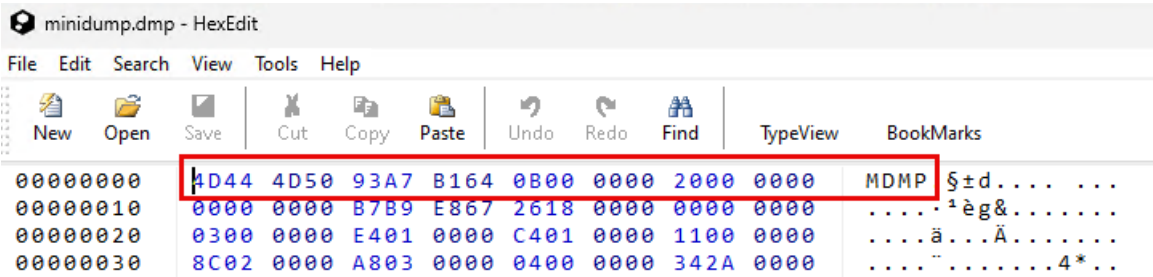


Figure 22: User-Mode Crash Dump File Header

Path	SizeMB	Modified	Type
-----	-----	-----	----
C:\Users\localuser\Desktop\Active Dump\MEMORY.DMP	5114.21	3/16/2025 7:44:27 PM	KernelDump (PAGEDU64 - 64-bit)
C:\Users\localuser\Desktop\ActiveDumpV2\MEMORY.DMP	5435.35	4/9/2025 6:36:49 PM	KernelDump (PAGEDU64 - 64-bit)
C:\Users\localuser\Desktop\Auto Dump\MEMORY.DMP	775.08	4/9/2025 6:36:37 PM	KernelDump (PAGEDU64 - 64-bit)
C:\Users\localuser\Desktop\DC Full Dump\MEMORY.DMP	8191.59	4/9/2025 6:36:24 PM	KernelDump (PAGEDU64 - 64-bit)
C:\Users\localuser\Desktop\Full Dump\MEMORY.DMP	8185.69	3/29/2025 8:49:26 PM	KernelDump (PAGEDU64 - 64-bit)
C:\Users\localuser\Desktop\Full Dump\Old\MEMORY.DMP	8185.7	2/8/2025 6:13:52 PM	KernelDump (PAGEDU64 - 64-bit)
C:\analysis\active_dump\minidump.dmp	63.52	3/29/2025 10:25:43 PM	UserDump (MDMP)
C:\analysis\chromedata_old\chrome-11016.dmp	300.47	2/8/2025 6:10:19 PM	UserDump (MDMP)
C:\analysis\chromedata_old\minidump.dmp	285.53	2/8/2025 6:10:35 PM	UserDump (MDMP)
C:\analysis\dc\minidump.dmp	161.44	3/15/2025 9:30:18 PM	UserDump (MDMP)

Figure 23: Proof-Of-Concept Script Output

4.2. Configuration Options

Crash dump generation options are configurable from the Windows registry (Microsoft, 2025). The relevant values exist under `HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\CrashControl`. The following chart explains the relevant registry keys in greater detail.

Registry Key	Registry Value	Description
CrashDumpEnabled	REG_DWORD 0x0	Crash Dump Disabled
CrashDumpEnabled	REG_DWORD 0x1	Complete Memory Dump
CrashDumpEnabled	REG_DWORD 0x2	Kernel Memory Dump
CrashDumpEnabled	REG_DWORD 0x3	Small Memory Dump
CrashDumpEnabled	REG_DWORD 0x7	Automatic Memory Dump
CrashDumpEnabled and FilterPages	REG_DWORD 0x1	Active Memory Dump
AutoReboot	REG_DWORD 0x0 – Disable REG_DWORD 0x1 - Enable	Reboot After Crash
DumpFile	REG_EXPAND_SZ	Crash Dump Path
LogEvent	REG_DWORD 0x0 – Disable REG_DWORD 0x1 - Enable	Write Crash Event to System Log
MinidumpDir	REG_EXPAND_SZ	Minidump Directory
Overwrite	REG_DWORD 0x0 – Disable REG_DWORD 0x1 - Enable	Overwrite Previous Crash Dump Upon Generation of New Crash Dump

Table 1: Registry Options for Crash Dump Configuration

4.3. Further Research Recommendations

This research focused primarily on user endpoint data exposed by crash dumps. The techniques covered in this research can likely be used to obtain sensitive data from crash dumps on application servers and domain controllers. Further research into the contents of server crash dumps could yield information that could provide a threat actor with data across the entire organization rather than a single endpoint.

Likewise, the applications evaluated were limited to popular line-of-business applications. There is value in exploring the data held in memory by other applications. Password managers could be of particular interest.

Finally, this research uncovered a scenario where user data was discoverable within an Automatic Memory Dump. Microsoft's documentation states that Kernel Memory Dumps, which contain the same information as an Automatic Memory Dump (Microsoft, 2021), does not include memory allocated to User-mode programs (Microsoft, 2025). Further research would be valuable in determining why user-mode information persisted within the Automatic Memory Dump.

5. Conclusion

The analysis of Windows crash dump files for offensive purposes represents an alternative look into a well-researched craft: exploring computer memory for forensic artifacts. While it is common for threat actors to dump processes containing sensitive data as part of their TTPs, utilizing system-generated crash dumps to derive similar information can provide a stealthier approach to obtaining sensitive process data. This research has demonstrated how threat actors can leverage memory forensics to obtain plaintext credentials, operating system secrets, files created by users, and other critical artifacts from system-generated crash dumps to facilitate privilege escalation, lateral movement, and data exfiltration within a compromised environment.

By exploring the techniques, tools, and real-world applications of crash dump exploitation, this research calls out the need for defensive practitioners to understand the

risk of memory artifacts within their organizations. Organizations should ensure that crash dumps are generated only in specific, monitored circumstances and are only stored when they provide active troubleshooting value.

As threat actors such as Storm-0558 become more creative, security practitioners must understand their methods and continue to be equally as creative in their defense and offensive operations capabilities.

References

- 0xNemo. (2025). *MemProcFS_lsass (Version 5.13.4) [Source code]*. Retrieved from GitHub: https://github.com/0xNemo/MemProcFS_lsass
- Bursztein, E., & Picod, J.-M. (2010). Recovering Windows Secrets and EFS Certificates Offline. *Workshop On Offensive Technologies (WOOT)*, 1-9.
- Center for Internet Security. (2024). CIS Microsoft Windows 11 Enterprise Benchmark. 958-959.
- Chua, Z. F. (2015). *Using Windows Crash Dumps for Remote Incident Identification*. SANS Institute.
- Cyber Safety Review Board. (2024). *Review of the Summer 2023 Microsoft Exchange Online Intrusion*.
- Delpy, B. (2019, February 18). You (Dis)Liked Mimikatz? Wait For Kekeo. *BlueHat IL*. Tel Aviv, Israel: Microsoft. Retrieved from <https://www.youtube.com/watch?v=sROKCsXdVDg>
- Frisk, U. (2023, 03 02). *The forensic directory*. Retrieved from MemProcFS Wiki: https://github.com/ufrisk/MemProcFS/wiki/FS_Forensic
- Frisk, U. (2025). *MemProcFS (Version 5.13.4) [Source code]*. Retrieved from Github: <https://github.com/ufrisk/MemProcFS>
- Harris, W. (2024, April 30). *Detecting Browser Data Theft Using Windows Event Logs*. Retrieved from Google Security Blog: <https://security.googleblog.com/2024/04/>
- Harris, W. (2024, July 30). *Improving the Security of Chrome Cookies on Windows*. Retrieved from Google Security Blog: <https://security.googleblog.com/2024/07/improving-security-of-chrome-cookies-on.html>
- Ligh, M. H., Case, A., Levy, J., & Walters, A. (2014). *The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory*. John Wiley & Sons.
- Jason Mull, research@jasonmull.com

- Marcho, C. (2019, March 15). *Understanding Crash Dump Files*. Retrieved from Microsoft: <https://techcommunity.microsoft.com/t5/ask-the-performance-team/understanding-crash-dump-files/ba-p/372633>
- Meckazin. (2024). *ChromeKatz (Version 0.6.1) [Source code]*. Retrieved from Github: <https://github.com/Meckazin/ChromeKatz>
- Microsoft. (2021, 12 14). *Active Memory Dump*. Retrieved from Microsoft Learn: <https://learn.microsoft.com/en-us/windows-hardware/drivers/debugger/active-memory-dump>
- Microsoft. (2021, 12 14). *Automatic Memory Dump*. Retrieved from Microsoft Learn: <https://learn.microsoft.com/en-us/windows-hardware/drivers/debugger/automatic-memory-dump>
- Microsoft. (2024, 07 18). *Collecting User-Mode Dumps*. Retrieved from Microsoft Learn: <https://learn.microsoft.com/en-us/windows/win32/wer/collecting-user-mode-dumps>
- Microsoft. (2024, 11 26). *Web browser cookies used in Microsoft Entra authentication*. Retrieved from Microsoft Learn: <https://learn.microsoft.com/en-us/entra/identity/authentication/concept-authentication-web-browser-cookies>
- Microsoft. (2025, 01 15). *Overview of Memory Dump File Options for Windows*. Retrieved from Microsoft Learn: <https://learn.microsoft.com/en-us/troubleshoot/windows-server/performance/memory-dump-file-options>
- Microsoft Security Response Center. (2024, March 12). *Results of Major Technical Investigations for Storm-0558 Key Acquisition*. Retrieved from Microsoft: <https://msrc.microsoft.com/blog/2023/09/results-of-major-technical-investigations-for-storm-0558-key-acquisition/>
- MITRE. (2024, August 13). *OS Credential Dumping: LSASS Memory*. Retrieved from MITRE ATT&CK: <https://attack.mitre.org/techniques/T1003/001/>
- MITRE. (2024, 10 14). *Steal Web Session Cookie*. Retrieved from MITRE ATT&CK: <https://attack.mitre.org/techniques/T1539/>
- Jason Mull, research@jasonmull.com

SigmaHQ. (2025). *Sigma - Generic Signature Format for SIEM Systems (Version r2025-02-03) [Source code]*. Retrieved from GitHub: <https://github.com/SigmaHQ>

SkelSec. (2025). *Pypykatz (Version 0.6.11) [Source code]*. Retrieved from Github: <https://github.com/skelsec/pypykatz>

The Volatility Foundation. (n.d.). Retrieved from The Volatility Framework: <https://volatilityfoundation.org/the-volatility-framework/>

The Volatility Foundation. (2019, 03 22). *Crash Address Space*. Retrieved from Volatility Wiki: <https://github.com/volatilityfoundation/volatility/wiki/Memory-Samples#crash-dumps>

Vidas, T., Kaplan, B., & Geiger, M. (2014). OpenLV: Empowering investigators and first-responders. *Digital Investigation 11*, S45-S53.

WithSecureLabs. (2024, December 28). *Chainsaw (Version 2.11.0) [Source code]*. Retrieved from GitHub: <https://github.com/WithSecureLabs/chainsaw>

Appendix A: Office Document Generation Script

```
# Load Outlook COM Object
$outlook = New-Object -ComObject Outlook.Application

# Define different types of sensitive data
$sensitiveDataList = @(
    "Password: SuperSecurePass123!",
    "API Key: sk_live_9876543210abcdef",
    "Session Token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "Database Credentials: user=admin; password=P@ssw0rd!",
    "Encryption Key: 0x4E5342AC3FA1994B...",
    "SSH Private Key: -----BEGIN RSA PRIVATE KEY-----...",
    "OAuth Access Token: ya29.a0AfH6SMC...",
    "Credit Card Info: 4111-1111-1111-1111 Exp: 12/26 CVV: 123",
    "JWT Token: eyJhbGciOiJIUzI1Ni...",
    "Internal Server Config: db_host=10.0.0.1; secret_key=XYZ987"
)

# Define file types for attachments
$fileTypes = @(".txt", ".pdf", ".docx", ".xlsx")

# Ensure the attachments directory exists
$attachmentDir =
"C:\Users\$env:USERNAME\Documents\GeneratedAttachments"
if (-not (Test-Path $attachmentDir)) {
    New-Item -Path $attachmentDir -ItemType Directory -Force | Out-Null
}

# Function to generate a Word Document
Function Create-WordDocument {
    param ($filePath, $content)
    $word = New-Object -ComObject Word.Application
    $word.Visible = $false
    $doc = $word.Documents.Add()
    $doc.Content.Text = $content
    $doc.SaveAs([ref] $filePath, [ref] 16)
    $doc.Close()
    $word.Quit()
}

# Function to generate an Excel Spreadsheet
Function Create-ExcelFile {
    param ($filePath, $content)
    $excel = New-Object -ComObject Excel.Application
    $excel.Visible = $false
    $workbook = $excel.Workbooks.Add()
```

```

    $sheet = $workbook.Sheets.Item(1)
    $sheet.Cells.Item(1,1) = $content
    $workbook.SaveAs($filePath)
    $workbook.Close()
    $excel.Quit()
}

# Function to generate a PDF (via Word)
Function Create-PDF {
    param ($filePath, $content)
    $word = New-Object -ComObject Word.Application
    $word.Visible = $false
    $doc = $word.Documents.Add()
    $doc.Content.Text = $content
    $pdfPath = $filePath -replace ".pdf", ".docx"
    $doc.SaveAs([ref] $pdfPath, [ref] 16)
    $doc.ExportAsFixedFormat($filePath, 17)
    $doc.Close()
    $word.Quit()
    Remove-Item $pdfPath
}

# Send 10 emails with different contents and dynamically created mixed-
type attachments
for ($i = 1; $i -le 10; $i++) {
    #Create Email
    $mail = $outlook.CreateItem(0)
    $mail.To = "colleague@company.com"
    $mail.Subject = "Sensitive Info #${i}"
    $mail.Body = "Hi, here is the requested sensitive
data:`n`n${sensitiveDataList[$i-1]}"

    # Choose a random file type for the attachment
    $fileType = $fileTypes | Get-Random
    $attachmentPath = "$attachmentDir\SensitiveData_${i}$fileType"

    # Generate the attachment
    if ($fileType -eq ".txt") {
        $sensitiveDataList[$i-1] | Out-File -FilePath $attachmentPath -
Encoding ASCII -Force
    } elseif ($fileType -eq ".pdf") {
        Create-PDF -filePath $attachmentPath -content
$sensitiveDataList[$i-1]
    } elseif ($fileType -eq ".docx") {
        Create-WordDocument -filePath $attachmentPath -content
$sensitiveDataList[$i-1]
    } elseif ($fileType -eq ".xlsx") {

```

```
        Create-ExcelFile -filePath $attachmentPath -content  
$sensitiveDataList[$i-1]  
    }  
  
    Write-Host "[*] Created attachment: $attachmentPath"  
  
    # Attach the file to the email  
    $mail.Attachments.Add($attachmentPath)  
  
    # Send the email  
    $mail.Send()  
    Write-Host "[*] Email #$i sent with $fileType attachment!"  
    Start-Sleep -Seconds 2  
}
```

APPENDIX B: Microsoft 365 Login Automation Script

```
# Define User Credentials
$0365Username = <Microsoft 365 Login>
$0365Password = <Password>

# Load Selenium module
Import-Module Selenium

#Setup GeckoDriver
$geckoDriverPath = "C:\ProgramData\chocolatey\bin"
$firefoxOptions = New-Object OpenQA.Selenium.Firefox.FirefoxOptions
$firefoxProfilePath = "$env:APPDATA\Mozilla\Firefox\Profiles"
$firefoxDefaultProfile = Get-ChildItem $firefoxProfilePath | Where-Object { $_.Name -like "*.default*" } | Select-Object -First 1
$firefoxFullProfilePath =
"$firefoxProfilePath\$($firefoxDefaultProfile.Name)"
$firefoxOptions.AddArgument("-profile")
$firefoxOptions.AddArgument($firefoxFullProfilePath)

#Setup ChromeDriver
$chromeDriverPath = "C:\ProgramData\chocolatey\bin"
$chromeOptions = New-Object OpenQA.Selenium.Chrome.ChromeOptions

# Use local user Chrome profile
$chromeProfilePath = "$env:LOCALAPPDATA\Google\Chrome\User Data"
$chromeOptions.AddArgument("--user-data-dir=$chromeProfilePath")
$chromeOptions.AddArgument("--profile-directory=Default")

# Execute Firefox Automation
$firefoxDriver = New-Object
OpenQA.Selenium.Firefox.FirefoxDriver($geckoDriverPath,
$firefoxOptions)
$firefoxDriver.Navigate().GoToUrl("https://mail.office365.com")
Start-Sleep -Seconds 3 # Give page time to load
$firefoxEmailField = $firefoxDriver.FindElementById("i0116") #Sends
Username
$firefoxEmailField.Clear()
$firefoxEmailField.SendKeys($0365Username)
Start-Sleep -Seconds 2 # Sleeps for input completion and click "Next"
button
$firefoxNextButton = $firefoxDriver.FindElementById("idSIButton9")
$firefoxNextButton.Click()
Start-Sleep -Seconds 3 #Wait for password field and enter password
$firefoxPasswordField = $firefoxDriver.FindElementById("i0118")
$firefoxPasswordField.SendKeys($0365Password)
```

```
Start-Sleep -Seconds 2 # Sleep for input completion and click "Sign In"
$firefoxSignInButton = $firefoxDriver.FindElementById("idSIButton9")
$firefoxSignInButton.Click()
Start-Sleep -Seconds 3 # Wait for the prompt to appear. Click "No" on
Stay Signed In prompt
$firefoxStaySignedInNoButton =
$firefoxDriver.FindElementById("idBtn_Back")
$firefoxStaySignedInNoButton.Click()

#Execute Chrome Automation
$chromeDriver = New-Object
OpenQA.Selenium.Chrome.ChromeDriver($chromeDriverPath, $chromeOptions)
$chromeDriver.Navigate().GoToUrl("https://mail.office365.com")
Start-Sleep -Seconds 3 # Give page time to load
$chromeEmailField = $chromeDriver.FindElementById("i0116") #Sends
Username
$chromeEmailField.Clear()
$chromeEmailField.SendKeys($0365Username)
Start-Sleep -Seconds 2 # Sleeps for input completion and click "Next"
button
$chromeNextButton = $chromeDriver.FindElementById("idSIButton9")
$chromeNextButton.Click()
Start-Sleep -Seconds 3 #Wait for password field and enter password
$chromePasswordField = $chromeDriver.FindElementById("i0118")
$chromePasswordField.SendKeys($0365Password)
Start-Sleep -Seconds 2 # Sleep for input completion and click "Sign In"
$chromeSignInButton = $chromeDriver.FindElementById("idSIButton9")
$chromeSignInButton.Click()
Start-Sleep -Seconds 3 # Wait for the prompt to appear. Click "No" on
Stay Signed In prompt
$chromeStaySignedInNoButton =
$chromeDriver.FindElementById("idBtn_Back")
$chromeStaySignedInNoButton.Click()
```

Appendix C: Crash Dump Generation Sigma Rule

```
title: Crash Dump Created By Operating System
id: 882fbe50-d8d7-4e29-ae80-0648a8556866
related:
  - id: 2ff692c2-4594-41ec-8fcb-46587de769e0
    type: similar
status: experimental
description: This rule detects the creation of crash dumps by the
operating system
author: Jason Mull
date: 2025-02-23
tags:
  - attack.credential-access
  - attack.collection
  - attack.t1003.002
  - attack.t1005
logsource:
  product: windows
  service: system
detection:
  selection:
    Provider_Name: 'Microsoft-Windows-WER-SystemErrorReporting'
    EventID: 1001
  condition: selection
level: medium
```

APPENDIX D – SentinelOne EDR Queries

```
# Query to search for relevant registry keys within Singularity Data Lake (SDL)
event.type = "Registry Value Create" AND (registry.keyPath contains:anycase '\\Control\\CrashControl\\LastCrashdump\\Info' OR registry.keyPath contains:anycase '\\Control\\CrashControl\\MachineCrash\\DumpFile')

#PowerQuery to search for crash dump generation events, grouped by endpoint and number of occurrences
| filter( event.type == "Registry Value Create" AND registry.keyPath contains:anycase( "\\Control\\CrashControl\\MachineCrash\\DumpFile" ) )
| group EventCount = count() by endpoint.name
| sort - EventCount
| limit 1000
```

APPENDIX E – PowerShell Script to Search for Crash Dump File Headers

```
#Hex encoded crash dump file signatures
#https://github.com/volatilityfoundation/volatility/wiki/Crash-Address-Space
$headerSignatures = @{
    '50411474544554D50' = 'KernelDump (PAGEDUMP - 32-bit)'
    '50411474544553634' = 'KernelDump (PAGEDU64 - 64-bit)'
    '4D444D50'          = 'UserDump (MDMP)'
}

$results = @()
#Loop through all local drives
Get-PSDrive -PSProvider FileSystem | ForEach-Object {
    #Search file system of each drive looking for header signatures
    Get-ChildItem -Path $_.Root -File -Recurse -ErrorAction
    SilentlyContinue | ForEach-Object {
        try {
            $bytes = New-Object byte[] 8
            $fileStream = [System.IO.File]::Open($_.FullName, 'Open',
            'Read', 'ReadWrite')
            $fileStream.Read($bytes, 0, 8) | Out-Null
            $fileStream.Close()

            $fileHex = ($bytes | ForEach-Object { $_.ToString('X2') })

            -join ' '

            #Append results if signatures match
            foreach ($sig in $headerSignatures.Keys) {
                if ($fileHex.StartsWith($sig)) {
                    $results += [PSCustomObject]@{
                        Path      = $_.FullName
                        SizeMB    = [math]::Round($_.Length / 1MB, 2)
                        Modified   = $_.LastWriteTime
                        Type       = $headerSignatures[$sig]
                    }
                    break
                }
            }
        } catch {}
    }
}

#Output results
$results | Sort-Object Type | Format-Table -AutoSize -Wrap
```