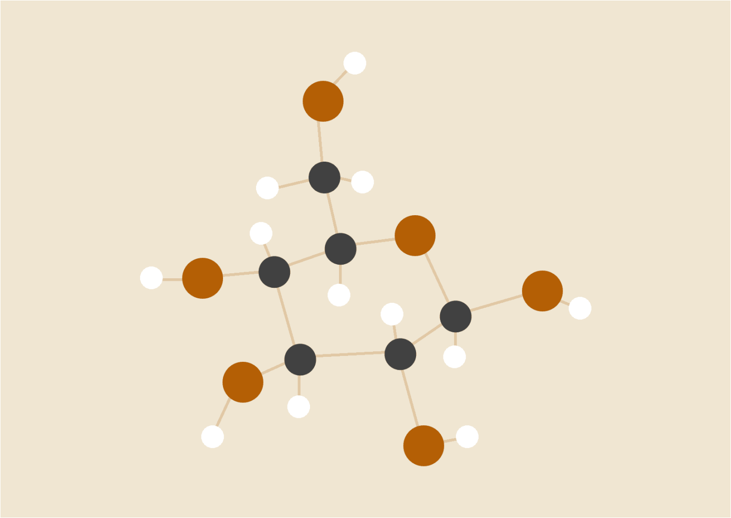
Malware Analysis

Vibe Stealer : White Mamba



INTRODUCTION

My friend recently told me he's become a *"vibe coder"*. Apparently, with the help of LLMs, he's now able to write and ship malware faster than ever. He even joked about starting his own *Malware-as-a-Service (MaaS)* operation.

He told me LLMs generate clean, high-level code that follows best practices, making it easy for him to pack it into a binary and deploy it. He sent me a sample of his latest creation and asked me to take a look.

He also claimed he already tested it on a victim, and *“it worked like a charm.”*
He insisted I check out what the victim was doing at the time of the infection and his browsing history .

Malware sample :

Malware sample : stc2_clean.exe

Tasks :

- Understand how the malware works
- Analyze how it communicates with the Command & Control (C2) server
- Figure out how data is exfiltrates
- Investigate the first victim's data (**only the first victim is relevant for this challenge**)
- Assemble the flag: it's split into **three parts** across the challenge

Note : this is not really needed to solve the challenge all you need is to find the flag

DATA: use those as base and answer in detailed manner , All script will be available in the github repo

Questions	Answer
Sha256 hash	4758a01dfe789d349da4624178dd2643290a1313b83d543f04f782a637574b18
Malware Tags	**Type:** Stealer **Language:** Python (packed) **Capabilities:** ● Screenshot capture ● Cookie theft ● Telegram C2 communication ● Executes PowerShell payloads

	🔍 **Analysis Links:** ● **ANY.RUN Sandbox:** https://app.any.run/tasks/8d4e6168-0fdf-483a-99a2-192728efcccf ● **VirusTotal Report:** https://www.virustotal.com/gui/file/4758a01dfe789d349da4624178dd2643290a1313b83d543f04f782a637574b18
packer	Nuitka is a well-known Python-to-C transpiler that compiles Python code into highly optimized C executables. To unpack binaries created with Nuitka, you can use the following tool: 🔧 nuitka-extractor – a project designed specifically to extract original Python source files from Nuitka-generated executables.
C2	Telgram bot 7421019055:AAHqU6fNWt1CRo1bAt-1x59JUMxoosrLqKA Exfill channel : -1002569946357
Data exfiltration	The malware uses a predefined secret that is also hardcoded into the Telegram C2 . This secret can be found by analyzing the extracted strings after unpacking the binary. Specifically, functions like `gen_key_current_time_and_secret_with_md5()` and `encrypt_rc4()` reveal how the encryption key is derived. To replicate the key derivation process, we reverse the logic and determine that the encryption key is generated as follows: `key = md5("timestamp" + "secret")` The key is used with RC4 to encrypt the zip field before sending them to telegram channel with the bot token

	We can estimate the approximate timestamp by checking the date the malware payload was uploaded to Telegram. However, because of potential timezone differences or system time drift, it's important to test a wide range of timestamps around the upload time. We iterate through possible timestamps, compute the corresponding keys, and use them to decrypt the files. We know we've found the correct key when the output begins with a valid ZIP file signature.
Flag	**First Part** **You can find the first part of the flag by looking at the bot's available commands.** **Among the results, you will see:** **[{'command': 'status', 'description': '#1nexus{T3legramC2_'}]** --- **Second Part** **There are two ways to obtain this part of the flag:** **1. Dynamic Analysis:** **When executing the malware, we observe that a PowerShell script runs as part of its behavior. One of the payloads is obfuscated, but after decoding it, we find that it takes a screenshot of the user's screen.** **In addition to capturing the screen, the script prints the second part of the flag on the screenshot:** **Pyth0n_4nd_P0wer5hell_** **2. Static Analysis:** **After decrypting all related files, one of the screenshots can be viewed. The accompanying description suggests:** **“Look at what the user was doing.”**

By analyzing the screenshot, we can visually identify the second part of the flag.

Third Part

The description hints at checking the user's browsing history.
Once the history file is decrypted, we can search (e.g., using `grep`) for strings like `flag` or `}`.
Within the browser history file, we find the final part of the flag in a URL:

http://rankiha.vercel.app/?flag_part3:With_C_m4ke_1337_Malware}

Final Flag:

nexus{T3legramC2_Pyth0n_4nd_P0wer5hell_With_C_m4ke_1337_Malware}