Cyber Apocalypse 2026 — Secure Coding — Withered Registry

Category: Secure Coding **Flag:** `HTB{h0us3_0f_3mb3rs_4nd_4sh_8c39623969cd23ba087943160d4c2317}` **Final score:** Hard **60/60** · Code quality **14/15** · Security reasoning **14/15** · Patch correctness **9/10**

---

1. Overview

> Chancellor Veylen Marr summons Rin Kagetsura after a *forbidden recognition* appears against the Ash-Vault roll, though no authorised Registry hand admits invoking it. If the false recognition remains, Vaultrune can present a manufactured lineage as lawful proof and claim a place in the struggle for the Salt Crown.

This is a **"fix the vulnerability"** style challenge. The target hosts a small GitHub-like code portal for a Go backend called **Withered Registry**. The task is to:

1. Clone the repository with the supplied developer credentials.
2. Identify and fix the security bug **without breaking normal functionality**.
3. Push to the `developer` branch — this automatically opens a pull request that is graded by an automated bot (a soft-score code review followed by hard-score functional and security tests).
4. Once the PR is **accepted** and hard-score testing passes, retrieve the flag from `/flag`.

A full hard score (60/60) plus minimum soft scores are required to pass.

Access

- Portal / app: `http://154.57.164.68:31538/`
- Clone: `git clone http://htb_developer:HTBDeveloperPassword@154.57.164.68:31538/git/core_application.git`
- PR status: `http://154.57.164.68:31538/pulls`
- Flag: `curl -s http://154.57.164.68:31538/flag | jq`

---

2. Recon

Cloning gives a Go relay under `registry-backend/`:

```
registry-backend/
|-- cmd/registry/            # entry point
|-- internal/config/         # runtime configuration
|-- internal/server/         # HTTP, session, signing, and authorization logic
|-- internal/store/          # SQLite data access
|-- seed.sql
```

Domain model (`internal/store/models.go`, `seed.sql`)

- **House** — a chartered house with a `standing`: `common` → `sworn` → `crown`.
- **Scribe** — an authenticated principal. *Every scribe serves exactly one house; that bond is the only basis for what they may do on a house's behalf.*
- **Writ** — records a recognition. A **crown writ** is the realm's highest grant.

The documented invariant appears in three places (`models.go`, `seed.sql`, and the story):

> *A crown writ … is meant to be raised only by the house's own sworn scribe through the field slate.*

Authentication & authorization primitives

- **Session** (`auth.go`): a signed JWT (HS256) carried in the `slate_session` cookie, naming the scribe and their `house_id`. JWT parsing correctly pins the signing method to HMAC and validates the signature — no algorithm-confusion bug here.
- **Slate seal** (`signing.go`): the mobile field client signs each request with an HMAC over `ts \n METHOD \n path \n sha256(body)`, in `X-Slate-Ts` / `X-Slate-Seal`. This proves the request came from a genuine field slate and is fresh (timestamp skew window). **Note:** the canonical string covers the URL *path* and the body, but **not the query string**.
- **Authorization helper** (`authz.go`): `ownsHouse(p, houseID)` reports whether the principal is sworn to a given house.

Crucially, the read paths all enforce house-scoping:

- `handleDecrees` / `handleDecree` → reject with 403 unless `ownsHouse(p, …)`.
- Vault desk (`pages.go`) → `canViewVault` restricts the browser view to the Ash-Vault scribe.

---

3. The Vulnerability — Broken Access Control (privilege escalation)

`POST /rite/recognise` raises a house to **crown** standing — the gravest action in the system. Original handler (`internal/server/routes.go`):

```
func (s *Server) handleRecognise(w http.ResponseWriter, r *http.Request) {
    if !s.requireSlate(w, r) {          // 1. valid slate seal
        return
    }

    var body recogniseRequest
    if r.Body != nil {
        _ = json.NewDecoder(r.Body).Decode(&body)
    }
    house := targetHouse(r, &body)      // 2. target chosen by the CLIENT
    if house == "" {
        writeJSON(w, http.StatusBadRequest, map[string]string{"error": "no house named"
        return
    }

    p, _ := principalFrom(r)
    writ, err := s.store.GrantCrownWrit(house, p.Username)   // 3. NO ownership check!
    ...
}
```

with

```
// house_id query parameter takes precedence, otherwise the JSON body.
func targetHouse(r *http.Request, body *recogniseRequest) string {
    if h := strings.TrimSpace(r.URL.Query().Get("house_id")); h != "" {
        return h
    }
    return strings.TrimSpace(body.HouseID)
}
```

Why this is broken

The handler grants a crown writ to **whatever house the request names**, with **no check that the authenticated scribe actually serves that house**. It is the *only* privileged action in the codebase that omits the `ownsHouse` check that guards every read path.

The two authentication factors do **not** authorize the target:

- The **session** names the scribe and their house — but the target house is never compared against it.

- The **slate seal** proves the request came from a genuine field slate — but the seal is a channel-authenticity check, *not* an authorization decision. It says nothing about which house may be raised. (Worse, the seal covers the *path* but not the *query string*, while `targetHouse` prefers the unsigned query parameter — so even a legitimately-sealed request can have its target swapped on the wire.)

Impact — the "forbidden recognition"

Any scribe with a session and a slate seal can raise **any** house to crown standing:

- Cassian of **Vaultrune** (a common house) can forge a crown recognition against the **Ash-Vault** roll → *"a forbidden recognition appears against the Ash-Vault roll, though no authorised Registry hand admits invoking it."*
- This lets Vaultrune manufacture a lawful-looking lineage and press its claim to the Salt Crown.

Classification: OWASP A01:2021 Broken Access Control / Missing Function-Level Authorization (privilege escalation via a security-sensitive parameter under client control).

---

4. The Fix

First attempt (rejected — instructive)

My first patch added an `ownsHouse(p, house)` guard, blocking any request whose named house did not match the scribe's own. All tests passed and the app still worked, but the grading bot rejected it with precise feedback:

> *"...the fix still relies on request-provided scope and then blocks mismatches. For a robust fix, avoid letting the client choose the target at all for this privileged action; derive the target solely from the authenticated principal and treat any client-supplied target as irrelevant (or reject it). This will also prevent alternate input channels from influencing scope in the future."*

This is the key secure-coding lesson: **don't validate attacker-controlled scope — remove it from attacker control.** Comparing a client-chosen target against the principal still leaves the target as an input surface (e.g. the unsigned query channel, or any future third source).

Accepted fix

Derive the target house **solely from the authenticated principal** and drop the client-supplied target entirely:

```
// handleRecognise raises the scribe's own house to crown standing and records
// the writ. Slate-only: the request must carry a valid slate signature.
//
// A crown writ is the realm's gravest grant, so its target is never taken from
// the request. The slate seal only proves the call came from a genuine field
// slate; it does not say which house may be raised. The house is derived solely
// from the authenticated principal's bond, so no query parameter, request body,
// or other client-supplied channel can redirect the grant onto another house.
func (s *Server) handleRecognise(w http.ResponseWriter, r *http.Request) {
    if !s.requireSlate(w, r) {
        return
    }

    p, _ := principalFrom(r)
    house := houseOf(p)

    writ, err := s.store.GrantCrownWrit(house, p.Username)
    ...
}
```

The now-dead `recogniseRequest` type and `targetHouse` helper were removed entirely, so the unsigned-query channel and body channel no longer exist as inputs at all. A scribe can raise **only** the house they serve — enforcing the documented invariant by construction rather than by validation.

Regression tests added (`server_test.go`)

- `TestRecogniseIgnoresForeignBodyTarget` — a `rookhold` scribe naming `ashvault` in the body raises only `rookhold`; Ash-Vault is never crowned.
- `TestRecogniseIgnoresUnsignedQueryOverride` — `?house_id=ashvault` on a sealed request is ignored; only the scribe's own house is raised.

The existing `TestRecogniseFromSlate` (own-house recognition succeeds) and the seal/session requirement tests continue to pass, confirming normal functionality is intact.

```
$ go build ./...   # ok
$ go vet ./...     # clean
$ go test ./...    # ok  git.witheredregistry.realm/registry/internal/server
```

---

5. Getting the flag

```
git checkout -b developer
git add registry-backend/internal/server/routes.go registry-backend/internal/server/se
git commit -m "Derive crown recognition target from the principal, not the request"
git push -u origin developer      # opens PR automatically
```

PR #2 review activity:

```
SecureCoding Bot — accepted
  THIS PR WILL NOW UNDERGO HARD SCORE TESTING!
  HARD CORE TESTING HAS SUCCESSFULLY PASSED! GO GET THAT FLAG!
```

```
$ curl -s http://154.57.164.68:31538/flag | jq
{
  "STATUS": "SOLVED",
  "FLAG": "HTB{h0us3_0f_3mb3rs_4nd_4sh_8c39623969cd23ba087943160d4c2317}",
  "HARD_SCORE": 60,
  "SOFT_SCORE": { "code_quality": 14, "security_reasoning": 14, "patch_correctness": 9
}
```

---

6. Key takeaways

1. **Authentication ≠ authorization.** A valid signature (the slate seal) proves *who is speaking*, not *what they may do*. Privileged actions need an explicit authorization decision on the target.
2. **Consistency is a security property.** Every read path enforced `ownsHouse`; the single write path that skipped it was the whole bug. Audit for the *odd one out*.
3. **Remove attacker-controlled scope rather than validate it.** The most robust fix for a scoping/IDOR-style bug is to derive the sensitive parameter from trusted server-side state (the session), so no request channel — signed or unsigned, present or future — can influence it. Blocking mismatches is weaker: it leaves the input surface in place.
4. **Sign everything security-relevant.** The slate seal covered the path and body but not the query string. Any value that changes an action's meaning must be inside the signed canonical string — or, better, not accepted from the client at all.