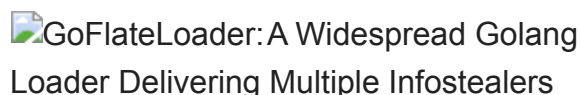


GoFlateLoader: A Widespread Golang Loader Delivering Multiple Infostealers

 gendigital.com/blog/insights/research/goflateloader-delivers-multiple-infostealers

Vojtěch Krejsa

June 10, 2026



Key Points

- Gen Threat Labs has been tracking GoFlateLoader, a widespread Golang loader used to deliver multiple infostealers, including Amatera, Remus, Lumma, Vidar and StealC.
- GoFlateLoader appears both in x86 (32-bit) and x86-64 (64-bit) variants, matching the bitness of the payload it is supposed to execute.
- The loader is designed for in-memory payload execution and is deliberately inflated with a massive PE overlay to hinder detection
- The two main delivery paths are via supposedly cracked software and a [malicious TDS recently documented by Check Point Research](#), which may redirect users to dedicated landing pages serving password-protected archives containing GoFlateLoader.

Introduction

Not every threat that matters is technically sophisticated, and that is also the case with GoFlateLoader, which is a rather simple loader written in Go, whose sole purpose is to decode and execute the payload in memory. What stands out the most is not what the loader does but rather what it does not do – it comes without anti-debugging, anti-VM, or sandbox-evasion checks, and also lacks API hashing or CFG obfuscation, the kind of tricks that loaders almost always come with. Instead, GoFlateLoader relies on one of the simplest yet still

effective tricks to stay under the radar – it appends a massive PE overlay at the end of the file, deliberately inflating the binary's size (hence the name GoFlateLoader).

Nevertheless, it is important to understand that malware does not necessarily need to be elegant or stealthy. In the end, the only thing that matters to attackers is whether it can successfully deliver the final payload. And if that is the only metric, GoFlateLoader appears to be working fairly well for its operators, as we continue to see it widely distributed across our user base. Since the beginning of April, we have protected more than 33,000 unique users from it, with the most affected countries including Brazil, India, Argentina, Mexico, Turkey and Spain.

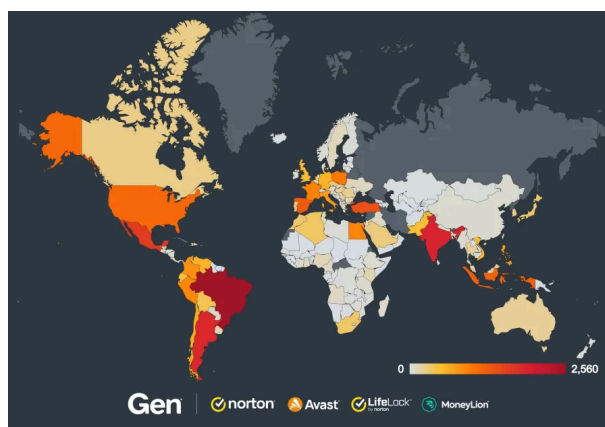


Figure 1: Global distribution of unique users protected by Gen from GoFlateLoader by country since April 2026.

GoFlateLoader's Execution Flow

If we strip away the obfuscation noise that the Go compiler and the author's decoy code add, the loader's execution flow is short and linear:

- Copy the encoded payload blob from the `.rdata` section onto the stack.
- Decode the payload using a small, multi-stage, custom byte-level transformation into a valid PE.
- Parse the PE headers of the decoded payload (image base, image size, sections, data directories, etc.).
- Allocate an RWX memory region using `VirtualAlloc` with `MEM_COMMIT | MEM_RESERVE` and `PAGE_EXECUTE_READWRITE` flags, trying the payload's preferred `ImageBase` first and falling back to an OS-assigned address if that fails.
- Map the decoded payload into the allocated memory region by copying the PE headers first, then walking the section table and writing each section to its target virtual address.
- Apply relocations (only if preferred base mapping fails).
- Resolve imports by walking the standard `IMAGE_IMPORT_DESCRIPTOR` table and rebuild the IAT in place via `LoadLibrary` and `GetProcAddress`.

- Transfer execution to the final payload using Go's `syscall.Syscall`.

This is essentially a textbook example of manual PE loading with the payload reconstructed and executed entirely in memory so that it never touches the disk.

[illegible]

Figure 2: Annotated decompiled code of GoFlateLoader's main routine.

Notably, GoFlateLoader transfers execution to the final payload via `syscall.Syscall`. In Go, `syscall.Syscall` has the signature `Syscall(trap, nargs, a1, a2, a3)`, where `trap` is the function pointer to invoke, `nargs` is the argument count, and `a1`, `a2`, and `a3` are the first three arguments passed to the target function. Here, `trap` is set to point to the final payload's entry point, `nargs` is set to 1, and since the execution is transferred to that entry point, the remaining arguments serve only as filler values and are never used by the invoked payload. In other words, GoFlateLoader is not issuing a `syscall` at all, but abusing `syscall.Syscall` as a generic call gate to hand control to the payload.

It is worth emphasizing that GoFlateLoader consistently sets these arguments to the hardcoded values 1, 2, 3, and 4, which is unusual enough to stand out as a potential detection opportunity (combined with other patterns).

```
(call resolve_ptr imports
mov eax, [esp+000072]
mov ecx, [eax+12]
mov ecx, [esp+1d0]
add ecx, ecx
mov [esp], ecx
mov dword ptr [esp+4], 1
mov dword ptr [esp+8], 2
mov dword ptr [esp+12], 3
mov dword ptr [esp+16], 4
call external_resolve)
```

Figure 3: GoFlateLoader transferring execution to the final payload's entry point via `syscall.Syscall` with dummy arguments.

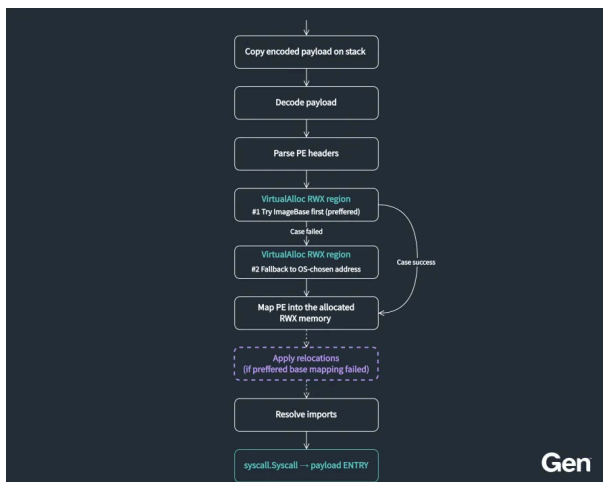


Figure 4: GoFlateLoader's execution flow.

The loader also carries substantial junk/decoy code, which varies across builds and serves no purpose other than padding the control flow and complicating static analysis.

Inflated PE Overlay

GoFlateLoader's defining characteristic is its massive size (typically 700-950 MB), caused by an artificially inflated PE overlay appended to the original loader.

Name	Ext	Size	Attr
[res_raw]		<DIR>	----
[resource]		<DIR>	----
[Certificate]		2,176	----
[Imports]	txt	0	----
[no-PE-Overlay]		2,348,544	----
[PE-Header]		1,024	----
[PE-Overlay]		861,933,538	----
[Relocations]	txt	0	----
section [.data]	02	87,040	----
section [.idata]	03	1,024	----
section [.rdata]	01	1,368,576	----
section [.reloc]	04	28,672	----
section [.rsrc]	06	232,448	----
section [.symtab]	05	97,280	----
section [.text]	00	532,480	----

Figure 5: Structure of a GoFlateLoader sample highlighting a massive PE overlay.

This is a deliberate design choice intended to bypass detection, as excessively large executable files may pose challenges for AVs and EDRs, primarily due to performance and resource constraints. As a result, to preserve performance and avoid noticeable impact to end users, these solutions often enforce practical file-size limits for deep scanning and/or emulation.

The same logic extends to automated analysis pipelines – bloated files frequently cause timeouts during heuristic analysis and effectively prevent the sample from being uploaded to cloud-based sandboxes or threat intelligence platforms, which enforce their own strict upload limits to manage bandwidth and storage costs. A prime example is VirusTotal, which enforces a strict 650 MB upload limit. The fact that GoFlateLoader consistently sits just above this threshold strongly suggests that VirusTotal is one of the platforms it specifically aims to evade.

In most cases, GoFlateLoader's PE overlays are filled with null bytes, though we have also observed builds with random padding.

[PE-Overlay] x																	0123456789ABCDEF
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
0000h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0010h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0020h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0030h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0040h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0050h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0060h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0070h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0080h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0090h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00A0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00B0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00C0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Figure 6: GoFlateLoader's PE overlay filled with null bytes.

Crucially, adding such a PE overlay comes at virtually no distribution cost for attackers, as the inflated data compresses to a fraction of its original size (assuming constant padding).

Delivered Final Payloads

The final payloads distributed through GoFlateLoader are predominantly information stealers, with the most prevalent currently being Amatera, [Remus](#), and Lumma. Beyond these, we have also observed it distributing Vidar, StealC, and SvitStealer.

GoFlateLoader comes in both x86 and x86-64 variants, with each variant matched to the architecture of the payload it is intended to execute.

As for how GoFlateLoader binaries are distributed, we have observed at least two notable delivery paths. The first is through supposedly cracked software. The second involves the payload being served via [the malicious TDS recently described by Check Point Research](#), which redirects users to a landing page that offers a downloadable archive and separately displays the password required to extract it. This approach provides several advantages for the threat actor. First, compression significantly reduces the binary size due to the constant-padded PE overlay. Second, because the archive password is displayed separately on the landing page, it makes it harder for AV and EDR solutions to obtain it and decrypt the archive. As a result, the payload may remain concealed until the victim manually extracts it.

Summary

GoFlateLoader is a reminder that prevalence and sophistication are not the same thing. At its core, it is a rather simple manual in-memory PE loader written in Go, with no novel anti-analysis tricks under the hood. Whatever evasion it achieves comes not from obfuscation or other anti-analysis tricks but from its size – the appended PE overlay is built to slip past size-constrained scanners and automated analysis pipelines. Yet, despite its low level of sophistication, GoFlateLoader remains an active and relevant threat, used to deliver a wide range of prevalent information stealers.

Gen products protect our users against both GoFlateLoader and the payloads it delivers.

Indicators of Compromise (IoCs)

Note: The files listed in the IoCs that are not archives all exceed 650 MB in size and are therefore not available on VirusTotal.

b88c5744975d2abb447aecc6c090fee9f8580413f4612eecdc6ed1973e8a1739 (password-protected archive containing GoFlateLoader x64 variant loading Remus; pwd: 1234)

ed5ae7f36453c5a23e9868a5729d67e0549a11f6dea54f5f52d654a8f51d4902 (archive containing GoFlateLoader x64 variant loading Remus)

841c9297cb8a2e0ff89433d13c05bfc760eb2e98e251cb8fa785d2ad7cbac05f (archive containing GoFlateLoader x86 variant loading Amatera)

ece7c48eb411b24f26762ede83badb4a644c41d5777129381ac2541804d64fc2 (archive containing GoFlateLoader x86 variant loading Lumma)

421ce2d2f49c23bbe9f60ef3b9cd38d7eb912ce02e56a61837656210069bd9e2 (archive containing GoFlateLoader x64 variant loading Vidar)

121c2dc793b3873f75a29ec02241f94136de19c049382a50a50d0d5b99507073
(GoFlateLoader x64 variant loading StealC)

2415db5081cec9bfd14ad6da1a66169fd96f13a49010c319a73d1ed6fafd4efa (GoFlateLoader x64 variant loading Vidar)

d9917ade3b4c125a95b5d3e6343cde26145dfbf569bd7e2a843fd0c6fc8ddc28 (GoFlateLoader x64 variant loading Remus)

4cf6893756f441522b94b36f10e5de0e47aead4743f95c51650746d1ecf97e3d (GoFlateLoader x64 variant loading SvitStealer)

8b89d6c9152d3aab97aadd515ecb69ca72654db2f25425759ba4b646853d737d
(GoFlateLoader x86 variant loading Lumma)

90ce4ff9da23ac150da0a8e17930cab1e369aa349fdc1b65691b70369145664a
(GoFlateLoader x86 variant loading Amatera)

More on this topic

[Fake hiring pages abuse FIFA and other major brands to steal work credentials – From Research](#)



[Who's Really Using Your Home Internet Connection? – From Research](#)



[When Hotel Scams Know Your Booking: 350 Compromised Accommodations Across 50 Countries – From Research](#)

