Go: which habits still turn into vulnerabilities?

Before your next release, rebuild every Go binary with Go 1.27.3 or 1.26.10, as Go 1.25 is unsupported[1,2], and make `govulncheck` and `gosec` required CI checks. Judge everything else by whether outside data can reach SQL, a shell, a template, a path or a URL unchecked, or hold memory, a connection or a goroutine without a limit.

Reviewed	Next review	Version	Written for
5 Nov 2026	**5 Dec 2026**	**1.1**	**Developers and tech leads writing production Go services and tools, and the security people who review them**

Latest version
sheets.einblick.dev/secure-go/

If you only do five things

1. **Run a supported Go and rebuild on every point release.** Only the two newest major releases get security fixes[1]: today 1.27.3 and 1.26.10[2]. Rebuild to apply them; `go version -m` shows which Go built a binary[3].
2. **Make govulncheck and gosec required checks.** `govulncheck ./...` narrows findings to vulnerabilities your code can reach and exits non-zero on them[6]. gosec's taint rules G701 to G705 cover SQL, commands, paths, SSRF and XSS[4,8].
3. **Bound every request in time and size.** Zero `http.Server` timeouts mean none, and `http.DefaultClient` has none either[10]. Set `ReadHeaderTimeout`, wrap bodies in `http.MaxBytesReader` and time out every client.
4. **Open files from input only through os.Root.** Path traversal ranks 6th in the CWE Top 25[9]. `filepath.Join` cleans `..` but doesn't confine it[11]; `os.OpenInRoot` and `os.Root` refuse names and symlinks that leave the directory[12].
5. **Test with the race detector and fuzz your parsers.** Races on interfaces, maps, slices and strings can corrupt memory[15], and `go test -race` finds only those your tests execute[16]. Give every parser a `go test -fuzz` target[17].

Review a Go pull request

1. **Where does outside data enter?** Find the handlers, gRPC methods and queue consumers it touches. Each should cap the body, decode into a struct and check the caller's right to the record: missing authorization ranks 4th in the CWE Top 25[9].
2. **Does it start a goroutine?** Ask what stops it, whether it gets the request's `context.Context` and what it shares without a lock[15].
3. **Does input reach SQL, a command, a template or a path?** Search for `fmt.Sprintf` near `Query` or `Exec`, `"sh", "-c"`, `text/template`, `template.HTML(`, `filepath.Join`, `os.Open` and archive readers.
4. **Does it call out over the network?** Check URLs from input, clients without a `Timeout`, `InsecureSkipVerify` and `ReverseProxy.Director`, deprecated in Go 1.26 because clients can strip the headers it adds[5].
5. **Does it switch off Go's checks?** `unsafe`, cgo, `//go:debug` lines and `godebug` blocks in `go.mod` need a second reviewer.

Go traps and what to use instead

- **`filepath.Join(dir, name)` and `http.Dir`.** `filepath.Join("a/b", "../../../xyz")` returns `../xyz`[11], and `http.Dir` follows symlinks out of its tree and serves dotfiles such as `.git`[10]. Use `os.OpenInRoot`, `os.Root` and `http.FileServerFS(root.FS())`[12] on a tree without dotfiles, and patch: `os.Root` escapes were fixed in March, April, July and October[13,14].
- **`zip.NewReader` and `tar.NewReader`.** Both accept entry names like `../x` unless `GODEBUG` sets `zipinsecurepath=0` or `tarinsecurepath=0`[19]. Create each entry through an `os.Root` and copy with `io.CopyN` to cap its size, as gosec G110 asks[8].
- **`text/template`, `template.HTML(s)` and data inside `<script>`.** Only `html/template` escapes, and it trusts template authors and `template.HTML` values[20]. Go fixed seven `html/template` escaping bugs in 2026, five in script contexts[21]. Pass data in `data-` attributes, and never parse templates from user input. Cross-site scripting ranks 1st in the CWE Top 25[9].
- **`db.Query(fmt.Sprintf(...))` and ORM string methods.** Go's docs call SQL built with `fmt` a security risk: pass values as arguments with your driver's placeholder, such as `?` or `$1`[22]. GORM doesn't escape `Order`, `Select`, `Table`, `Group` or Raw, and `db.First(&u, id)` with a string `id` is injectable[23]. Map sort columns from a fixed list.
- **`exec.Command("sh", "-c", cmd)`.** `os/exec` deliberately runs no shell[24], and `sh -c` brings back OS command injection, 9th in the CWE Top 25[9]. Pass each argument separately, put `--` before values that may start with `-`, and don't clear `exec.ErrDot`, which stops `Command` from resolving a name to a program in the current directory[24].
- **`http.Get(r.FormValue("url"))`.** The default client has no timeout and follows up to 10 redirects[10]. Allowlist hosts, also in `CheckRedirect`, then reject loopback, private and link-local addresses in `net.Dialer.ControlContext`, which runs just before each connection is made[25]. Server-side request forgery ranks 22nd in the CWE Top 25[9].
- **`json.Unmarshal(body, &req)`.** The v1 decoder matches names case-insensitively, lets the later of duplicate names win and ignores unknown fields, so two services can read one request differently[26]. On Go 1.27, use `encoding/json/v2`, which rejects duplicates and invalid UTF-8 and matches names exactly, with `json.RejectUnknownMembers(true)`[26].
- **`go func() { ... }()` in a handler.** `net/http` recovers a panic in the handler itself[10], but an unrecovered panic in a goroutine it starts ends the program. Nil map writes, nil pointers, bad indexes and one-value type assertions all panic[18]. Use `v, ok := x.(T)` and a deferred `recover` in goroutines that handle input.
- **`int32(n)` and other integer conversions.** They truncate with no indication of overflow, and arithmetic overflows without a panic[18]. Parse with the target size, as in `strconv.ParseInt(s, 10, 32)`, and check lengths before converting them, which gosec G115 flags[8].

Tools that catch it

- **govulncheck.** Source mode checks the standard library of the go on your PATH, `-mode binary` a built binary. Calls through `reflect` are invisible, and `-json` or `-format sarif` output never fails the build[6].
- **gosec.** Fix G112 (no ReadHeaderTimeout), G115 (integer conversion), G204, G304, G305, G402, G404 (`math/rand`) and G701 to G705 first[8].
- **Race detector and fuzzer.** `go test -race` can use 5 to 10 times the memory and 2 to 20 times the time[16]: run it in CI. Fuzz one target per run with `-fuzztime`[17].
- **Goroutine leak profile.** Go 1.27's `goroutineleak` profile reports goroutines that can never wake up[4].

What you can skip

- **Setting `GOFLAGS=-mod=readonly`.** Build commands have behaved that way since Go 1.16[29].
- **FIPS 140-3 mode you don't need.** Go says applications without a compliance need shouldn't enable it, and `fips140=only` isn't meant for production[30].
- **Rushing govulncheck's informational findings.** It lists vulnerabilities your code doesn't call separately and says you may not need to act[7]. Fix them in routine updates.

What changed in 2026

- **Go 1.27 shipped on 19 August,** ending support for Go 1.25[2,1]. Old go lines no longer turn TLS 1.0 or 3DES back on by default[5,4].
- **Go made its first URGENT release.** Go 1.27.3 and 1.26.10, out on 19 October, distrust eMudhra intermediates whose keys were trivially weak[3].
- **Malicious proxies could bypass the checksum database** until fixes in May, August and October. Upgrade the installed Go, as pinning `GOTOOLCHAIN` isn't enough[27]. The supply-chain sheet covers the rest.
- **A compiler bug dropped bounds checks.** Go 1.26.2 and 1.25.9 fixed loop arithmetic that could let code without `unsafe` corrupt memory (CVE-2026-27143)[28].

Mistakes that cost the most

- **Patching the image, not the binary.** On Linux, Go checks certificates itself, without revocation checks[3]. Binaries rebuilt with 1.27.3 or 1.26.10 reject the eMudhra intermediates whatever trust store the image ships[3]. Scan what you deploy with `govulncheck -mode binary`.
- **Leaving the go line in `go.mod` behind.** Most GODEBUG defaults follow it, so a `go 1.23` module built with Go 1.27 still accepts RSA keys under 1,024 bits. Check with `go list -f '{{.DefaultGODEBUG}}'`[19].

What this looks like for your code

New project

- For services, run `go get go@1.27.3` after `go mod init`, which writes an older go line[5], and use `encoding/json/v2` and `os.Root` from the start.
- Give each `http.Server` timeouts, and put `http.NewCrossOriginProtection().Handler` in front of the mux against CSRF[10].
- Make `govulncheck`, `gosec` and `go test -race` required checks.

Existing codebase

- Leave Go 1.25, raise the `go` line and compare `DefaultGODEBUG` before and after[19].
- Search for `ListenAndServe(`, `filepath.Join(`, `"sh", "-c"`, `InsecureSkipVerify`, `Director:` and `math/rand`.
- Run `govulncheck -mode binary` on the binaries you deploy[6].

High-assurance or regulated

- For FIPS 140-3, build with `GOFIPS140=certified`: Go Cryptographic Module v1.0.0 holds CMVP certificate #5247[30].
- Fuzz every parser continuously in your own CI.
- Require CodeQL's `security-extended` suite; private code needs a paid GitHub Code Security license[31]. Build releases with `CGO_ENABLED=0` unless a reviewed package needs cgo; `-race` needs it[16].

Version history

5 Nov 2026, version 1.1 Reviewed for company readers: CodeQL on private code needs paid Code Security, and OSS-Fuzz takes only open-source projects.

Sources

1. Go Security Policy. Go, checked 5 Nov 2026. go.dev
2. Release History. Go, checked 5 Nov 2026. go.dev
3. Our Response To The eMudhra Key Disclosure Incident. Go Blog, 26 Oct 2026. go.dev
4. Go 1.27 Release Notes. Go, Aug 2026. go.dev
5. Go 1.26 Release Notes. Go, Feb 2026. go.dev
6. govulncheck command. Go, v1.9.0, 11 Oct 2026. pkg.go.dev
7. Tutorial: Find and fix vulnerable dependencies with govulncheck. Go, checked 5 Nov 2026. go.dev
8. gosec rules. securego, v2.30.0, 23 Oct 2026. github.com
9. 2025 CWE Top 25. MITRE CWE, Dec 2025. cwe.mitre.org
10. Package net/http. Go, checked 5 Nov 2026. pkg.go.dev
11. Package path/filepath. Go, checked 5 Nov 2026. pkg.go.dev
12. Package os. Go, checked 5 Nov 2026. pkg.go.dev
13. Vulnerability reports: os. Go, checked 5 Nov 2026. pkg.go.dev
14. GO-2026-4864 (CVE-2026-32282). Go, 7 Apr 2026. pkg.go.dev
15. The Go Memory Model. Go, 6 Jun 2022. go.dev
16. Data Race Detector. Go, checked 5 Nov 2026. go.dev
17. Go Fuzzing. Go, checked 5 Nov 2026. go.dev
18. The Go Programming Language Specification. Go, 26 May 2026. go.dev
19. Go, Backwards Compatibility, and GODEBUG. Go, checked 5 Nov 2026. go.dev
20. Package html/template. Go, checked 5 Nov 2026. pkg.go.dev
21. Vulnerability reports: html/template. Go, checked 5 Nov 2026. pkg.go.dev
22. Avoiding SQL injection risk. Go, checked 5 Nov 2026. go.dev
23. Security. GORM docs, checked 5 Nov 2026. gorm.io
24. Package os/exec. Go, checked 5 Nov 2026. pkg.go.dev
25. Package net. Go, checked 5 Nov 2026. pkg.go.dev
26. Package encoding/json/v2. Go, checked 5 Nov 2026. pkg.go.dev
27. Vulnerability reports: cmd/go. Go, checked 5 Nov 2026. pkg.go.dev
28. GO-2026-4868 (CVE-2026-27143). Go, 7 Apr 2026. pkg.go.dev
29. Go Modules Reference. Go, checked 5 Nov 2026. go.dev
30. FIPS 140-3 Compliance. Go, checked 5 Nov 2026. go.dev
31. CodeQL CLI. GitHub Docs, checked 5 Nov 2026. docs.github.com