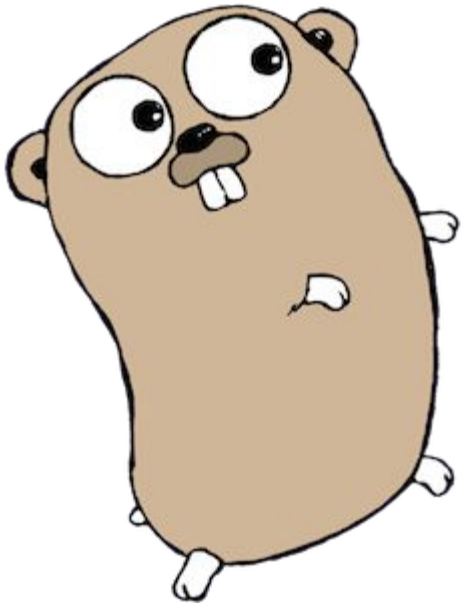


Supply Chain Security with Go

Michael Stapelberg
<stapelberg@golang.org>

GPN, 2024-May-31

Agenda

- Part 1: Keeping your Go build environment up to date
- Part 2: Is my program vulnerable?
- Part 3: Can we trust Go modules? [proxy, sumdb]
- Part 4: Best practices: separation, least privilege, sandboxing
- Part 5: supply chain minimalism: gokrazy

Context / Lens

- You become aware of a security vulnerability! What now?

- This talk tries to answer that question for various common scenarios.

Part 1: Keeping your Go environment up to date

Setting the scene

- POV: developer or admin — you're responsible for running a Go program

- We're using Go modules (introduced in Go 1.11 in 2018),
  meaning we have a go.mod file like this (`go mod init`, `go mod tidy`):

```
module github.com/robustirc/robustirc

require github.com/google/renameio/v2 v2.0.0
// …
```

Is my build environment up to date?

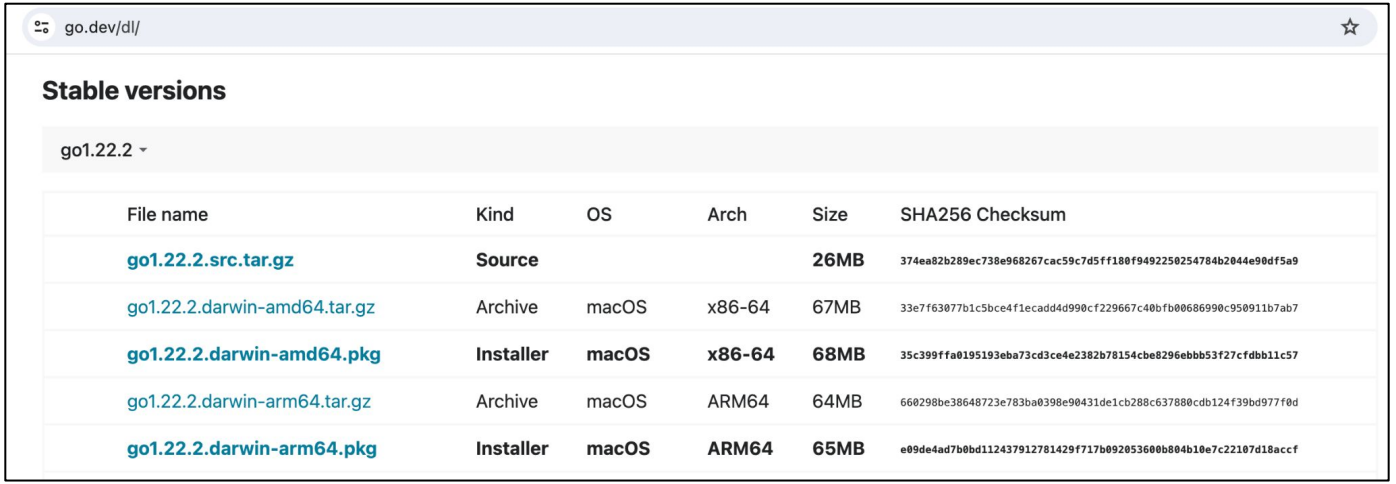
- Compare your version with go.dev/dl

```
% go version
go version go1.22.0 darwin/arm64
```

Is my build environment up to date?

- Compare your version with [go.dev/dl](go.dev/dl)

```
% go version
go version go1.22.0 darwin/arm64
```



Can you use the latest toolchain?

- Mental model: the latest Go version (1.22.2) is the best implementation of a certain Go language version (1.22)

- But what if your program was developed for an older Go version?

- Go 1.21 improved both backward and forward compatibility!

Backward compatibility: GODEBUG

- Changes that risk breaking programs are tied to a `GODEBUG` setting

  e.g. `GODEBUG=x509sha1=1` if you need insecure SHA-1 hashes

- `GODEBUG` defaults are tied to the go.mod language version line:

  Go 1.22 introduced `gotypesalias=0`

  Go 1.23 will change the default to `gotypesalias=1`

- → go.dev/blog/compat

Forward compat: build with a newer Go toolchain

- Since Go 1.21 (August 2023), main modules can require minimum toolchain versions (if they depend on a specific fix, for example) and those toolchains are downloaded on demand (adjust with `GOTOOLCHAIN` env var):

```
% go mod edit -toolchain go1.22.2

% go install ./cmd/scan2drive
go: downloading go1.22.2 (darwin/arm64)

% go version -m =scan2drive
/Users/michael/go/bin/scan2drive: go1.22.2
[…]
```

Forward compat: build with a newer Go toolchain

- Can you trust these toolchains? What's inside?

- Go 1.21 toolchain and newer versions are perfectly reproducible:

  now built without cgo, built with `-trimpath`, etc.

  → details at go.dev/blog/rebuild

  → daily reports at go.dev/rebuild

- → go.dev/blog/toolchain

Recap: go.mod file

```
module github.com/robustirc/robustirc

go 1.21

toolchain go1.22.2

require github.com/google/renameio/v2 v2.0.0

// …
```

Go language version

minimum Go toolchain version (optional)

Where can you get informed about new releases?

- Subscribe to the golang-announce mailing list:

  Go is released in February and August of each year

  pre-announcement 3-7 days before security fixes (per the Security Policy)

Hello gophers,

We have just released Go versions 1.22.2 and 1.21.9, minor point releases.

These minor releases include 1 security fixes following the security policy:

- http2: close connections when receiving too many headers

  Maintaining HPACK state requires that we parse and process all HEADERS and CONTINUATION frames on a connection. When a request's headers exceed MaxHeaderBytes, we don't allocate memory to store the excess headers but we do parse them. This permits an attacker to cause an HTTP/2 endpoint to read arbitrary amounts of header data, all associated with a request which is going to be rejected. These headers can include Huffman-encoded data which is significantly more expensive for the receiver to decode than for an attacker to send.

  Set a limit on the amount of excess header frames we will process before closing a connection.

  Thanks to Bartek Nowotarski (https://nowotarski.info/) for reporting this issue.

  This is CVE-2023-45288 and Go issue https://go.dev/issue/65051.

Part 2: Is my program vulnerable?

Is my program vulnerable?

- Since Go 1.18 (March 2022), go embeds information about the build:

```
% go version -m ./scan2drive
./scan2drive: go1.22.2
    path    github.com/stapelberg/scan2drive/cmd/scan2drive
    mod github.com/stapelberg/scan2drive (devel)
    dep github.com/gorilla/sessions  v1.2.0
    [...]
    build   vcs=git
    build   vcs.revision=7e8a2ca85438f0bcc43603bde2337fd0c644b9d2
    build   vcs.time=2023-03-07T07:51:30Z
    build   vcs.modified=false
```

Is my program vulnerable?

- buildinfo gives us a chance to locate the corresponding source

- govulncheck does static analysis (to reduce spurious reports):

  `go install golang.org/x/vuln/cmd/govulncheck@latest`

- gorilla/sessions report: https://pkg.go.dev/vuln/GO-2024-2730

govulncheck example

```
% govulncheck ./...

Scanning for dependencies with known vulnerabilities...
Found 1 known vulnerability.

Vulnerability #1: GO-2024-2730
    Directory traversal in FilesystemStore in github.com/gorilla/sessions
  More info: https://pkg.go.dev/vuln/GO-2024-2730
  Module: github.com/gorilla/sessions
    Found in: github.com/gorilla/sessions@v1.2.0
    Fixed in: N/A
    Example traces found:
      #1: internal/webui/web.go:93:30: webui.UI.indexHandler calls
sessions.FilesystemStore.Get
      #2: internal/webui/web.go:76:25: webui.UI.constantsHandler calls
sessions.Session.Save
```

Mitigating vulnerabilities (1): updating

- Easiest way: `go get` to update to a fixed version

- Only needed in the main module!

  Other languages require updating versions in your dependency modules,

  but Go uses Minimum Version Selection

  → main module's version effectively overrides dependencies' versions

- → research.swtch.com/vgo-mvs

Mitigating vulnerabilities (2): patching

- Create a writable working copy of your dependency:

```
git clone https://github.com/google/renameio
```

- Add a replace directive to your go.mod file to pick up this directory:

```
replace github.com/google/renameio => /home/michael/renameio
```

Mitigating vulnerabilities (2): patching

- Make your changes, verify your binary picked them up:

```
% go version -m ./scan2drive
/home/michael/go/bin/scan2drive: go1.22.2

    path github.com/stapelberg/scan2drive/cmd/scan2drive
    mod  github.com/stapelberg/scan2drive (devel)
    dep  github.com/golang/protobuf  v1.5.2
h1:ROPKBNFfQgOUMifHyP+KYbvpjbdoFNs+aK7DXlji0Tw=
    dep  github.com/stapelberg/airscan    v0.0.0-20230123183513-bed4bafc7ef4
    =>   /home/michael/go/src/github.com/stapelberg/airscan (devel)

    dep  go.opencensus.io   v0.22.4  h1:LYy1Hy3MJdrCdMwwzxA/dRok4ejH+RwNGbuoD9fCjto=
    [...]
```

Mitigating vulnerabilities (3): removing

- If not possible, maybe you don't ***absolutely*** need the feature right now?

  → Open your editor, comment out the code.

  → Verify: Does the module disappear from go.mod after `go mod tidy`?

Easy & Fast Rollouts means Fast Mitigation!

- I have a `deploy-all.sh` script, which consists of lines like these:

```
(cd smtp-dkim-proxy && make push)
(cd authelia && make push)
...
```

- Each project has the `push` Makefile target defined like so:

```
CGO_ENABLED=0 GOOS=linux GOARCH=amd64 \
    go build -o bin/proxy -trimpath ./cmd/proxy && \
    rsync -rav bin exo1:/srv/ && \
    ssh exo1 systemctl restart proxy
```

Part 3: Can we trust Go modules?

Go Modules

- Go modules are always safe to download

  The go tool never runs code when downloading a module

  Module authors run `go generate` and submit the resulting code to git

- Go modules are always generated from source

  → not maintainer-provided tarballs

- → go.dev/ref/mod

Go Module Proxy

- published module versions are immutable in the go module proxy

- Enabled by default if you install from go.dev

  Some Linux distributions may disable the proxy by default (slow!)

- Very clear one-page privacy policy at proxy.golang.org/privacy

  Working with private modules? set `GOPRIVATE=` env var

  Better: run your own (company-internal) proxy

go.sum / Go Checksum Database

- go.sum stores a cryptographic hash of the module contents on first use, which the go tool verifies when later downloading a module:

```
github.com/google/renameio/v2 v2.0.0 h1:UifI23ZTGY8Tt29JbYFiuyIU3eX+RNFtUwefq9qAhxg=
github.com/google/renameio/v2 v2.0.0/go.mod h1:BtmJXm5YlszgC+TD4HOEEUFgkJP3nLxehU6hfe7jRt4=
```

- Checksum database stores these checksums centrally in a verifiable way
  → allows safely using an otherwise untrusted proxy

- → go.dev/blog/module-mirror-launch

Part 4: Best practices: sandboxing, separation, least privilege

Best practice: separation

- Split out untrusted clusters of dependencies into their own process, or container, or VM, or machine, …

- For example: if you move QR code generation into its own process, the module authors of the QR code module (and dependencies!) cannot run arbitrary code in your main process

Best practice: least privilege / sandboxing

- Run different services under their own dedicated user account,
  not everything as root or any other shared user

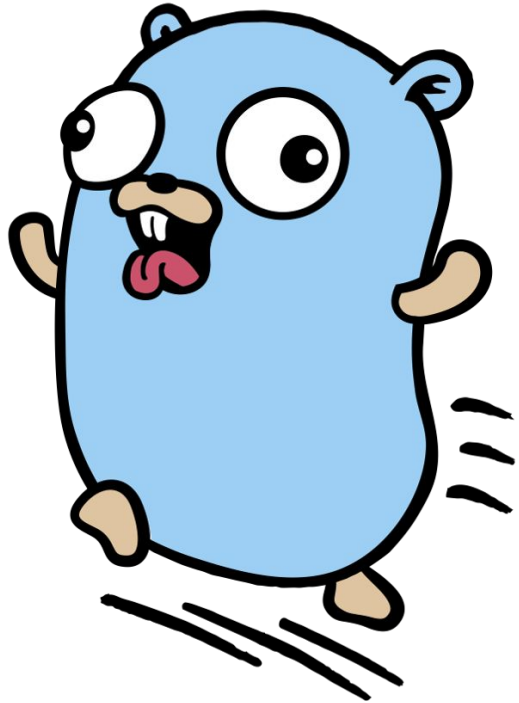
- When deploying as a systemd service, enable syscall filtering (seccomp)
  → `systemd-analyze security`
  → example hardened systemd `.service` file for a Go service:
  https://github.com/stapelberg/qrbill/blob/master/systemd/qrbill.service

Best practice: immutability

- make as much as possible immutable / read-only

- When deploying with systemd: use `ProtectSystem=strict`

Part 5: Supply chain minimalism: the gokrazy appliance platform



gokrazy

- from-scratch appliance platform built entirely in Go

  Linux kernel + (firmware) + gokrazy Go userland + <your app(s)>

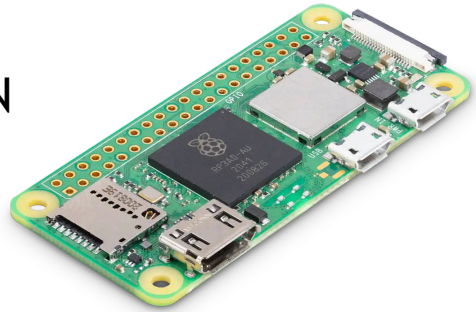
  no C userland or runtime environment! no glibc, OpenSSL, xz, …

- `gok new` # create ~/gokrazy/hello/config.json

  `gok overwrite --full /dev/sdx` # write SD card for Raspberry Pi

- → gokrazy.org/quickstart

gokrazy: how far will it get you?



- Example use-case: want to remotely trigger Wake-on-LAN

- Deploy on a Raspberry Pi Zero 2 W, which runs at ≈1W!

  Just need a free USB plug somewhere, the Pi Zero 2 W has WiFi

  (gokrazy supports encrypted WiFi without `wpa_supplicant` or similar)

- Use Tailscale to make the service reachable over the internet!

  mesh VPN, handles authentication, NAT traversal, etc.

gokrazy: what can you build this way?

- Document management: I scan physical mail with scan2drive

- Smart home: instead of using Home Assistant or Node RED,
  I integrate my smart home components with a few lines of Go
  (I also replaced my HomeMatic controller with a Go one)

- Internet router: router7.org is a small home internet router

gokrazy: supply chain

- github.com/gokrazy/kernel — auto-updated Linux upstream kernel
  github.com/gokrazy/firmware — auto-updated Raspberry Pi firmware files

- `gok get` wraps `go get`, all gokrazy programs are Go modules
  can use `govulncheck` for security analysis

- `gok update` updates an instance over HTTP(S)
  automate it from a cron job or similar

Conclusion

- Supply chain management can be tedious, but Go makes it easy enough
- Think about your supply chain and how you could make it smaller
- Minimalist solutions like gokrazy can help for select use-cases

  Makes me sleep better at night, and I hope it gives you all some peace, too

Thank you for your attention!

- More details in the Go blog: go.dev/blog

  Details about gokrazy are at gokrazy.org

- Questions? Talk to me after the presentation :)

- Give me feedback on this presentation!