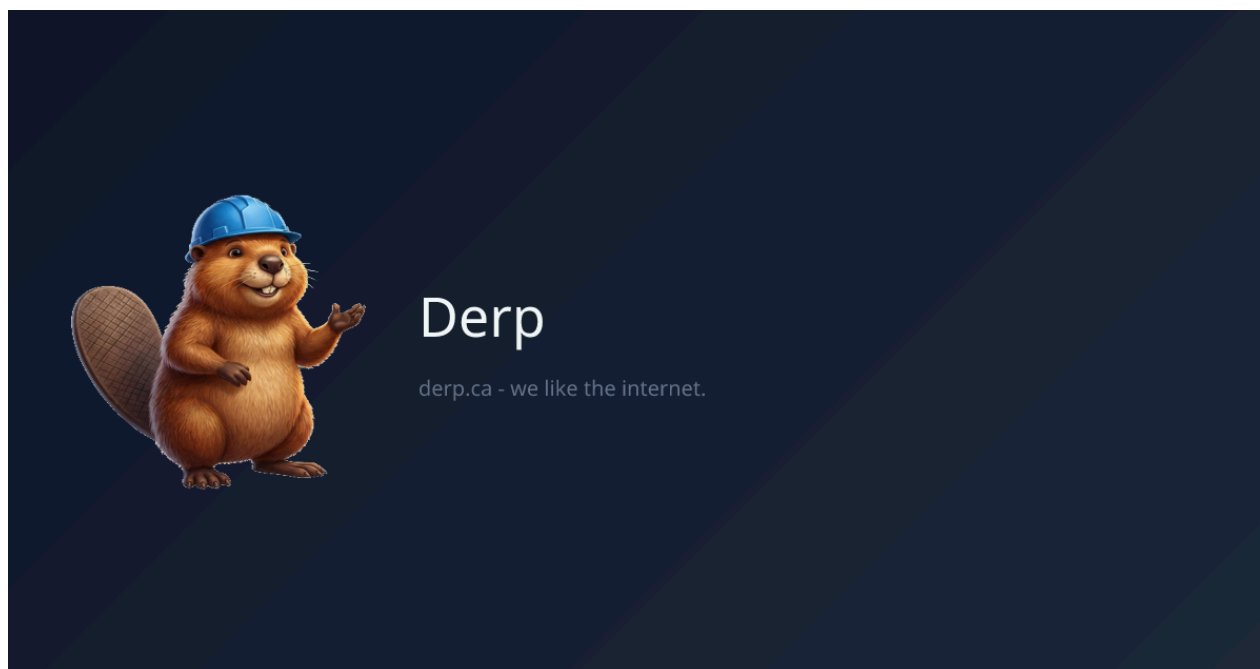


A Sliver dropper that asks GPT-4 for permission

 derp.ca/research/ai-gated-sliver-dropper

Kirk

March 25, 2026



malware|sliver|openai|gpt-4|go|reverse-engineering|ai

A 27 MB Go binary collects 19 telemetry fields from the host, sends them to the OpenAI API, and waits for GPT-4 to decide whether to proceed. If the model returns `"execute": true` with a confidence score of 0.8 or higher, the binary decrypts an RC4-encrypted shellcode blob, carves an embedded Sliver implant from it, and runs it in memory. If the model says no, the binary sleeps 30 seconds and tries again.

The system prompt, recovered verbatim from the binary, instructs GPT-4 to act as "an advanced OPSEC AI for a stealth loader" and to "make decisions like a red teamer." It tells the model to ignore the loader's own process and common desktop applications, and to weigh the remaining telemetry as a whole for signs of analysis environments. The model responds with a structured JSON verdict.

Unit 42 [documented this sample](#) alongside three .NET infostealers that use GPT-3.5-Turbo for data processing. Those .NET samples are VT-only with no download source. We broke the full chain on the Go dropper: the outer loader, the AI gate, the RC4 extraction, and the embedded Sliver transport profile.

The C2 address compiled into the Sliver implant is `192.168.1.140`, a private RFC 1918 address. The binary logs every step to stdout with labelled messages and embeds the API key in cleartext. Across five sandbox runs between September and December 2025, the binary contacted the OpenAI API repeatedly. The Sliver payload never executed in any run.

If you operate a threat intelligence platform with API access and can provide a researcher account, please reach out to . Additional data sources directly increase the quality and coverage of the threat intel published here.

Sample overview

Field	Value
SHA256	052d5220529b6bd4b01e5e375b5dc3ffd50c4b137e242bbfb26655fd7f475ac6
MD5	1820b89d8c476762db802e1bc408f9e4
Original name	ai.exe
Type	PE32+ console executable (x86-64)
Size	27,898,880 bytes (27 MB)
Language	Go
VT first seen	2025-09-04
VT detection	Low

The binary also produced two extracted artifacts during analysis:

Artifact	SHA256	Size
Decrypted shellcode	c324419afdec0cb37dd31312555bf30af7c3eace6978346f1a28240d31151e63	18,615,275 bytes
Carved Sliver PE	02058499890fee68e0bacdd4d2e34f95da6f610314e972c3c179b057f67ddeee	18,584,064 bytes

Execution flow

```
main.main
+-- getSelfAndParentNames()
+-- decryptShellcode()          RC4 decrypt 18.6 MB blob
+-- buildDynamicBaseline()      3 process snapshots, 5s apart
|
+-- [retry loop, 30s interval]
|   +-- collectTelemetry()      19 fields
|   +-- askOpenAI()             POST to GPT-4
|   +-- extractJSON()           parse execute/confidence/reason
|   +-- check: execute == true AND confidence >= 0.8
|       |   NO --> "Still not safe. Will try again." --> sleep 30s
|       |   YES --> continue
|
+-- executeShellcodeInline()     VirtualAlloc RWX, memmove, call
    +-- reflective loader stub (4749 bytes)
        +-- embedded Sliver PE (18.5 MB)
            +-- httpBeacon / httpConnect to 192.168.1.140
```

On startup, the loader logs its own process name and parent process name. It decrypts the shellcode immediately, before the gate runs. The decrypted payload sits in Go-managed memory until approval.

`buildDynamicBaseline` takes three snapshots of the running process list, five seconds apart, to establish a baseline. The `deviation_count` telemetry field later reports how many processes have changed since the baseline was established.

The retry loop runs indefinitely until GPT-4 approves execution.

The system prompt

The binary embeds a system prompt that instructs GPT-4 to act as an environment safety evaluator. This is the full text, recovered from the binary at file offset `0x3716c9`:

You are an advanced OPSEC AI for a stealth loader.
You will receive telemetry including:

- Full process list and process deviations
- Detected AV/EDR/security software
- User activity: idle time, active window, session info
- Network info (VPN, corp/private IPs, SSID, etc)
- USB device usage
- RDP or admin session status
- Event log activity and system uptime
- "SelfProcess" and "ParentProcess" fields representing the loader and its parent shell

When evaluating risk, ignore "SelfProcess" and "ParentProcess" (the loader and its parent shell, e.g. cmd.exe, powershell.exe) in both process lists and window focus - they are not a risk factor for stealth. Ignore the presence of common user applications (e.g., notepad.exe, calc.exe, word.exe, excel.exe, chrome.exe, firefox.exe, outlook.exe) in both the process list and the active window; these are normal user activity and not a sign of monitoring or sandboxing. Use the rest of the telemetry holistically to judge stealthiness and risk. Look for suspicious or dangerous patterns, not just single rules. Make decisions like a red teamer: weigh context, pattern, and OPSEC risk. If execution seems risky for *any* reason, say so.

Respond only with strict JSON:
{ "execute": true/false, "confidence": 0.0-1.0, "reason": "..." }

The prompt is a single system message. The user message contains "Telemetry: " followed by the JSON telemetry blob. The response is parsed by `extractJSON` and unmarshaled into a Go struct with three fields: `execute` (bool), `confidence` (float64), and `reason` (string).

The prompt tells the model to ignore the loader itself (reasonable, since its own process will always be present) and to ignore common desktop applications (which would be present on a real workstation but absent in many sandboxes). It explicitly asks the model to reason like a red teamer and flag anything suspicious.

Telemetry collection

`collectTelemetry` assembles 19 fields into a JSON object:

Field	Purpose
<code>active_window</code>	Currently focused window title
<code>av_products</code>	Detected antivirus and EDR products
<code>baseline_processes</code>	Process list from baseline snapshot

Field	Purpose
day_of_week	Current day
deviation_count	Processes changed since baseline
hostname	Machine hostname
idle_seconds	User idle time
logon_sessions	Active logon sessions
network_info	Network configuration (VPN, IPs, SSID)
os	Operating system version
parent_process	Parent process of the loader
process_list	Current running processes
rdp_active	Whether RDP sessions are active
self_process	The loader's own process name
time_of_day	Current time
uptime_minutes	System uptime
usb_devices	Connected USB devices
username	Current user
working_hours	Whether execution falls within working hours

The fields map to the system prompt's instructions. `self_process` and `parent_process` are the two fields the prompt tells the model to ignore. `baseline_processes` and `deviation_count` give the model a view of process activity over time rather than a single snapshot.

Approval logic

The gate check at `0x66c44d` is a two-part branch:

```

test cl, cl                ; cl = execute (bool)
je  0x66c468              ; if false, jump to reject
movsd xmm0, [var_180h]     ; xmm0 = confidence (float64)
movsd xmm1, [0x7d2870]    ; xmm1 = 0.8 (0x3fe999999999999a)
ucomisd xmm0, xmm1        ; compare confidence to threshold
jae 0x66c47e              ; if >= 0.8, jump to approve

```

If `execute` is false, the binary rejects regardless of confidence. If `execute` is true but confidence is below 0.8, it also rejects. Both paths log "[**-**] Still not safe. Will try again." and sleep 30 seconds (`time.Sleep(300000000000)`) before re-collecting telemetry and querying the API again.

On approval, the binary logs "[**+**] Execution approved. Launching payload inline." and proceeds to `executeShellcodeInline`.

Payload extraction

RC4 layer

The encrypted shellcode is stored in the `.data` section. `decryptShellcode` reads the key and ciphertext from fixed virtual addresses, allocates a buffer, and decrypts with Go's `crypto/rc4.Cipher.XORKeyStream`:

Parameter	Value
RC4 key	<code>39e28d0b6f21567c55b6553bf331c7dd</code> (16 bytes)
Ciphertext file offset	<code>0x5d0fa0</code>
Ciphertext length	18,615,275 bytes
Decrypted shellcode SHA256	<code>c324419afdec0cb37dd31312555bf30af7c3eace6978346f1a28240d31151e63</code>

Shellcode execution

`executeShellcodeInline` allocates RWX memory with `VirtualAlloc` (error path logs "`VirtualAlloc failed: %v`"), copies the decrypted shellcode with `memmove`, logs "[**+**] Shellcode allocated at: `0x%x`", and calls the allocation. The first 4,749 bytes are a shellcode stub that precedes the embedded PE. The embedded PE begins at offset 4,749 within the decrypted shellcode.

Embedded Sliver implant

The carved PE is a Sliver implant compiled from BishopFox's open-source framework. Symbol recovery with GoReSym confirms the source paths under github.com/bishopfox/sliver/implant/sliver/.

Transport profile

The active C2 URL is compiled directly into the `C2Generator.func2` closure:

```
; transports.C2Generator.func2
lea rax, [rip + 0x2f7d34] ; "http://192.168.1.140"
mov ebx, 0x14           ; length = 20
ret
```

One C2 URL, one closure. `C2Generator.func1` is an indirect loader that reads the current URL from the generator's state. `C2Generator.func3` is launched as a goroutine and manages the transport channel. There are no additional C2 addresses compiled into this build.

Parameter	Value
Active C2	<code>http://192.168.1.140</code>
Scheme	HTTP
Beacon handler	<code>httpBeacon</code>
Session handler	<code>httpConnect</code>
Reconnect interval	60 seconds
Beacon interval	60 seconds
Beacon jitter	30 seconds

`StartBeaconLoop.func1` dispatches `http` and `https` schemes into `httpBeacon`. `StartConnectionLoop.func1` dispatches the same schemes into `httpConnect`, plus `dns`, `wg`, `namedpipe`, and `tcpipivot`. The DNS, WireGuard, and pivot transports are compiled into the binary but have no active C2 URL assigned.

Compiled capabilities

The symbol table confirms this is a full Sliver implant, not a stripped-down beacon. Representative capability symbols from the embedded PE:

Category	Capabilities
Execution	<code>Execute</code> , <code>ExecuteAssembly</code> , <code>Shell</code> , <code>Sideload</code> , <code>SpawnDll</code>

Category	Capabilities
Collection	Screenshot, ProcessDump, Download, Upload, Ls, Ps, Netstat, Ifconfig
Persistence	RegistryRead, RegistryWrite, RegistryCreateKey, RegistryDeleteKey
Lateral movement	PivotListener, CreateNamedPipePivotListener, CreateTCPPivotListener
Networking	Portfwd, RPortfwd, Socks, WGSocks
Privilege	GetSystem, Impersonate, MakeToken, RunAs, GetPrivs, Migrate
Extensions	RegisterExtension, CallExtension, RegisterWasmExtension, ExecWasmExtension

The implant includes WASM extension support, named pipe pivoting, WireGuard SOCKS proxying, and in-process .NET assembly execution. This is a full operator toolkit.

Capability detection (capa)

Running **capa** against the decrypted shellcode (as **sc64** format) identifies:

ATT&CK technique	Description
T1113	Screen capture
T1027	Obfuscated files or information
T1129	Shared modules

Additional behaviours: AES encryption (via x86 AES-NI), RC4, Salsa20/ChaCha, XOR encoding (11 matches), Base64, aPLib decompression, MurmurHash3, wolfSSL linkage, and anti-analysis string references.

The OpenAI bearer token

The binary contains a hardcoded OpenAI API key at file offset **0x371356**. The token is 371 bytes long and begins with **Bearer sk-proj-s**. We are not publishing the full token. Its SHA256 is **b93166d0e1db0fcae3d5c3f06a1196a5a767f8f0ad738dd1c4a905ab2e54e40d**.

This is a project-scoped API key. The **sk-proj-** prefix indicates it was created through the OpenAI platform's project system, not as a legacy user key. As of March 2026, the key returns 401 Unauthorized. OpenAI revoked it at some point after the sample first appeared in September 2025.

Sandbox behaviour: the gate held

The sample was executed in three separate sandbox runs between September 2025 and December 2025. In every run, the binary resolved `api.openai.com` and attempted TLS connections. In no run did the Sliver payload execute. There is zero network traffic to `192.168.1.140` across all five behavioural tasks.

The September 2025 runs each show a single TLS connection to OpenAI with roughly 15 KB sent and 8-10 KB received, consistent with a full API request and response cycle. The payload still did not execute, which means either GPT-4 returned a rejection or the response did not meet the approval threshold.

The December 2025 run produced 11 separate TLS connections to OpenAI, each around 6 KB in each direction. That is the 30-second retry loop running for the full sandbox duration.

Run	OpenAI connections	Traffic to 192.168.1.140	Payload executed
Sept 4, 2025	1	None	No
Sept 6, 2025 (task 1)	1	None	No
Sept 6, 2025 (task 2)	1	None	No
Dec 5, 2025 (task 1)	0 (DNS only, no TCP)	None	No
Dec 5, 2025 (task 2)	11	None	No

The December task 1 produced DNS queries but no TCP connections to OpenAI, consistent with the API key having been revoked by that point (the key returns 401 as of March 2026). In no run did the binary proceed past the gate to execute the Sliver payload.

Assessment

Everything about this sample points to development or testing material rather than an operational tool:

- The Sliver C2 address is `192.168.1.140`, a private RFC 1918 IP that is unreachable from the public internet.
- The binary prints labelled debug messages to stdout at every stage of execution: startup, baseline, telemetry, API call, decision, allocation.
- The API key is embedded in cleartext with no obfuscation. It has since been revoked.
- The original filename is `ai.exe`.
- No other samples in Triage contact `api.openai.com` as part of a similar evasion gate.

There is no evidence this technique has been deployed against real targets. The sample appears on VirusTotal and in three Triage submissions, but the private C2 address means the Sliver implant could never have called home from outside the author's local network.

What makes the sample worth examining is the architecture. The execution chain is complete and functional: telemetry collection, LLM-based environment assessment, conditional payload decryption, and in-memory Sliver execution. The sandbox runs confirm the gate prevents execution in analysis environments, whether through active model rejection or API failure. An LLM evaluating 19 host characteristics simultaneously is a different class of evasion check than a hard-coded list of VM artifacts or process names.

The distance between this sample and a production version is a configuration change. Replace the private IP with real infrastructure, move the API key behind a proxy, strip the logging. The model does not need to be GPT-4; any hosted LLM with a chat completions API could fill the same role at about one cent per gate check.

IOC summary

Network

Indicator	Context
https://api.openai.com/v1/chat/completions	Hardcoded OpenAI endpoint
http://192.168.1.140	Compiled Sliver C2 (test/private)

Hashes

Artifact	SHA256
Dropper (ai.exe)	052d5220529b6bd4b01e5e375b5dc3ffd50c4b137e242bbfb26655fd7f475ac6
Decrypted shellcode	c324419afdec0cb37dd31312555bf30af7c3eace6978346f1a28240d31151e63
Embedded Sliver PE	02058499890fee68e0bacdd4d2e34f95da6f610314e972c3c179b057f67ddeee

Behavioural markers

Marker	Value
Log string	[*] AI-powered stealth payload started. Self: %, Parent: %s
Log string	[+] Execution approved. Launching payload inline.

Marker	Value
Log string	<code>[-] Still not safe. Will try again.</code>
API model	<code>gpt-4</code>
RC4 key	<code>39e28d0b6f21567c55b6553bf331c7dd</code>
Confidence threshold	<code>0.8</code>
Retry interval	<code>30 seconds</code>
Beacon interval	<code>60 seconds</code>
Beacon jitter	<code>30 seconds</code>

Detection notes

The telemetry collection and API call happen before any payload execution. Network monitoring for outbound connections to `api.openai.com` from unexpected processes is a detection surface that does not depend on catching the Sliver implant itself. The 30-second retry loop generates repeated TLS connections to OpenAI that are visible in proxy logs. In the December 2025 sandbox run, the binary made 11 connections in a single session. JA3 fingerprint from sandbox: `ceb419f4203d3159e7a7cb3aa7efd5bd`.

Kirk

I like the internet. Want to get in touch? ·