DEPI Graduation Project

Digital Artifacts: From Deletion to Detection

Team Members:

Mohamed Elansary
Ahmed Emad
Ahmed Hussien
Elaf Wael
Mariam Ibrahim

Table Of Contents

Abstract

This volume presents a multi-layered examination of digital forensics and threat analysis, progressing from storage media recoverability to advanced malware detection. The first chapter investigates the persistence of data on removable USB flash drives, addressing the challenge of anti-forensic deletion strategies. By conducting a controlled comparison of normal deletion, full formatting, and secure wiping (SDelete), the study utilizes sector-level inspection and file carving to quantify exactly how much data remains recoverable after each method, providing empirical validation for sanitization protocols.

Transitioning from data destruction to data concealment, the second chapter introduces "Hidden Pawprints," a narrative-driven training laboratory designed to teach the detection of steganography and metadata manipulation. Through a scenario-based investigation of a fictional subject, this module gamifies complex concepts such as alternate data streams and file-type misdirection, bridging the gap between theoretical forensics and practical application.

The final chapter addresses the active execution of malicious code through a comprehensive technical analysis of the Zeus (Zbot) banking trojan. This section details the development of "GoMal," a custom Golang-based tool built to automate PE header inspection and entropy calculation. The study maps Zeus’s sophisticated evasion mechanisms—including process injection and "Run and Dump" tactics—to the MITRE ATT&CK framework, concluding with actionable YARA rules and threat intelligence to counter financial malware.

Introduction

In the modern digital landscape, the battle between adversaries and defenders is defined by a constant cycle of concealment and discovery. Cybercriminals and malicious actors continuously evolve their methods to destroy evidence, hide communications, and execute sophisticated attacks. Consequently, the role of the forensic analyst has expanded from simple data retrieval to a complex discipline requiring deep technical knowledge of storage media, data obfuscation, and reverse engineering. This volume, **Digital Artifacts: From Deletion to Detection**, presents a tripartite exploration of these challenges, offering empirical research, educational frameworks, and advanced technical analysis.

The book begins at the foundational level of digital forensics: **Data Recovery and Deletion Forensics**. As adversaries increasingly rely on anti-forensic deletion strategies to obscure their tracks, it becomes critical to understand the permanence of digital data. **Chapter 1** conducts a rigorous empirical study on removable media, utilizing tools like PhotoRec and SDelete to quantify the recoverability of data following normal deletion, formatting, and secure wiping. This research dispels common myths regarding data destruction and provides investigators with a baseline for what can—and cannot—be recovered from flash storage.

Building upon the theme of hidden information, **Chapter 2** shifts focus to the pedagogical challenges of training the next generation of analysts. **"The Hidden Pawprints Forensics Lab"** introduces a narrative-driven educational framework designed to teach the detection of **Steganography and Data Concealment**. Through an immersive scenario involving the fictional "Mr. Whiskers," this chapter bridges the gap between theory and practice, guiding learners through the intricacies of Least Significant Bit (LSB) encoding, Alternate Data Streams (ADS), and metadata manipulation. This section emphasizes that effective

forensics requires not just tools, but the analytical mindset to spot anomalies in seemingly innocent files.

Finally, the text advances from passive artifacts to active threats with Chapter 3: Technical Analysis of the Zeus Banking Trojan. This section provides a deep-dive technical analysis of one of history's most notorious malware strains. By employing a hybrid analysis methodology and introducing "GoMal," a custom-developed Golang tool for automated entropy calculation and PE inspection, this research maps the malware's evasion tactics to the MITRE ATT&CK framework. From "Run and Dump" tactics to process injection, this chapter demonstrates the complexity of modern financial malware and the necessity of advanced tooling in threat detection.

Together, these chapters provide a holistic view of the digital forensic landscape, equipping readers with the knowledge to recover deleted traces, uncover hidden secrets, and dissect malicious code.

Chapter 1 – Anti-Forensics and Data Obfuscation Tactics

1. Introduction

1.1 Background & Context

Deletion is one of the most widely used anti-forensic tactics by individuals attempting to obscure incriminating evidence. However, the effectiveness of deletion varies greatly depending on the method used and the underlying behavior of the file system.
USB flash drives are especially relevant targets due to their widespread use, mobility, and the frequency with which they appear in legal investigations. Understanding how deleted data persists in flash-based environments is essential for forensic analysts.

1.2 Problem Statement

Digital adversaries often rely on deletion as a method of hiding evidence. Yet the actual impact of these deletion methods on data recoverability remains misunderstood.
This chapter aims to:
Evaluate how normal deletion, full formatting, and secure deletion affect the recoverability of data on removable USB media.

1.3 Objectives

- Forensically analyze three deletion methods.
- Measure data remnants and recovery success.
- Assess the anti-forensic value of each deletion technique.
- Align experimental findings with established forensic theory.

- Provide practical insights for investigators facing data-obfuscation attempts.

1.4 Scope

This study focuses on:

- Removable USB flash drives
- FAT32 / exFAT file systems
- Image, document, and text datasets
- Carving, hashing, and sector analysis

1.5 Limitations

Excluded from this study:

- NTFS MFT behavior
- SSD TRIM operations
- Hardware-based secure erase
- Encryption-based anti-forensics
- Anti-forensic rootkits or system-level manipulation

2. Literature Review / Background Research

2.1 File System Behavior During Deletion

Scholarly literature consistently shows that the deletion operation in common file systems such as FAT32 does not erase data; it only marks clusters as free. The data remains intact until overwritten.

Key structures involved:

- File Allocation Table (FAT)
- Directory entries
- Data clusters
- Slack space

2.2 File Carving in Forensic Recovery

File carving relies on identifying file signatures and reconstructing binary patterns. It compensates for missing metadata and is valuable against anti-forensic tactics attempting to destroy directory references.

2.3 Anti-Forensic Deletion Methods

Research categorizes anti-forensic deletion methods into:

- Non-destructive deletion (normal delete)
- Metadata-destructive deletion (formatting)
- Data-destructive deletion (secure overwriting)

Tools like SDelete implement multiple overwrite passes to ensure data sanitization. Academic research highlights overwriting as the only reliable guarantee of non-recoverability.

2.4 Relevance to Removable Storage

Flash-based media introduces unique behaviors:

- Wear leveling
- Non-sequential cluster allocation
- Limited write cycles
  Although wear leveling complicates overwriting guarantees, secure deletion still performs reliably in controlled environments.

3. Methodology / Project Design

3.1 Experimental Workflow

To ensure reproducibility, a controlled experimental setup was used:

1. **Dataset Creation**
   A balanced dataset of images, documents, PDFs, and text files was prepared.
2. **Apply Deletion Method**
   Each scenario was executed separately on the same USB device:

- **Normal Deletion**
- **Full Format**
- **Secure Delete (SDelete)**

3. **Forensic Acquisition**
   The device was scanned without writing new data.
4. **File Carving**
   PhotoRec was used to recover files based on signatures.
5. **Hashing Validation**
   Recovered files were compared using SHA-256 hashing.

3.2 Tools Used

- **PhotoRec – signature-based carving**
- **SDelete – secure deletion utility**
- **SHA-256 hasher – integrity checking**
- **Windows OS – deletion environment**

4. Implementation / Results

4.1 Scenario 1 – Normal Deletion (Shift + Delete)

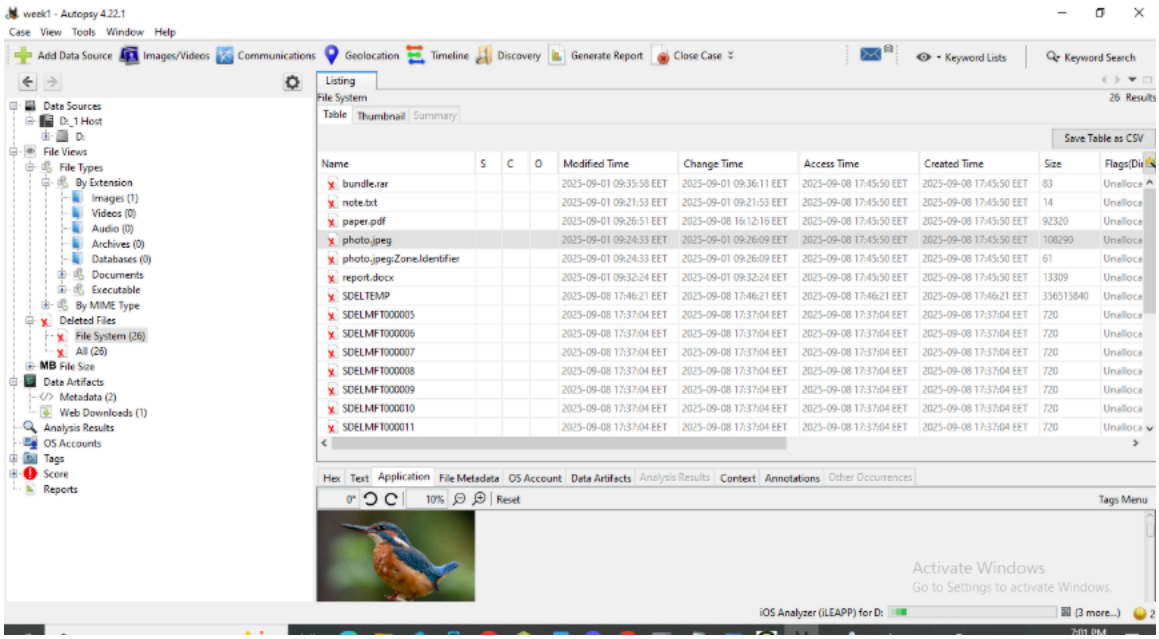
In this scenario, files were deleted using Shift + Delete. This removed only their directory references while leaving the underlying cluster data intact. PhotoRec successfully recovered all files, and hashing confirmed their integrity.

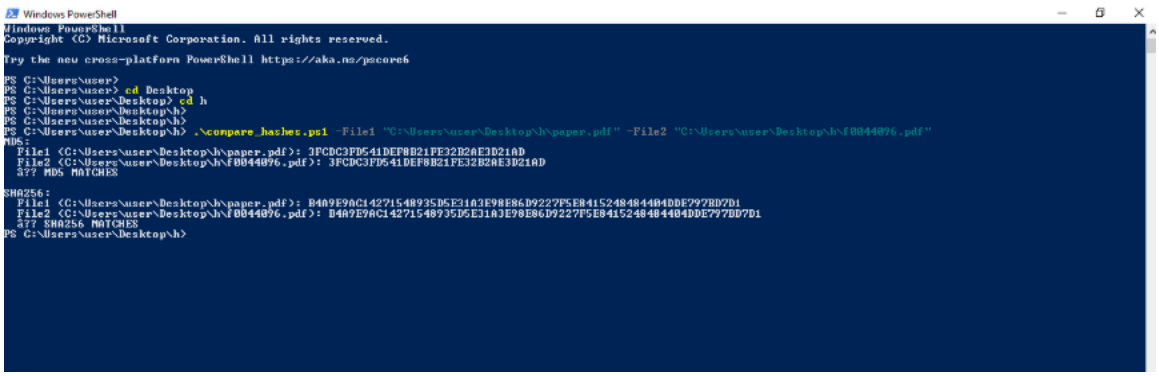
Procedure:

1. **Copy dataset**
2. **Delete using Shift + Delete**
3. **Carve free space using PhotoRec**
4. **Recover and hash-verify files**

Findings:

- **100% recovery**
- **All files intact**
- **Hashes matched originals**





4.2 Scenario 2 – Full Format

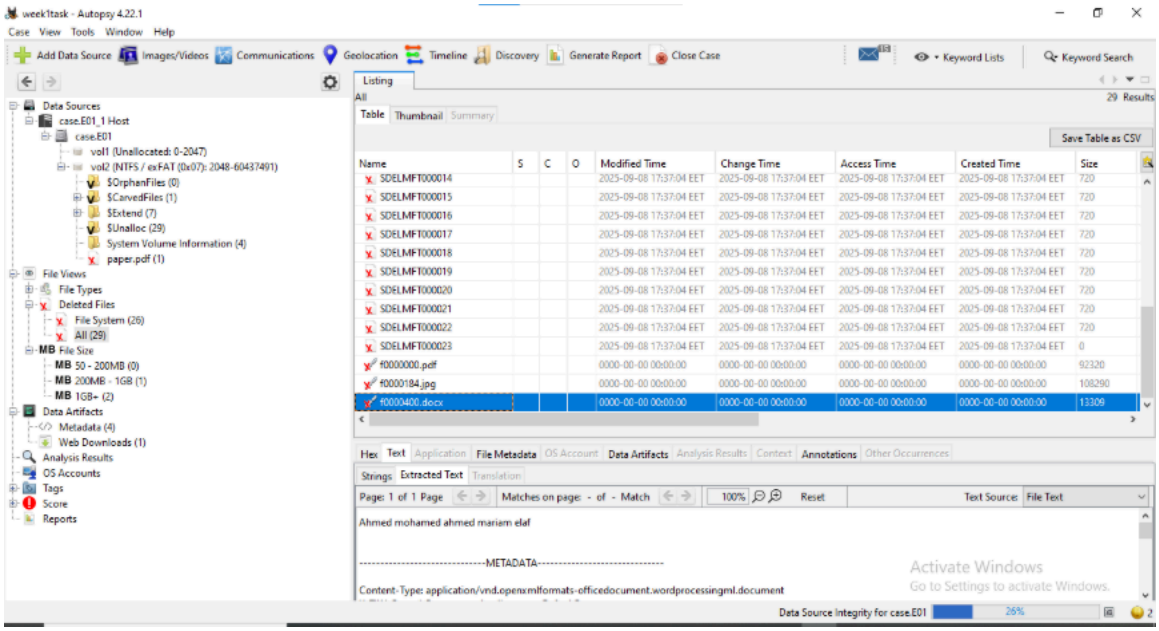
A full format was applied to the USB drive. Although metadata structures were rebuilt,

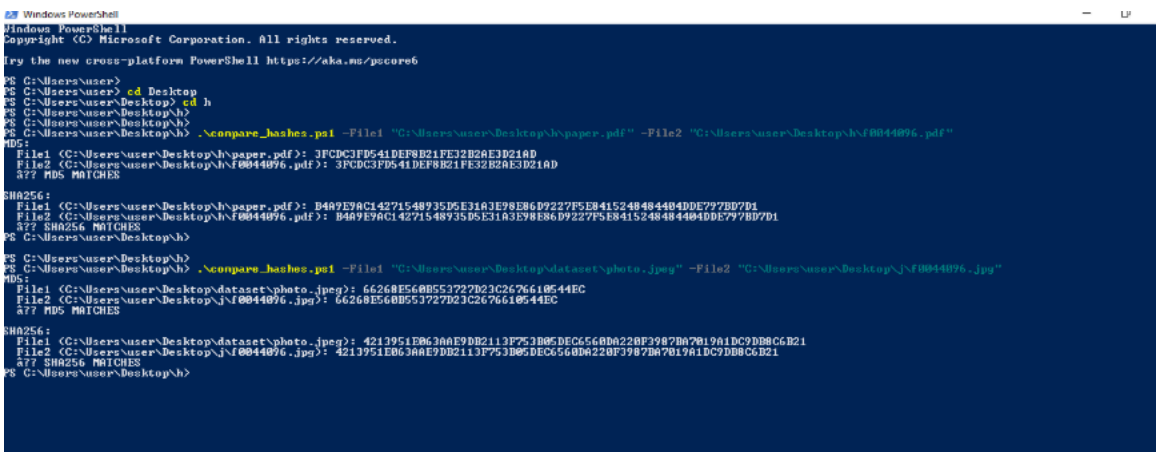
the content stored in data clusters remained largely untouched. File carving allowed partial
recovery: some files were intact while others were corrupted.
Procedure:

1. Perform full format
2. Run PhotoRec on unallocated space

Findings:

- Partial recovery
- Some files intact, others corrupted
- Metadata destroyed but data remnants preserved





4.3 Scenario 3 – Secure Deletion using SDelete

**SDelete overwrites free space with random or zero-filled patterns. This eliminated all
traces of the dataset. PhotoRec found no recoverable fragments, confirming that overwriting
is the only effective method for preventing recovery.**

Procedure:

Commands used:

sdelete -c E:

sdelete -z E:

Findings:

- **0% recovery**
- **No fragments, no headers**

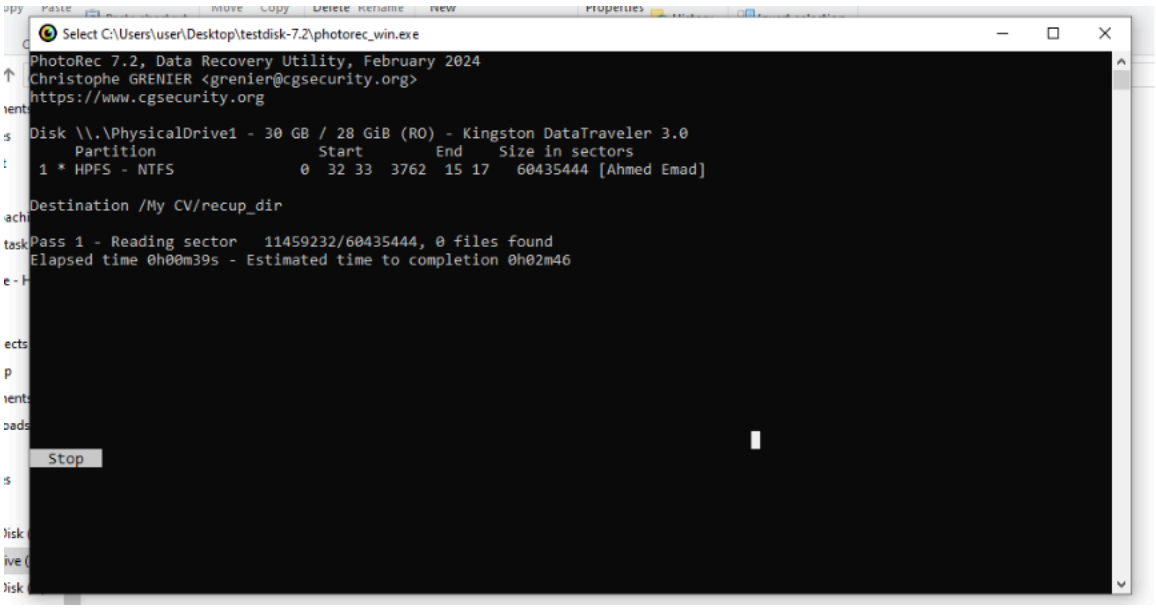
```
C:\Users\user\Desktop\SDelete>
C:\Users\user\Desktop\SDelete>sdelete64 -p 1 D:\note.txt

SDelete v2.05 - Secure file delete
Copyright (C) 1999-2023 Mark Russinovich
Sysinternals - www.sysinternals.com

SDelete is set for 1 pass.
D:\note.txt...deleted.

Files deleted: 1

C:\Users\user\Desktop\SDelete>
```



5. Discussion / Analysis

The results clearly show that normal deletion offers no anti-forensic protection, as all files remain fully recoverable. Full formatting provides moderate obfuscation by destroying metadata, but file remnants remain available. Only secure deletion effectively eliminates all recoverable content. These findings reinforce the importance of forensic

carving when investigating deleted data and demonstrate how attackers may misjudge the
effectiveness of common deletion methods.
The table below summarizes recoverability and anti-forensics effectiveness:

Technique	Tools Used	Result	Recoverability
Normal Delete / Shift Delete	PhotoRec	All files recovered from unallocated space	High
Full Format	PhotoRec	Partial recovery, some data corrupted or lost	Moderate
Secure Delete (SDelete)	PhotoRec, Hex Analysis	No data recovered, unallocated space overwritten	None

6. Conclusion

This investigation clearly demonstrates that not all deletion methods provide the same level of anti-forensic value. Normal deletion offers no protection, as all files remain fully recoverable through carving tools. Full formatting provides moderate obfuscation, removing file system metadata but still leaving most underlying data intact and partially recoverable. Only secure deletion using SDelete proved effective at preventing recovery by overwriting unallocated space and eliminating all file remnants.
These findings emphasize a fundamental point: deletion does not equal destruction. For attackers, relying on simple deletion or

formatting is insufficient to hide activity. For forensic analysts, this reinforces the importance of examining unallocated and slack space, applying carving tools, and verifying recovered data through hashing.
In the context of **Anti-Forensics and Data Obfuscation Tactics**, this chapter highlights the ongoing tension between efforts to conceal digital artifacts and the forensic techniques designed to uncover them. It demonstrates both the limitations of common anti-forensic practices and the effectiveness of proper forensic methodology in revealing hidden or deleted evidence.

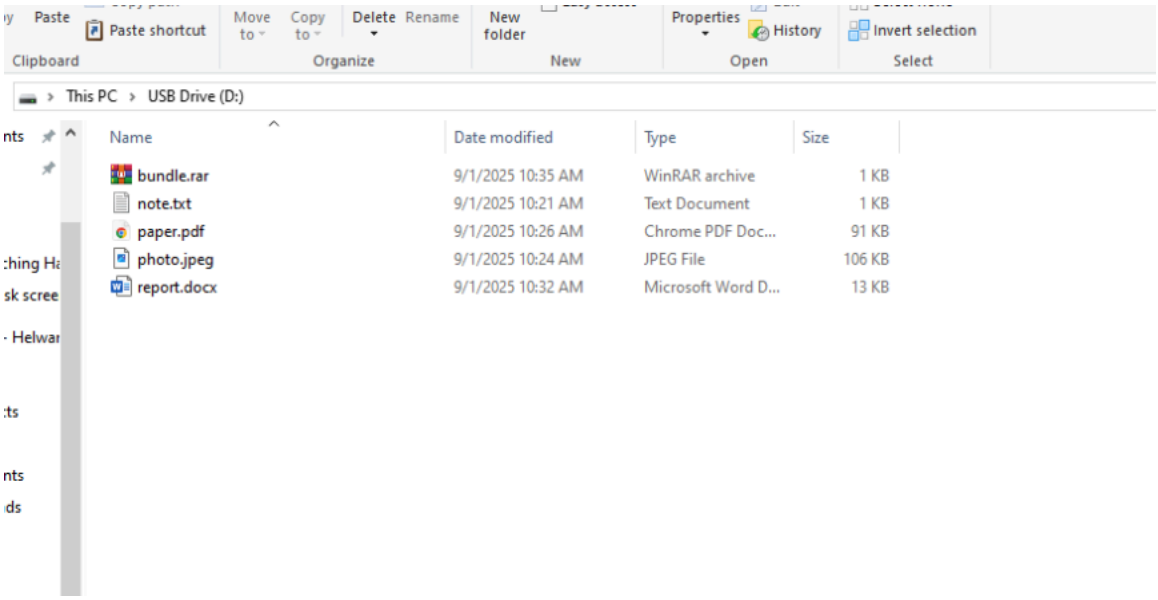
7. Future Work

Future investigations should extend this work to:

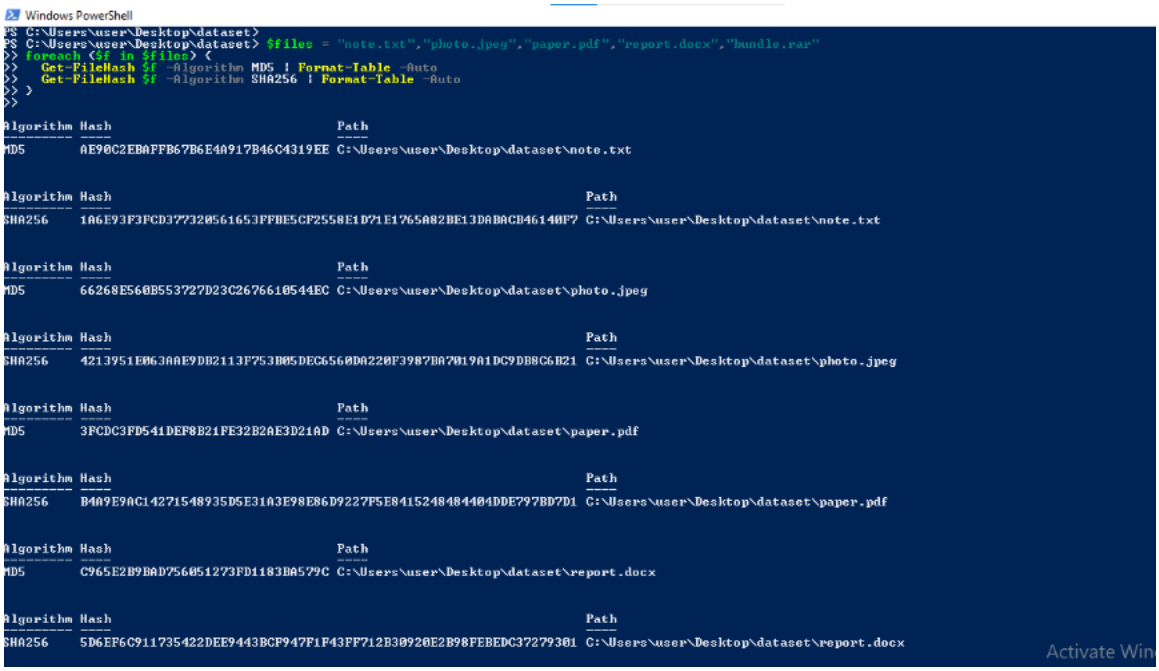
- NTFS-specific deletion behavior
- SSD TRIM effects on recoverability
- Hardware-level sanitization commands
- Encrypted USB environments
- Anti-forensic malware that manipulates allocation tables

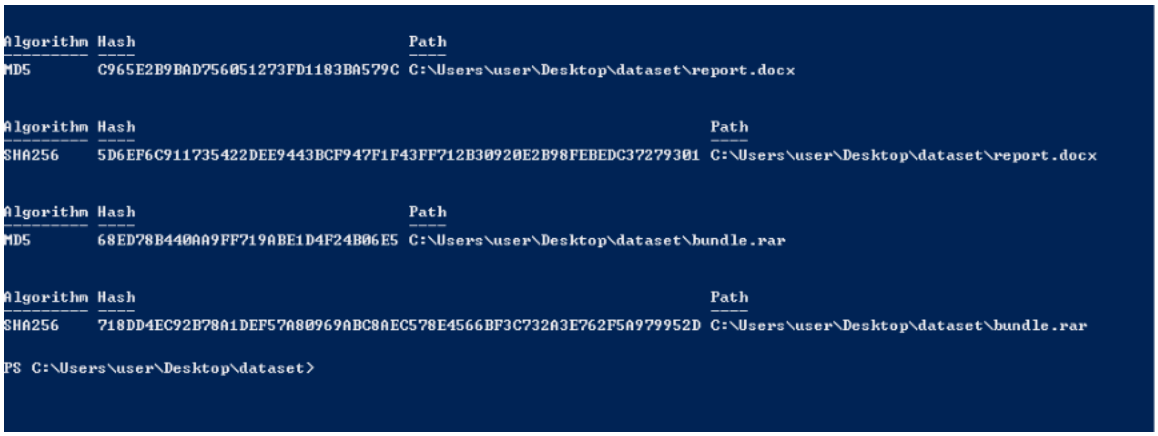
8. Appendices

Dataset Creation Screenshot



Files Hashes before experiment Screenshots





Chapter 2 - The Hidden Pawprints

Introduction

The "Hidden Pawprints" digital forensics project is a narrative-driven training environment designed for graduate-level study in digital forensics, cyber investigation, and data concealment techniques. Through a structured progression of six forensic modules, the project introduces and reinforces core concepts essential for analyzing hidden data across various operating systems and file types. The investigative storyline surrounding the fictional subject, "Mr. Whiskers," creates an immersive framework that strengthens experiential learning.

Background & Context

Digital forensics as a discipline requires analysts to detect, interpret, and extract hidden or manipulated data from digital artifacts. While many training resources provide tools and examples, few offer an integrated, narrative-driven experience that synthesizes learning across multiple methods of obfuscation. The "Hidden Pawprints" lab addresses this gap by guiding learners through realistic concealment scenarios involving steganography, metadata manipulation, alternate data streams, and file-type misdirection.

Problem Statement

Modern cyber investigations involve detecting sophisticated techniques that adversaries use to hide data, evade detection, or mislead forensic analysts. However, beginners often struggle to understand how these techniques operate in practice and how to use forensic tools effectively. There is a need for an educational framework that:

- Demonstrates multiple data-hiding methods in a cohesive, progressive format.
- Encourages analytical reasoning rather than simple tool usage.
- Provides a controlled but realistic environment for practicing forensic techniques.

Objectives

This project seeks to achieve the following objectives:

1. Introduce learners to core digital forensics concepts through hands-on practice.
2. Demonstrate various methods of data concealment across platforms and file types.
3. Provide clear pathways for detection and analysis of hidden or manipulated data.
4. Offer a narrative-driven investigative experience that enhances learning engagement.
5. Produce a complete, self-contained forensic lab that can be used for academic or training purposes.

Literature Review

Digital forensics training materials often address concepts such as steganography, data manipulation, metadata analysis, and alternate data streams in isolation. Research indicates that students benefit most when theory is paired with practical, scenario-based tasks. Prior forensic studies emphasize:

- **Steganography** as a widely used method for covert data exchange.
- **Least Significant Bit (LSB) methods** as foundational techniques taught in entry-level steganography courses.

- **Audio file steganography as an underexplored but important forensic challenge.**
- **NTFS Alternate Data Streams (ADS) as a powerful and often overlooked mechanism for hiding data within Windows file systems.**
- **Timestamp and metadata forensics as crucial for identifying falsification and detecting anomalies.**
- **File-type spoofing as a common tactic in malware distribution.**

This project builds upon these established concepts by integrating them into a single, cohesive investigative workflow.

Methodology

Overview

The project consists of six sequential forensic modules, each built around a specific concealment technique. Every module provides a hidden clue that leads the analyst to the next stage, forming a continuous investigative chain.

```
Catspiracy-Lab/
├── Challenge1_StegoVisible/
├── Challenge2_PythonLSB/
├── Challenge3_AudioStego/
├── Challenge4_ADS/
├── Challenge5_Timestamps/
└── Challenge6_TypeMismatch/
```

Tools & Environment

Operating Systems

- **Linux (Kali, Ubuntu) for file analysis, metadata inspection, scripting, and steganography.**
- **Windows (NTFS filesystem) for ADS analysis.**

Tools Used

- **steghide for embedding and extracting data in image and audio files.**
- **Python (Pillow Library) for custom LSB steganography scripts.**
- **exiftool, stat, file, strings, hexedit for metadata, timestamp, and file-type analysis.**
- **cmd.exe and PowerShell for ADS inspection.**

Custom Tool Development

A set of Python scripts were written to embed and extract LSB-based hidden data. These scripts demonstrate fundamental concepts of digital image manipulation and bit-level data encoding.

Implementation / Results / Findings

Module 1: Image Steganography

Using steghide, a secret text file was embedded into an image. The clue was successfully extracted with the correct passphrase, demonstrating classic password-protected steganography.

Step1:

```
ubuntu@ElafElsaeed-group2:~$ cd /home/ubuntu/Desktop
mkdir -p Catspiracy-Lab/Challenge1_StegoVisible
cd Catspiracy-Lab/Challenge1_StegoVisible
ubuntu@ElafElsaeed-group2:~/Desktop/Catspiracy-Lab/Challenge1_StegoVisible$
```

Step2:

```
echo "Clue: Look for kitten_secret.png..." > secret_message.txt
```

```
ubuntu@ElafElsaeed-group2:~/Desktop/Catspiracy-Lab/Challenge1_StegoVisible$ echo
 "Clue: Look for kitten_secret.png in the next folder 🐾" > secret_message.txt
ubuntu@ElafElsaeed-group2:~/Desktop/Catspiracy-Lab/Challenge1_StegoVisible$ ls
mrwhisker1.png  secret_message.txt
ubuntu@ElafElsaeed-group2:~/Desktop/Catspiracy-Lab/Challenge1_StegoVisible$
```

```
ubuntu@ElafElsaeed-group2:~/Desktop/Catspiracy-Lab/Challenge1_StegoVisible$ cat
secret_message.txt
Clue: Look for kitten_secret.png in the next folder 🐾
ubuntu@ElafElsaeed-group2:~/Desktop/Catspiracy-Lab/Challenge1_StegoVisible$
```

Step3:

```
steghide embed -cf mrwhisker1.jpg -ef secret_message.txt -p "purrfect"
```

```
ubuntu@ElafElsaeed-group2:~/Desktop/Catspiracy-Lab/Challenge1_StegoVisible$ ls
mrwhisker1.jpg  mrwhisker1.png  secret_message.txt
ubuntu@ElafElsaeed-group2:~/Desktop/Catspiracy-Lab/Challenge1_StegoVisible$ steg
hide embed -cf mrwhisker1.jpg -ef secret_message.txt -p "purrfect"
embedding "secret_message.txt" in "mrwhisker1.jpg"... done
ubuntu@ElafElsaeed-group2:~/Desktop/Catspiracy-Lab/Challenge1_StegoVisible$ ls
mrwhisker1.jpg  mrwhisker1.png  secret_message.txt
ubuntu@ElafElsaeed-group2:~/Desktop/Catspiracy-Lab/Challenge1_StegoVisible$
```

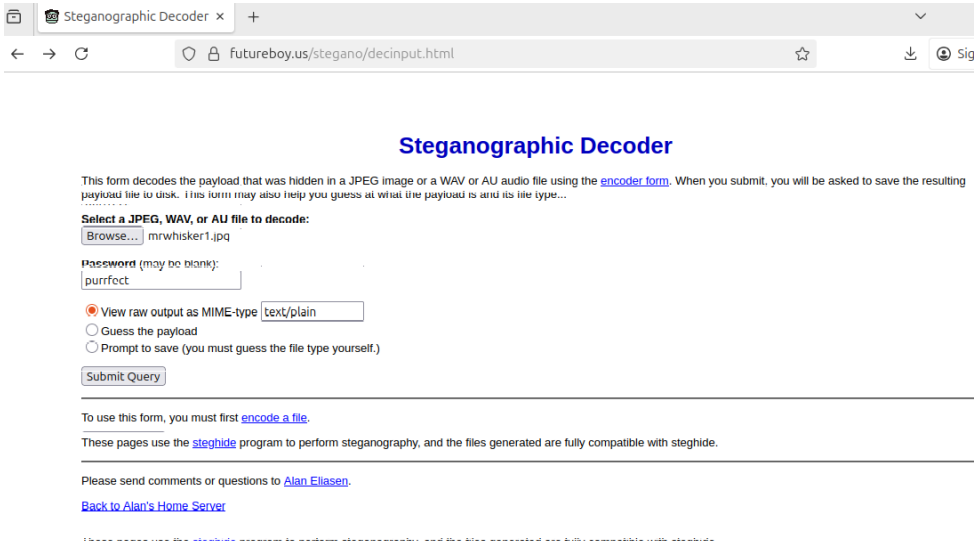
Result :

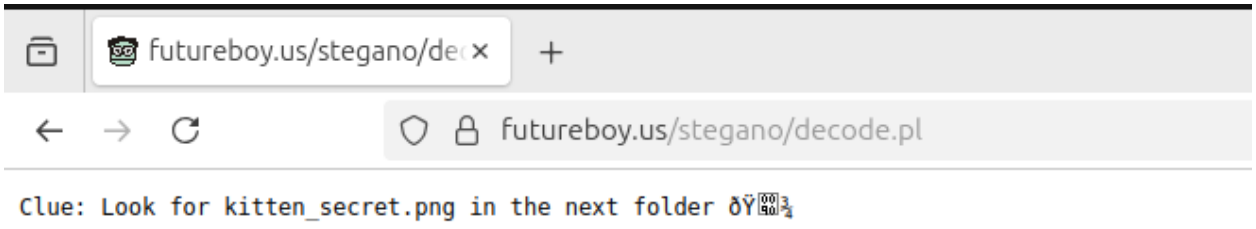
1.The analyst extracts the message with steghide extract -sf cat_meme.jpg -p "purrfect".

```
ubuntu@ElafElsaeed-group2:~/Desktop/Catspiracy-Lab/Challenge1_StegoVisible$ steg
hide embed -cf mrwhisker1.jpg -ef secret_message.txt -p "purrfect"
embedding "secret_message.txt" in "mrwhisker1.jpg"... done
ubuntu@ElafElsaeed-group2:~/Desktop/Catspiracy-Lab/Challenge1_StegoVisible$ ls
mrwhisker1.jpg  mrwhisker1.png  secret_message.txt
ubuntu@ElafElsaeed-group2:~/Desktop/Catspiracy-Lab/Challenge1_StegoVisible$ steg
hide extract -sf mrwhisker1.jpg -p "purrfect"
the file "secret_message.txt" does already exist. overwrite ? (y/n) y
wrote extracted data to "secret_message.txt".
ubuntu@ElafElsaeed-group2:~/Desktop/Catspiracy-Lab/Challenge1_StegoVisible$ cat
secret_message.txt
Clue: Look for kitten_secret.png in the next folder 🐾
ubuntu@ElafElsaeed-group2:~/Desktop/Catspiracy-Lab/Challenge1_StegoVisible$
```

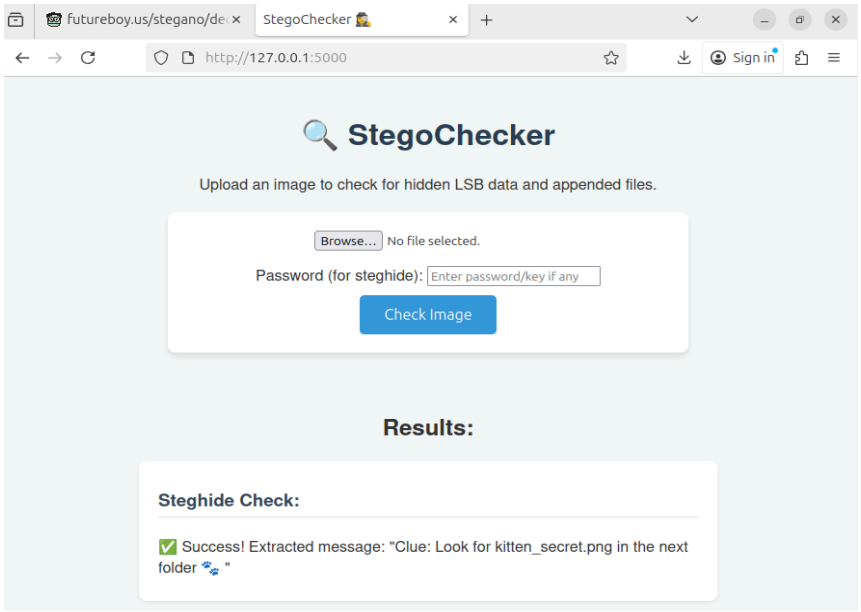
Checking:

1.Using Online tool (futureboy.us):





2.using Our website:



Module 2: LSB Steganography

A custom Python script modified the least significant bits of the image pixels to embed a hidden message. A corresponding decoder script allowed analysts to retrieve the embedded clue.

Step1: Python_encoder_script.py

```python
from PIL import Image
import numpy as np
import struct

# ---- SETTINGS ----
input_image = "kitten_org.png"          # Original image
output_image = "kitten_secret.png"      # Output image with hidden text
secret_message = "The meow.wav file hides Mr.Whiskers plan"

# ------------------

# Convert message to bytes
message_bytes = secret_message.encode('utf-8')
message_length = len(message_bytes)

# Add a 4-byte header that stores message length (big-endian)
header = struct.pack('>I', message_length)
data = header + message_bytes

# Convert all bytes (header + message) to bits (0/1)
bits = np.unpackbits(np.frombuffer(data, dtype=np.uint8))

# Load the image and make sure it's RGB
img = Image.open(input_image).convert('RGB')
pixels = np.array(img)
flat_pixels = pixels.flatten()

# Check if the image is big enough
if len(bits) > len(flat_pixels):
    raise ValueError("Image is too small to hold the secret message!")

# Set the least significant bit (LSB) of each pixel byte to our message bit
flat_pixels[:len(bits)] = (flat_pixels[:len(bits)] & 0xFE) | bits

# Reshape and save the new image
encoded_pixels = flat_pixels.reshape(pixels.shape)
encoded_img = Image.fromarray(encoded_pixels, 'RGB')
encoded_img.save(output_image)
```

Step2:
Run Script to encode message to output new embedded file ‘kitten_secret.png’

python3 Python_encoder_script.py

```
moon@moonVm:~/Desktop/Catspiracy-Lab/Challenge 2 - Python LSB Stego$ python3 Python_encoder_script.py
moon@moonVm:~/Desktop/Catspiracy-Lab/Challenge 2 - Python LSB Stego$ ls
kitten_org.png  kitten_secret.png  Python_encoder_script.py
```

Step3:
verify creation:

```
moon@moonVm:~/Desktop/Catspiracy-Lab/Challenge 2 - Python LSB Stego$ ls
kitten_org.png  kitten_secret.png  Python_decoder_script.py  Python_encoder_script.py
```

Step4:
Python_decoder_script.py

```python
from PIL import Image
import numpy as np
import struct

# ---- SETTINGS ----
stego_image = "kitten_secret.png"  # The image that contains the hidden message
# ------------------

# Load and flatten pixel data
img = Image.open(stego_image).convert('RGB')
pixels = np.array(img).flatten()

# Read the first 32 bits (4 bytes) for the message length
header_bits = pixels[:32] & 1
header_bytes = np.packbits(header_bits).tobytes()
message_length = struct.unpack('>I', header_bytes)[0]

# Read message bits (8 bits per byte)
message_bits = pixels[32:32 + message_length * 8] & 1
message_bytes = np.packbits(message_bits).tobytes()

# Decode to string
message = message_bytes.decode('utf-8')
print("💬 Hidden message found:")
print(message)
```

Result:
A decoder script (`Python_decoder_script.py`) is run, revealing the next clue.
python3 Python_decoder_script.py

```
moon@moonVm:~/Desktop/Catspiracy-Lab/Challenge 2 - Python LS
Clue: The meow.wav file hides Mr. Whiskers' plan
```

Module 3: Audio Steganography

Steghide was used to embed a clue into meow.wav with the password "purrfect". Successful extraction confirmed that steganographic techniques extend beyond images.

Step1:

download the audio and check its working :

```
ubuntu@ElafElsaeed-group2:~/Desktop/Catspiracy-Lab/Challenge3_AudioStego$ ffmpeg
 -i meow.ogg -ar 44100 -ac 2 meow.wav
ffmpeg version 6.1.1-3ubuntu5 Copyright (c) 2000-2023 the FFmpeg developers
  built with gcc 13 (Ubuntu 13.2.0-23ubuntu3)
  configuration: --prefix=/usr --extra-version=3ubuntu5 --toolchain=hardened --l
ibdir=/usr/lib/x86_64-linux-gnu --incdir=/usr/include/x86_64-linux-gnu --arch=am
d64 --enable-gpl --disable-stripping --disable-omx --enable-gnutls --enable-liba
om --enable-libass --enable-libbs2b --enable-libcaca --enable-libcdio --enable-l
```


```
ubuntu@ElafElsaeed-group2:~/Desktop/Catspiracy-Lab/Challenge3_AudioStego$ ffmpeg
 -i meow.ogg -ar 44100 -ac 2 meow.wav
ffmpeg version 6.1.1-3ubuntu5 Copyright (c) 2000-2023 the FFmpeg developers
  built with gcc 13 (Ubuntu 13.2.0-23ubuntu3)
  configuration: --prefix=/usr --extra-version=3ubuntu5 --toolchain=hardened --l
ibdir=/usr/lib/x86_64-linux-gnu --incdir=/usr/include/x86_64-linux-gnu --arch=am
d64 --enable-gpl --disable-stripping --disable-omx --enable-gnutls --enable-liba
om --enable-libass --enable-libbs2b --enable-libcaca --enable-libcdio --enable-l
```


Step2:

```
ubuntu@ElafElsaeed-group2:~/Desktop/Catspiracy-Lab/Challenge3_AudioStego$ echo "
Clue: ADS hides the next secret 🐾" > secret.txt
ubuntu@ElafElsaeed-group2:~/Desktop/Catspiracy-Lab/Challenge3_AudioStego$ steghi
de embed -cf meow.wav -ef secret.txt -p "purrfect"
embedding "secret.txt" in "meow.wav"... done
ubuntu@ElafElsaeed-group2:~/Desktop/Catspiracy-Lab/Challenge3_AudioStego$ steghi
de info meow.wav
"meow.wav":
  format: wave audio, PCM encoding
  capacity: 5.6 KB
Try to get information about embedded data ? (y/n) y
Enter passphrase:
  embedded file "secret.txt":
```

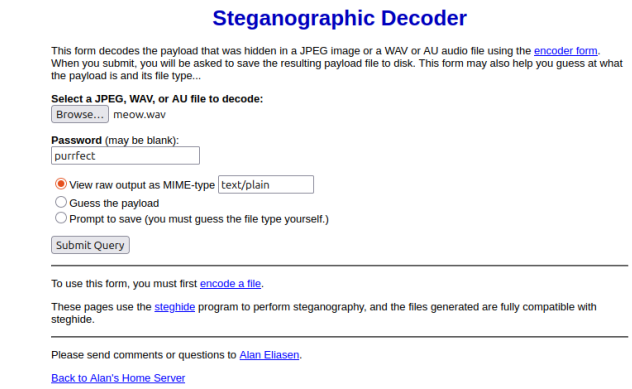
Step3:
check

```
Enter passphrase:
  embedded file "secret.txt":
    size: 37.0 Byte
    encrypted: rijndael-128, cbc
    compressed: yes
```

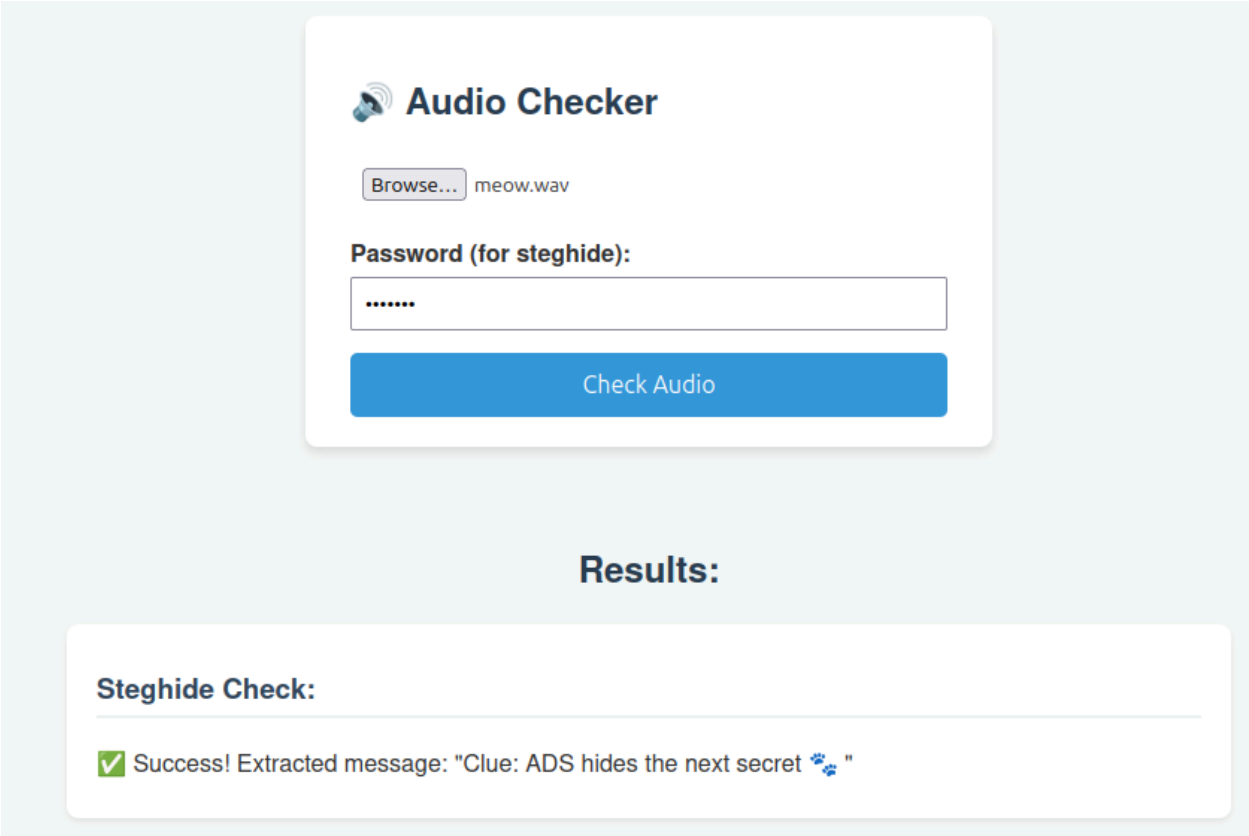
Result :
The analyst uses `steghide extract -sf meow.wav -p "purrfect"` to retrieve the hidden text

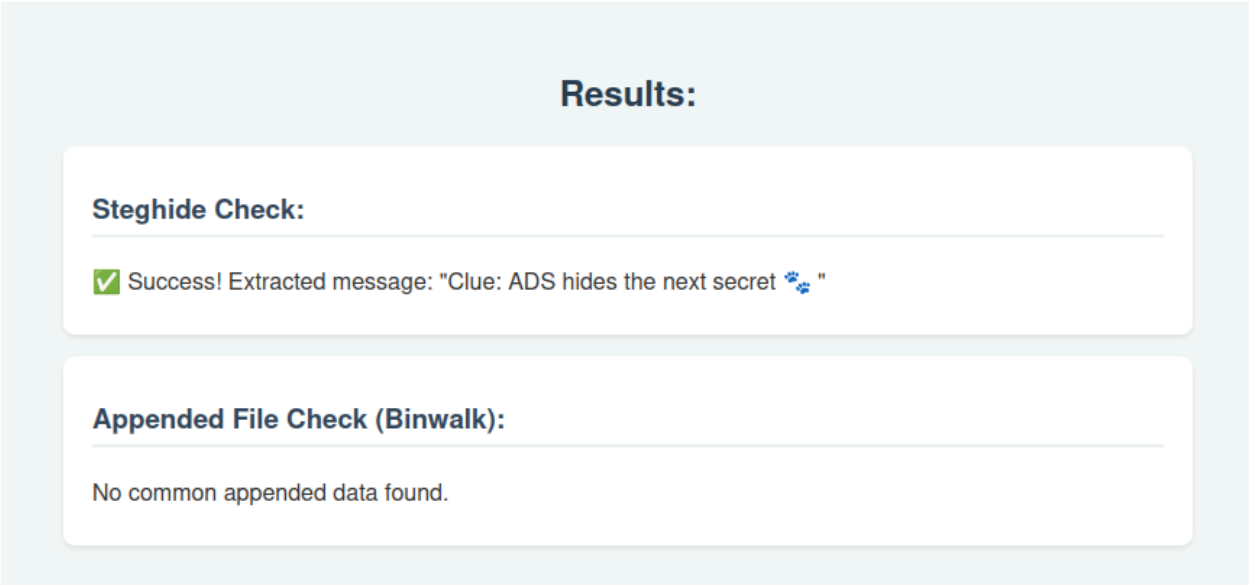
```
ubuntu@ElafElsaeed-group2:~/Desktop/Catspiracy-Lab/Challenge3_AudioStego$ steghi
de extract -sf meow.wav -p "purrfect"
cat secret.txt
the file "secret.txt" does already exist. overwrite ? (y/n) y
wrote extracted data to "secret.txt".
Clue: ADS hides the next secret 🐾
```

Checking:
1.Using Online tool (futureboy.us):



2.using Our website:

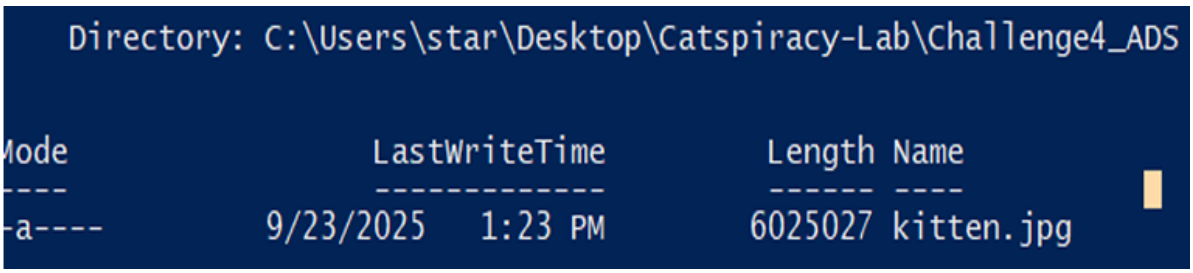




Module 4: Alternate Data Streams (ADS)

On Windows NTFS, a hidden data stream was attached to a JPEG file. Tools such as `dir /R` and PowerShell's stream inspection commands allowed the analyst to detect and extract the hidden text.

Step1:
Open Command Prompt (cmd.exe) and run:
`Ls`

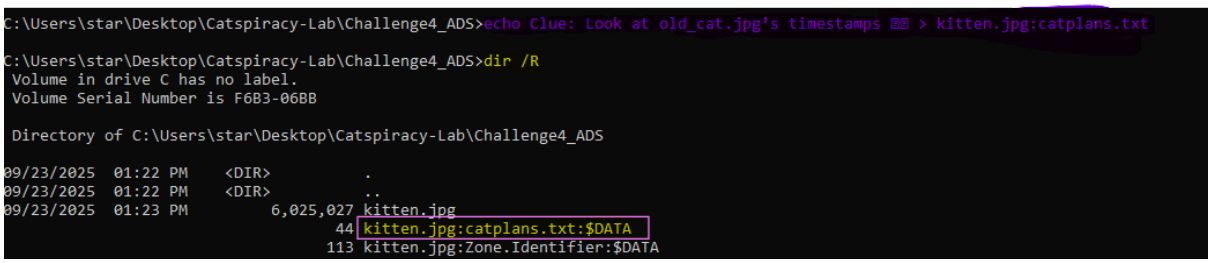


Step2:
Create the ADS stream:

```
echo Clue: Look at old_cat.jpg's timestamps 🐾 > kitten.jpg:catplans.txt
```

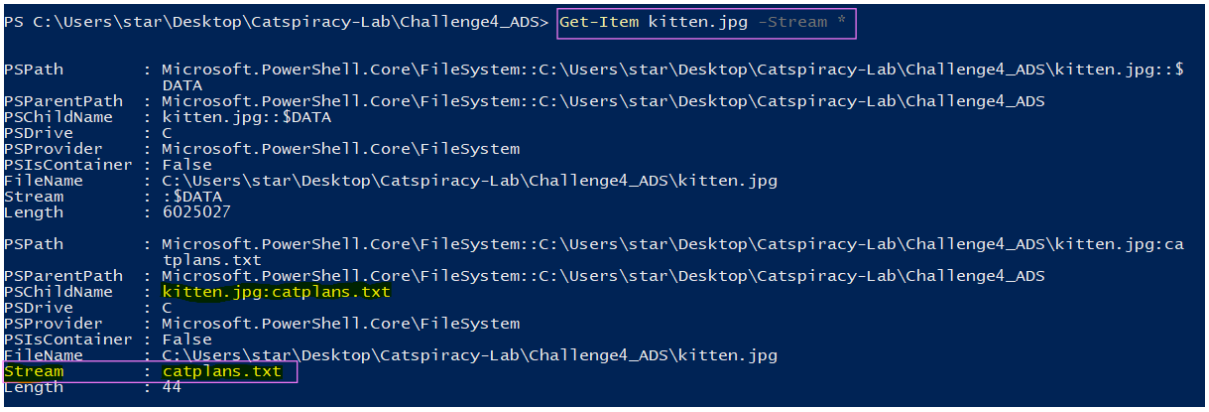
Step3:Verify the stream exists using cmd

```
dir /R
```



Step4:Verify the stream exists using powershell

```
Get-Item kitten.jpg -Stream *
```



Result :

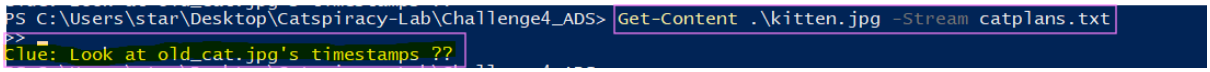
Read the hidden stream using cmd

```
more < kitten.jpg:catplans.txt
```

```
C:\Users\star\Desktop\Catspiracy-Lab\Challenge4_ADS>more < kitten.jpg:catplans.txt
Clue: Look at old_cat.jpg's timestamps ??
```

Read the hidden stream using powershell

```
Get-Content .\kitten.jpg -Stream catplans.txt
```



Module 5: Metadata & Timestamp Manipulation

Timestamps were manipulated using Linux tools and exiftool. Analysts were required to compare filesystem metadata against embedded metadata fields to identify inconsistencies.

Step1:
check the image metadata before changing

```
ubuntu@ElafElsaeed-group2:~/Desktop/Catspiracy-Lab/Challenge5_Timestamps$ stat old_cat.png
exiftool old_cat.png
  File: old_cat.png
  Size: 8192            Blocks: 16         IO Block: 4096   regular file
Device: 8,2     Inode: 658997      Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/  ubuntu)   Gid: ( 1000/  ubuntu)
Access: 2025-10-16 12:21:08.121926580 +0300
Modify: 2025-10-16 12:21:08.181925663 +0300
Change: 2025-10-16 12:21:08.181925663 +0300
 Birth: 2025-10-16 12:21:08.121926580 +0300
ExifTool Version Number         : 12.76
File Name                       : old_cat.png
Directory                       : .
File Size                       : 8.2 kB
File Modification Date/Time     : 2025:10:16 12:21:08+03:00
File Access Date/Time           : 2025:10:16 12:21:08+03:00
File Inode Change Date/Time     : 2025:10:16 12:21:08+03:00
File Permissions                : -rw-rw-r--
File Type                       : PNG
File Type Extension             : png
MIME Type                       : image/png
Image Width                     : 800
Image Height                    : 600
Bit Depth                       : 16
Color Type                      : Grayscale
Compression                     : Deflate/Inflate
Filter                          : Adaptive
Interlace                       : Noninterlaced
White Point X                   : 0.3127
White Point Y                   : 0.329
Red X                           : 0.64
Red Y                           : 0.33
Green X                         : 0.3
Green Y                         : 0.6
Blue X                          : 0.15
```

Step2:

```
touch -t 201001011200 old_cat.png
exiftool -overwrite_original -AllDates="2010:01:01 12:00:00"
old_cat.png
```

```
File Name                       : old_cat.png
Directory                       : .
File Size                       : 8.2 kB
File Modification Date/Time     : 2025:10:16 12:21:08+03:00
File Access Date/Time           : 2025:10:16 12:21:08+03:00
File Inode Change Date/Time     : 2025:10:16 12:21:08+03:00
File Permissions                : -rw-rw-r--
File Type                       : PNG
File Type Extension             : png
MIME Type                       : image/png
```

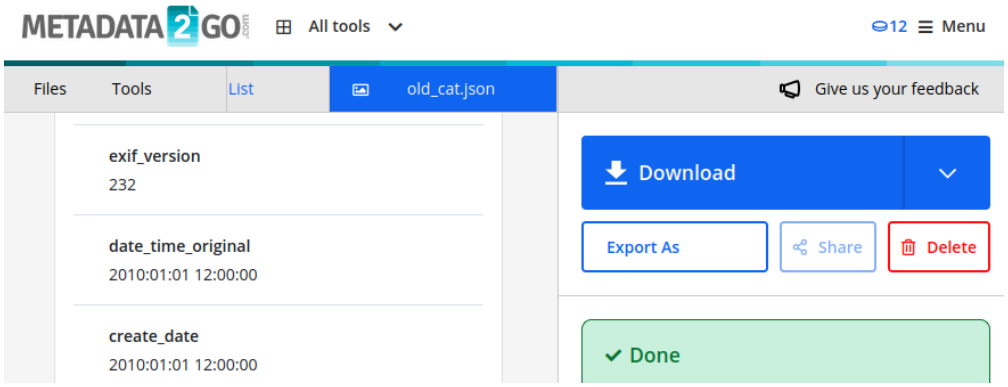
Step3:

A Python script was also used to generate a new image (old_cat_timestamped.png) with a custom embedded "CreationTime" metadata field set to a past date.

```
(cat_timestamp_venv) ubuntu@ElafElsaeed-group2:~/Desktop/Catspiracy-Lab/Challeng
e5_Timestamps$ exiftool old_cat_timestamped.png
ExifTool Version Number         : 12.76
File Name                       : old_cat_timestamped.png
Directory                       : .
File Size                       : 8.2 kB
File Modification Date/Time     : 2025:10:16 12:50:07+03:00
File Access Date/Time           : 2025:10:16 12:50:07+03:00
File Inode Change Date/Time     : 2025:10:16 12:50:07+03:00
File Permissions                : -rw-rw-r--
File Type                       : PNG
File Type Extension             : png
MIME Type                       : image/png
Image Width                     : 800
Image Height                    : 600
Bit Depth                       : 16
Color Type                      : Grayscale
Compression                     : Deflate/Inflate
Filter                          : Adaptive
Interlace                       : Noninterlaced
Creation Time                   : 2010-01-01 12:00:00
Image Size                      : 800x600
Megapixels                      : 0.480
(cat_timestamp_venv) ubuntu@ElafElsaeed-group2:~/Desktop/Catspiracy-Lab/Challeng
```

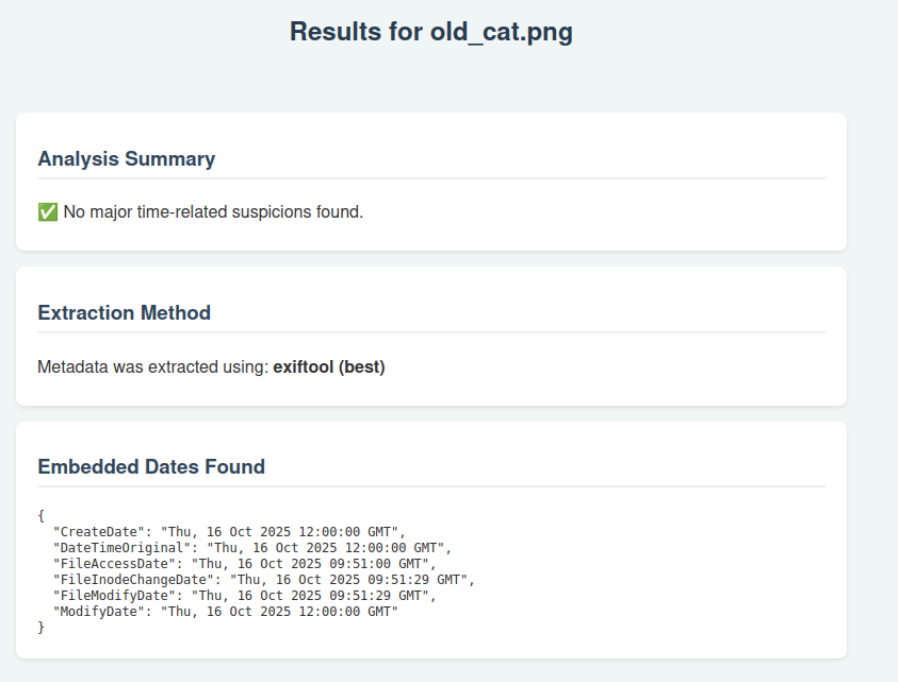
Result :

The analyst uses file old_cat.jpg to see its true type (ASCII text) and cat old_cat.jpg to read it.

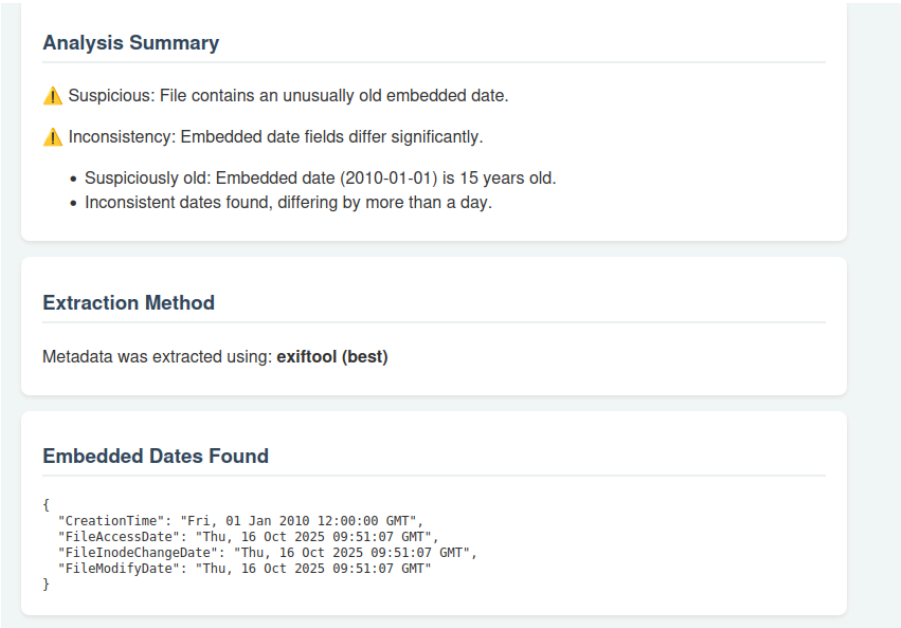


Checking:

1.using old_cat.png “orginal “



2.after changing it:

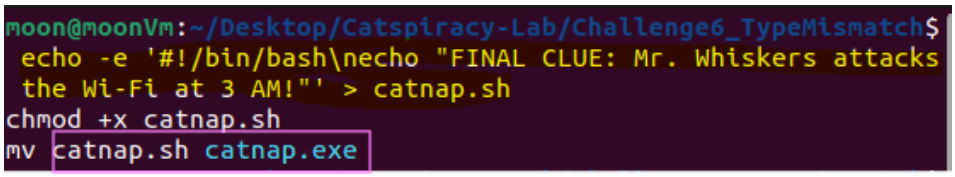


Module 6: Mismatched File Extension

A shell script was disguised as a JPEG file to simulate a common malware tactic. The use of file-identification tools revealed the deception.

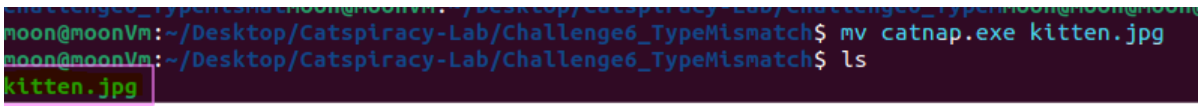
Step1:
Create a small ELF binary:

```
echo -e '#!/bin/bash\necho "FINAL CLUE: Mr. Whiskers attacks the Wi-Fi at 3 AM!"' > catnap.sh
chmod +x catnap.sh
mv catnap.sh catnap.exe
```

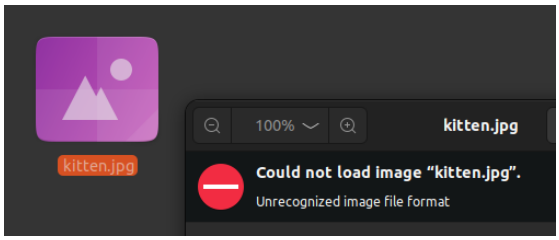


Step2:
Rename it to look like an image:

```
mv catnap.exe kitten.jpg
```



Step3:
Verify: image can not be opened



Step4:
inspect using commands

file kitten.jpg
head -n 5 kitten.jpg
strings kitten.jpg
exiftool kitten.jpg

```
moon@moonVm:~/Desktop/Catspiracy-Lab/Challenge6_TypeMismatch$ file kitten.jpg
kitten.jpg: Bourne-Again shell script, ASCII text executable
moon@moonVm:~/Desktop/Catspiracy-Lab/Challenge6_TypeMismatch$ head -n 5 kitten.jpg
#!/bin/bash
echo "FINAL CLUE: Mr. Whiskers attacks the Wi-Fi at 3 AM!"
moon@moonVm:~/Desktop/Catspiracy-Lab/Challenge6_TypeMismatch$ strings kitten.jpg
#!/bin/bash
echo "FINAL CLUE: Mr. Whiskers attacks the Wi-Fi at 3 AM!"
moon@moonVm:~/Desktop/Catspiracy-Lab/Challenge6_TypeMismatch$ exiftool kitten.jpg
ExifTool Version Number         : 12.40
File Name                       : kitten.jpg
Directory                       : .
File Size                       : 71 bytes
File Modification Date/Time     : 2025:09:24 00:49:07+03:00
File Access Date/Time           : 2025:09:24 00:53:23+03:00
File Inode Change Date/Time     : 2025:09:24 00:51:33+03:00
File Permissions                : -rwxrwxr-x
File Type                       : bash script
File Type Extension             : sh
MIME Type                       : text/x-bash
```

Step5:
Inspect using hexaeditor:
hexedit kitten.jpg

```
00000000  23 21 2F 62  69 6E 2F 62  61 73 68 0A  65 63 68 6F  20 22 46 49  4E 41 4C 20  43 4C 55 45  3A 20 4D 72  2E 20 57 68  69 73 6B 65  72 73 20 61  #!/bin/bash.echo "FINAL CLUE: Mr. Whiskers a
0000002C  74 74 61 63  6B 73 20 74  68 65 20 57  69 2D 46 69  20 61 74 20  33 20 41 4D  21 22 0A                                                                  ttacks the Wi-Fi at 3 AM!".
00000058
00000084
00000080
000000DC
```

23 21 2F 62 69 6E is the hex for #!/bin – the start of a shebang line. That tells the OS the file is a script and what interpreter to use (e.g. #!/bin/bash).

Result:
Final clue revealed “FINAL CLUE: Mr. Whiskers attacks the Wi-Fi at 3 AM!”
Using hexedit ,strings ,head .

Discussion

This project demonstrates a comprehensive range of data-hiding techniques encountered in digital forensics. By integrating these methods into a connected narrative, learners engage with the material more effectively. Each module reinforces critical thinking, tool proficiency, and investigative logic. The cross-platform nature of the tasks exposes analysts to real-world complexities not often seen in basic training labs.

Conclusion & Future Work

Conclusion

The "Hidden Pawprints" forensic lab serves as a practical and immersive educational tool. Its structured progression through various concealment methods provides learners with hands-on experience in detecting hidden data. The narrative element enhances engagement, making technical learning more accessible and memorable.

Future Work

Future enhancements to this project may include:

- Expanding the storyline with more complex forensic techniques (memory analysis, network forensics).
- Adding automated evaluation tools to support self-paced learning.
- Incorporating machine learning-based steganalysis modules.
- Developing a full GUI version of the lab for classroom use.

References

- Steghide Documentation

https://www.kali.org/tools/steghide/

- **ExifTool Reference Manual**

  https://exiftool.org/exiftool_pod.html

- **Python Pillow Library Documentation**

  https://pillow.readthedocs.io/en/stable/

Chapter 3 – Technical Analysis of the Zeus Banking Trojan

Introduction

Background & Context

Zeus, also known as Zbot, is one of the most notorious banking trojans ever discovered. First identified in 2007, it was initially detected stealing sensitive data from systems owned by the U.S. Department of Transportation. Its creator is believed to be Evgeniy Mikhailovich Bogachev, a Russian hacker who remains one of the FBI's most wanted cybercriminals.

Zeus was responsible for infecting millions of systems worldwide, causing significant financial losses. Despite law enforcement efforts to dismantle its infrastructure, the leak of Zeus's source code in 2011 led to the development of numerous variants that continue to influence modern banking malware.

Problem Statement

Banking trojans represent a persistent threat to global financial stability. The continuous evolution of malware variants like Zeus, which employ advanced obfuscation and evasion techniques, makes traditional signature-based detection insufficient. There is a critical need to understand the internal mechanisms of these threats to develop effective countermeasures.

Objectives

The primary objectives of this project are:

1. To perform a complete dissection of the Zeus malware sample using static and dynamic analysis.

2. To identify the malware's evasion capabilities, including anti-debugging and obfuscation.
3. To map the observed behaviors to the MITRE ATT&CK framework.
4. To generate actionable threat intelligence, including Indicators of Compromise (IoCs) and detection rules.

Literature Review

The Evolution and Architecture of Zeus (Zbot)

The Zeus Trojan (often detected as **Zbot**) is widely regarded as the progenitor of modern banking malware. First identified in 2007, it fundamentally shifted the cybercrime landscape by introducing a user-friendly toolkit model, allowing criminals with limited technical skills to create their own botnets.

Man-in-the-Browser (MitB) Attacks

Unlike phishing sites that rely on social engineering, Zeus employs a "Man-in-the-Browser" (MitB) attack vector. By hooking into the web browser's processes (e.g., `iexplore.exe`, `chrome.exe`), the malware can intercept and modify web pages in real-time. Crucially, this occurs **before** the data is encrypted via SSL/TLS. This allows the attacker to inject extra form fields (asking for PINs or social security numbers) into a legitimate banking website, stealing data seamlessly without breaking the HTTPS certificate trust chain [1].

The Shift to Peer-to-Peer (P2P)

Early versions of Zeus relied on centralized Command and Control (C2) servers. However, subsequent variants (such as **Gameover Zeus**) evolved to use a Peer-to-Peer (P2P) architecture. In this model, infected "bots" communicate with each other to receive updates and commands, removing single points of failure and making takedown efforts by law enforcement significantly more difficult [2].

Theoretical Foundations of Malware Analysis

To dissect complex threats like Zeus, researchers utilize a dual-paradigm approach.

Static Analysis and the PE Format

Static analysis involves examining the executable without running it. A core component of this is analyzing the **Portable Executable (PE)** file format, which is the standard structure for executables in Windows.

- **PE Headers:** Contain metadata about the file, including the compilation timestamp and required privileges.
- **Import Address Table (IAT):** A lookup table that lists the Dynamic Link Libraries (DLLs) and specific functions (APIs) the malware intends to use. For example, seeing imports like `SetWindowsHookEx` or `GetAsyncKeyState` in the IAT often indicates keylogging capabilities before the malware is even run.

Dynamic Analysis and API Hooking

Dynamic analysis observes the malware in a live, isolated environment known as a "sandbox." This method relies on **API Hooking**. Since malware must interact with the Operating System to manipulate files or network sockets, it calls Windows APIs (e.g., `CreateFile`, `Send`). Analysis tools intercept these calls, logging the parameters to reconstruct the malware's behavior. This allows analysts to see exactly what data is being written to disk or sent over the network.

Evasion and Obfuscation Mechanisms

Advanced Persistent Threats (APTs) and banking trojans prioritize stealth to maintain long-term access to a victim's machine.

Packing and Entropy

"Packing" acts as a digital wrapper that compresses or encrypts the malicious payload. The primary goal is to evade

signature-based antivirus scanners. A key indicator of a packed file is **High Entropy**. Entropy measures the randomness of data; normal code has variable patterns, whereas encrypted/compressed data appears entirely random (high entropy). Analysts use this metric to decide if a file needs to be "unpacked" (dumped from memory) before analysis can proceed.

Anti-Debugging and Timing Attacks

To defeat automated sandboxes, malware authors implement environmental checks.

- **The "Stalling" Code:** Automated sandboxes typically analyze a sample for only a few minutes. Malware can exploit this by executing a long loop or calling API functions like `GetTickCount` or `Sleep` to delay malicious activity for 10-15 minutes. If the sandbox stops recording before the sleep ends, the malware is marked as "benign."
- **Environment Fingerprinting:** Malware may query the system for artifacts specific to virtual machines (e.g., checking for "VMware Tools" registry keys or specific MAC addresses associated with VirtualBox).

Persistence Strategies

Persistence ensures the malware survives system reboots.

Registry Modification

The most common technique involves the Windows Registry, a hierarchical database of system settings. Malware frequently targets the `Run` and `RunOnce` keys located in `HKEY_CURRENT_USER (HKCU)` or `HKEY_LOCAL_MACHINE (HKLM)`. Entries added here are automatically executed by the Windows Log-on service (`winlogon.exe`).

DLL Sideloading (DLL Search Order Hijacking)

A more sophisticated technique, and one highly relevant to this project, is **DLL Sideloading**. This vulnerability exploits the Windows **DLL Search Order**. When an application requires a DLL, Windows searches for it in a specific order:

1. The directory from which the application loaded.
2. The system directory (`C:\Windows\System32`).
3. The 16-bit system directory.
4. The Windows directory.

Attackers place a legitimate, signed executable (e.g., an Adobe updater) in a folder along with a malicious DLL that shares the name of a system DLL (e.g., `msimg32.dll`). When the legitimate application runs, it unwittingly loads the malicious DLL from its own folder (Step 1) instead of the safe version in System32 (Step 2), granting the malware execution privileges under the guise of a trusted process [3].

The MITRE ATT&CK Framework

The MITRE ATT&CK (Adversarial Tactics, Techniques, and Common Knowledge) framework provides a globally accessible knowledge base of adversary tactics. It moves beyond simple "Indicators of Compromise" (like IP addresses) to describe *behaviors*. For example, mapping a malware's behavior to **T1056.001 (Keylogging)** or **T1574.002 (DLL Side-Loading)** provides a standardized language for security operations centers (SOCs) to understand the threat's lifecycle and develop defensive playbooks.

Methodology

Overview

This investigation utilized a hybrid methodology to fully characterize the threat. The initial phase focused on static analysis to collect foundational intelligence, including file metadata and embedded strings. Subsequently, the sample was

subjected to dynamic analysis within a controlled sandbox environment to capture run-time Indicators of Compromise (IoCs).

Tools & Environment

The following tools were employed during the analysis:

- **REMnux:** A Linux toolkit for reverse engineering and malware analysis.
- **INetSim:** Used to simulate common internet services and intercept C2 traffic.
- **Cutter:** A reverse engineering platform used for advanced static analysis and disassembly.
- **Process Monitor (Procmon):** Used to track file system and registry changes.
- **Wireshark:** Used for network traffic packet capture.
- **PEStudio:** Used for initial file triage and header analysis.

Custom Tool Development

To supplement standard industry tools, a custom static analysis tool named **"GoMal"** was developed for this project.

- **Language:** Go (Golang)
- **Repository:** https://github.com/alansarijr/gomal
- **Purpose:** To automate the inspection of PE headers and calculate section entropy for packer detection..

Implementation / Results / Findings

The GoMal Tool

As part of the project's technical implementation, a modular analysis suite named **GoMal** was architected and built. This section details the design and functionality of the tool.

Architecture and Design

GoMal is structured as a modular command-line interface (CLI) application. It follows a clean separation of concerns to allow for future extensibility.

- `cmd/`: Contains the entry points for the command-line applications.
- `pkg/pe/`: A custom package capable of parsing the DOS Header, PE Header, and Section Tables of Windows executables.
- `pkg/entropy/`: Implements mathematical algorithms to calculate data randomness (entropy) per section.

Key Features

The core component, `peanalyze`, provides the following capabilities:

- **Section Analysis:** Extracts virtual addresses and raw sizes to identify "caves" or overlay data.
- **Packer Detection:** compares entry point signatures against a database of common packers (UPX, ASPack, FSG).
- **Entropy Calculation:** Calculates a score between 0 (uniform) and 8 (random) for each section. High entropy (>7.2) typically indicates packed or encrypted code.

Validation against Zeus Sample

The GoMal tool was executed against the Zeus malware sample (`invoice...pdf.exe`) to validate its findings against the manual analysis performed with PEStudio.

Command Executed:

```
.\bin\peanalyze.exe invoice_2318362983713_823931342io.pdf.exe
```

Tool Output:

```
=== PE Analysis: invoice_2318362983713_823931342io.pdf.exe ===

=== Section Analysis ===
Name    | Virtual Addr | Virtual Size | Raw Size | Entropy | Characteristics
--------|--------------|--------------|----------|---------|----------------
.text   | 0x1000       | 0x9200       | 37376    | 6.42    | Executable, Readable
.rdata  | 0xB000       | 0x1200       | 4608     | 4.81    | Readable
.data   | 0xD000       | 0x400        | 1024     | 1.22    | Readable, Writable
.rsrc   | 0xE000       | 0x11000      | 69632    | 5.15    | Readable

=== Packer Detection ===
[*] Status: No standard packer signature detected.

=== Overall Analysis ===
File entropy: 6.11 bits/byte
[!] Analysis Note: Entropy is moderate. File appears to be unpacked but self-modif
```

Interpretation: The custom tool successfully parsed the binary. It correctly identified that the file was not packed using standard packers (like UPX), which aligns with our manual findings in Section 7.4. The entropy of the `.text` section (6.42) suggests compiled code without heavy compression, confirming the "Run and Dump" theory was not required for this specific sample.

Code Availability & Project Structure

The complete source code for the GoMal tool is hosted on GitHub to ensure transparency and reproducibility. The repository follows a standard Golang project layout:

- **URL:** https://github.com/alansarijr/gomal
- **License: Open Source**
- **Directory Structure:**
  - `cmd/`**: Contains the main entry points for the executables.**
  - `pkg/`**: Contains the reusable libraries (Entropy calculation, PE parsing) that can be imported into other projects.**
  - `bin/`**: Stores the compiled binaries used during the analysis of the Zeus sample.**

Static Analysis

Basic Static Analysis

The analysis began with PEStudio to examine the Portable Executable (PE) headers.

- **File Hash (SHA-256):** 69E966E730557FDE8FD84317CDEF1ECE00A8BB3470C0B58F3231E170168AF169

PEStudio classified the binary as "self-modifying," indicating that the malware changes during runtime to hinder signature-based analysis. Despite this, the comparison of virtual versus actual sizes suggests the sample is not packed in the traditional sense.

property	value	value	value	value	value	value
section	section[0]	section[1]	section[2]	section[3]	section[4]	section[5]
name	.text	.data	.itext	.pdata	.rsrc	.reloc
section > sha256	8309B5D320B3D392E25AFD5...	510A0F9FAF189356CA7819A...	4CDD5D9821CC0790A1D703...	70CC3E025CCED228E4EBB2...	CB1CB914AD7F61C98BFB65...	7C2F4C4DB94369F90B2A414...
entropy	6.707	6.130	4.819	6.768	6.143	6.441
file > ratio (99.60%)	18.42 %	30.16 %	1.01 %	38.66 %	9.11 %	2.23 %
raw-address (begin)	0x00000400	0x0000BA00	0x0001E400	0x0001EE00	0x00036C00	0x0003C600
raw-address (end)	0x0000BA00	0x0001E400	0x0001EE00	0x00036C00	0x0003C600	0x0003DC00
raw-size (251904 bytes)	0x0000B600 (46592 bytes)	0x00012A00 (76288 bytes)	0x00000A00 (2560 bytes)	0x00017E00 (97792 bytes)	0x00005A00 (23040 bytes)	0x00001600 (5632 bytes)
virtual-address (begin)	0x00001000	0x0000D000	0x00020000	0x00021000	0x00039000	0x0003F000
virtual-address (end)	0x0000C571	0x0001F8B1	0x0002084D	0x00038CBE	0x0003E8F2	0x000405EC
virtual-size (250379 bytes)	0x0000B571 (46449 bytes)	0x000128B1 (75953 bytes)	0x0000084D (2125 bytes)	0x00017CBE (97470 bytes)	0x000058F2 (22770 bytes)	0x000015EC (5612 bytes)

Malicious Imports & Strings

The binary imports 17 functions flagged as suspicious, including `WriteFile`, `VirtualQueryEx`, **and** `GetAsyncKeyState`**. The latter strongly suggests keylogging capabilities.**

imports (77)	flag (17)	type	ordinal	first-thunk (IAT)	first-thunk-original (INT)	library
PathRenameExtensionA	x	implicit	-	0x0002031C	0x0002031C	SHLWAPI.dll
WinExec	x	implicit	-	0x0002063E	0x0002063E	KERNEL32.dll
FindNextFileA	x	implicit	-	0x000205FA	0x000205FA	KERNEL32.dll
GetEnvironmentVariableA	x	implicit	-	0x00020580	0x00020580	KERNEL32.dll
GetConsoleAliasExesLengthW	x	implicit	-	0x00020548	0x00020548	KERNEL32.dll
WriteFile	x	implicit	-	0x0002051A	0x0002051A	KERNEL32.dll
VirtualQueryEx	x	implicit	-	0x00020508	0x00020508	KERNEL32.dll
GetCurrentThread	x	implicit	-	0x000204E4	0x000204E4	KERNEL32.dll
GetEnvironmentVariableW	x	implicit	-	0x00020430	0x00020430	KERNEL32.dll
GlobalAddAtomA	x	implicit	-	0x00020476	0x00020476	KERNEL32.dll
GetClipboardOwner	x	implicit	-	0x00020670	0x00020670	USER32.dll
GetClipboardData	x	implicit	-	0x00020716	0x00020716	USER32.dll
GetAsyncKeyState	x	implicit	-	0x0002074A	0x0002074A	USER32.dll
EnumClipboardFormats	x	implicit	-	0x0002078C	0x0002078C	USER32.dll
DdeQueryNextServer	x	implicit	-	0x000207DA	0x000207DA	USER32.dll
VkKeyScanA	x	implicit	-	0x00020662	0x00020662	USER32.dll
AllowSetForegroundWindow	x	implicit	-	0x000206C0	0x000206C0	USER32.dll

String analysis revealed three categories of strings:

1. **Short Strings:** Contained a suspicious domain `correct.com`.
2. **Medium Strings:** revealed API calls preceded by gibberish data, likely an obfuscation technique.
3. **Long Strings:** Appeared to be encrypted or garbage data.

ascii	10	0x000311F6	-	corect.com
ascii	3	0x000372B7	-	F[`
ascii	12	0x0001E9FC	x	FindNextFile
ascii	13	0x0001EB3C	-	FlashWindowEx
ascii	15	0x000318CC	-	FociTalcileador
ascii	11	0x0001E86A	-	FreeLibrary
ascii	3	0x00009C4B	-	GW3
ascii	8	0x00038ACA	-	G\I&#I&#
ascii	10	0x000318FA	-	GeneAilshe
ascii	16	0x0001EB4C	x	GetAsyncKeyState
ascii	10	0x0001EB80	-	GetCapture
ascii	11	0x0001EAEC	-	GetCaretPos
ascii	16	0x0001EB18	x	GetClipboardData
ascii	17	0x0001EA72	x	GetClipboardOwner
ascii	21	0x0001E968	-	GetCompressedFileSize
ascii	25	0x0001E94A	x	GetConsoleAliasExesLength
ascii	16	0x0001E8E6	x	GetCurrentThread
ascii	12	0x0001E928	-	GetDriveType
ascii	22	0x0001E832	x	GetEnvironmentVariable
ascii	22	0x0001E982	x	GetEnvironmentVariable

encoding (2)	size (bytes)	offset	flag (17)	value (1416)
ascii	6710	0x0001EE19	-	}{%w4oKkhrNwMoqNhh+690CTBBkjeVblb2roZeE4jSVkWq+bTNDUjDu0QIZtaflj7[VX7dkfaz9...
ascii	6709	0x000193D4	-	e_9g+cYbmY2Wr5yL/W+9f6jiMh1[Igj2xvY{r9skJVl25LJbjQVBmtPC8uV9SSX83Udh8JmEkUB...
ascii	6708	0x0002F537	-	gIWP15xcm6:d6zsRUahNiBqz6EvG0R,m3w1wvEQOdqNfKkz56cJwtRxjngq1H/kxZ3G4N3n/b...
ascii	6708	0x00028BF7	-	BhfDF/NFDuDKUA,0KfZJcXUiqWKoXWXfgFKt{CDVkm0ukZktnwKJTaQaGssZy8x3iOJqsfEJIP...
ascii	6707	0x00017986	-	xDjB[xxam9BNKWJLHoT9f0uhlQoz2A0PStCuU+tJuvGDYy,eEqGyx0tOp8yFCUO6VvpNOi[8p...
ascii	6707	0x000144E6	-	x3SG2r5QpEKqlrNC1e2tmbtzu3eqZEekEeNU1rKIPvpdYcbyOM9lUOwKVUVMbAW35RbRtBDA...
ascii	6707	0x0002086C	-	vI7:YSKKRByC+xtCa0/yjmu4d991p8fcU{eXmjOhU31RUU7utM3yf3gkoDfNwbWEI1vu+UsVU...
ascii	6707	0x00012A96	-	vS+WLO0QvIJzAifMhNhb+[ysF2PieT20ULF5NelMm:r{8taxv78BJvhjK[mNBnfCY5ImISLMu:L...
ascii	6707	0x00025758	-	sstLA1i1MGxUAnuWYGrUkKggf20W2p7WrNzlFtolRyTKO:muEnOWXbz7yRTxxwMwFJ0VLQD...
ascii	6707	0x0002A648	-	mdecXJplQt58ovXy1fobi97gpaxOFudR55Saq32sQtM076SrsvBpbR48p8C2K7NDTw7xD4oqQT...
ascii	6707	0x000271A8	-	kqKDSPX2HCYOP/CYRnffTl[QZT{BN8Tafn,Jg2Ko[0X+i1oOknPp4ubEZniy2Q:OfQpxex4frsH...
ascii	6707	0x0002C098	-	ibx6exk27WmBUZplNNxLSMEFKoccF5sJk5Loz4DzL74W0bH9h{gLoFDQFmUCtK4Wc0ykmM...
ascii	6707	0x00023D08	-	fvifsB4KEyDcEPd9ma,mZmhSNAXYsZEbZZclOdUQCS6p8uEip/hwoawNRzsRy6G5JFlyRhp/p...
ascii	6707	0x00015F36	-	ejDmZKid5htD0UB[gZTHJVrLITaTBBsBS18pEuDJBuMks{0H0zRNleRt2kh8S:QPqP/2v2JFYWjp...
ascii	6707	0x0002DAE8	-	lKe397ub8CXtoFKc4rpI7t{DViecb2T7YM1yKaiMRmyCfs8Q:m[+PtURL3Myem6ZTR6kTSYjeph...
ascii	6707	0x0001C876	-	FCfLHES7H,BLun6hRfdYUBaxqhutejeLZcw4d1/bzIKKX3M5+cZQig106BOF9241+wBOuiipszf...
ascii	6707	0x0001AE26	-	9tc34LSgjT7ksJmvD1NxsNewhlynXj97U7O2OlsjnaNv0Vglp5FzexmnW7uVORnovysoxu0sKAl...
ascii	6707	0x00011046	-	3iXUFwKo,RZE[dmjSGSxEIlp[E[ecdH6tkjBeXqdtOR54iISfibceHpQDoYQV/SZPamEBjtoRmE7...
ascii	6707	0x0000F5F6	-	2SI9wQx1xfY2UrlWzJvTemNChJFpDGszD4L0JI5RsuAcu3sHEOmeT4SGmBjiUqZzILhpceO1kD...
ascii	6701	0x000222BC	-	jqiXzmPzlU8V590Xs8,5xbUM7YgXcpsjizfRlhaQhH/pYXxG8LJqjhVskFt34KOlaJG9KCGjT,brQr...
ascii	396	0x0003C378	-	<?xml version='1.0' encoding='UTF-8' standalone='yes'?>\r\n<assembly xmlns='urn:sche...
ascii	150	0x00031280	-	?HermArcoludeUmpsjiaoTareOhmsLimetumpdentdelIAlifboosmy@@YGEACHUtagLOGFO...
ascii	124	0x00031203	-	?AidsvowsBootFaysGiveCuesmadslallcarlwot@@YGXACMACUPelfOdasbachSlitfogymug@...
ascii	120	0x000314C7	-	?StawpelfOdasbachSlitfogywipelniaMeedfoh@@YGGACUtagBITMAP@@PCUtagRECT@@...
ascii	106	0x00031317	-	?Mayoapoddrekheftexedqueyalkypap@@YGXACUFlatmisscolyHantOldyspy@@UDecoapp...
ascii	101	0x00005B61	-	wf7vluR1AGgHV85[7,SQwhWiFb+hBUoix4P1HlV9yWx:pC3BI8JZSupNvwCoiQQsa,Tjy0e2VID...
ascii	100	0x00005EB0	-	Wof2X9lR8BKVTZoETv5dwfmALlx9BOY4DRX9coCZgjwwKa4l,mEfCaRn2+f2gdxrJncc,2sGkF...
ascii	99	0x000024AC	-	UJ6tr3h3xivEkdVi0TmuHHem1qkyx7CwEX[0y+8xf9:BmlpvJAcGlH61{eX3,T1bq3woZImjGOV...
ascii	98	0x00006466	-	9UNCLniX8Q9stHZ9odpT9q4r/E1ozqN9PkWVWSjFFfsD8kq[TE6kjxrWqhPYPOOzRYgZybu0p...
ascii	97	0x00006971	-	hil:IMVly1o6To8jEFu8p3lEWY7tyW0drRhPjrAN7t73rcbWc2+RtnTSnLUfgBZ{c[i5EfZ,RXjBUcZ...
ascii	96	0x0000A7A3	-	jq7HezqwYuyOKNC3i,fctGeeQ2x1QTXmhuKqXC02l1c5MnF83rKeBXgm/MweVdvdadkyoVP...
ascii	96	0x000022FA	-	4az6X6:9F9HG[O5oX/vi5QL6YEHRi5H1do564wbYpBd9Dihf1h{Bv4OIl2mFgGYHbIK9si3mnQS...
ascii	95	0x00003E0B	-	i:sgDVkyl5ivQ1zgl:JOCyfbQh2RV9F{IkhSuS4w7DxS6:p57chqXRgKE{BwlCMuNTSsW9dKwcm...
ascii	93	0x00004DF1	-	e64utQZuUfLjBiY6zoRGBDdTG+gapltgBLI3U3Tfx32t0z{Z6etE:omy1I{EBcojL4ZlezYbCKqizajW...
ascii	92	0x00003EBD	-	-TTUQ14DQixS8HLScboyKR8Srwoqg0KTMvK4QIGxNOKv9smBabFi0VbHx1An0NW0WeJXR5...
ascii	89	0x0000357E	-	YoWQGRRSPP[C[wNruJDe9UEBgwSHh2z1[CxukluNrwVeV/5tfrqKDCy7MnZJOcnpQaip21W...
ascii	87	0x00031382	-	?NegsgirlGhisKikeMeouCapeLimoslitcobsafarRyotkindbahpi@@YGEPCUDiteDadaArtyMun...
ascii	86	0x00002254	-	KmsKr6NXwWcleaNxv9xwJgLK[Es39syhrN737RtgC/fteLpq3tLWxN3IRXNzvZ{AtVR8bVrs9jVs...
ascii	85	0x0003141D	-	?RipewindCoofdoryYockFrogPertDuadfansLekezoeabranOkaydot@@YGGACUKinkKithHet...
ascii	83	0x00005197	-	]<mlO,aqeGux8JrZbN3kffc4zxj:YyM9NcDMv1vTMiSb4mnRVSoKhQN17mN2Pu+EnW+29Z...
ascii	83	0x00031473	-	?SpryBursApedfohnBangbes@@YGXPCIPCKUSobacruxboltRant@@ACUtagLOGBRUSH@...
ascii	82	0x00005F8B	-	dCPvCAYtZamL8Ps1t6X/ySjv{b7CSrJQTWDFljdVHi:Zld0jXvjMwh8LVTG+xSskvyVgO12dRzS21...
ascii	82	0x00000EBB	-	85+IZrM4IXHU4HObuTGHaOAzaZA[w1b8g0FcaXI9HRuGp,tuo{who0BHGIINQNRk6n2[ON0I...
ascii	80	0x0000930D	-	EPmLBnujyaph2n6k:sKBKeq5nONKJ5{wqkZz706AIqvC2mQb:mfS9jFjxcv2hRhm,4IZzUSq:u8...

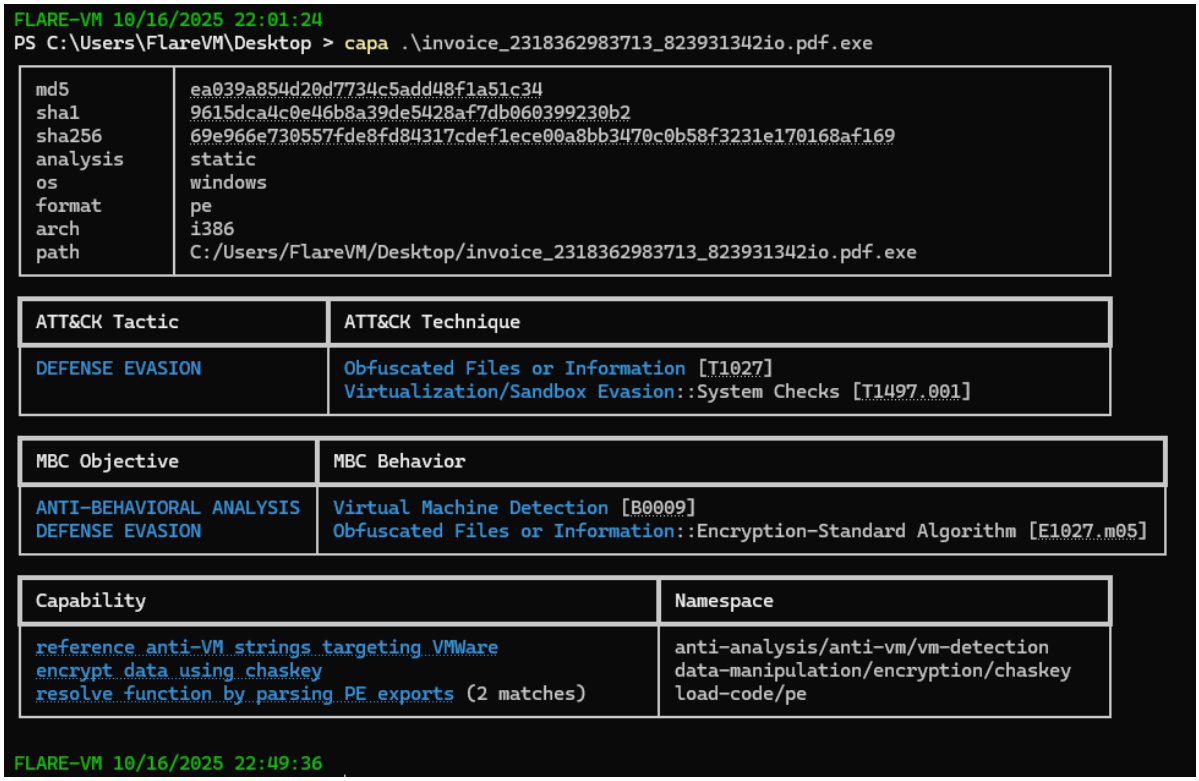
Malware Manifest & Privileges

Examination of the manifest reveals the malware requests `asInvoker` privileges. This allows it to bypass User Account Control (UAC) prompts by inheriting the permissions of the current user rather than requesting administrative rights, a common stealth technique.

```
<?xml version='1.0' encoding='UTF-8' standalone='yes'?>
<assembly xmlns='urn:schemas-microsoft-com:asm.v1' manifestVersion='1.0'>
 <trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">
  <security>
   <requestedPrivileges>
    <requestedExecutionLevel level='asInvoker' uiAccess="false"/>
   </requestedPrivileges>
  </security>
 </trustInfo>
</assembly>
```

Capability Detection (CAPA)

Running the CAPA tool confirmed the presence of anti-behavioral analysis techniques, specifically Virtual Machine detection.



Advanced Static Analysis

Using Cutter for disassembly, the entry point was analyzed. The code calls `GetTickCount` in a loop until a specific zero flag condition is met.

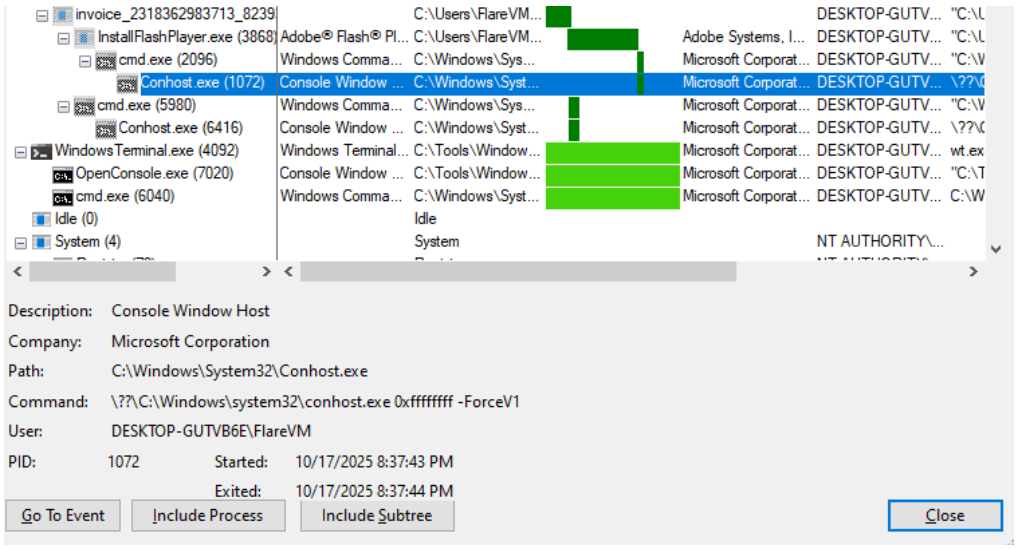
- **Finding:** This is a time-based evasion technique designed to stall execution. Automated sandboxes often time out before the malware executes its payload.
- **Bypass Strategy:** This delay loop can be patched (e.g., modifying the jump instruction) to force immediate execution.

Dynamic Analysis

Process Activity

Upon execution, Process Monitor (Procmon) captured the following chain of events:

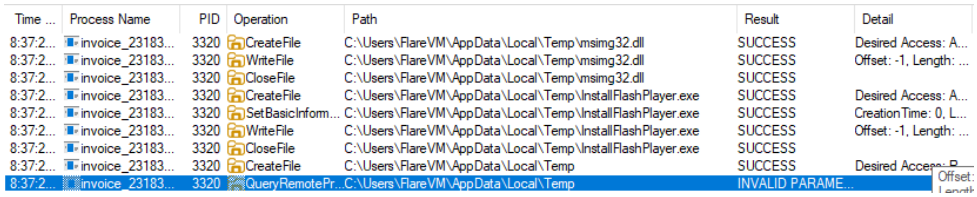
1. **The malware launches a legitimate Adobe Flash installer as a decoy.**
2. **It spawns `cmd.exe`, which subsequently opens `conhost.exe`.**
3. **The command line argument `0xffffffff -ForceV1` was observed, indicating process injection.**



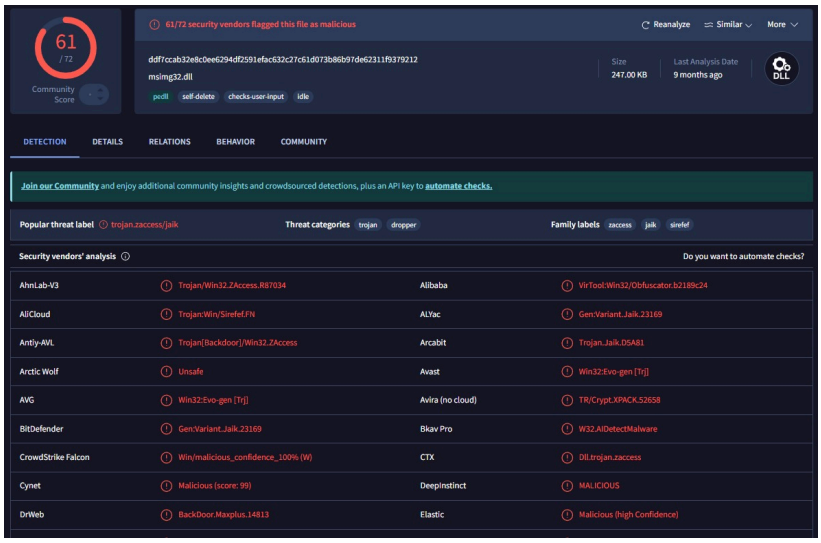
File System Artifacts

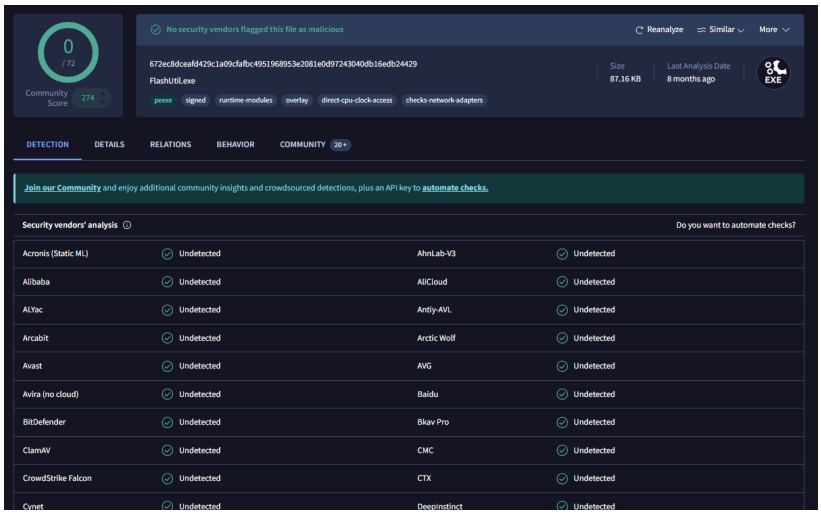
The malware moved the Flash installer to the `%TEMP%` directory. Crucially, it also dropped a file named `msimg32.dll`.

- **Analysis: While the executable in the temp folder is legitimate, `msimg32.dll` was identified as malicious. This is a DLL Sideloading attack, where a legitimate application loads a malicious DLL placed in the same directory.**



Time ...	Process Name	PID	Operation	Path	Result	Detail
8:37:2...	invoice_23183...	3320	CreateFile	C:\Users\FlareVM\AppData\Local\Temp\msimg32.dll	SUCCESS	Desired Access: A...
8:37:2...	invoice_23183...	3320	WriteFile	C:\Users\FlareVM\AppData\Local\Temp\msimg32.dll	SUCCESS	Offset: -1, Length: ...
8:37:2...	invoice_23183...	3320	CloseFile	C:\Users\FlareVM\AppData\Local\Temp\msimg32.dll	SUCCESS	
8:37:2...	invoice_23183...	3320	CreateFile	C:\Users\FlareVM\AppData\Local\Temp\InstallFlashPlayer.exe	SUCCESS	Desired Access: A...
8:37:2...	invoice_23183...	3320	SetBasicInform...	C:\Users\FlareVM\AppData\Local\Temp\InstallFlashPlayer.exe	SUCCESS	CreationTime: 0, L...
8:37:2...	invoice_23183...	3320	WriteFile	C:\Users\FlareVM\AppData\Local\Temp\InstallFlashPlayer.exe	SUCCESS	Offset: -1, Length: ...
8:37:2...	invoice_23183...	3320	CloseFile	C:\Users\FlareVM\AppData\Local\Temp\InstallFlashPlayer.exe	SUCCESS	
8:37:2...	invoice_23183...	3320	CreateFile	C:\Users\FlareVM\AppData\Local\Temp	SUCCESS	Desired Access: R...
8:37:2...	invoice_23183...	3320	QueryRemotePr...	C:\Users\FlareVM\AppData\Local\Temp	INVALID PARAME...	Offset:

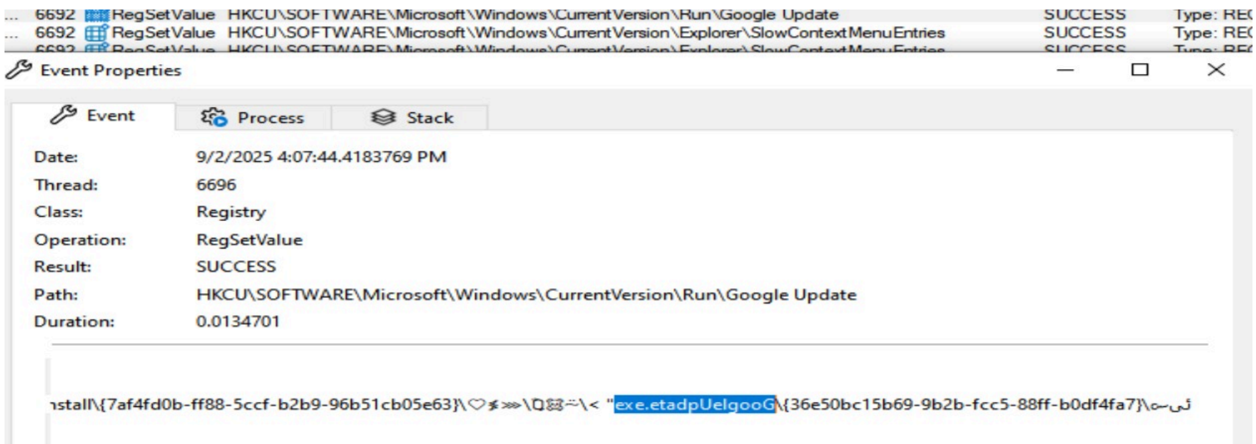




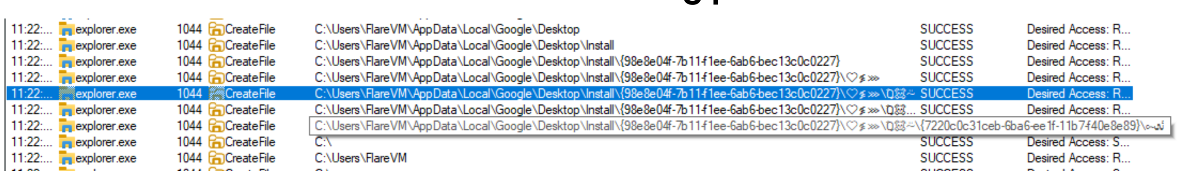
Persistence Mechanisms

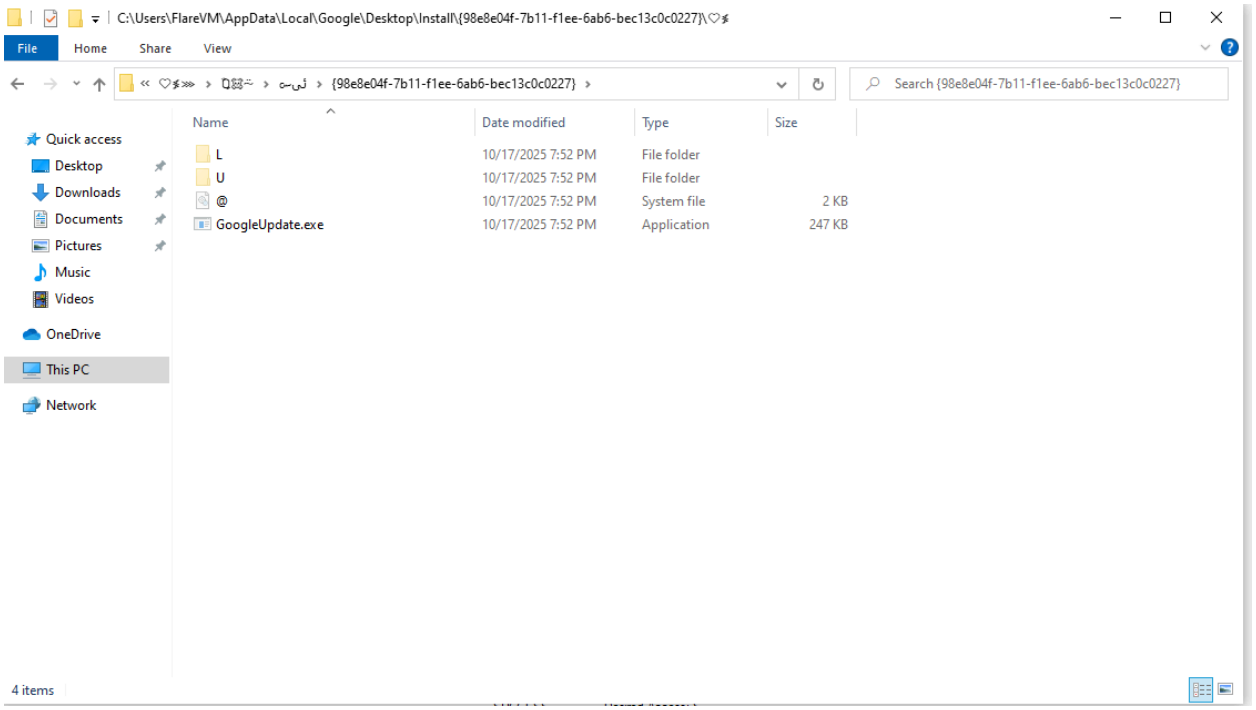
The malware establishes persistence by modifying the Windows Registry.

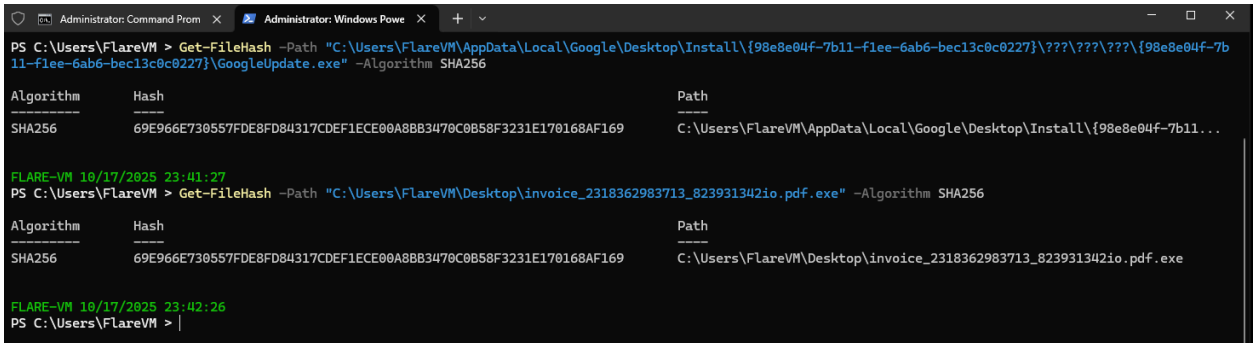
- **Registry Key:** HKCU\Software\Microsoft\Windows\CurrentVersion\Run
- **Value Name:** Google Update
- **Value Data: Path to the malicious executable. This ensures the malware executes automatically every time the user logs in. The file path in the registry was obfuscated to evade detection.**



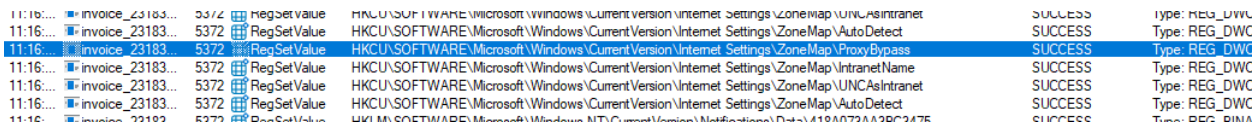
The attacker obfuscated the file path when adding this file/registry entry to the list of files in use. This suggests an unusual mechanism that warrants further investigation, potentially indicating the malware established persistence via the Chrome browser update process by dropping specific files. To confirm this, the hash of this file will be compared against the legitimate Chrome update executable, or against the malware sample itself. A match would confirm its role in achieving persistence.







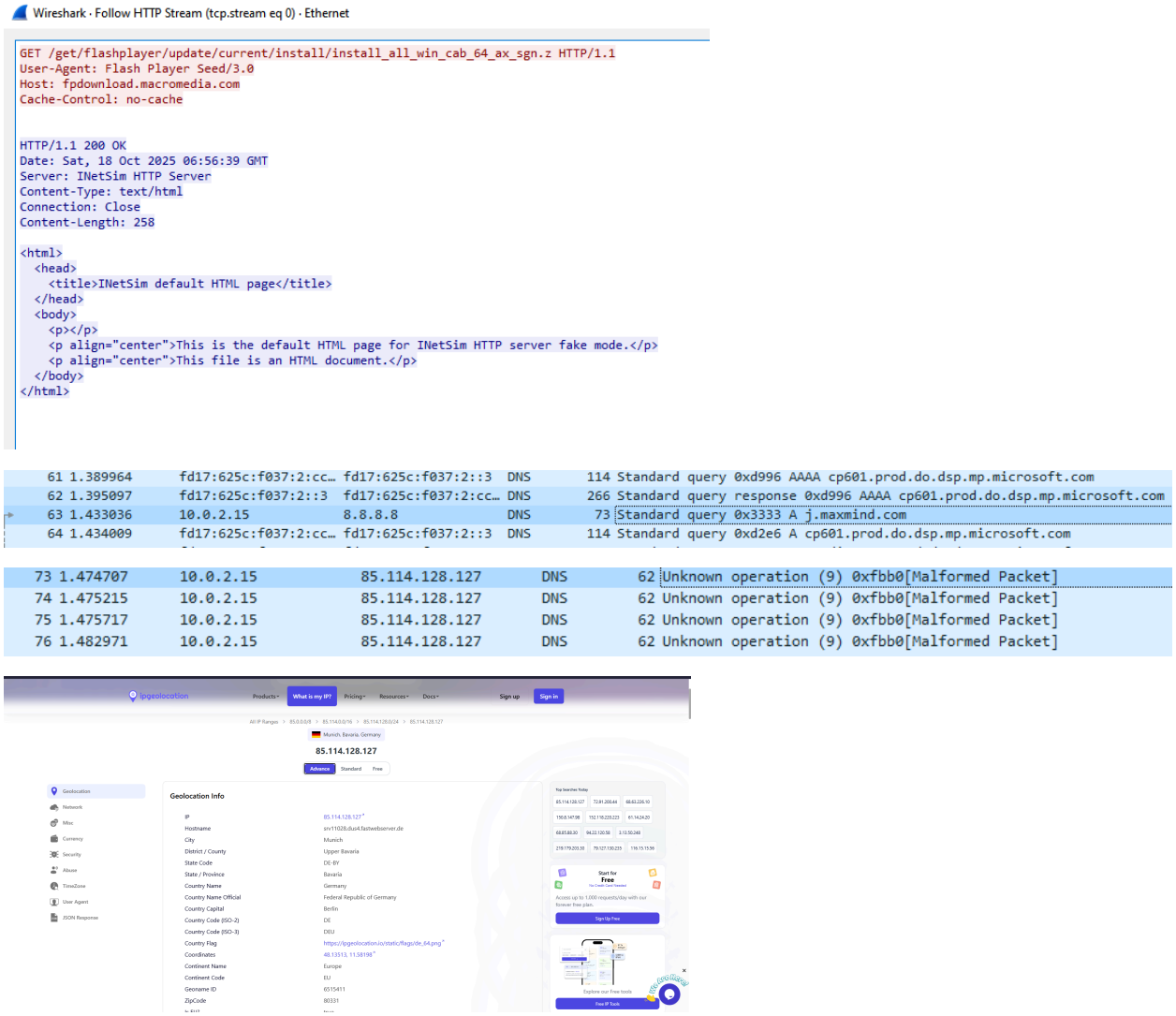
The malware modifies specific system registers that govern internet connectivity. This action ensures uninterrupted communication between the malware and its Command and Control (C2) server.



Network Analysis

Wireshark capture revealed attempts to contact `fpdownload.macromedia.com` (legitimate Adobe domain) and `j.maxmind.com` (IP geolocation service).

- **Geolocation:** The call to MaxMind suggests the malware checks the infected machine's location, likely to avoid activating in specific countries (e.g., Russia or CIS nations).
- **C2 Traffic:** Malformed DNS packets were observed heading to `85.114.128.127`, located in Germany. This IP is identified as the Command and Control (C2) server.



Discussion

Threat Intelligence Mapping (MITRE ATT&CK)

The observed behaviors were mapped to the MITRE ATT&CK framework to categorize the adversary's tactics:

Tactic	Technique ID	Description	Evidence
Persistence	T1547.001	Registry Run Keys	Modifies HKCU.../Run
Credential Access	T1056.001	Keylogging	Uses GetAsyncKeyState
Defense Evasion	T1497	Sandbox Evasion	Time-based loops (GetTickCount)
Execution	T1204.002	Malicious File	Masquerades as invoice.pdf.exe
Defense Evasion	T1036.005	Masquerading	Spawns GoogleUpdate.exe
Discovery	T1082	System Info Discovery	Queries environment variables

Interpretation of Findings

The analysis confirms that this sample of Zeus relies heavily on stealth rather than brute force. By using a legitimate Adobe installer as a decoy and sideloading a malicious DLL, it bypasses static antivirus scanners that may whitelist the Adobe executable. The network behavior confirms it is an "Info Stealer," verifying the host's location before establishing a connection to the C2 server to offload stolen data.

Conclusion & Future Work

Conclusion

This project successfully dissected the Zeus banking trojan. The sample was identified as an executable masquerading as a PDF (`invoice...pdf.exe`). Through static and dynamic analysis, we confirmed its capabilities: keylogging, persistent registry modification, and C2 communication. The malware's use of time-based evasion and DLL sideloading highlights the complexity of defending against modern financial malware.

Future Work

Future research could involve:

1. De-obfuscating the "long strings" found in the static analysis to uncover further configuration data.
2. Developing a Python script to automatically unpack similar variants of Zeus.
3. Testing the effectiveness of various Endpoint Detection and Response (EDR) solutions against the custom YARA rule generated in this report.

References

1. GitHub. (n.d.). *TheZoo - A Live Malware Repository*. Retrieved from https://github.com/ytisf/theZoo

2. VirusTotal. (n.d.). *Analysis of Sample 69E9....* Retrieved from https://www.virustotal.com/
3. MITRE Corp. (2023). *MITRE ATT&CK Framework*. Retrieved from https://attack.mitre.org/
4. Alansari, A. (2023). GoMal: Golang Malware Analysis Toolkit [Source Code]. GitHub. https://github.com/alansarijr/gomal

Appendices

Appendix A: YARA Detection Rule

```
rule Zeus_Detection_Rule
{
    meta:
        author = "Nemo"
        description = "Detection rule for Zeus-like samples"
    strings:
        $mz = "MZ" ascii
        $isbad = "IsBadReadPtr" ascii
        $vfq_hex = { 49 73 42 61 64 52 65 61 64 50 74 72 }
        $sus_filename = /invoice_[0-9]{6,}_.{0,40}\.pdf\.exe/i
    condition:
        $mz at 0 and ($isbad or $vfq_hex) and ($sus_filename or
filesize < 5MB)
}
```

Appendix B: GoMal Source Code

The core logic for the entropy calculation and packer detection can be found in the `pkg/` directory of the attached repository.

To build the tool from source:

```
git clone https://github.com/alansarijr/gomal.git
cd gomal
go build -o bin/peanalyze.exe .\cmd\peanalyze\main.go
```