gomodjail: library sandboxing for Go modules

Akihiro Suda, NTT

https://github.com/AkihiroSuda/gomodjail

Overview

- Open source is under attack
- Malicious libraries everywhere
- gomodjail puts untrusted Go modules into the jail

```
require (
    example.com/module v1.0.0 // gomodjail:confined
)
```

Just add a directive comment in go.mod,
and run the program with gomodjail

Demo

Background

Open source is under attack

NTT

- Nowadays it is practically impossible to write software without depending on third-party libraries

- **Docker CLI v29**: 94 dependencies
- **Docker daemon v29**: 239 dependencies
- **Kubernetes v1.35**: 208 dependencies

- Bad actors have been trying to inject malicious dependencies

xz/liblzma backdoor incident (2024)

NTT

- CVE-2024-3094: a backdoor was injected to xz (liblzma) by a maintainer (not by the original author)
- The culprit had been making harmless contributions for 2+ years; suddenly injected the backdoor and faded out from the community
- Even widely adopted libraries can be compromised
- **Even maintainers cannot be trusted** 😞
- This case was not about Go, but similar attacks could happen with Go

https://research.swtch.com/xz-timeline

Fake Go modules (2025-)

- In the spring of 2025, hundreds of fake Go modules were published on GitHub
- The repos looked genuine but contained malicious `wget|bash` code
- For some repos, the numbers of the GitHub stars ⭐ even exceeded those of the genuine repos
- Most repos seem now banned, but some of them are still alive

https://socket.dev/blog/wget-to-wipeout-malicious-go-modules-fetch-destructive-payload

“Slopsquatting”

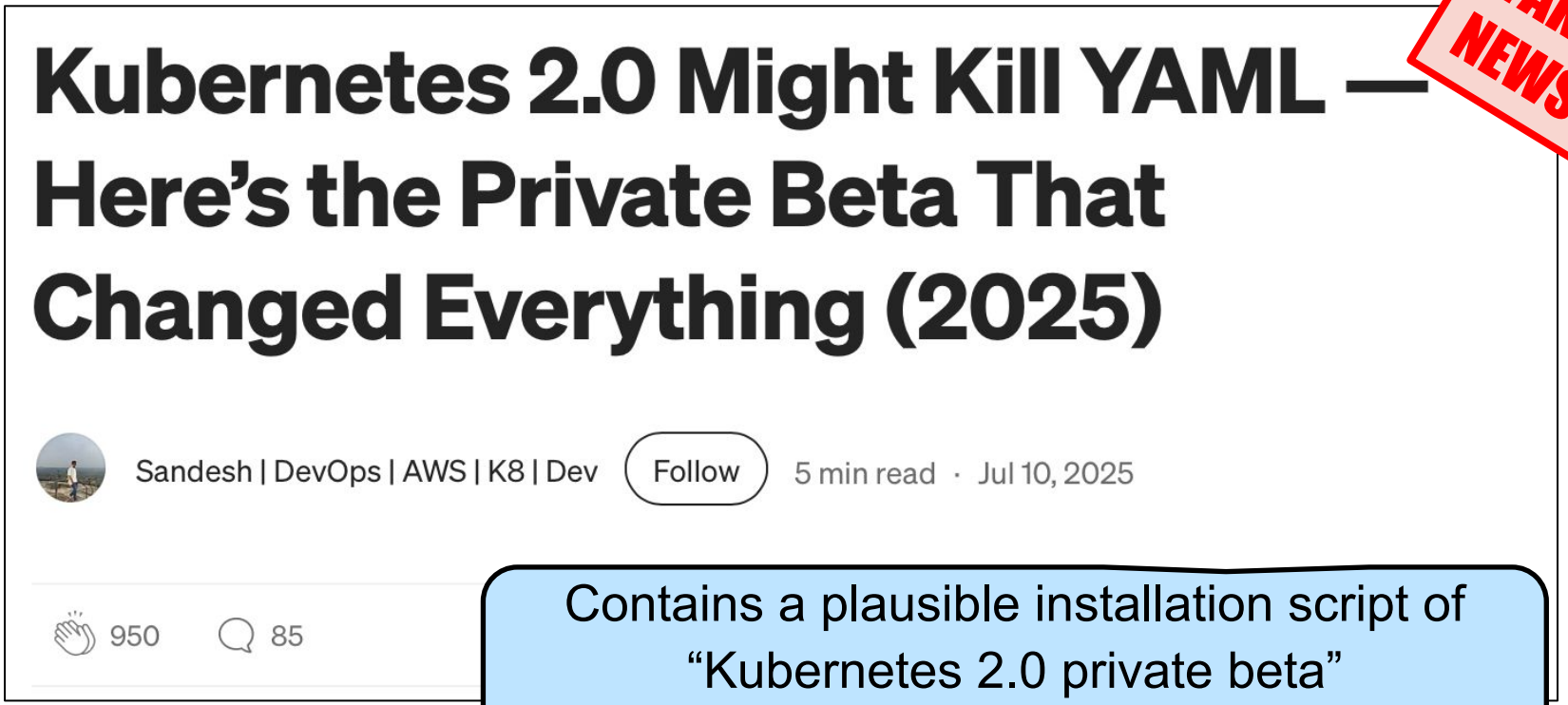
- AI coding agents may hallucinate to inject malicious dependencies with plausible package names

```
# 🤖 Here is a code to store passwords securely...
import securehashlib
```

- Even when an LLM itself doesn’t hallucinate, it can be still deceived by fake sites on the Internet via the web search tool
  - Some sites are just slops
  - Some sites are published with malice

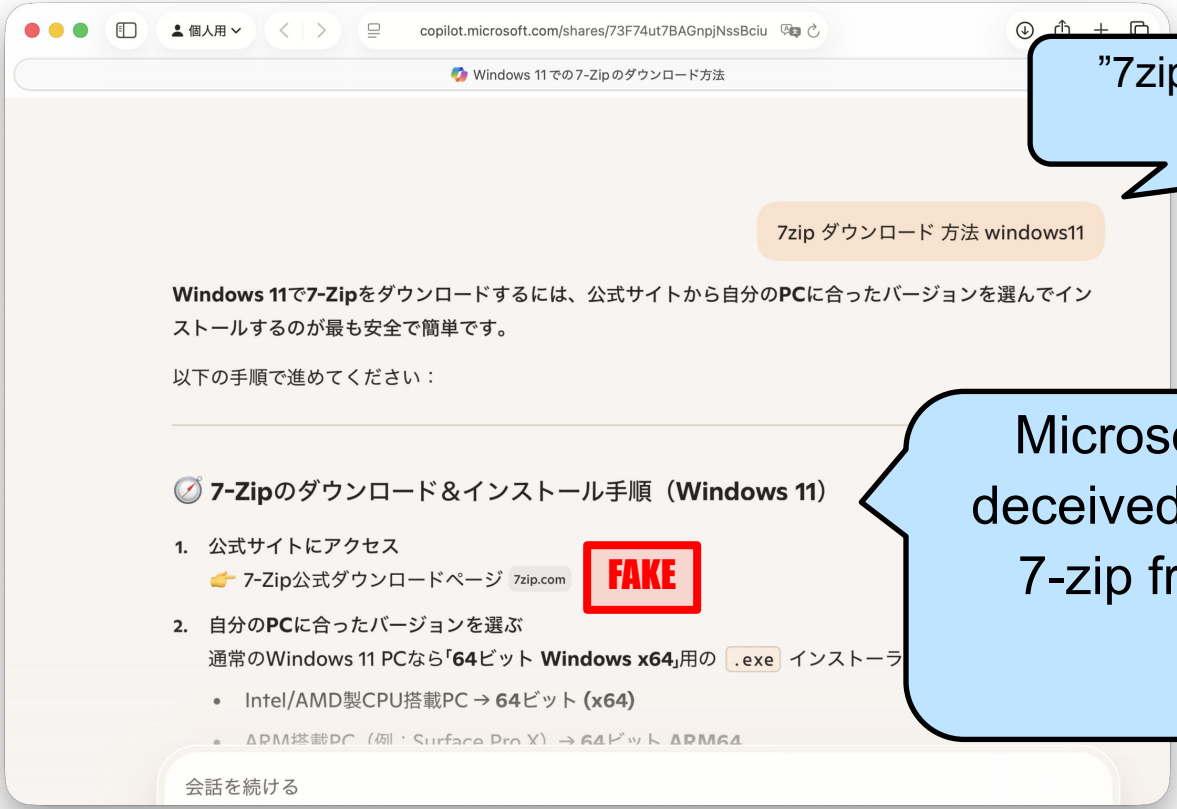
“Importing Phantoms: Measuring LLM Package Hallucination Vulnerabilities” (A.Krishna et al., 2025)
https://arxiv.org/html/2501.19012v1

“Slopsquatting”



Contains a plausible installation script of
“Kubernetes 2.0 private beta”
that does not even exist

“Slopsquatting”



”7zip download method windows11” (Japanese)

Microsoft Copilot was actually deceived to suggest downloading 7-zip from a fake “official” site

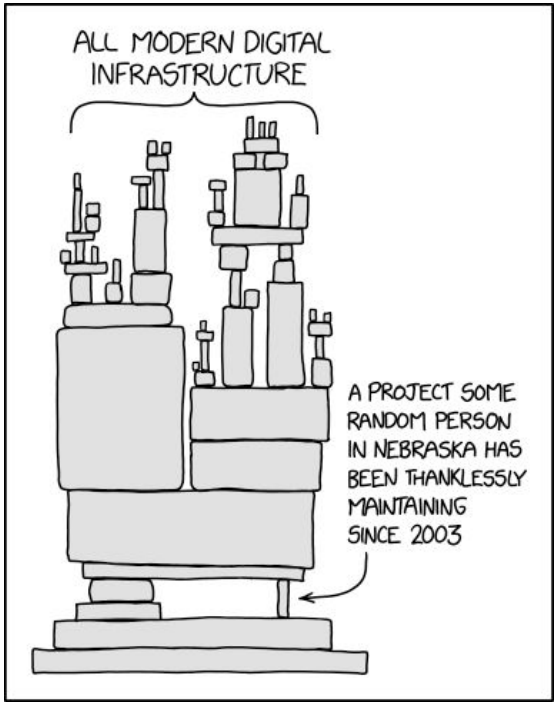
Fake: 7zip[.]com

Real: 7-zip.org

The chat session was shared by a Twitter user @longer_n (Jan 22, 2026)
https://x.com/longer_n/status/2014335971505123760

Review all the dependencies!

- Everybody knows that they should do so
- But who can *actually* do?



https://xkcd.com/2347/

Library sandboxing

- A genre of technology to confine capabilities of a library

- Similar to containers (as in Docker containers), but in finer granularity
  - process granularity vs library granularity

- Several designs are possible
  - Seccomp
  - WebAssembly
  - Compilation-time assertion

Similar work: Google Sandboxed API

NTT

- Wraps C/C++ library calls using Linux seccomp and namespaces
- A function caller and a callee are executed in separate processes that communicate with each other via IPC
- Adoption does not seem straightforward, as it requires massively modifying the source codes and the build scripts
- Does not seem widely adopted outside Google

https://developers.google.com/sandboxed-api/

Similar work: RLBox

- Wraps C library calls using WebAssembly
- Adopted in Firefox since 2021
  - libgraphite, libogg, libhunspell, libexpat, libwoff2, libsoundtouch, …
- Adoption does not seem straightforward either
  - *“On average, sandboxing a library takes only a few days”*

"Retrofitting Fine Grain Isolation in the Firefox Renderer" (S. Narayan et al., 2020)
https://www.usenix.org/conference/usenixsecurity20/presentation/narayan

Similar work: isolated-vm

- Wraps nodejs library calls using another instance of v8 engine
- Adopted by Fly, Algolia, Tripadvisor, etc.
- Adoption is relatively wide, but not really easy to get started
  - Several host functions have to be manually exposed to an isolated instance of v8 engine

https://github.com/laverdet/isolated-vm

Similar work: depguard

- A linter to enforce an allowlist/denylist of Go modules
- Widely adopted via golangci-lint
- Too strict: no way to allow a module with limited capabilities

- Yet another similar linter: gomodguard
  - Not to be confused with my *gomodjail*

https://github.com/OpenPeeDeeP/depguard
https://github.com/ryancurrah/gomodguard

Goal: make it easy to get started

- No modification to the source code
- Little modification to the build script etc.
- Caveats may apply, as a trade-off for simplicity

gomodjail

gomodjail: library sandbox for Go

- Focuses on simplicity
- Applicable in just two steps
- **Step 1**: Add `gomodjail:confined` comment to go.mod

```
require (
    example.com/module v1.0.0 // gomodjail:confined
)
```

Confined modules are disallowed to open or execute files

- **Step 2**: Run the target program with `gomodjail run`

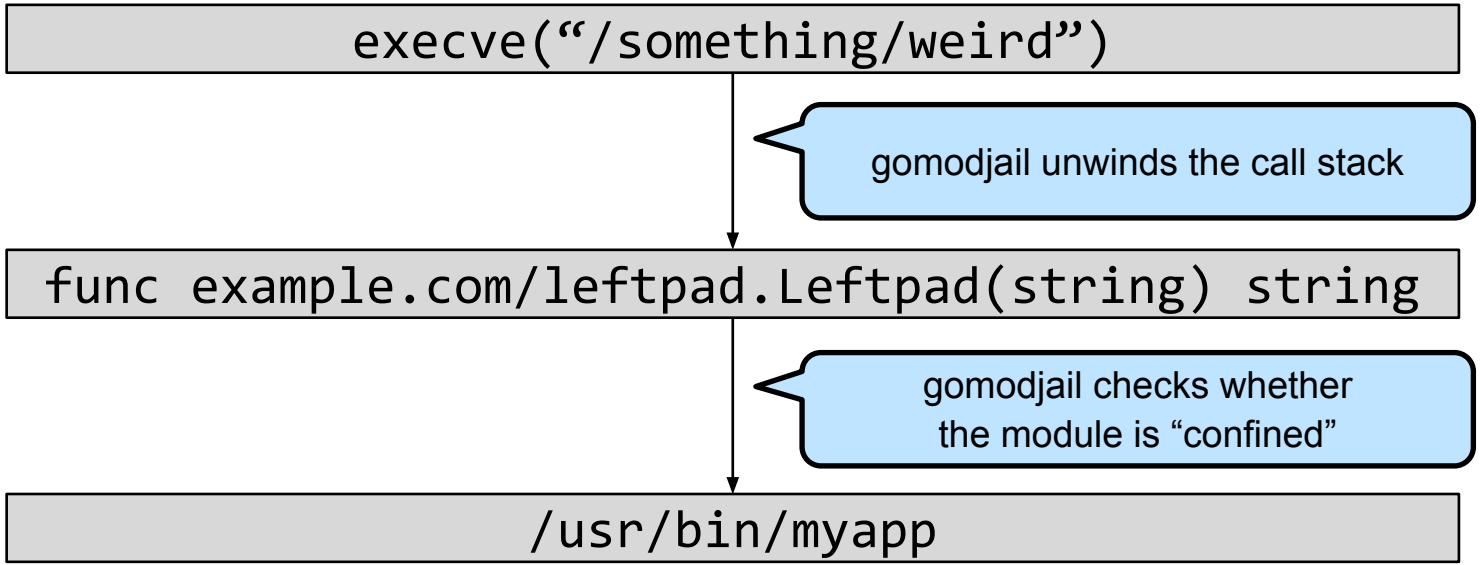
```
gomodjail run --go-mod=go.mod -- ./prog
```


https://github.com/AkihiroSuda/gomodjail

How it works

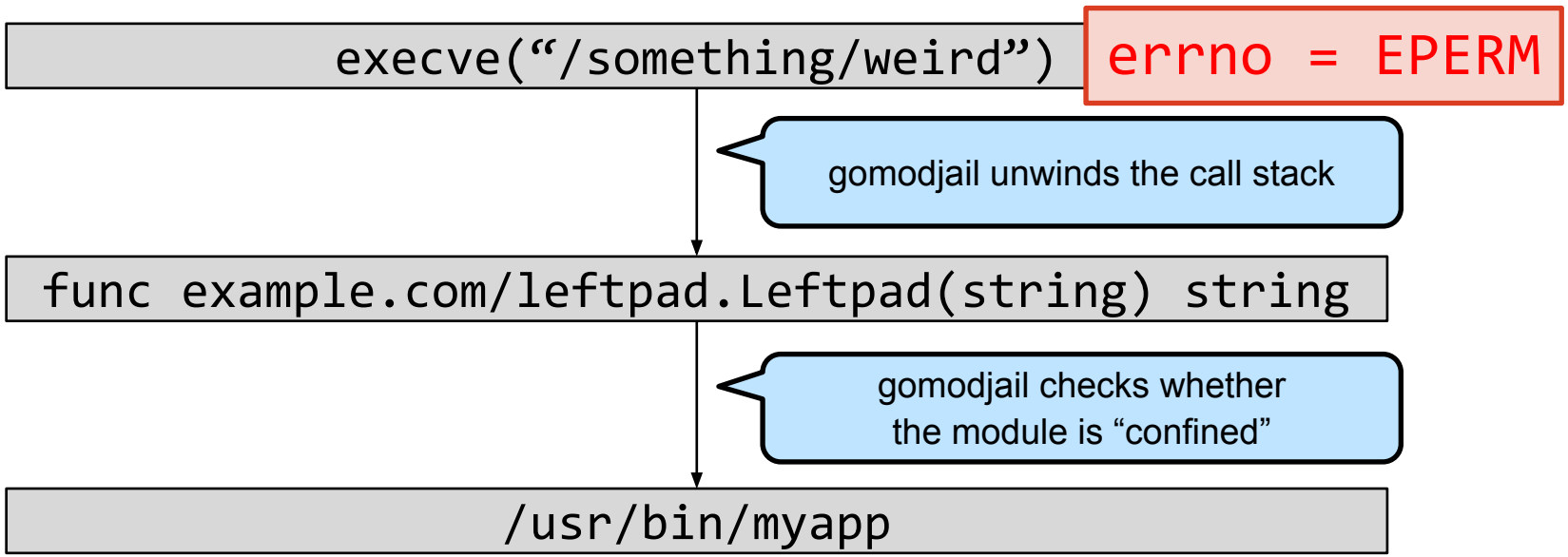
- Hooks “dangerous” syscalls
  - **File**: `open()`, `creat()`, `mkdir()`, `unlink()`, ...
  - **Process**: `execve()`, `posix_spawn()`, ...
  - **Network**: `connect()`, `listen()`, ...
- Unwinds the call stack to identify the Go function that invoked the syscall
  - The `.gopclntab` ELF section contains symbols, even when stripped
- If the function belongs to a confined module, blocks the syscall

How it works




```
require (
    example.com/leftpad v1.0.0 // gomodjail:confined
)
```

How it works




```
require (
    example.com/leftpad v1.0.0 // gomodjail:confined
)
```

Supported GOOS and GOARCH

- linux/amd64
- linux/arm64
- darwin/amd64 (no support for stripped binaries)
- darwin/arm64

Implementation for Linux

- `SECCOMP_RET_TRACE` is used to hook syscalls
- `PTRACE_PEEKDATA` is used to unwind the call stack

- **FAQ**: why not `SECCOMP_USER_NOTIF` ?
  - › Because it lacks the frame pointer information
- **FAQ**: why not use Landlock ?
  - › Because no way to associate Go modules with threads to be landlocked

Implementation for macOS

- `DYLD_INSERT_LIBRARIES` to hook syscalls
  - Equates to `LD_PRELOAD` on glibc/Linux
  - No need to support static binaries on macOS
    - System calls are always invoked via `libSystem`

- `libgomodjail_hook_darwin.dylib` unwinds the stack using `_dyld_get_image_vmaddr_slide()`
  - Unlike on Linux, unwinding is *quite* complicated as the call stack switches when calling `libSystem`
    - Fetches `struct g` that represents the goroutine (See the next slide)
    - Parses `g->m.libcallsp` , `g->m.libcallpc` etc.

Implementation for macOS

- How to fetch the pointer to the `struct g`
  - Fetch the value of the `runtime.tls_g` variable (see below)
  - Read the TLS register (ARM: `tpidrro_el0`, Intel: `%gs:0x30`)
  - `g` is located at `tls_base + runtime.tls_g`

- How to fetch the value of the `runtime.tls_g`
  - Non-stripped: symbols of global variables are available
    - Just find the pointer to the `runtime.tls_g` variable in `__gosymtab`
  - Stripped: symbols of functions are still available
    - Find the pointer to `func runtime.load_g()` in `__gopclntab`
    - Call the function
    - `runtime.tls_g` is “leaked” into a temporary register `x27`

Benchmark

- Applied gomodjail to Docker CLI (not to daemon)
- **Workload**: `docker run --rm hello-world`

	Normal	With gomodjail
Linux	119.5 ± 9.2 ms	124.6 ± 7.6 ms (**overhead**: 4.27%)
macOS	124.4 ± 6.8 ms	135.7 ± 7.1 ms (**overhead**: 9.08%)

Measured mean ± σ, 100 times
gomodjail v0.3.0, Docker v29.1.5, Ubuntu 24.04.3, macOS 26.2 (both ARM64)

Caveats

- Not applicable to a Go binary built by non-trustworthy thirdparty
  - › The symbol information might be faked
- No isolation of file descriptors across modules
  - › A confined module can still read/write an existing FD
- The target binary must not be replaced during execution

Caveats

- Not applicable to a module that imports:
  - `unsafe`
  - `reflect`
  - `plugin`
  - `//go:linkname`
  - C
  - asm

Static analyzer to detect incompatible imports

```
$ gomodjail analyze ./...
golang.org/x/sys@v0.39.0/execabs/execabs.go:22:2:
  unsafe or cgo-related package imported: "unsafe"
```

Adoption status of gomodjail

NTT

- Adopted in several projects under my own maintainership
  - **Lima** (https://lima-vm.io)
    - Linux VM, optimized for containers and AI agents
  - **nerdctl** (https://github.com/containerd/nerdctl)
    - contaiNERD CTL
  - **Alcoholless Homebrew** (https://github.com/AkihiroSuda/alcless)
    - Homebrew with User ID isolation for security

Lima

nerdctl

- Looking for wider adoptions

Packing gomodjail with target program for easier distribution

```
$ gomodjail pack --go-mod=go.mod ./program
$ ./program.gomodjail
```

Side topics

Future work: compilation-time mode

- Run-time sandboxing is fragile and has lots of caveats

- The future plan is to introduce the compilation-time mode
  - Would lint/translate the source code to control accessing dangerous functions
  - Any chance to get it accepted in the upstream Go compiler?
  - Maybe reuse AspectGo (2016) ? https://github.com/AkihiroSuda/aspectgo
    - Aspect-Oriented Programming framework for Go
    - Similar to AspectJ (Java)

Future work: support other languages

- The threat to the software supply chain is not specific to Go
- Just began with Go because it is the primary language in the Cloud Native communities
- Other language ecosystems need improvements in supply chain security too

Another approach to prevent supply chain attacks

- **gosocialcheck**: social reputation checker for Go modules
  - Checks whether dependencies have been already adopted by matured projects (CNCF Graduated projects)
    - **Caveat**: even Graduated maintainers may overlook malicious packages
  - Future version may add more reputation checks
    - e.g., check OpenSSF Scorecard?
    - GitHub Stars will never be counted, as they can be faked up with bot accounts
  - Social approaches are complementary to tech approaches such as gomodjail

https://github.com/AkihiroSuda/gosocialcheck

Recap

- Open source is under attack
- Malicious libraries everywhere
- gomodjail puts untrusted Go modules into the jail

```
require (
    example.com/module v1.0.0 // gomodjail:confined
)
```

Just add a directive comment in go.mod,
and run the program with gomodjail


NTT