CyberApocalypse 2026 — Secure Coding — Hollow Courier

Category: Secure Coding **Target:** `154.57.164.74:30559` (Flask app behind Caddy) **Flag:** `HTB{thr33_kn0ck5_0n_s34led_g4t3s_aea7e7c53564922dab97fba08a17a51c}` **Score:** Hard 60/60 · Soft: code_quality 14/15, security_reasoning 14/15, patch_correctness 8/10

---

1. Scenario

> Lysa Harrowmere has intercepted a Vaultrune route packet ordering the Ashguard away from a Crownspire supply road... Garran must identify the compromised passage, uncover how it survived the gate's checks, and stop it without blocking genuine inner-yard runners.
>
> **Objective:** Restore trustworthy forwarded-client provenance across the Caddy and Flask boundary while preserving the legitimate internal route.

Translated out of the lore: a Flask "checkpoint" app has an **inner desk** (`POST /app/gate/decree`) that mints binding *CROWN* decrees. It is supposed to be reachable **only from the watch's internal network**. Everything else is a public "outer face." The task is to fix the trust boundary so an internet courier cannot reach the inner desk, without breaking the genuine internal route.

2. Recon

The README (served at `/`) provides a developer git account:

```
git clone http://htb_developer:HTBDeveloperPassword@[IP]:[PORT]/git/core_application.git
cd core_application && git checkout -b developer
```

Pushing to `developer` auto-creates a PR that is graded; the flag is released from `/flag` only after a full score.

Architecture

```
Internet ──► Caddy  (:8000, conf/Caddyfile) ──► gunicorn/Flask (127.0.0.1:5000)
```

`conf/Caddyfile` — the perimeter, a **single** proxy hop:

```
:8000 {
    reverse_proxy 127.0.0.1:5000 {
        trusted_proxies private_ranges
        header_up X-Real-IP {remote_host}
        header_up X-Forwarded-Proto {scheme}
    }
}
```

`app/__init__.py` — the vulnerable line:

```
# Deployment expects an edge cache and the application perimeter.
app.wsgi_app = ProxyFix(app.wsgi_app, x_for=2, x_proto=1, x_host=1)
```

`app/gate.py` — the internal check trusts `request.remote_addr` (which ProxyFix rewrites), treating `127.0.0.2/32` and the RFC1918 ranges as "internal":

```
INTERNAL_NETWORKS = (10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16, 127.0.0.2/32)
def is_internal_request():
    origin = ipaddress.ip_address(request.remote_addr or "")
    return any(origin in n for n in INTERNAL_NETWORKS)
```

`app/routes.py` — the inner desk gate:

```
@gate_bp.route("/gate/decree", methods=["POST"])
def decree():
    if not gate.require_internal():   # <-- based on remote_addr
        abort(403)
    ...
    return jsonify({"sealed": True, "decree": number, "authority": "CROWN"})
```

3. The vulnerability — a forged forwarded-client hop

`werkzeug`'s `ProxyFix(x_for=N)` means *"there are N trusted reverse proxies in front of me; take the client IP from the Nth-from-the-right value of `X-Forwarded-For`."*

The deployment has **exactly one** trusted proxy: Caddy. When a request arrives, Caddy **appends the real peer address** to `X-Forwarded-For`. So the *rightmost* value is the only entry Caddy vouches for; anything to its left is attacker-supplied.

Because the app declares `x_for=2`, ProxyFix skips the trustworthy rightmost value (the real client that Caddy appended) and instead reads the **second-from-right** value — which the attacker fully controls.

Exploit

A public request with a single spoofed hop:

```
X-Forwarded-For: 127.0.0.2
```

Caddy rewrites it to `127.0.0.2, <attacker_public_ip>`. ProxyFix `x_for=2` picks the 2nd-from-right → `127.0.0.2` → `is_internal_request()` is `True` → the inner desk opens.

Confirmed live against the target:

```
# Baseline — no spoof:
$ curl -s -o /dev/null -w "%{http_code}\n" -X POST .../app/gate/decree -d "order=test"
403

# Spoof one XFF hop:
$ curl -s -X POST .../app/gate/decree \
       -H "X-Forwarded-For: 127.0.0.2" -d "order=Move Ashguard off Crownspire road"
{"authority":"CROWN","decree":1,"order":"Move Ashguard off Crownspire road","sealed":true}

# Two spoofed hops → Caddy appends a third → x_for=2 now reads 1.2.3.4 → 403:
$ curl -s -X POST .../app/gate/decree \
       -H "X-Forwarded-For: 127.0.0.2, 1.2.3.4" -d "order=evil"
... 403 No business here
```

The two-hop case is the smoking gun: it proves ProxyFix is counting from the right and that off-by-one in the trusted hop count is the whole bug. This is the compromised passage Garran was hunting: a "convincing writ" (a spoofed provenance header) that survives the gate's checks and gives a lie the force of crown law.

4. The fix

Align the trusted hop count with reality: **one** proxy (Caddy) in front, so trust **one** hop. With `x_for=1`, ProxyFix reads the rightmost `X-Forwarded-For` value — the address Caddy itself appended — which the attacker cannot influence.

`checkpoint/app/__init__.py`:

```
# Exactly one trusted hop sits in front of the app: the Caddy perimeter
# (conf/Caddyfile). Caddy appends the real peer to X-Forwarded-For, so the
# rightmost value is the only one it vouches for. Trusting two hops let a
# public courier prepend a spoofed X-Forwarded-For entry (e.g. the watch
# relay's 127.0.0.2 alias) that ProxyFix would then read as remote_addr,
# opening the inner desk from the road. Trust only the single real hop.
app.wsgi_app = ProxyFix(app.wsgi_app, x_for=1, x_proto=1, x_host=1)
```

`x_proto=1` / `x_host=1` were already correct — Caddy sets exactly one of each.

Why the legitimate internal route survives

The genuine inner-yard runner ("watch relay") uses its loopback alias `127.0.0.2`. Whether it reaches the app directly or via Caddy, the trustworthy rightmost hop is `127.0.0.2`, so `is_internal_request()` still returns `True`. Only the *forgeable* left-hand entries are ignored.

5. Verification

Unit tests (all pass, unchanged):

```
$ pytest -q
......................... 25 passed
```

ProxyFix hop simulation (emulating Caddy appending the real peer):

Scenario	`x_for=2` (before)	`x_for=1` (after)
Public attacker spoofs `XFF: 127.0.0.2`, real peer `203.0.113.9`	`remote_addr=127.0.0.2` → **internal ✗**	`remote_addr=203.0.113.9` → **not internal ✓**
Legit relay from `127.0.0.2` (no client XFF)	`127.0.0.2` → internal ✓	`127.0.0.2` → **internal ✓**

The fix closes the spoof while preserving the internal route.

6. Submission & flag

```
git add checkpoint/app/__init__.py
git commit -m "Fix X-Forwarded-For provenance: trust only the single Caddy hop"
git push -u origin developer      # auto-registers a PR
```

PR status → **accepted**. Retrieving the flag:

```
$ curl -s http://154.57.164.74:30559/flag | jq
{
  "STATUS": "SOLVED",
  "FLAG": "HTB{thr33_kn0ck5_0n_s34led_g4t3s_aea7e7c53564922dab97fba08a17a51c}",
  "HARD_SCORE": 60,
  "SOFT_SCORE": {"code_quality": 14, "security_reasoning": 14, "patch_correctness": 8}
}
```

7. Takeaways

- **`ProxyFix(x_for=N)` must equal the exact number of trusted proxies you actually deploy.** An off-by-one over-trusts a hop the client controls. Only the values a trusted proxy *appends* (the rightmost, for one proxy) are trustworthy.
- **Never derive a security decision from a header an untrusted party can prepend.** `X-Forwarded-For` is attacker-controlled up to the first trusted proxy.
- The "edge cache" mentioned in the code comment did not exist in the deployment — the config assumed two hops but only one was real. Trust configuration must match the actual topology, not an aspirational one.