Operation Cybershadow

David Dudas

Faculty of Mathematics and Computer Science, West University of Timisoara, Timisoara, Romania.

Corresponding author(s). E-mail(s): david.dudas03@e-uvt.ro;

Abstract

This paper present the operation CYBERSHADOW: A Digital Whodunit. This operation is about a digital forensics investigation triggered by an infiltration of the Ministry of Strategic Technologies. This paper presents four chapters that describe the investigation, the tools used, and the findings.

Keywords: Cybersecurity, Digital Forensics, Investigation, C++

1 Introduction

This paper tends to present operation CYBERSHADOW: A Digital Whoduint that started at InterSec Division HQ.

The Ministry of Strategic Technologies was inflitrated by an unknown actor and the only clues left are a compromised machine, garbled traffic logs, a suspicious USB image, and a stash of potentially synthetic media.

In the following sections I will present how I approached the problem, what tools I used, and what I found.

1.1 Chapter 1: Shadows in the File System

The investigation begins with a laptop with a compressed archive on its drive. This archive contains a web of directories and files that may contain clues. The filenames and types are misleading, so we will need a script that can recursively scan a folder and identify the actual file types.

The script should be designed for the Windows operating system and I will write it in C++.

The script will offer a command line interface with the following options:

- `-i` or `--input` followed by a path: Specify the input file path.
- `-s` or `--sigs` followed by a path: Specify the file type map path.
- `-d` or `--depth` followed by a number: Specify the search depth (when input path is a directory).
- `-h` or `--help`: Display the help message.

We can use this script to scan the archive and identify the actual file types.

1.2 Chapter 2: Listening to Ghosts

A machine started acting strange after the user installed something searching for Google Authenticator. There are two separate LAN segments, each tainted by infection. We will analyze the traffic for each.

1.2.1 Operation MOONFALL: The False Authenticator

The infection took root after accessing a fake Google Authenticator page. In order to find out **infected Windows client's IP address, MAC address, host name** and also the **likely domain name used by the fake Authenticator site and the Command and Control (C2) server IP address**, we will analyze the traffic logs.

1.2.2 Operation GREENWIRE: The Domain Breach

There are some outbound connections to obscure hosts, odd DNS behavior, and encrypted chatter in a different AD environment. Seems to be a different malware strain, but it might be linked to the previous analysis.

Here we are interested in finding out the **IP adddress and host name** of the infected Windows machine, the associated **user account names**, the responsible **malware family**, the **exact UTC timestamp** when the infection began, and the **domain name** used within this AD environment.

1.3 Chapter 3: Echoes from the Drive

An USB device is next clue. I wat profesionally wiped, but the imaging team was able to extract a complete forensic dump: IMAGE.ISO.

We also know that there is a file inside the image that was encrypted using a 2-byte XOR cipher, the archives may be nested within the image, and one file is believed to be an image (BMP, PNG, or JPG), but it also be more than just pixels.

I will try to **recover** all the extractable files from the ISO image, **classify** them by type with as much accuracy as possible, decrypt the encrypted file, and examine image files.

1.4 Chapter 4: Faces Behind the Curtain

There is also a folder that has surfaced during the investigation. It contains dozens of images and videos depicting high-ranking officials in scenarios that could ignite international crises. However, there is something suspicious about them.

I will analyze the media files and will try to **determine which images and videos are authentic and which have been manipulated**.

2 Methodology

2.1 File Type Detection Script

In this section, I will present the C++ program used to detect the file types from the achive from chapter 1.1.

I will use the magic number of the file's headers to determine the file type. The magic number is a unique sequence of bytes at the beginning of a file that identifies its type. I will use Gary Kessler's magic number list[1] for this.

The program reads the json file using the nlohmann::json library[2] and populates a map where the magic number is the key and the file type details are the value. The program then recursively scans the input directory and reads the first eight bytes of each file. It checks if the magic number is present in the map and prints the file type details if it is found.

The program won't work if the input path and the signatures path are not provided. The depth is optional - if not provided, the program will scan the entire directory tree.

To compile the program, you need to have a C++ compiler, cmake, and optionally ninja installed. You can use the following commands to compile the program:

```
mkdir build
cd build
cmake -G Ninja ..
ninja
```

This will produce and executable file named `stdf.exe` and a test file named `stdf_tests.exe` in the `build` directory. However, if you do not want to compile the program, the executable files are also available in the `bin` directory of the repository.

2.2 Network Traffic Analysis

2.2.1 The False Authenticator

This investigation will focus on a pcap file that contains the traffic logs from the infected machine. The goal is to find out the likely domain name used by the fake Authenticator site, the infected Windows client's IP address,host name, and MAC address, and the C2 IP address.

To find out the **domain name** of the fake Authenticator site, I will look for DNS queries. Since I know that the domain name contains the word 'moon', I will filter the DNS queries for that keyword.

The **infected Windows client's IP address** can be found by looking at the source IP address of the DNS event.

The pcap file also contains some **NBNS** traffic, which can be used to find the **host name** of the infected machine. I will need to look for NBNS queries and responses that contain the infected machine's IP address. Also, these NBNS queries will contain the **MAC address** in the **Etherned II** header.

In order to find the **Command and Control (C2) IP address**, I will look for http, udp, and tcp traffic, only in external communication for the infected machine's IP address.

2.2.2 The Domanin Breach

During the investigation of this pcap file, I will focus on finding the AD enviroment's domain name, the malware family, the infected Windows machine's IP address and host name, the exact UTC timestamp when the infection began and the user account names associated with this incident.

I know that the AD environment's domain name contains the word 'green', so I will filter the DNS queries for that keyword.

To identify the **malware family**, I will look for suspicious traffic patterns and analyze the payloads of the packets.

The infected Windows machine's IP address and host name can be found after identifying the malware family. When I will find the malware, I will be able to identify the infected machine's IP address. The host name can be found in the NBNS queries and responses, similar to the previous investigation.

In order to find the **exact UTC timestamp** when the infection began, I will look for the trigger of the suspicious traffic.

For the **user account names**, I also need to know the malware family. Once I identify it, I expect to find the user account names implicated in the infection in the traffic logs.

2.3 Digital Foresics & Hidden Data Extraction

In this section, I will present how I will try to extract the files from the USB image.

The first step is to simply mount the image and copy-paste it's contents. But as we might expect, that's not all. I will also do some further analysis, deep scan on the extracted files.

For the encrypted file, I will simply brute-force the 2-byte XOR key. I will use a python script for that together with the 'file' linux command to identify the decrypted file type. There will be multiple possibilities, however I excpect that 90% of the results will be just 'data' and I will have to manually check the rest.

I will also use the 'binwalk' linux command to deep scan every file from the image. For image files (PNG, JPG, etc.), I will use both 'exiftool'[3] and 'steghide' to extract the metadata and hidden data.

2.4 Deepfake Detection

In this section, I need to analyze the media files and determine which images and videos are authentic and which have been manipulated. For this, I will do manual analysis, but I will also use some tools to help me, such as 'exiftool'.

In case of images, I will look for pixel structure anomalies, metadata and lighting inconsistencies, thin details such as hair strands, and other signs of manipulation. I will also use 'exiftool' to extract the metadata and check if the image was edited with a known software.

In case of videos, I will look for frame inconsistencies, audio-visual mismatches, irregularities in motion and lighting, and other signs of manipulation.

3 Results

3.1 File Type Detection Script Results

We used the File Type Detection Script 2.1 to scan the archive from chapter 1.1.

```
.\bin\stfd.exe -s .\chapter1\resources\file_sigs.json -i <
    path_to_chapter1_archive>
```

The output of the following is as follows:

```
'chapter1\archives\arch.7z': 7Z (7-Zip compressed file)
'chapter1\archives\ARCH2.bz2': DMG (Mac Disk image (BZ2 compressed))
'chapter1\archives\arch3.zip': XPT (eXact Packager Models)
'chapter1\archives\notes.jar': AVI|CDA|QCP|RMI|WAV|WEBP (Resource Interchange
    File Format)
'chapter1\docs\Chairing_Guidelines.doc': WPS (MSWorks text document)
'chapter1\docs\Meeting-agenda-template.docx': XPT (eXact Packager Models)
'chapter1\docs\meeting-checklist-template.docx': XPT (eXact Packager Models)
'chapter1\docs\meeting_staff.doc': AVI|CDA|QCP|RMI|WAV|WEBP (Resource
    Interchange File Format)
'chapter1\docs\more\docs\in\here\How to Survive a meeting.doc': WPS (MSWorks
    text document)
'chapter1\docs\more\docs\in\here\how-to-conduct-a-meeting-2.docx': XPT (eXact
    Packager Models)
'chapter1\fun\1711922908449.jpg': SYS (Windows executable)
'chapter1\fun\62fbe39b1c30c95b2cc78f2b_LB0250..png': PNG (PNG image)
'chapter1\fun\AdobeStock_407607989-scaled.jpeg': SYS (Windows executable)
'chapter1\fun\business-meeting.jpg': SYS (Windows executable)
'chapter1\fun\images (1).jpg': SYS (Windows executable)
'chapter1\fun\images.jpg': SYS (Windows executable)
'chapter1\fun\leading-a-meeting-1024x683.jpg.optimal.jpg': SYS (Windows
    executable)
'chapter1\fun\stock_photo.png': XPT (eXact Packager Models)
'chapter1\fun\types-of-meetings.webp': AVI|CDA|QCP|RMI|WAV|WEBP (Resource
    Interchange File Format)
'chapter1\music\emotional_music_561.mp3': VXD|386 (Windows virtual device
    drivers)
'chapter1\music\eona-emotional-ambient-pop-351436.mp3': SYS (Windows
    executable)
'chapter1\music\future-design-344320.mp3': SYS (Windows executable)
'chapter1\music\jungle-waves-drumampbass-electronic-inspiring-promo-345013.mp3
    ': SYS (Windows executable)
'chapter1\presentations\1 Setup.pptx': XPT (eXact Packager Models)
'chapter1\presentations\AWKUM-offical-PowerPoint-Template.pptx': XPT (eXact
    Packager Models)
'chapter1\presentations\CCB_MeetingMgmt.ppt': WPS (MSWorks text document)
'chapter1\presentations\Firefighter-Silhouette-PowerPoint-Diagram.pptx': XPT (
    eXact Packager Models)
'chapter1\presentations\JS2-Mod1.pptx': XPT (eXact Packager Models)
```

```
'chapter1\presentations\planning_call_meeting.ppt': VXD|386 (Windows virtual
    device drivers)
'chapter1\presentations\samplepptx.pptx': XPT (eXact Packager Models)
'chapter1\presentations\slide-zoom-powerpoint-template.pptx': XPT (eXact
    Packager Models)
'chapter1\unknown\file1': XPT (eXact Packager Models)
'chapter1\unknown\file2': SYS (Windows executable)
'chapter1\unknown\file3': VXD|386 (Windows virtual device drivers)
'chapter1\unknown\file4': VXD|386 (Windows virtual device drivers)
'chapter1\unknown\file5': Unknown
'chapter1\unknown\file6': Unknown
'chapter1\unknown\file7': Unknown
'chapter1\unknown\file8': Unknown
'chapter1\unknown\file9': Unknown
'chapter1\videos\bear_fun.mp4': VXD|386 (Windows virtual device drivers)
'chapter1\videos\COWS_AT_THE_GRASS.mp4': AVIF (High Efficiency Image Container
     (HEIC)_1)
'chapter1\videos\Good_dog.mp4': AVIF (High Efficiency Image Container (HEIC)_1
    )
'chapter1\videos\MVI_2350.mov': AVIF (High Efficiency Image Container (HEIC)_1
    )
'chapter1\videos\waterfall2.avi': AVI|CDA|QCP|RMI|WAV|WEBP (Resource
    Interchange File Format)
```

3.1.1 The False Authenticator Results

We used the network traffic analysis tools to analyze the pcap file from opeartion MOONFALL 2.2.1.

The first step was to filter the DNS queries for the keyword 'moon' in order to find the **likely domain** name used by the fake Authenticator site. The result was: `bluemoontuesday.com`.

The **infected Windows client's IP address** was found in the DNS event: `10.1.17.215`.

The **host name** of the infected machine was found in the NBNS queries' and responses' NetBIOS Name Service: `DESKTOP-L8C6GSJ`.

The **MAC address** of the infected machine was found in the NBNS queries' and responses' Ethernet II header: `00:d0:b7:26:4a:74`.

In order to find the **Command and Control (C2) IP address**, I looked for http, udp, and tcp traffic, only in external communication for the infected machine's IP address. There were multiple HTTP GET requests at /1517096937 to an IP address that seems to be the C2 server: `5.252.153.241`.

3.1.2 The Domain Breach Results

In the second part of the network traffic analysis, I focused on the pcap file from operation GREENWIRE 2.2.2.

The first step was to filter the DNS queries for the keyword 'green' in order to find the **AD environment's domain name**. The result was: `fargreentech.com`.

While looking for the **malware family**, I noticed some SMTP traffic with suspicious patterns. Fromt the STMP traffic, I was able to extract some base64 encoded strings that were usernames and passwords. There were multiple authentication attempts, some of them successful. Because of the suspicious patterns, I belive that the malware family is a variant of the **Emotet malware**[4].

All this SMTP traffic was generated by the **infected Windows machine with IP address** `10.12.3.66`. Given the IP address, I was able to find the **host name** of the infected machine: `DESKTOP-LUOABV1`.

Regarding the **exact UTC timestamp** when the infection began, I found a suspicious DNS query at 18.742200 (relative to the pcap file's start time) that was made by the infected machine. The query was for the domain name `fargreentech.com`.

The **user account names** associated with this incident were found in the SMTP traffic. The usernames were:

- darin.figueroa
- marlon.gamba
- minami.hinaka
- tenpo.kaihatsu
- noboyasu.takahashi
- elena.chavez

3.1.3 Digital Foresics & Hidden Data Extraction Results

After mounting the USB image, and copying its contents 2.3, I got 5 files:

- `data1.bin`
- `data2.bin`
- `data3.bin`
- `data4.bin`
- `data5.bin`

The first step was to see the realy type of each file. I used the 'file' linux command to do that. The results were:

```
data1.bin: data
data2.bin: 7-zip archive data, version 0.4
data3.bin: PE32+ executable for MS Windows 10.00 (GUI), x86-64, 8 sections
data4.bin: Microsoft Word 2007+
data5.bin: PC bitmap, Windows 3.x format, 1927 x 1080 x 24, image size
    6246720, resolution 2835 x 2835 px/m, cbSize 6246774, bits offset 54
```

As expected 1.3, there is an encrypted file (**data1.bin**) for which the 'file' command returned 'data'. I wrote the following python script to brute-force the 2-byte XOR key:

```python
import itertools
import sys

input_file = "data1.bin"

with open(input_file, "rb") as f:
```

```
    data = f.read()

def is_digit(byte):
    return 0x30 <= byte <= 0x39

print("")
for b1 in range(256):
    for b2 in range(256):
        if is_digit(b1) or is_digit(b2):
            print(f"Trying key: {b1:02x}{b2:02x}", end='\r')
            sys.stdout.flush()

            key = bytes([b1, b2])

            decrypted = bytearray()
            for i in range(len(data)):
                decrypted.append(data[i] ^ key[i % 2])

            output_file = f"o/output_{b1:02x}_{b2:02x}.bin"
            with open(output_file, "wb") as out:
                out.write(decrypted)
            print(f"[+] Saved {output_file} with key {b1:02x}{b2:02x}")
```

The script saves each attempt to an output file named `output_XX_YY.bin`, where `XX` and `YY` are the hexadecimal representation of the two bytes of the key.

After running the script, I used the 'file' command to check the output files. Most of them were just 'data', but for `0x62 0x30` there was a **zip archive**.

After extracting the zip archive, I found two files:

- `data1.bin`: a simple text file with the content: `'I'd tell you a joke about a buffer overflow... but you might not be able to handle it.%'`
- `data2.bin`: an image file:

Fig. 1 The image file extracted from the data1.bin.

For the **data2.bin** file, since in reality it was a 7zip file, I was able to simply unzip it with '7z x data2.bin'. The result is another file: `data1.bin`, that after running the 'file' command, I found out that it is a mpv4 video file about WTF News:

Fig. 2 A screenshot of the video extracted from the data2.bin file.

The **data3.bin** file seemed to be a Windows executable, but I tried to run an extrac scan on it using 'binwalk -e data3.bin'. This extracted the following files:

```
2C490/image.png
40568/image.png
786A8/image.png
83328/image.png
97400/image.png
A80B8/image.png
BC190/image.png
CE650/image.png
```

The **data4.bin** file was a Microsoft Word 2007+ document. However, I still tried to run 'binwalk -e data4.bin' on it, and it seems like there was also a Microsoft PowerPoint document inside it. The result of the binwalk command was two folders:

- `0`: the actual Microsoft Word document.
- `9940`: the Microsoft PowerPoint document.

For **data5.bin**, I also tried to run 'binwalk -e data5.bin' on it, but without any success. I've also tried 'steghide', but still no results. However, after running 'zsteg data5.bin', I found the following results:

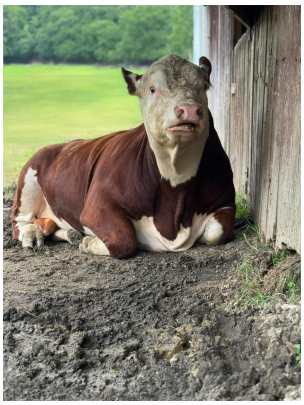
- **text**: `'We don't do late submissions... we do denial of service on deadlines.'`
- **file**: `Targa image data - RGBA 1290 x 514 x 1 +259 +512 - 3-bit alpha ' 001'`
- **file**: `OpenPGP Public Key`
- **file**: `Applesoft BASIC program data, first line number 138`

3.1.4 Deepfake Detection Results

In this section, I will present the results of the deepfake analysis for each media file.

set1/p1.jpg

set1/p2.jpg



set1/p3.jpg

set1/p1.jpg - This image appears to be **authentic**. The pixel structure is consistent, the lighting is natural, and there are no signs of manipulation.

set1/p2.jpg - It's really hard to find any inconsistencies or irregularities in hair strands, edges, lighting, shadows, or pixel structure. Even if some parts in grass, in the tree's leaves, and in the cow's front legs are a bit strange, I can't tell if this is a result of a manipulation or not.

set1/p3.jpg - Here it is clearly visible that the edges and the hair strands are not consistent and they are significantly worse quality. All around the cow there is a blur-ish area that removes the end of the hair strands, mostly around the cow's ears. This image is **manipulated**.

set2/p1.jpg

set2/p2.jpg

set2/p1.jpg - It seems like there is nothing wrong with this image. I think it is **authentic**.

set2/p2.jpg - Here the hair strands make it obvious that the background was changed. All around the hair it is really blurry and the end of the hair strands are not visible. This image is **manipulated**.

set2/p3.jpg

set2/p4.jpg

set2/p3.jpg - Here we have an 'AI generated content' warning in the bottom left corener of the image. This might be a clue. Also the girl's hair seems to be very well edited, I couldn't tell that this image was manipulated based on her hair. However, the woman's hair lose a lot of details at the top, it get's blurred and the end of the hair strands are not visible or have very low quality. This image is **manipulated**.

set2/p4.jpg - This image is very similar to the previous one. The girl's hair is very well edited again. Now there is no 'AI generated content' warning, but the woman's hair is again blurred at the top and the end of the hair strands. This image is also **manipulated**.

set3/p1.jpg

set3/p2.jpg

set3/p3.jpg

set3/p1.jpg - I can't find any inconsistencies in this image. The pixel structure is consistent, the lighting is natural, and there are no signs of manipulation. This image is **authentic**.

set3/p2.jpg - The pixel structure is consistent, the lighting is natural. I can't find any signs of manipulation. The hair strands are also consistent. This image is **authentic**.

set3/p3.jpg - Compared to p2.jpg, the girl's teeth are more white. However, I couldn't tell this without comparing the two images. Also, the man it was obviously edited. The shadow it's on the left side of the women's face, but for the man is on the right side. The line between the man and the woman is blury, unnatural and inconsistent. The man has contours in some places. This image is **manipulated**.

set2/p4.jpg

set2/p4.jpg

set4/p1.jpg - There are multiple blurry areas in this image, and the cars from the left bottom corner seems to be distorsioned and blurred. Other blurry areas around the left tower and at the front wheel of the grey car from right. These blurry areas are not consistent with the rest of the image. This image is **manipulated**.

set4/p2.jpg - Here I can't find any inconsistencies. The pixel structure is consistent, the lighting is natural. This image is **authentic**.

Fig. 3 video1.mp4

In this video every text (car numbers, subtitles, etc.) is gibberish. The movement of the lips are unnatural and the people from this video always look in strange directions, not where you would expect a person to look. For example, there is hard to find a normal eye-contact between to people that are talking to each other. At `0:31` the man leaving the water is not as wet as he naturally should be. The old woman's mouth at the end of the video is very unnatural. This video is **manipulated**.

Fig. 4 video2.mp4

In this video, the man and BigFoot are booth looking in the same direction, and both of their mouths are moving, even if only BigFoot is talking. BigFoot. BigFoot is talking. The movement of BigFoot's mouth is unnatural. His face is slightly blurred, and slightly different in each section. BigFoot has human-like eyes and teeth. When BigFoot bites the "flower bag", some "flower powder" appears in his mouth from nowhere. The subtitles are gibberish. This video is **manipulated**.

4 Conclusion

In this paper, I presented the operation CYBERSHADOW: A Digital Whodunit. I described each step of the investigation, the tools and techniques used, and the results obtained.

The investigation started with a file type detection. I wrote a C++ script that recursively scans a directory and identifies the actual file type for each file based on the magic number of the file's header.

The next step was to analyze the network traffic logs. I used Wireshark to visualize the traffic and extract the relevant information. Here it was necessarry to look for patterns and suspicious traffic.

The third step was to analyze the USB image. I mounted the image and extracted the files. I also decrypted an encrypted file using a brute-force approach. I used the 'file' command to identify the file types and 'binwalk' to deep scan the files. I also used 'exiftool' and 'steghide' to extract the metadata and hidden data from the image files.

The last step was to analyze the media files. I used a combination of manual analysis and tools to determine which images and videos were authentic and which

had been manipulated. I looked for pixel structure anomalies, metadata and lighting inconsistencies, thin details such as hair strands, and other signs of manipulation.

References

[1] Kessler, G.: File Signatures. https://web.archive.org/web/20250620072537/https://www.garykessler.net/software/index.html#filesigs (2025)

[2] Lohmann, N.: JSON for Modern C++. https://github.com/nlohmann/json (2025)

[3] Suhardjono, S., Handayani, P., Sugiarto, H., Aisyah, N., Putra, A.S.: Forensic analysis video metadata authenticity detection using exiftool. Journal of Innovation Research and Knowledge **1**(12), 1727–1734 (2022)

[4] Kuraku, S., Kalla, D.: Emotet malware—a banking credentials stealer. Iosr J. Comput. Eng **22**, 31–41 (2020)