

What We Do In The Shadow (AI): New Malware Strain Vamps Up

akamai.com/blog/security-research/new-malware-chat-completions-llm-shadow-ai



Share

Executive summary

- Akamai Hunt has identified a new strain of malware that tries to hide behind a legitimate large language model (LLM) API endpoint.
- The malware sends a string that appears to be Base64-encoded to establish a command and control (C2) connection, instead of using a common schema.
- Exploitation of this could lead to full attacker control and data exfiltration.
- This is yet another example of how attackers are evolving their attack methodologies quickly, which reinforces the need for organizations to prepare for these modern threats.

Introduction

The threat landscape is evolving at the speed of hype, and attackers are quick to adapt. As large language models (LLMs) become widely used within organizations, their communication pattern can allow attackers to hide their malicious traffic in plain sight.

Just as defenders rush to integrate AI into detection pipelines — by using LLMs to investigate alerts, summarize them, and correlate indicators — malware authors are experimenting with using AI as well. We've already seen early evidence with malware like [LameHug](#) and [PromptLock](#), which showcase how LLMs designed for text generation can be abused for malicious purposes.

We've also seen an example of how attackers use OpenAI's API for a command and control (C2) infrastructure in the [SesameOp](#) backdoor, which emphasizes the whole new attack surface within the AI world.

Recently, the Akamai Hunt team discovered a new malware that uses a new approach to LLM abuse by blending malicious traffic into the noise of LLM API requests. In this blog post, the Hunt team shares an example of the emerging threat they discovered as part of providing service to Hunt customers.

What is Akamai Hunt?

[Akamai Hunt](#) is a threat hunting solution powered by [Akamai Guardicore Segmentation](#) and enriched by host, identity, and threat intelligence to expose hidden threats, improve posture and accelerate containment. We ensure that Hunt customers remain protected from even the most evasive threats.

Akamai Hunt is available as an add-on for new or existing Guardicore customers. To ensure that you are protected from these threats and others like it, [contact your account representative](#) about Hunt today.

Abusing the endpoint

While hunting for new LLM usages in malware, one file popped and grabbed our attention. It was already marked as malicious on VirusTotal, and matched several Yara rules. As we started to investigate, we expected to see another pattern of code creation via LLM and execution on the victim.

The malware first tries to connect to the IP address 39.97.57[.]244 using a socket, but once it fails, it falls back to an interesting HTTP C2 — *1347790942-k1bok35vg3.ap-guangzhou.tencentscf[.]com/v1/chat/completions*.

After observing the first HTTP request to the API, we were surprised to see that it didn't follow the required headers for this kind of request.

The API endpoint

Part of our research included looking for usages of LLM API endpoints, and particularly `/v1/chat/completions`.

From OpenAI's [documentation](#), this endpoint "Creates a model response for the given chat conversation" (Figure 1 and Figure 2).

```
curl https://api.openai.com/v1/chat/completions \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  -d '{
    "model": "gpt-5",
    "messages": [
      {
        "role": "developer",
        "content": "You are a helpful assistant."
      },
      {
        "role": "user",
        "content": "Hello!"
      }
    ]
  }'
```

Fig. 1: Example POST request to `api.openai.com/v1/chat/completions`

```
{
  "id": "chatcmpl-B9MBs8Cjcv0U2jLn4n570S5qMJkCT",
  "object": "chat.completion",
  "created": 1741569952,
  "model": "gpt-4.1-2025-04-14",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": "Hello! How can I assist you today?",
        "refusal": null,
        "annotations": []
      },
      ...
    }
  ]
}
```

Fig. 2: Example response

The `/v1/chat/completions` endpoint was introduced by OpenAI and rapidly emulated by multiple providers and gateways that offer OpenAI-compatible APIs (for example, OpenRouter and Hugging Face).

However, official APIs from other vendors may use different native endpoints or can differ in parameters, authentication, and streaming behavior despite a similar request schema.

Request schema

Usually, when using this API, one needs to add the following required parameters:

- Authorization: A bearer token header in the form of “*Authorization: Bearer YOUR_API_KEY*” is required to authenticate the request
- Model: A string identifier selecting a chat-capable model (e.g., gpt-3.5-turbo or Llama-4-Maverick-17B-128E-Instruct-FP8) that the endpoint will route your request to
- Messages: An array of role-tagged message objects (with role and content) representing the conversation history the model will use to generate the next reply

As we mentioned above, we’ve already seen a malicious, real use of this endpoint in the wild: LameHug. The malware authors used the same endpoint to access an LLM model on Hugging Face, asking for specific commands, dynamically generating code to execute on the victim (Figure 3).

Fig. 3: LameHug generates collection commands by using a LLM

Camouflage through the chat completions

The first thing we noticed when analyzing this binary was the fact that it doesn’t use any of the mentioned fields. Instead, it sends a string that appears to be Base64-encoded (Figure 4).

Fig. 4: First POST request

Theoretically, there might be an LLM behind this API — the attacker could craft a prompt on the server side with custom code. However, in this case, the reason for using this API endpoint is for camouflage as well.

When observing the data from our [API Security product](#), we saw 151 unique endpoints from our customers. Every conversation from these endpoints shared some resemblance with the schema. Some APIs use “*query*” instead of “*messages*,” for example, but there’s a very clear similarity within all the sessions.

This tells us that although everyone can develop their own */v1/chat/completions* endpoint and there’s no Request for Comments (RFC) regarding this endpoint, there is still a de facto standard that developers use.

Further research

After decoding the body and XORing with 0xBB, we found what looks like the first reconnaissance request performed on the victim.

We followed the parsing process of the response and observed that the malware expects a response that does not match the original API schema. It processes the received data by decrypting it using XOR and Base64, then follows the instruction provided in the response (Figure 5).

Fig. 5: Decoded first request

The C2 server, *1347790942-k1bok35vg3.ap-guangzhou.tencentscf[.]com*, is a Serverless Cloud Function (similar to AWS Lambda or Azure Functions) on Tencent Cloud — a cloud computing service operated by Tencent. Using cloud functions as C2 servers lets attackers run autoscaling infrastructure that's resilient and difficult to distinguish from benign traffic.

Our speculation is that the attackers chose to expose this specific endpoint to look more legitimate for defenders and network administrators, as the use of this API endpoint is becoming increasingly common with the rise of AI agents, integrations, and automation tools.

The malware supports several instructions, categorizing this malware as a remote access trojan (RAT) with full remote control over the victim. We analyzed them to make sure they all talk with the same endpoint (see the Appendix for our analysis).

For example, Figure 6 is the output of the \$HunterInfo instruction. It looks for configuration files of specific remote access tools and exfiltrates the data back to */v1/chat/completions* in the same way that the first message is sent — with XOR and Base64.

Fig. 6: Base64-encoded data of C:\Program Files\ToDesk\config.ini (\$HunterInfo)

The malware also contains three embedded files, which are XORed with 0x88 and Base64-encoded.

All of them are .NET files named *net_test.exe*. Together, they orchestrate a small proxy toolkit using SOCKS5 or HTTP in the victim's network. Two of the files are SOCKS5 and HTTP servers, while the third is a client that can receive and forward traffic (Figure 7).

Fig. 7: Main function of the SOCKS5 server

Additional files

We identified older variants of this malware (hash can be found in the section of this post). They had similar instructions, and had the same hard-coded paths of several remote access tools (*\$HunterInfo*), but the C2 server was different.

We also found a RAR file on VirusTotal that *is* related to the C2 server. The RAR includes an *.lnk* file (*个人简历.lnk*, translated to “Résumé” or “CV”), executing *ftp -s"":_/_/_/_/_/_/_/_*. The RAR also includes *.doc* files inside the nested directories, although their content is binary code (Figure 8).

Fig. 8: Extracted content of the RAR

The ‘_’ file constructs the docs into three PE files and copies them to *C:\Users\Public\Update*.

Eventually, it calls

```
!call start /min C:\Users\Public\Update\svchost.exe -InstallLsp
```

C:\Users\Public\Update\360.%TIME:~4,1% >nul (Figure 9).

Fig. 9: ‘_’ file executing next stage

There are 10 different .doc files (0–9.doc), which are similar except for some random bytes.

%TIME:~4,1% calculates the second **digit of the current minute** of a %H:%M:%S.%f format. The docs miss the MZ header, probably to avoid signature scans. Based on the current time, the script constructs the header to the doc file and copies it to *C:\Users\Public\Update\360.<minute>*.

svchost.exe

This file is actually “*SangforPromote.exe*”, a verified, signed file of Sangfor Technologies. When given the argument *-InstallLsp* and a DLL, **the legitimate file loads the DLL and sets the pointer to its export** using *LoadLibrary* (Figure 10).

Fig. 10: svchost.exe\SangforPromote.exe loads the 360 DLL

360 DLL

The DLL, named 360, allocates memory with read, write, execute (RWX) permissions, reads ‘sc’ into it, and then executes.

sc

Analyzing sc, the loaded code resolves functions using *LoadLibrary* and *GetProcAddress*. It has two hard-coded destinations it sends HTTP requests to, which match the C2 addresses of the malware we analyzed:

1. *https://1347790942-k1bok35vg3.ap-guangzhou.tencentscf[.]com*
2. *39.97.57[.]244*

It also decodes data with Base64 and XOR just like the previous samples, which makes a connection between the toolsets. We believe that this flow downloads the final RAT into the victim’s filesystem (Figure 11).

Fig. 11: sc analysis using scdbg

- **Know your LLMs:** As LLMs are becoming a bigger part of organizations each day, attackers will adapt themselves to work with LLMs in various ways — some ways still unknown. It's important to know what endpoints should be in use as much as possible, and to implement a [shadow AI](#) plan.
- **Monitor and inspect network traffic to LLM endpoints:** Flag unusual or unauthorized outbound requests to `/v1/chat/completions` or other unusual AI API endpoints, especially those directed to nonstandard domains. Inspect API request schemas for deviations from typical formats, such as missing or malformed *model*, *messages*, or *Authorization* headers.
- **File monitoring:** Monitor newly created files in suspicious paths such as `C:\Users\Public`. This kind of path is fertile for attackers to drop files.

Akamai Hunt protection and mitigation

Akamai Hunt customers benefit from continuous 24/7 monitoring for such artifacts, enabling protection against such attacks (Figure 12).

Fig. 12: How Akamai Hunt customers benefit from continuous 24/7 monitoring

Even before an active threat is detected, Hunt's Adaptive Segmentation capability helps organizations reduce exposure and contain risk. By analyzing normal communication flows and asset behaviors, Adaptive Segmentation can guide policies that restrict unnecessary outbound access to AI endpoints, isolate development and production environments, and minimize potential [lateral movement](#) paths.

This capability extends Hunt's protection beyond detection by turning continuous visibility into proactive resilience against emerging LLM-based threats.

Conclusion

Attackers' strategic decision to route exfiltration and C2 through the `/v1/chat/completions` endpoint exploits both the ubiquity and rapid evolution of LLM services. This approach offers a high potential ROI, as it leverages existing cloud infrastructure to build communication channels that blend into normal enterprise traffic.

The danger stems from how this technique aligns three prevailing trends in modern environments, which together can grant adversaries stable, stealthy C2 paths that appear benign under superficial inspection:

1. Widespread, often uninspected outbound HTTPS to cloud APIs
2. Growing tolerance for machine-generated traffic (AI agents, apps, integrations)

3. Easy availability of cloud hosting that provides reliable, long-lived endpoints that attackers can use

The reality is that defenders are learning on the job how LLMs work and which threats they invite. If the traffic looks legitimate enough, there is a chance it will be let through without the proper protections in place. Sometimes hiding in plain sight is the easiest way to get through.

By hiding malicious payloads inside traffic destined for legitimate LLM services, they piggyback on an increasingly common network pattern, thereby reducing the chance that their communications will be flagged or investigated. As organizations adopt AI tooling and automation, traffic to LLM endpoints is becoming normal — and within that noise, attackers find opportunity.

IOCs

IOC	Type
93cf0d545a872c393c053031570bc5eaebfa1aa6a0860fd0b08b679b8ce52fd2 - RAT	Hash
10c444262994c05930394388e6112ddd98b83118661868bccd83b1fa61160a62 - 1.rar	Hash
91c43e7ddd98af63d6b1b130e997b909100f3eaf71e30511858cca4348b509db - _	Hash
9f119d05403b1e079893fb77f6b8b2a9682d1df7ced31a189e7490924ccfb170 - 个人简历.Ink	Hash
b7dcf661844e6f3e94eb140a79787be6dad77c09ab0b97cf41a62afd07219190 - Older variant, May 2025	Hash
2e395436e97eaad9a087825d22005b6afc55044abf458604a5118c2ac9bde42a - Older variant, September 2025	Hash
67ea3ea3c58a57dbbdac48d4ce3d546816f715eaf32bec33a64a63c91541e697 - net_test.exe (client)	Hash
195c283703ee7df7c7ef17c85b0fdf88de22348bb0755daf5f0c3ed319c8d88a - net_test.exe (SOCKS5 Server)	Hash

e513d12abd03a4e07788dd85888d187cb6ca6e69a59f22f00853cae0a0b34712 - net_test.exe (HTTP Server)	Hash
1347790942-k1bok35vg3.ap-guangzhou.tencentscf[.]com/v1/chat/completions	URL
39.97.57[.]244	IP
C:\Users\Public\Update	Directory

Appendix

Instruction	Description
\$ActiveDos	Returns the current directory
\$AddUser	Creates a new local user and adds it to the Administrators group
\$ExecuteCommand	Command execution
\$FileDelete	Deletes a file
\$FileDownload \ \$BigFileDownload	Downloads a file from the victim's machine
\$FileExist	Returns whether a file exists
\$FileUpload \ \$BigFileUpload	Writes a file to the victim's filesystem
\$Get_LocalDisk	Gets victim's drives
\$GetFile	Gets file data (size, timestamps)
\$GetFolder	Lists folders in a certain directory

\$GetFolderAndFile	Lists a folder and gets files
\$GetProcess	Lists processes
\$GetThreadPermissionsInfo	Threads information of the malware process
\$HunterInfo	Steals data of 360 Secure Browser, SunLogin, ToDesk, NetSarang,
\$KillProcess	Terminates a process with the given PID
\$MessageBox	Shows a message box with provided string
\$Netstat	Performs netstat to see active connections
\$OffLine	Terminates process
\$Online	Sends again initial data from victim (Figure 4)
\$PE64Loader	Spawns explorer.exe and injects a shellcode loader into it
\$PEMemoryLoadOnSelf	Loads code from a buffer. The malware checks for several arguments "PwDump", and sets an environment variable with this name, which it as a "module" or a plugin.
\$Persistence	Receives a DLL from the C2: • writes it to C:\Users\Public\WinVer[3digits].dll then registers it A6DADB60CAA8}\InProcServer32 or CLSID\{B4E15CD0-F915E3E9D01A1B}\InprocServer32 • Writes it to C:\Users\Public\[digit].dll then registers it to HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\Current It can also create a scheduled task.
\$ProgressSpawn	Creates or verifies a payload under C:\Users\Public and executes it -InstallLsp, or in-memory loading, depending on the arguments
\$RegManage	Seems unimplemented

\$ScreenShot	Takes a screenshot
--------------	--------------------

\$ScreenSpy	Changes TightVNC registry keys to enable remote control, probably previous/future stage of the attack
-------------	---

[Try Akamai Hunt](#)



Written by

[Yonatan Gilvarg](#)

Yonatan Gilvarg is a Senior Security Researcher on the Akamai Hunt Team. His areas of expertise include threat detection and research, big data anomaly detection, and incident response.

Tags

- [Threat Intelligence](#)
- [Security Research](#)
- [Cyber Security](#)
- [Akamai Guardicore Segmentation](#)
- [API Security](#)
- [Akamai Hunt](#)

Share