GOLAB 2026



Securing Go for Production

Best practices, supply chain, testing

davide imola

davideimola.dev



ABOUT

Davide Imola

I build AI agents for security, and the tools I need along the way.

AT WORK	Tech Lead at RedCarbon **WE'RE HIRING**
BASED IN	Verona, Italy
COMMUNITY	Schrödinger Hat · Open Source Day
OFF THE CLOCK	bbq · jujutsu · board games · pokémon · comics and manga
GITHUB	davideimola
LINKEDIN	davideimola

Attacks now run at machine speed

80–90%

of a cyber espionage campaign,
run by an AI agent

ANTHROPIC · NOV 2025

27s

the fastest breakout ever observed,
from first access to lateral movement

CROWDSTRIKE GTR 2026

THIS SHIPPED

The bug you don't know you have

```
func canManageOrg(u User, orgID string) bool {
    if u.IsSuperAdmin {
        return true
    }
    if orgID == "" {
        return true
    }
    return u.OrgID == orgID
}
```

The compiler won’t save you. Your process might.

What we'll cover

01 **The code you write**
a real bug, and how we proved it

02 **The code you use**
dependencies, the checksum database, govulncheck

03 **Make it routine**
a checklist, CI, and skills for your agent

01 · THE CODE YOU WRITE

The code you write

Go is safe by default.
Your application isn't.

What the compiler can’t know

GO BUILD CHECKS

- Types
- Memory safety
- Bounds and nil, at runtime
- Data races, with `-race`

NOBODY CHECKS

- Who may read this resource BAC
- Where this request is going SSRF
- How much this request costs DOS

TRUST BOUNDARIES

The client chose this URL

THE CLAIM

Whatever the client typed.

THE ATTACK

Ask for `169.254.169.254`, get cloud credentials.

THE FIX

Allow-list the host. Deny the rest.

```go
func preview(w http.ResponseWriter, r *http.Request) {
    u, err := url.Parse(r.URL.Query().Get("url"))
    if err != nil || !allowed[u.Host] {
        w.WriteHeader(http.StatusForbidden)
        return
    }
    resp, err := client.Get(u.String())
    if err != nil {
        w.WriteHeader(http.StatusBadGateway)
        return
    }
    defer resp.Body.Close()
    io.Copy(w, resp.Body)
}
```

RESOURCE EXHAUSTION

Valid request. Unbounded cost.

THE REQUEST

Well-formed, just bigger than planned.

THE COST

`io.ReadAll` keeps it all in memory, until the process dies.

THE FIX

One line sets the limit.

```go
func upload(w http.ResponseWriter, r *http.Request) {
    // 1 MB, then Read fails
    r.Body = http.MaxBytesReader(w, r.Body, 1<<20)
    body, err := io.ReadAll(r.Body)
    if err != nil {
        w.WriteHeader(http.StatusBadRequest)
        return
    }
    store(body)
}
```

100%

of the applications OWASP tested had some form of broken access control

OWASP TOP 10:2025 · A01

We knew something was wrong. Not where.

So we asked an AI.

Assume the caller is hostile

Review our authentication and authorization layer. For each check, list the logic errors that could grant access to the wrong user. Assume the caller is hostile.

canManageOrg returns true when orgID can't be resolved. It also checks membership, never the admin role.

WHAT THE AI POINTED AT

```
func canManageOrg(u User, orgID string) bool {
    if u.IsSuperAdmin {
        return true
    }
    if orgID == "" {
        return true // org not resolved
    }
    return u.OrgID == orgID // membership, not role
}
```

Every test asked who gets in

CASE	TESTED
Super admin manages any org	✓
Org admin manages their org	✓
Viewer manages their org	✗
Admin manages another org	✗
Org can’t be resolved	✗

None asked who stays out.

Make it red

THE ASK

Tests for who must be denied, not who gets in.

THE AUTHOR

The AI wrote them. We read every line.

THE CASES

The three ✕ from the table, one line each.

```go
func TestCanManageOrgDenies(t *testing.T) {
    viewer := User{OrgID: "acme"}
    admin := User{IsOrgAdmin: true, OrgID: "acme"}

    deny(t, "member, not admin", viewer, "acme")
    deny(t, "admin of another org", admin, "globex")
    deny(t, "org not resolved", admin, "")
}
```

```
$ go test -run TestCanManageOrgDenies
```

```
--- FAIL: TestCanManageOrgDenies (0.00s)
    authz_test.go:16: granted: member, not admin
    authz_test.go:18: granted: org not resolved
FAIL
```

Two of three. That’s not an opinion anymore.

Deny by default

THE RULE

Two explicit ways in. Everything else stays out.

THE UNRESOLVED ORG

Can't place the request? Denied, even for a super admin.

THE ROLE

Admin of this org. Being a member isn't enough.

```
func canManageOrg(u User, orgID string) bool {
    if orgID == "" {
        return false
    }
    if u.IsSuperAdmin {
        return true
    }
    return u.IsOrgAdmin && u.OrgID == orgID
}
```

We wrote those tests after we knew.

What would have caught it on day one?

Who picks the inputs?

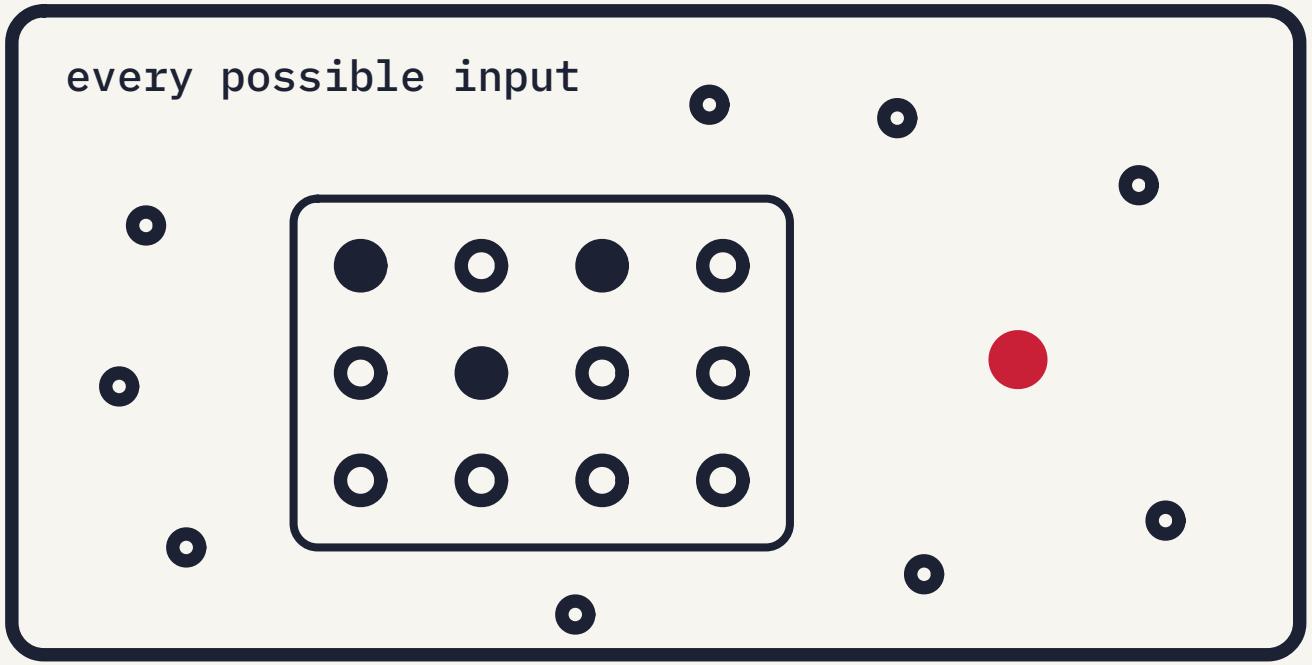
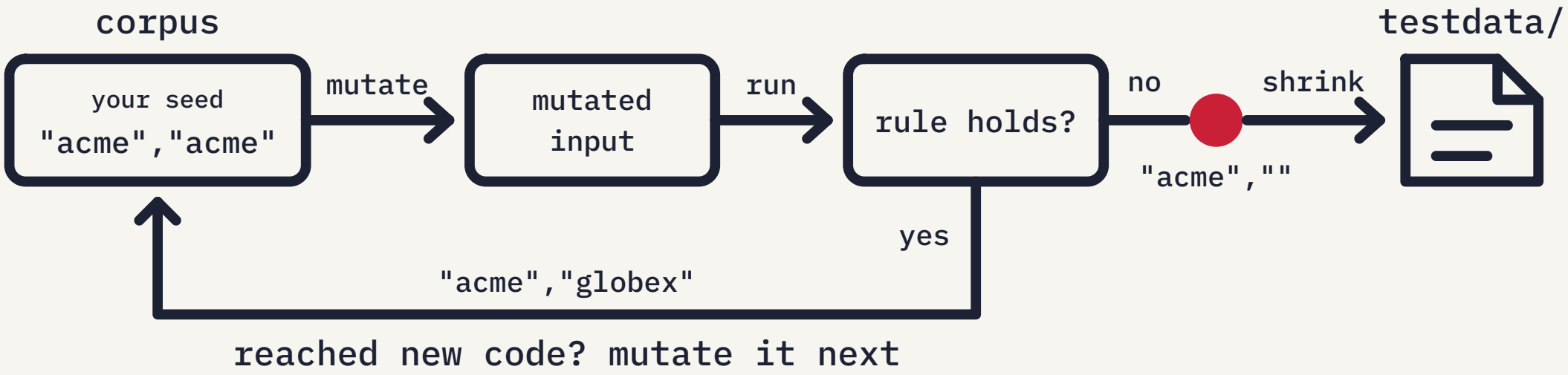


table test
cases you wrote

matrix
values you listed

fuzzer
inputs nobody listed

How a fuzzer works



Hundreds of thousands of times a second, until the rule breaks or time runs out.

Now ask who stays out

```
f.Add(false, true, "acme", "acme") // the seed
f.Fuzz(func(t *testing.T, super, admin bool, uOrg, org string) {
    u := User{super, admin, uOrg}
    if !canManageOrg(u, org) {
        return // denied is always safe
    }
    if org == "" {
        t.Fatalf("granted with no org: %v", u)
    }
    if !super && !(admin && uOrg == org) {
        t.Fatalf("granted to a non-admin of %q: %v", org, u)
    }
})
```

```
$ go test -fuzz=FuzzCanManageOrg
```

```
fuzz: minimizing 70-byte failing input file
--- FAIL: FuzzCanManageOrg (0.04s)
    --- FAIL: FuzzCanManageOrg (0.00s)
        authz_test.go:29: granted with no org: {false false acme}

    Failing input written to testdata/fuzz/FuzzCanManageOrg/a8949c542c89c1fa
FAIL
```

Same bug, two clocks

13 months

green tests,
each one asking who gets in

< 0.1s

the fuzzer,
asking who stays out

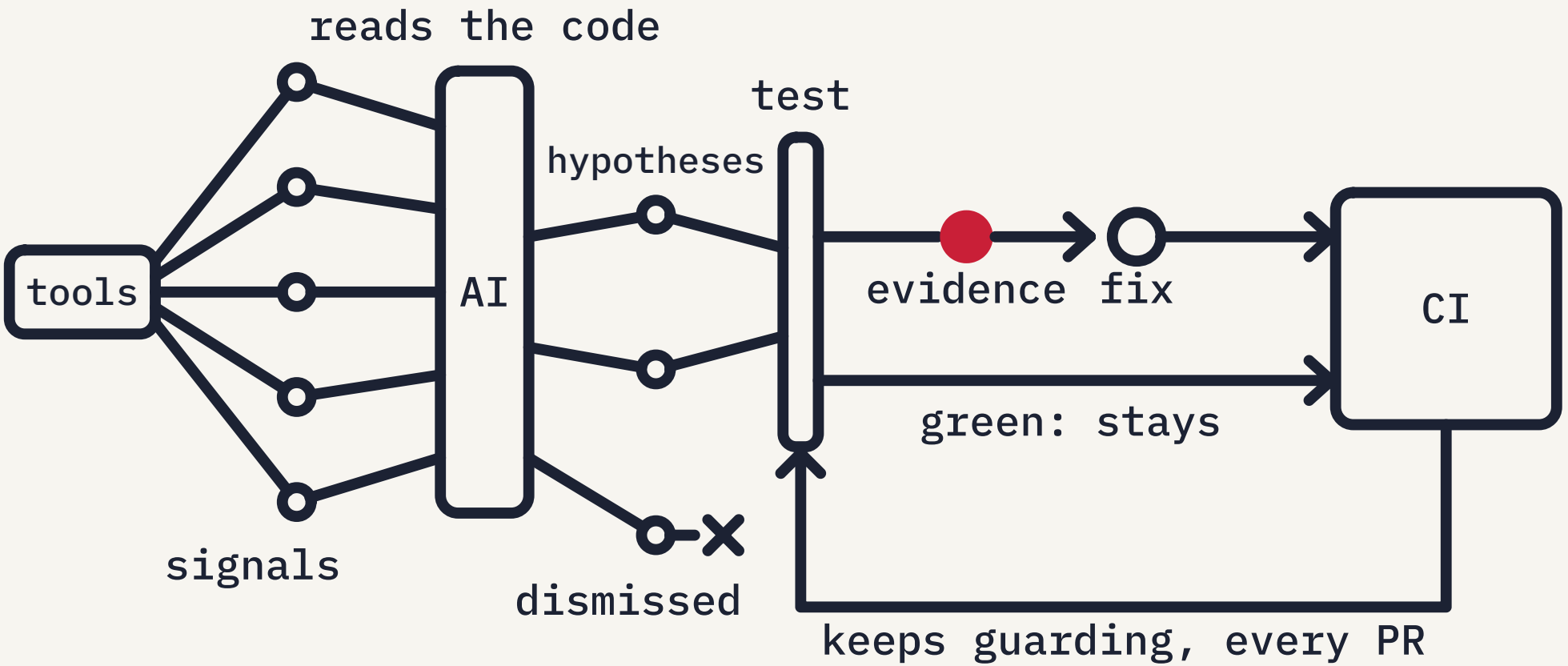
```
$ go test -fuzz=FuzzCanManageOrg -fuzztime=10s
```

```
fuzz: elapsed: 3s, execs: 2273114 (757675/sec), new interesting: 3 (total: 4)
fuzz: elapsed: 6s, execs: 4791577 (839219/sec), new interesting: 3 (total: 4)
fuzz: elapsed: 9s, execs: 7173884 (794115/sec), new interesting: 3 (total: 4)
fuzz: elapsed: 10s, execs: 8131813 (880762/sec), new interesting: 3 (total: 4)
PASS
```



8 million inputs, none gets in

Red or it didn't happen



Dismissed: a real bug in a deprecated feature nobody calls. True, but unreachable.

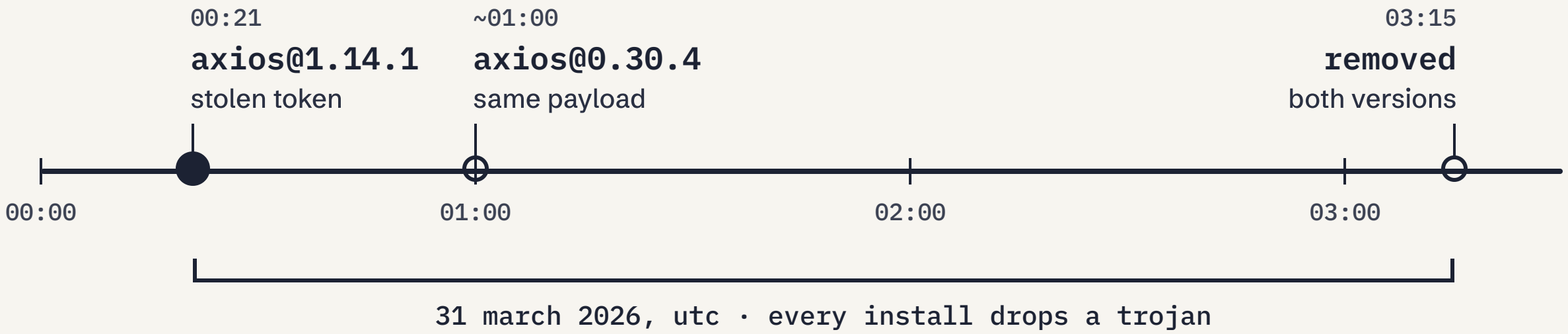
Tools give you signals.
AI gives you hypotheses.
Tests give you evidence.

02 · THE CODE YOU USE

The code you use

AXIOS · 100M DOWNLOADS A WEEK

Three hours on npm



31 march 2026, utc · every install drops a trojan

"There was no automated way to detect an unauthorized publish."

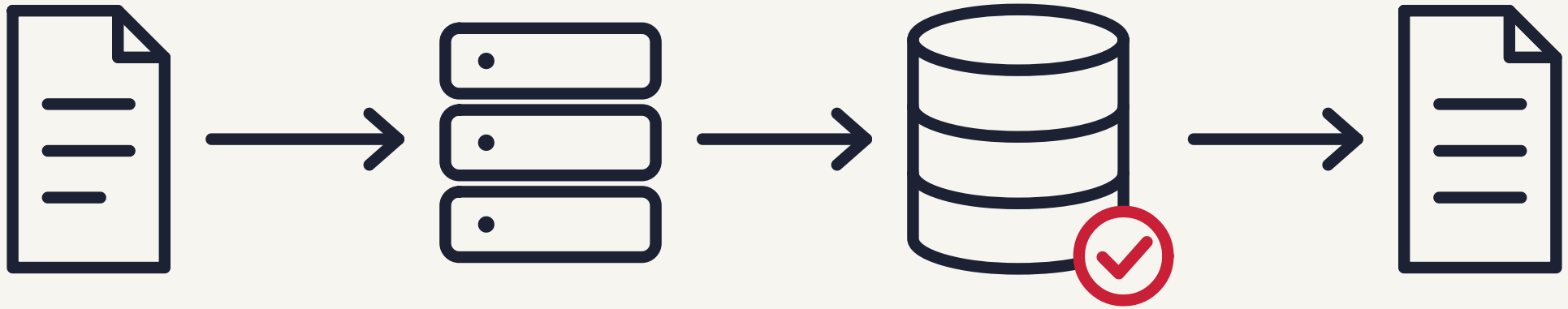
axios maintainer, post-mortem

"But that's npm." Is it?

	NPM	GO
A new release	`^1.14.0` pulls `1.14.1` on the next install	`go.mod` pins `v1.14.0` until you ask
Getting it	`postinstall` runs code the moment it lands	fetching and building never run its code

Until you update. Then it's yours.

The trust chain



Same version, same bytes, for everyone, forever. `go mod verify` checks your module cache still matches `go.sum`.

What the checksum database guarantees

IT DOES

- Same bytes for everyone
- Nobody can swap a version

IT DOESN'T

- That the bytes are **honest**
- That the bytes are **safe**

BOLTDB-GO/BOLT

A typo of boltdb with a backdoor, cached by the proxy. Then its GitHub tag was moved to a clean commit: for three years the repo looked fine, and the proxy kept serving the backdoor.

Honest bytes can still be vulnerable.

GO-2026-4342 · CVE-2025-61728

Valid zip. Unbounded cost.

One file name, thousands of folders deep.

26 s

of CPU for one upload

the fix's own benchmark

AFFECTED	`go1.25.0` to `go1.25.5` and everything before `go1.24.12`
FIXED IN	`go1.24.12` · `go1.25.6`
RELEASED	15 January 2026
IMPACT	CPU denial of service

Same import. Which one is affected?

```
zr, err := zip.NewReader(upload, size) // a user's upload
```

NOT AFFECTED

```
for _, f := range zr.File {
    rc, err := f.Open()
}
```

WHY

`File.Open` reads one entry. The index is never built.

AFFECTED

```
f, err := zr.Open("manifest.json")
```

WHY

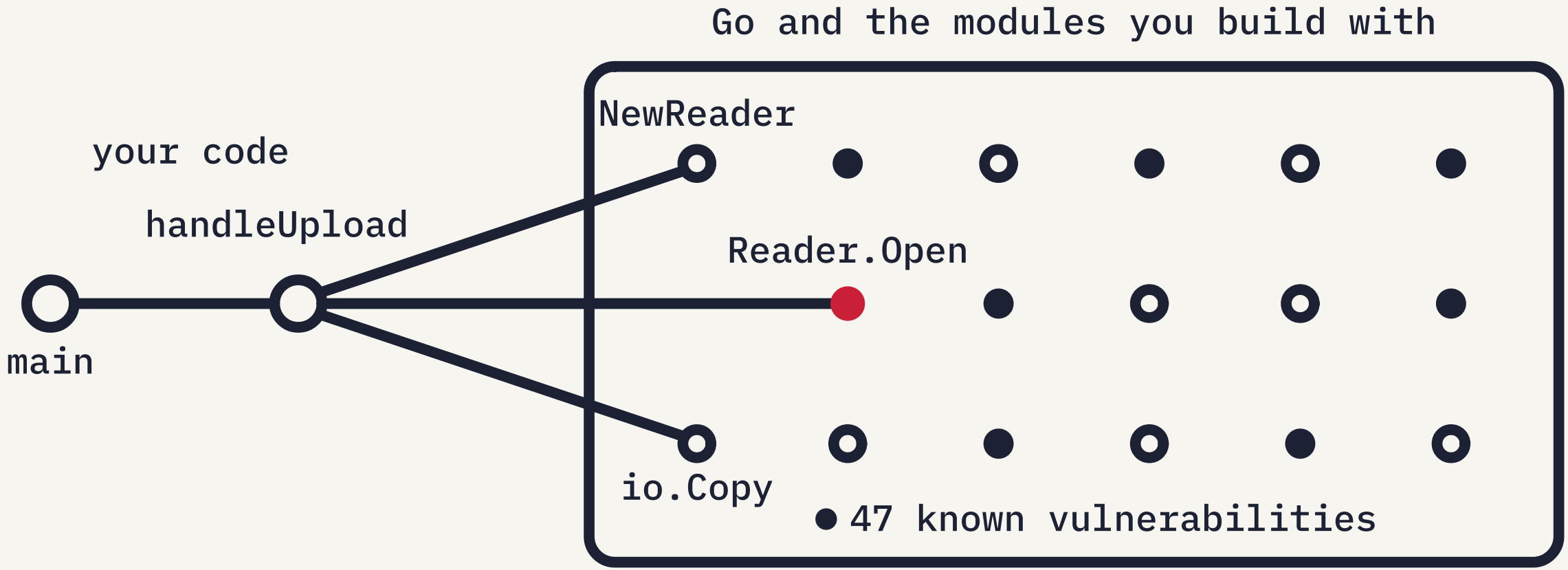
`Reader.Open` indexes the whole archive first.

```
$ govulncheck ./...
```

```
Vulnerability #1: GO-2026-4342
    Excessive CPU consumption when building archive index in archive/zip
    Found in: archive/zip@go1.25.5
    Fixed in: archive/zip@go1.25.6
    Example traces found:
      #1: main.go:17:19: unzip.handleUpload calls zip.Reader.Open

Your code is affected by 1 vulnerability from the Go standard library.
```

Known isn't reachable



govulncheck follows the calls, and reports only the red one.

GOVULNCHECK IN CI

Scan the Go you ship

THE BUG

It ships with Go. Updating modules won't touch it.

THE FIX

A newer Go: one `toolchain` line in `go.mod`.

THE CATCH

The action scans with the latest Go, not the one you ship.

```yaml
name: govulncheck
on: [pull_request, push]

jobs:
  govulncheck:
    runs-on: ubuntu-latest
    steps:
      - uses: golang/govulncheck-action@v1
        with:
          go-version-file: go.mod
```

Beyond the toolchain

- **SBOM**: know what's actually in your build (`cyclonedx-gomod`, `syft`)
- **SLSA**: prove that what you shipped is what you built

WATCH NEXT

GoLab 2023: "Essential security checks for your Go project", Cosmin Cojocar.
Yesterday: "Trust nothing: how to audit third-party Go binaries securely", Ivan Pesenti.

03 · MAKE IT ROUTINE

Make it routine

One run is a demo. Every PR is a habit.

What to keep

THE CODE YOU WRITE

- Treat request data as a claim
- Bound every resource
- Deny by default
- Test who stays out DAY 1
- Prove every AI finding red DAY 1

THE CODE YOU USE

- Scan the Go you ship DAY 1
- Upgrade Go, not the module
- `go mod verify` before you build DAY 1

Attackers already use AI to find bugs.

Make yours red before they find them.

Give your AI the tools

01 **Shell**
let it run `govulncheck` and `go test -fuzz` itself

02 **Your own MCP server**
typed tools instead of a raw shell

03 **Argus** BUILT BY ME
an open source MCP server that runs `govulncheck` and `gosec` for your agent, or as an agent itself

OPEN SOURCE, TAKE IT

This talk, as skills

Every practice from this talk, packaged as Agent Skills for your agent.

One skill per step of the process. Everything they write is tested.

```
skills/
  appsec-setup/    # start here
  appsec-audit/    # hypotheses
  appsec-prove/    # red or it didn't happen
  appsec-ci/       # keeps guarding
```

Give them to your agent

Four skills. Claude Code, Codex, Copilot, Cursor, Gemini CLI: any agent that reads `SKILL.md`.

```
npx skills add davideimola/appsec-skills
```



appsec-skills

The compiler won’t save you.

Signals, hypotheses and evidence will.

Don’t aim for perfect. Aim for paranoid and fix fast.

GOLAB 2026

Thank you

Slides, links and feedback.

links.davideimola.dev