

# Build better go release binaries



# About me

- Linux OS Distribution developer
  - Debian, Ubuntu, Clear Linux, Wolfi
  - SysV init, Upstart, Systemd, Grub, Linux, OpenSSL ...
  - Principle Engineer at Chainguard
  - Focused on building production-grade container images
- 
- Linux focused, broad application, tested at scale on x86\_64 & arm64
  - Production builds
  - Transferable to other arches / OSes

## go build -ldflags -w

- Debug information is on by default
- It can be quite large in size
- Often unused in production

### Resolution:

- go build -ldflags -w

### Alternative

- Strip & Store in debuginfod server: strip --strip-debug

# go build -trimpath

- Full file paths leak into binary
- Takes up space & lead to non-reproducible builds

Resolution:

- go build -trimpath

Caveat:

- Removes -X importpath.name=value definitions from buildinfo
- Upstream bug report in progress to resolve this

# go build -tags netgo,osusergo

- Default libc/NSS for DNS and username resolution
- Old hammer CGO\_ENABLED=0
- Can elect golang implementation & maintain CGO access
- Helps with boringcrypto, go-openssl, go-msft-fips

## Resolution:

- For containers / portable binaries: `go build -tags netgo,osusergo`
- For explicit Host os resolution: `CGO_ENABLED=1 go build`

# GOAMD64=v2 GOARM64=v8.0

- Production hardware is not 20 years old
- Optimisations available that can be used by default
- Depending on your deployments can use v3, v4 too

## Resolution:

- Environment variables: `GOAMD64=v2 GOARM64=v8.0`

## Caveat:

- Check your production hardware SKUs, some have broken v3 support

# go build -buildmode=pie

- Position Independent Code/ Executable can improve security
- Should use for dynamic binaries
- Security scanners / hardening-check report this

## Resolution:

- go build -buildmode=pie

## Alternative:

- Also available with external linker

```
go build -ldflags -X main.Version=$(git describe...)
```

- go build sets commit & date; and go install sets tag; but not both
- Do not use trimpath
- Check with go version -m & scanners look for it too

Solution:

```
$ go build -ldflags -X main.Version=$(git describe --tags --dirty --always --abbrev=12)
```

```
$ go version -m mybinary | grep -e main.Version
```

```
build      -ldflags="-X main.Version=1.19.0"
```



# CGO Hardening

- Accelerated code & C library access is convenient
- [OpenSSF Compiler Options Hardening Guide for C and C++](#)
- CFLAGS / CXXFLAGS ignored by go
- Distributions (rpmspec / dpkg-buildflags) typically do not set these

## Resolution:

- Use CGO\_CFLAGS CGO\_CXXFLAGS etc
- Use wrapper for gcc -specs with file
- Use clang .cfg automatically loaded config files

# Gcc spec file example

Gcc wrapper:

```
#!/bin/sh
```

```
exec /usr/bin/${0##*/} -specs "${GCC_SPEC_FILE:-openssf.spec}" "$@"
```

Reduced OpenSSF spec file for GCC-14:

```
*self_spec:
```

```
+ -O2 -fhardened -Wl,--as-needed,-O1,--sort-common,-z,noexecstack,  
-z,relro,-z,now -fno-delete-null-pointer-checks -fno-strict-overflow  
-fno-strict-aliasing -fno-omit-frame-pointer -mno-omit-leaf-frame-pointer
```

# [security] Vulnerability in golang.org/x/net

Hello gophers,

We have tagged version v0.33.0 of [golang.org/x/net](https://golang.org/x/net) in order to address a security issue.

`x/net/html`: non-linear parsing of case-insensitive content

Version v0.33.0 of [golang.org/x/net](https://golang.org/x/net) fixes a vulnerability in the [golang.org/x/net/html](https://golang.org/x/net/html) package which could cause a denial of service.

An attacker can craft an input to the Parse functions that would be processed non-linearly with respect to its length, resulting in extremely slow parsing.

Thanks to Guido Vranken for reporting this issue.

This is CVE-2024-45338 and Go issue <https://go.dev/issue/70906>.

Cheers,

Go Security team

# Keep symbols tables!

- [Govulncheck](#) in -mode=binary reports **module** level CVEs
- If binary has symbols tables, it reports **symbol** level CVEs

## Example from Wolfi

- Packages/Sub-packages found with x/net : 1065
- x/net/html Symbols found in binaries : 242
- Govulncheck : 13

This often removes need to upgrade, rebuild, re-release binaries

# Example coredns v1.11.1

```
$ go install -ldflags -s github.com/coredns/coredns@v1.11.1
```

```
$ govulncheck -mode=binary coredns
```

Your code is affected by **15** vulnerabilities from **7** modules and the Go standard library.

This scan found no other vulnerabilities in packages you import or modules you require.

# Example coredns v1.11.1

```
$ go install github.com/coredns/coredns@v1.11.1
```

```
$ govulncheck -mode=binary coredns
```

Your code is affected by **8** vulnerabilities from **4** modules and the Go standard library.

This scan **also found 2 vulnerabilities** in packages you import and **5** vulnerabilities in modules you require, **but your code doesn't appear to call these vulnerabilities.**

# How to keep symbols tables?

- Do **not** use: `go build -ldflags -s`
- Do **not** use: `strip --strip-all`
- Verify with: `go tool nm`

## Caveats:

- There is binary size penalty
- Worth it, if production binary is subject to security scans
- Check if your security scanners support this
- Reflection (urgh)

# Rince & repeat - rebuild & re-release

- Go toolchains have bug fixes & CVE fixes
- Dependencies have bug fixes & CVE fixes
- Continuously release binaries, or tag micro-point releases
- Same code built with new go toolchain can be safer

## Solution

- Bump “toolchain go1.21.3” in go.mod
- Go get dependencies in go.mod
- Tag a micro-point releases
- Automate low-key micro-point releases



# Thank you!

Will try to contribute this to OpenSSF Working Group

I have Chainguard Linky stickers!

Questions?

