TDX ARENA

Certification Report

Kevin Martinho
Final Assessment Report Submission

Imperial Memory: [Insider Threat Data Exfiltration via Memory Forensics]

December 5, 2025

Executive Summary

I investigated a suspicious file left on a desktop by Jules, a departing cyber researcher. He said the answer to his success would be "kept in memory," which turned out to be a huge hint. I found an encrypted archive (gift.7z) and a memory dump file (Emperor.vmem) on the desktop.

By analyzing the memory dump and searching for keywords, I recovered the password: `G6Vmc$0d5cpM8ee#Ca=x&A3`. This let me unlock the archive and extract suspicious.docx. But here's where it got interesting—when I converted the DOCX to a 7z archive and opened it, I discovered it had hidden internal files. Inside was secrets.txt, which contained classified forensic methodology documentation about the "Law of Individuality" and "Law of Probability" and "Law of Circumstantial Facts."

This turned out to be a data exfiltration incident. Jules didn't just leave a puzzle—he left evidence of stealing classified information using sophisticated encryption and obfuscation techniques. The password being stored in plaintext in the memory dump is a huge security issue that let me recover it easily.

Overview

The Imperial Memory lab is a digital forensics challenge that simulates a realistic insider threat scenario. The objective is to analyze a memory dump file to recover encrypted credentials, decrypt password-protected archives, and identify exfiltrated classified information. The lab tests proficiency in memory forensics, evidence recovery, archive manipulation, and threat identification. It requires combining multiple forensic techniques to uncover a multi-layer obfuscation scheme used to conceal sensitive data.

Technical Findings

#1: Emperor.vmem Memory Dump (1 GB)

File Location (Lab): /home/derrek/Desktop/Emperor.vmem
File Location (Original System): C:\Users\Aaron\Desktop\Emperor.vmem
File Type: Virtual Memory Dump
Size: 1,073,741,824 bytes (1 GiB)
Created: 2022-11-24 11:19:28

The memory dump contained volatile data from running processes on Aaron's Windows system. String extraction from this dump revealed plaintext credentials and sensitive command-line arguments. The presence of unencrypted passwords in memory demonstrates a critical vulnerability in secure coding practices where sensitive data should be cleared from memory immediately after use.

#2: gift.7z Encrypted Archive

File Location: C:\Users\Aaron\Desktop\gift.7z
File Type: 7-Zip Archive
Size: 10.2 KB (packed: 10,240 bytes)
Compression: LZMA2:14
Encryption: 7zAES:19 (AES-256)
Created: 2022-01-20 07:06:28

The archive uses strong encryption standards (AES-256 with LZMA2 compression). However, the encryption key was compromised by recovery of the plaintext password from the memory dump.

#3: Recovered Password from Memory

Password: G6Vmc$0d5cpM8ee#Ca=x&A3
Recovery Method: String extraction with grep filtering
Source: Process command-line artifacts in Emperor.vmem
Full Command Context: C:\Users\Aaron\Desktop\gift -p'G6Vmc$0d5cpM8ee#Ca=x&A3'

The password was extracted using the command:

```
strings Emperor.vmem | grep -E 'gift.7z|Jules|secret|password' -I > password.txt
```

The password appeared in plaintext within the memory space of the process handling the archive encryption, proving that the encryption key was not properly protected in volatile memory.

#4: suspicious.docx (DOCX Container)

File Location: Extracted from gift.7z
File Type: Microsoft Office Open XML (.docx)
Size: 12.8 KB (unpacked: 12,800 bytes)
Packed Size: 10.1 KB
Created: 2022-01-20 07:06:28
MD5 Hash: 347765E4

The DOCX file is actually a ZIP archive containing XML files and resources. Initial examination showed only the text "I am empty!" However, the internal structure contained hidden files in compressed form.

#5: secrets.txt (Classified Payload)

File Location: Hidden within suspicious.docx internal structure
File Type: Plain text (.txt)
Size: 1.3 KB (1,331 bytes)
Packed Size: 708 bytes
Created: 2022-01-20 07:05:59
MD5 Hash: 0f235385d25ade312a2d151a2cc43865

Content Summary: Classified forensic methodology documentation including:

Law of Individuality

- "Every object has a unique attribute that is not replicated in any other object."
- For example, fingerprints. Billions of fingerprints were inspected and never did one matched another. Although any two objects such as grains of sand, salt, seeds, twins, or man-made objects such as laptops, suits, typewriters, files, printers, etc., may seem similar, a unique attribute can always be found in each of them separately.

Law of Probability

- "All identifications definite or indefinite, made consciously or unconsciously, are on the basis of probability."
- For example, an unidentified male is found murdered in a forest. The man's body revealed a gold-plated tooth, a healed arm fracture, and a large tattoo on his left foot. A male with the same specifications was also reported missing. Can it then be assumed that there is a high probability that the unknown corpse is that of the missing man?
- Probability is a mathematical concept. It determines the chances of occurrence of a particular event through evidential and correlational points and signs.

Law of Circumstantial facts

- "Facts do not lie, men can and do."
- Pieces of evidence provided by eyewitnesses or victims are not always necessarily accurate.

- Since static facts are incontrovertable, and do not lie, circumstantial evidence is often more powerful than verbal evidence.

This is the actual classified payload that was deliberately concealed through multi-layer obfuscation.

#6: DOCX Internal Structure

Discovered After Format Conversion:
suspicious.7z (re-archived from DOCX)

- docProps/
- word/
- _rels/
- [Content_Types].xml
- secrets.txt

The secrets.txt file was only visible after converting the DOCX to 7z format and opening it with an archive manager. Standard document viewers do not expose this internal structure, making it an effective obfuscation technique.

#7: Evidence Chain of Custody

secrets.txt MD5 Hash: 0f235385d25ade312a2d151a2cc43865
Hash Command: md5sum secrets.txt
Hash Generation Time: 2025-12-05

The cryptographic hash was generated to preserve evidence integrity and establish forensic chain of custody. This hash value can be used to verify the file has not been modified since extraction.

Recommendations

#1: Immediate Access Revocation

Jules departed with unauthorized access to classified forensic methodologies. All credentials, access tokens, and permissions must be revoked immediately:

- Disable all user accounts and service accounts
- Revoke VPN, SSH, and remote access credentials
- Terminate active sessions and force logout
- Reset passwords for any shared accounts he had access to
- Remove API keys and authentication tokens
- Revoke any sudo or administrative privileges

#2: Preserve all evidence from Jules' workstation and access patterns

- Capture memory dump and full disk image of his workstation before shutdown
- Preserve system logs and event logs covering his entire employment period
- Extract email archives for communications with external parties
- Document all file access logs showing what data he accessed
- Preserve network logs showing all outbound connections from his user account
- Check cloud storage services (OneDrive, Dropbox, Google Drive) for uploaded files
- Review USB device logs and external drive connections

#3: Assume all credentials Jules had access to are compromised

- Force password resets for all systems he could access
- Rotate encryption keys and master passwords
- Audit SSH keys and revoke compromised keys
- Regenerate API tokens and service credentials
- Review authentication logs for suspicious login attempts
- Implement mandatory password changes for affected systems

#4: Identify all systems containing the stolen forensic methodologies

- Catalog all systems storing classified forensic procedures
- Implement digital watermarking on sensitive documents
- Add forensic canaries to detect if stolen data appears externally
- Deploy enhanced monitoring and audit logging on these systems
- Implement version control and access tracking
- Create backup copies in secure locations
- Consider proactive publication of sanitized versions to reduce value of stolen data

#5: Search for indicators of broader data exfiltration

- Scan all systems for other encrypted archives (7z, ZIP, RAR files)
- Look for suspicious file creation dates matching Jules' employment period
- Search for unusual process execution patterns (encryption tools, archiving utilities)
- Audit file access logs for mass data gathering or bulk downloads
- Check for other suspicious archives on shared drives or storage
- Review email forwarding rules or rules that might have been set up
- Investigate his communication patterns for coordination with external parties

#6: Initiate formal investigation procedures

- Notify legal department immediately of potential corporate espionage
- Preserve all evidence in accordance with legal requirements
- Document chain of custody for all forensic evidence
- Coordinate with HR on exit interview findings
- Consult with legal about potential law enforcement involvement
- Review employment agreement for non-disclosure and IP clauses

- Determine if criminal charges should be filed

#7: Monitor for external disclosure or misuse of stolen information

- Set up alerts for dark web, forums, and paste sites for leaked data
- Monitor for your organization name + "forensic" keywords
- Check competitor activity for sudden methodology changes
- Contact third parties who might be affected by disclosure
- Monitor for increased attacks using knowledge of your investigation techniques
- Subscribe to threat intelligence feeds for related indicators
- Set up Google Alerts for your organization and forensic-related terms

#8: Deploy tools to detect and prevent future exfiltration attempts

- Monitor for unusual archive creation (7z, ZIP, RAR, TAR files)
- Alert on large files being packaged or compressed
- Detect USB device connections and file transfers
- Monitor emails for encrypted attachments to external recipients
- Track access patterns suggesting data gathering behavior
- Implement endpoint protection with USB restrictions
- Deploy network DLP to detect exfiltration attempts

#9: Strengthen protection of sensitive encrypted archives

- Require multi-factor authentication for accessing sensitive archives
- Implement Hardware Security Modules (HSMs) for key management
- Use key derivation functions with secure random seeds
- Implement secure deletion protocols for sensitive archives
- Avoid password-based encryption for highly sensitive data
- Use ephemeral encryption keys that are never stored permanently
- Maintain strict access controls with compartmentalization

#10: Review and update investigation methodologies since they're now compromised

- Update forensic procedures that Jules had access to
- Rotate investigation methodologies where possible
- Implement compartmentalization so single individuals can't access all techniques
- Add additional security clearances for sensitive methodology access
- Create honeypot cases to detect if stolen procedures are being used
- Review sensitive case files for potential exposure
- Update training materials to reflect new procedures

#11: Strengthen incident response capabilities

- Conduct tabletop exercises for insider threat scenarios
- Update incident response playbooks for data exfiltration
- Deploy memory forensics tools (Volatility) across infrastructure
- Train incident response team on memory analysis techniques

- Establish forensic imaging procedures for rapid evidence collection
- Create incident response runbooks for insider threats
- Schedule regular drills for credential compromise scenarios

#12: Implement awareness programs to prevent similar incidents

- Conduct security awareness training on data handling and classification
- Educate employees on insider threat indicators
- Implement training on secure credential management
- Establish clear data handling policies for departing employees
- Create reporting mechanisms for suspicious behavior
- Share lessons learned from this incident (sanitized version)
- Conduct background checks and security reviews during hiring

#13: **Strengthen overall system security posture**

- Implement kernel patch protection to prevent memory-based attacks
- Enable process monitoring and alerting for suspicious memory access
- Deploy Endpoint Detection and Response (EDR) solutions
- Implement continuous monitoring of sensitive systems
- Enable audit logging for all access to classified information
- Review and strengthen firewall rules
- Implement network segmentation for sensitive systems

#14: **Assess and communicate impact to affected parties**

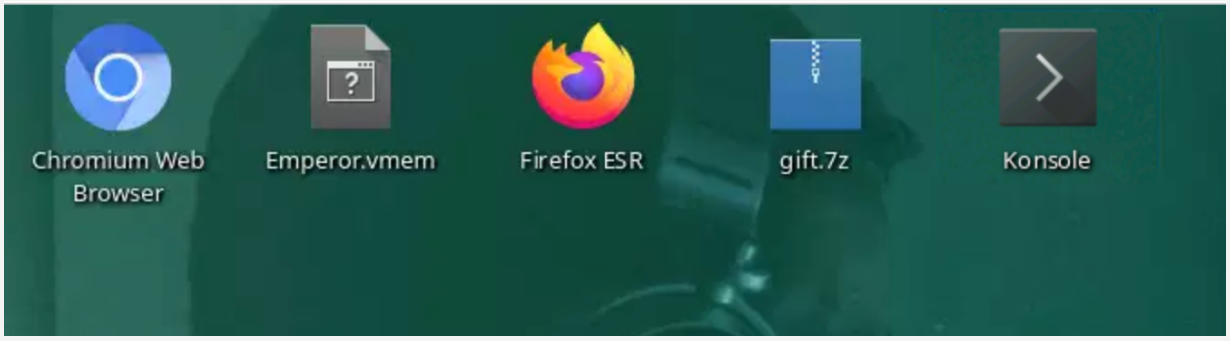
- Determine what classified information was specifically exposed
- Assess risk to customers, partners, and operations
- Notify affected parties according to legal requirements
- Prepare incident statement for stakeholders
- Document lessons learned and improvements made
- Consider offering affected parties credit monitoring or support
- Maintain transparency while protecting ongoing investigations

Tools and Technology Used

Investigation Process:

Step 1: Initial Artifact Assessment

I started by examining the desktop. The file browser showed Emperor.vmem (1.0 GB) and gift.7z (10.2 KB) along with other files. Jules' hint about "memory" combined with a memory dump and encrypted archive made it clear what I needed to do.



Step 2: Volatility Framework Attempt

I tried using Volatility to analyze the memory dump. I ran:

vol.py -f Emperor.vmem --profile=Win7SP1x64 pslist

However, it failed with profile compatibility errors. Multiple Windows profiles didn't work—each one showed "No suitable address space mapping found" or "Incompatible profile" errors. Since Volatility wasn't working with the available profiles, I needed a different approach.

```
derrek@ubuntu:~/Desktop$ vol.py -f Emperor.vmem --profile=Win7SP1x64 pslist
Volatility Foundation Volatility Framework 2.6.1
No suitable address space mapping found
Tried to open image as:
 MachOAddressSpace: mac: need base
 LimeAddressSpace: lime: need base
 WindowsHiberFileSpace32: No base Address Space
 WindowsCrashDumpSpace64BitMap: No base Address Space
 WindowsCrashDumpSpace64: No base Address Space
 HPAKAddressSpace: No base Address Space
 VirtualBoxCoreDumpElf64: No base Address Space
 VMWareMetaAddressSpace: No base Address Space
 QemuCoreDumpElf: No base Address Space
 VMWareAddressSpace: No base Address Space
 WindowsCrashDumpSpace32: No base Address Space
 SkipDuplicatesAMD64PagedMemory: No base Address Space
 WindowsAMD64PagedMemory: No base Address Space
 LinuxAMD64PagedMemory: No base Address Space
 AMD64PagedMemory: No base Address Space
 IA32PagedMemoryPae: No base Address Space
 IA32PagedMemory: No base Address Space
 OSXPmemELF: No base Address Space
 MachOAddressSpace: MachO Header signature invalid
 LimeAddressSpace: Invalid Lime header signature
 WindowsHiberFileSpace32: No xpress signature found
 WindowsCrashDumpSpace64BitMap: Header signature invalid
 WindowsCrashDumpSpace64: Header signature invalid
 HPAKAddressSpace: Invalid magic found
 VirtualBoxCoreDumpElf64: ELF Header signature invalid
 VMWareMetaAddressSpace: VMware metadata file is not available
 QemuCoreDumpElf: ELF Header signature invalid
 VMWareAddressSpace: Invalid VMware signature: 0xf000ff53
 WindowsCrashDumpSpace32: Header signature invalid
 SkipDuplicatesAMD64PagedMemory: Incompatible profile Win7SP1x64 selected
 WindowsAMD64PagedMemory: No valid DTB found
 LinuxAMD64PagedMemory: Incompatible profile Win7SP1x64 selected
 AMD64PagedMemory: No valid DTB found
 IA32PagedMemoryPae: Incompatible profile Win7SP1x64 selected
 IA32PagedMemory: Incompatible profile Win7SP1x64 selected
 OSXPmemELF: ELF Header signature invalid
 FileAddressSpace: Must be first Address Space
 ArmAddressSpace: No valid DTB found
```

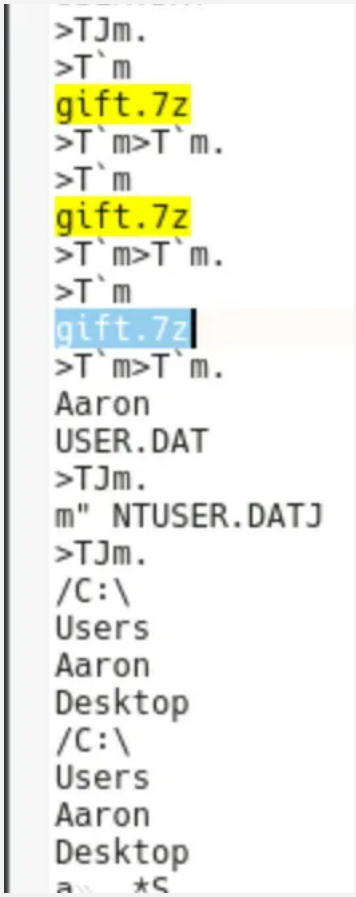
Step 3: Strings and Grep Command

I switched to a direct keyword search approach. Since the memory dump is just binary data, I used strings to extract readable text and grep to filter for relevant keywords:

strings Emperor.vmem | grep -E 'gift.7z|Jules|secret|password' -I > password.txt

This extracted all readable ASCII and Unicode text from the memory dump, filtered it for keywords related to the archive and encryption, and saved the results to password.txt.

```
File   Edit   View   Bookmarks   Settings   Help
derrek@ubuntu:~/Desktop$ strings Emperor.vmem | grep -E '|gift.7z|Jules|secret' > search
password.txt
```



Step 4: Password Recovered from Memory

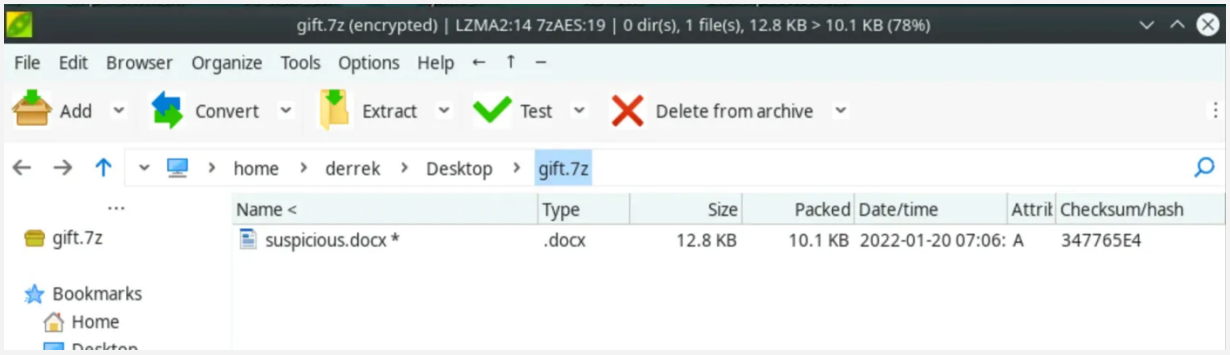
Looking through the filtered results, I found the password in plaintext within the memory dump. The command line that had been running was:

C:\Users\Aaron\Desktop\gift -p'G6Vmc$0d5cpM8ee#Ca=x&A3'

This showed the password was sitting unencrypted in memory:
G6Vmc$0d5cpM8ee#Ca=x&A3

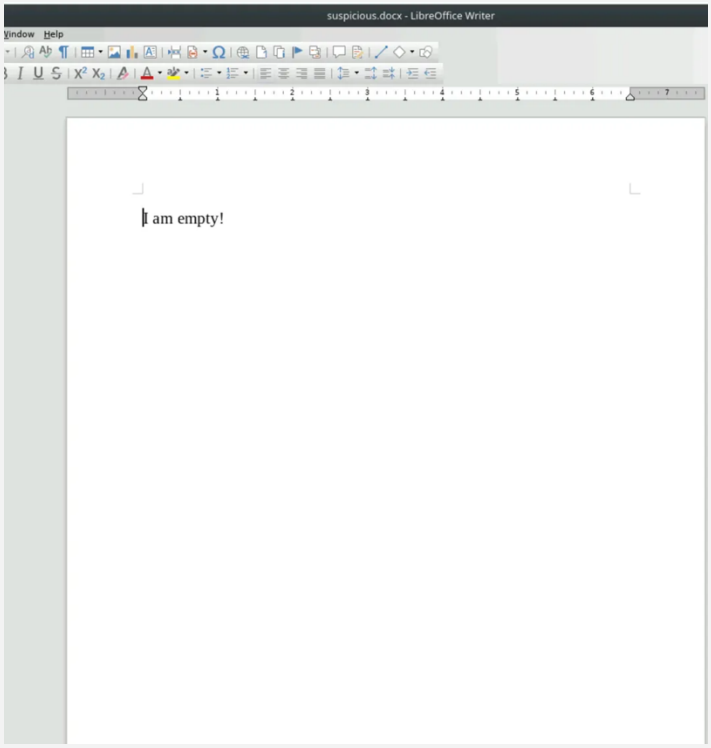
Step 5: Decrypting the Archive

I opened gift.7z with PeaZip. It recognized the file as password-protected (LZMA2:14, 7zAES:19). I entered the recovered password, and it successfully decrypted the archive, revealing suspicious.docx inside.



Step 6: Opening suspicious.docx

I extracted suspicious.docx and opened it in LibreOffice Writer. It displayed only the text "I am empty!" which seemed like a dead end or a decoy message. However, I knew that DOCX files are actually ZIP archives internally containing XML files and resources.



Step 7: Converting DOCX to 7z Format

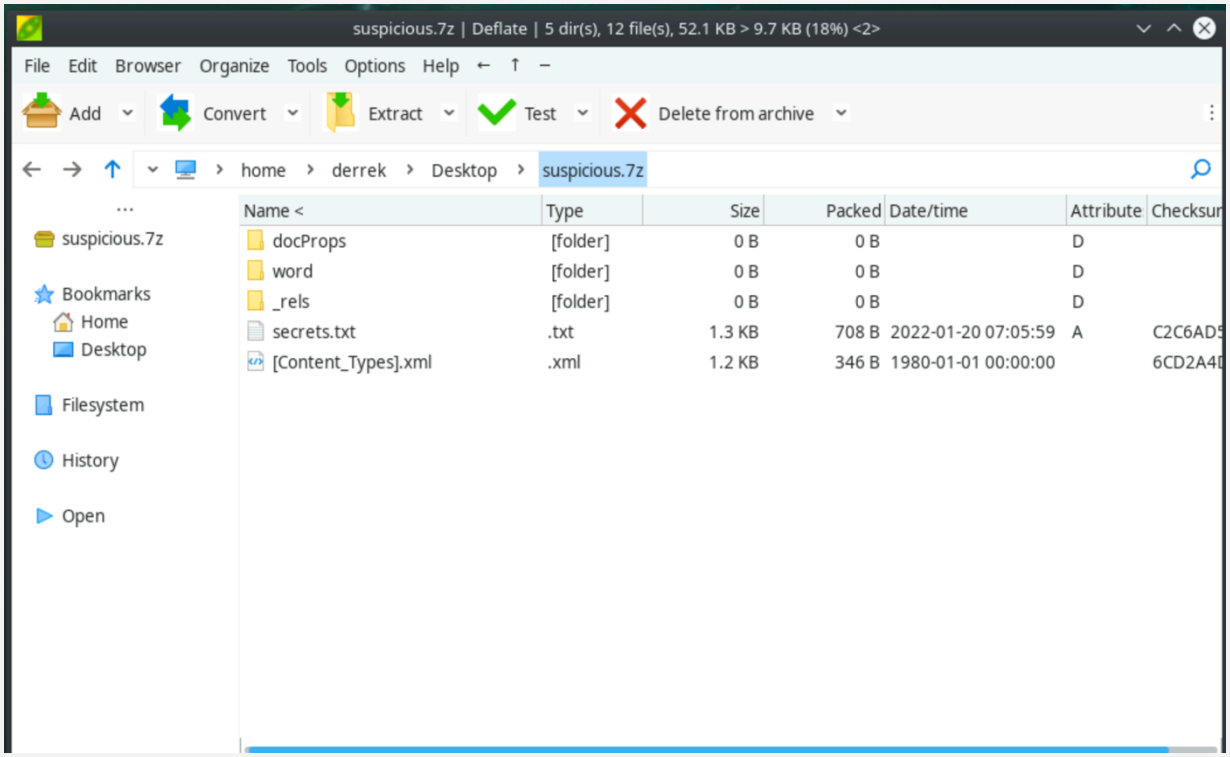
To examine the internal structure, I re-archived the DOCX file in 7z format:

7z a suspicious.7z suspicious.docx

When I opened this converted file in PeaZip, I could see the full internal structure that a normal document viewer wouldn't show:

- docProps/ (folder)
- word/ (folder)
- _rels/ (folder)
- [Content_Types].xml
- **secrets.txt** (1.3 KB)

The secrets.txt file was hidden in the DOCX internal structure, which is why it wasn't visible when opening the document normally.

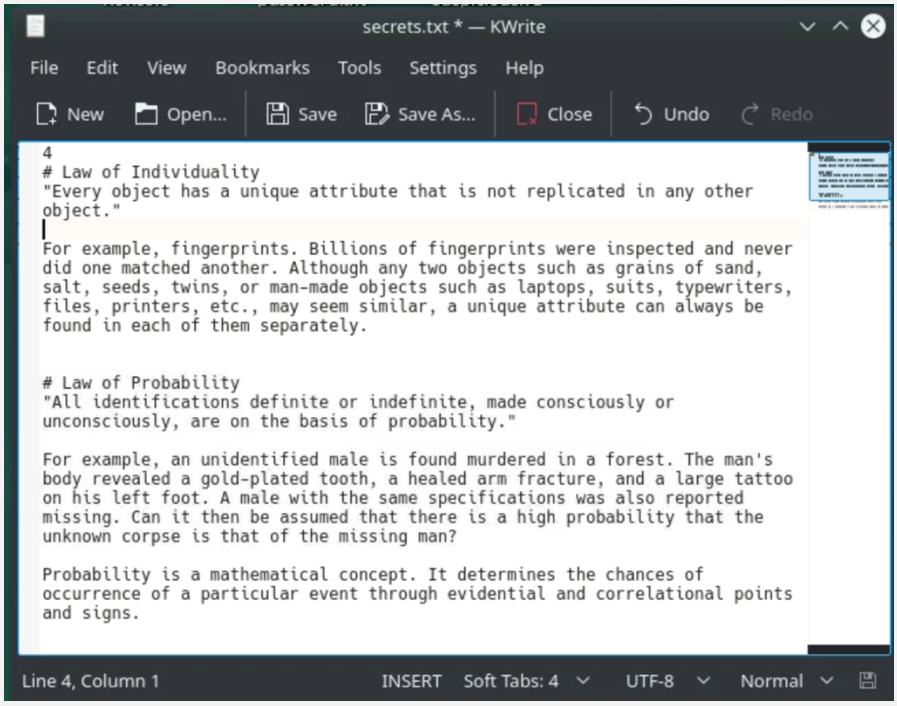


Step 8: Examining secrets.txt

I extracted and opened secrets.txt in KWrite. It contained classified forensic documentation:

Within the secrets.txt document were three laws, “Law of Individuality, Law of Probability, Law of Circumstantial Facts.”

This is the classified information Jules hid using multi-layer obfuscation.



Step 9: Hashing for Evidence Integrity

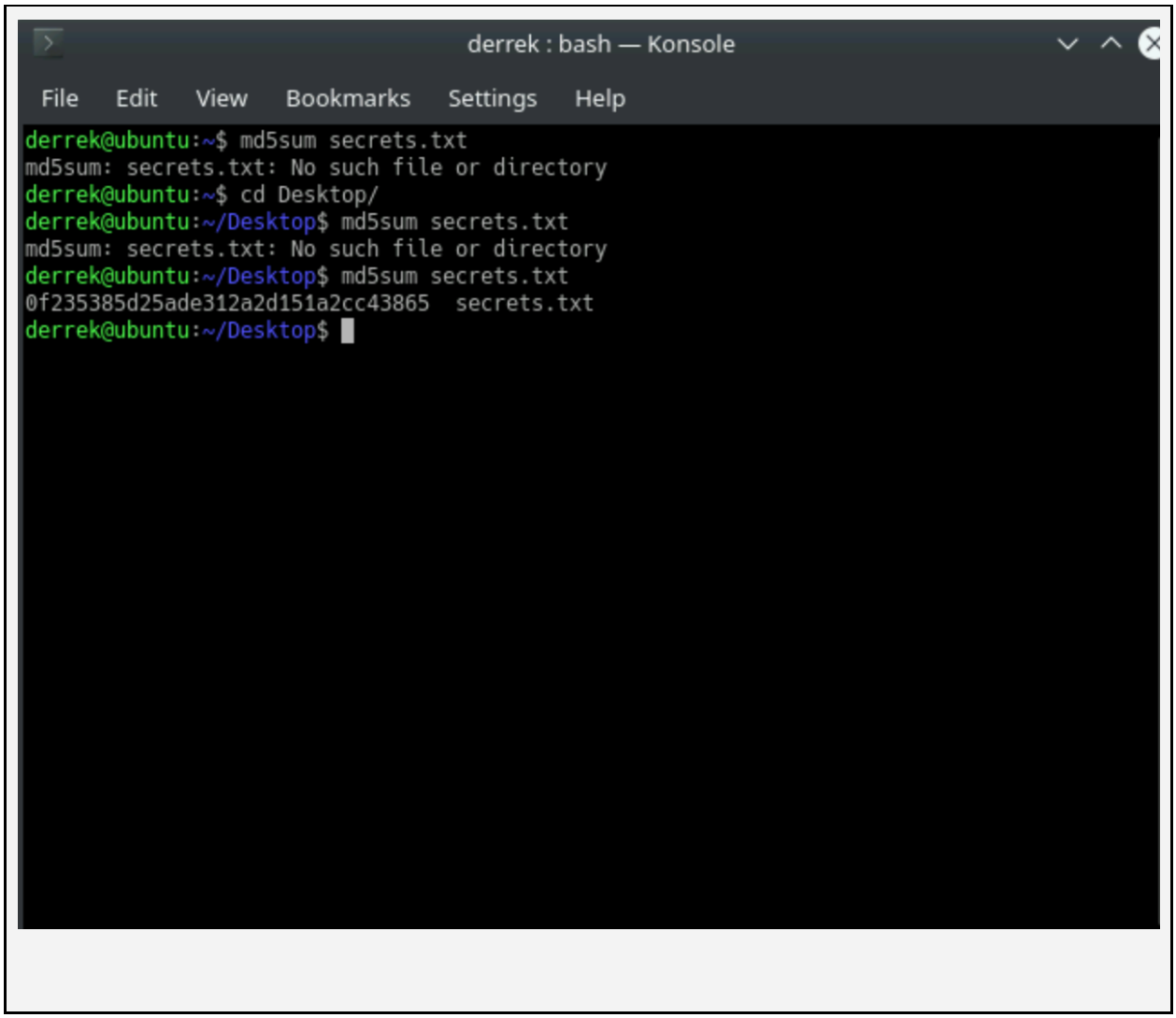
To maintain proper chain of custody, I generated an MD5 hash of secrets.txt:

md5sum secrets.txt

0f235385d25ade312a2d151a2cc43865 secrets.txt

Hash value: 0f235385d25ade312a2d151a2cc43865

This cryptographic fingerprint proves the file hasn't been modified since I extracted it.


derrek : bash — Konsole
File Edit View Bookmarks Settings Help
derrek@ubuntu:~$ md5sum secrets.txt
md5sum: secrets.txt: No such file or directory
derrek@ubuntu:~$ cd Desktop/
derrek@ubuntu:~/Desktop$ md5sum secrets.txt
md5sum: secrets.txt: No such file or directory
derrek@ubuntu:~/Desktop$ md5sum secrets.txt
0f235385d25ade312a2d151a2cc43865 secrets.txt
derrek@ubuntu:~/Desktop$

Findings and Analysis

Finding	Finding Details	Description
User Account	Aaron	System user account associated with the compromised workstation where the memory dump was captured. This account had access to encrypted archives and classified information.
Malicious File	gift.7z, suspicious.docx, secrets.txt	Multiple files used in the data exfiltration chain. The encrypted 7z archive contained a DOCX file which itself contained hidden classified forensic documentation.
Hash	0f235385d25ade312a2d151a2cc43865	MD5 hash of secrets.txt for forensic evidence integrity and chain of custody.
Attack Technique	T1027 - Obfuscation & Defense Evasion, T1052 - Exfiltration Over Physical Medium	Multi-layer encryption, nested archive structures, misleading document content, and password protection used to conceal and stage classified information for unauthorized exfiltration.
Process	7-Zip Archive Handler, String Extraction Process	Processes involved in creating encrypted archives and handling sensitive data. The process memory contained plaintext passwords and command-line arguments revealing the encryption scheme.

Summary of Investigation Findings

The investigation exposed a sophisticated data exfiltration scheme:

- AES-256 encrypted archive (gift.7z)
- DOCX container disguised as empty document
- Classified payload hidden in DOCX internal file structure
- Encryption password stored unencrypted in process memory