The Hood
MARKETS

AUDIT REPORT

The Hood — Full Security Assessment

Contracts, the prediction market, frontend, API, database and performance — audited by Claude Fable 5

DATE	APPLICATION	STATUS	AUTHOR
2026-08-15	General	Draft	Claude Fable 5

Contents

1. Disclaimer — read this first
2. Executive summary
3. Scope and method
4. Severity model
5. Section A — Smart contracts
6. Section B — Frontend
7. Section C — Backend & API
8. Section D — Database
9. Section E — Performance
10. Section F — Prediction market
11. Centralization & privilege
12. Controls that held
13. Test coverage
14. Reproducing this assessment

---

1. Disclaimer — read this first

This assessment was performed by Claude Fable 5 at the request of the project team.

What this assessment does: it reads the code, writes executable exploits and use-case tests, runs them against the real contracts (including against live Uniswap V4 on a fork of Robinhood Chain mainnet), and reports what it found with evidence anyone can reproduce. The contract findings were fixed and re-tested; the application findings from this round are reported with their status stated honestly — **several are open**, and are marked as such.

What it does not do: formal verification, or economic modelling beyond the specific attacks listed. **These contracts are deployed on mainnet and hold real value.** Anyone deciding whether to use the platform should weigh that.

---

2. Executive summary

The Hood was assessed across six surfaces: smart contracts, the prediction market, frontend, backend/API, database, and performance. The contract layer has been through six adversarial passes and is the most hardened; the application layer — most of it written recently, including a new "report a problem" feature and a per-launch graduation-target refactor — was assessed here for the first time.

Section	Critical	High	Medium	Low	Info
A — Smart contracts	2 *(fixed)*	1 *(fixed)*	2 *(fixed)*	2	1
B — Frontend	0	0	1 *(open)*	2	2
C — Backend & API	0	0	1 *(open)*	2	1
D — Database	0	1 *(fixed)*	1 *(open)*	2	0
E — Performance	0	0	2 *(open)*	2	0
F — Prediction market	1 *(fixed)*	2 *(fixed)*	3 *(2 fixed)*	3	3

The contract layer is in good shape. Every exploitable contract issue found across six passes — including two that could have caused total loss of a launch's funds — has been fixed, deployed, and locked behind a regression test. The refactor that made the graduation target a per-launch choice was attacked specifically (Section A, pass 6) and holds on every rung.

The application layer has two items worth prompt attention, neither in the money path:

- **D-1 (High):** a schema/migration drift that will break the indexer and leaderboards on a fresh production deploy. Confirmed with evidence. Does not affect funds — it affects whether the site's data surfaces come up at all.
- **The retention gap (Medium), found independently by all three application reviews:** the privacy notice promises problem reports are kept 12 months and deleted on request, and **no code implements either** deletion or a retention sweep. A compliance gap and an unbounded-growth risk on the one unauthenticated endpoint.

The prediction market (`src/predict`) was audited in full and had its own serious cluster. It is a Hood-native product — parimutuel markets on The Hood's launch outcomes, holding real staked ETH/ERC20 — and no longer connected to Seedify in any way. Its money mechanics and authenticity model are solid, but its **settlement oracle and anti-sybil census were both exploitable**, proven with three working PoCs (Section F): a settlement could be flipped to steal the pot, and the holder census that gates project-currency listing could be defrauded for ~0.01% of supply. *(Status update: both were fixed and regression-tested, and the five prediction-market contracts were deployed to Robinhood Chain mainnet on 15 August 2026 after the fixes landed — see the Remediation status below and Section F.)*

Remediation status. Every Critical and High finding across the whole codebase is fixed and verified — F-1 (Critical, settlement manipulation), F-2 (High, forced graduation), F-3/F-4 (High, census sybil), and D-1 (High, migration drift) — along with two settlement/census mediums (F-5, F-6). The remaining open items are medium/low: the application layer (report retention, ingest cap, RPC-URL disclosure, redaction label, DB index/aggregates, and performance) and the predict low-severity ladder items (F-7–F-11). 418 automated tests pass (74 of them adversarial security tests), with zero static-analysis warnings.

3. Scope and method

Repository: `thehood` monorepo. **Chain:** Robinhood Chain mainnet, id 4663. **Contracts:** solc 0.8.28, legacy pipeline, optimizer on.

Surface	What was reviewed
Contracts (launchpad)	`contracts/src/launchpad/**` — factory, curve, token, pool, V4 adapter, locker, fee splitter, maths
Contracts (prediction market)	`contracts/src/predict/**` — parimutuel market, factory, outcome oracle, settlement/creator registries, holder census, predicates
Frontend	`apps/launchpad/src/**` — the report-a-problem widget, admin, legal, layout, and app-wide client security
API	`apps/launchpad/src/app/api/**` — report ingest and admin mutation routes, session, admin gate
Database	`packages/accounts/prisma/**` — schema, the `problem_reports` migration, the client
Performance	`apps/launchpad/src/lib/chain.ts` and the contract gas profile

Method. Aligned to the structure common to CertiK and Hacken: an SWC-registry sweep, CertiK's category model (with centralization treated as first-class), and Hacken's Impact × Likelihood severity. Every contract finding is an executable exploit that runs in the repository's test suite; attacks on the Uniswap V4 path run against the **real deployed V4 contracts** on a mainnet fork, not mocks. The frontend, API and database were reviewed by dedicated passes with file-and-line evidence for every finding. An attack that failed is recorded as a control that held (Section 11), not omitted.

4. Severity model

Severity	Meaning
Critical	Direct loss or permanent freezing of user funds, no recovery path; or unprivileged/unwarned exploit.
High	Loss/freeze of protocol or creator revenue; a launch-specific denial of a core function; or a defect that breaks a core service on deploy.
Medium	Bounded or recoverable value/fairness/compliance impact, or one needing unusual conditions.

Severity	Meaning
Low	Limited impact, or fully mitigated by correct operational behaviour.
Informational	No direct security impact; relevant to operators and integrators.

5. Section A — Smart contracts

Six adversarial passes. **All exploitable findings are fixed, deployed and regression-tested.** The detailed write-ups of A-1 through A-9 (with the exploit for each) are in the companion report `the-hood-security-assessment-2026-08-11`; they are summarised here and joined by pass 6, new to this round.

ID	Severity	Finding	Status
A-1	Critical	Pool squatting permanently bricks any launch (the whole raise becomes unreachable)	Fixed
A-2	Critical	Owner could redirect graduation and take every future raise	Fixed — 2-day timelock
A-3	High	A fee recipient that refuses ETH permanently wedges the locker	Fixed — deferred credit
A-4	Medium	Deploy-script memory aliasing put the test config onto production	Fixed
A-5	Medium	Any holder could shift the graduation price by donating tokens	Fixed — donations burned
A-6	Low	Anti-snipe cap is defeated by splitting across wallets	Acknowledged (speed bump)
A-7	Low	Owner test launches obtain 75% of supply for a negligible sum	Acknowledged, labelled
A-8	Low	`CurveDeployer` is permissionless; unregistered curves can be forged	Acknowledged — use the registry
A-9	Info	Graduation is MEV-visible, but the obvious sandwich loses money (−0.0575 ETH measured)	Measured

A-10 — Per-launch graduation target (this round's refactor)

The graduation target changed from a protocol-wide constant to a per-launch argument, with the launch fee derived from it. A caller-supplied number that feeds both the curve maths and the fee is precisely where this kind of change goes wrong, so it was attacked as hostile input across nine tests (`Pentest6.t.sol`). **It holds.**

- **The ladder is enforced.** Only the five rungs 2/4/6/8/10 ETH are accepted; anything below the floor, above the ceiling, odd, or between rungs is refused (`_requireLadderTarget`). This matters because the curve's integer identity — the curve closes at exactly the pool's opening price — requires an even target, and every rung is even.
- **The fee matches the chosen target exactly**, is derived and never stored (so no two figures can disagree), is monotonic, and never rounds to zero. A launch one wei short of its fee is refused.
- **The price identity holds on every rung**, not just the old default — each of the five graduates with the pool opening at the curve's closing price, verified end to end.
- **The dev exemption stays owner-only** and still requires an even target; no public argument combination reaches it or launches below the floor.
- **Overpayment funds the opening buy or is refunded** — the factory keeps only its accrued fees — and a creator's opening buy is bound by the same early-wallet cap as everyone else.

Contract gas was reviewed and is clean: no unbounded loop over user-growable state, the target-ladder and pagination getters are bounded, and the one micro-optimisation available (caching a couple of warm `SLOAD`s in `_buy`) is a deliberate clarity-over-gas choice, correct for immutable mainnet money that cannot be patched if the optimisation turns out to be wrong.

6. Section B — Frontend

The report-a-problem feature is unusually well-built — client-side redaction with consent integrity, no `dangerouslySetInnerHTML` in the user path, dual client+server validation, and every standing UI rule (no native date pickers, no CDN fonts, file-upload not URL-field for images, Ctrl+Click preserved, exact errors) passes. Findings are governance and hardening, not code execution.

ID	Severity	Finding	Status
B-1	Medium	Privacy notice promises 12-month retention + delete-on-request; no code implements either	Open
B-2	Low	Rate limiter is per-process in-memory; effective limit scales with instance count	Open (documented)
B-3	Low	`x-forwarded-for` left-most token trusted for the rate-limit key — spoofable off Vercel	Open

ID	Severity	Finding	Status
B-4	Info	`marked` output to `dangerouslySetInnerHTML` on the static audit page, no sanitizer	Open (input is trusted, build-time)
B-5	Info	Ingest validator shape-checks a 200-event sample, not every event	Open (documented CPU trade-off)

B-1 is the one to act on. The privacy page states, as binding text, that reports are kept 12 months and "deleted outright — not marked hidden" on request. There is no delete route, no admin delete control, and no retention sweep anywhere in the codebase. It is both a GDPR Art. 17 gap and, combined with the unbounded ingest endpoint, an unbounded-growth risk.

7. Section C — Backend & API

The API is tightly built: reports are readable only by the on-chain protocol owner, checked live per request; ids are `cuid()` (no enumeration); authentication is an HMAC-signed, `HttpOnly`, constant-time-compared session; there is no raw SQL; input is validated on both sides; and errors carry a stable code and details. No IDOR, injection, or auth-bypass was found.

ID	Severity	Finding	Status
C-1	Medium	Unauthenticated report ingest is a storage/cost DoS — spoofable + per-instance limiter, no global cap, no retention	Open
C-2	Low–Med	On an RPC-read failure, the admin error returns the server's RPC endpoint URLs to a signed-in non-owner; if `RPC_URL` embeds a provider key, the key leaks	Open (**confirmed**)
C-3	Low	`redacted` is an unverified client claim defaulting to true; a non-widget caller can store unredacted PII labelled "redacted"	Open
C-4	Info	The unauthenticated 500 fallback echoes raw internal error text	Open (deliberate exact-error policy)

C-2 was confirmed against the code. `checkProtocolOwner` reads `owner()` on-chain for any signed-in session before the owner comparison, so a signed-in non-owner reaches the failure path; that path throws `ADMIN_OWNER_UNREADABLE` (503) whose message and `details.endpoints` embed `serverRpcUrls()`, which puts the configured `RPC_URL` first. The public fallbacks are keyless, so the leak is conditional on `RPC_URL` carrying a credential — but if it does, a signed-in non-owner can extract it during an RPC outage, and the 503 also makes the admin route fingerprintable where the

model otherwise promises a flat 404. Separately, `POST /api/upload` pins to IPFS unauthenticated with no rate limit — a distinct abuse surface worth its own look.

8. Section D — Database

Data modelling is sound where it counts: wei stored as `Decimal(78,0)` and read as strings (never floats), insert-only ledgers with unique idempotency anchors, `submitterAddress` stamped server-side, and IP never persisted. Two findings need attention.

ID	Severity	Finding	Status
D-1	**High**	Schema/migration drift: four indexer models exist in the schema with no migration creating them, and live code queries them	**Fixed**
D-2	Medium	Retention/delete policy promised to users and the regulator is entirely unimplemented	Open (same root as B-1)
D-3	Low	Redundant standalone `@@index([status])` duplicated by a composite index	Open
D-4	Low	Inbox runs two whole-table aggregates per page load	Open (scale-dependent)

D-1, confirmed. `IndexerCursor`, `IndexedLaunch`, `IndexedGraduation` and `IndexedTrade` are defined in `schema.prisma`, and `packages/indexer/src/persist.ts` calls `prisma.indexedLaunch.upsert` and its siblings — but **no migration creates those tables** (the only mention in the migrations folder is a prose comment in the `problem_reports` migration explaining they were removed from the diff). The documented production command is `prisma migrate deploy`, which applies migrations only. On a freshly migrated database the tables will not exist and the indexer and every leaderboard query will throw `relation "IndexedLaunch" does not exist` at runtime. It is invisible in development, where the tables were created by `db push`. **Fix:** add a migration that creates the four tables, their indexes and the two enums to match the schema, and verify with `prisma migrate status`.

9. Section E — Performance

ID	Severity	Finding	Status
E-1	Medium	Launch grid does N+1 chain reads (~10 per card × page); "one multicall" comment is really JSON-RPC batching	Open

ID	Severity	Finding	Status
E-2	Medium	Reads are not block-pinned — the footer's provenance block is a different block from the data	Open
E-3	Low	`readLaunches` refetches `offset+limit` from every registry per page	Open (won't scale)
E-4	Low	Per-instance locker memo does a genesis→head log scan on a cold instance	Open (acceptable)

E-2 matters for this project specifically. The module states every figure on screen comes from the chain "at a known block, which is also why the block number is rendered in the footer." In fact each read defaults to `latest` and the footer block is fetched separately, so under load a card's price/reserve/progress figures can straddle 0.1-second blocks and the displayed block is not the one they came from — a figure that cannot be checked against the block it claims. For a platform whose whole pitch is verifiability, pinning every read to one block (which E-1's multicall fix also enables) is worth doing.

10. Section F — Prediction market

The prediction market is a Hood-native product: binary, parimutuel markets on questions about The Hood's own launches ("will this launch graduate", "will it have raised X"), staked in ETH or in a project token that has proved its distribution (no stablecoin exists on this chain to list), holding real staked funds. It is **not tied to Seedify** in any way. It was audited in full — six contracts, their interfaces and the predicate library — with two new adversarial suites (`test/predict/PentestOracle.t.sol` , `test/predict/PentestCensus.t.sol`) joining the 181 tests it already carried. *(Status update: the prediction-market contracts were deployed to Robinhood Chain mainnet on 15 August 2026, wired to the launch factory, after the fixes recorded in this section landed.)*

The money mechanics are sound. The parimutuel accounting was attacked and holds: payouts plus fee never exceed the pot, truncation dust stays in the contract and is never paid twice, a void or a one-sided book refunds every staker at par with no fee, double-claims revert, stakes are credited by measured balance delta (fee-on-transfer safe), and payments are pull-only behind `nonReentrant` with checks-effects-interactions. The authenticity model holds too: a market cannot be written about a fake curve, a registered market's subject is frozen, and there is no way to grant, buy or forge a creator tier.

Two subsystems were exploitable, each proven with a working PoC; both are fixed and regression-tested (see the Status note at the end of this section).

ID	Severity	Finding	Status
F-1	**Critical**	Non-monotonic curve metrics are flash-manipulable at settlement — flip the outcome, steal the pot	**Fixed** *(PoC + regression)*
F-2	High	`graduated` can be forced NO->YES for gas when a curve is complete-but-not-graduated	**Fixed** *(PoC + regression)*
F-3	High	The anti-sybil holder census double-counts across sweep pages — self-list a controlled token	**Fixed** *(PoC + regression)*
F-4	High	`submitHolders` checks only instantaneous balance — cheap census stuffing / DoS, and the enabler for F-3	**Fixed** *(with F-3)*
F-5	Medium	Creator-tier flash-manipulation via `syncAndSettle` bypasses the freshness check	**Fixed** *(tier now non-settleable)*
F-6	Medium	`sweep` / `pruneHolder` ignore exclusions — an excluded venue, once submitted, is counted forever	**Fixed**
F-7	Medium	External graduation venue (V4 pool / adapter / locker) is not auto-excluded from the census	Open
F-8	Low	Monotonic creator counts can be bumped within the settlement window (graduate/create)	Open
F-9	Low	Listing ladder enforces monotonicity only on `minHolders`, not covered-supply or net-raise	Open
F-10	Low	Listing ladder has no timelock, unlike the creator-tier ladder	Open
F-11	Info	Three informational items (uncapped counted-supply, eligibility-view underflow, floor lock-in)	Open

F-1 — Settlement is a manipulable spot read (Critical)

Settlement is permissionless and reads the subject curve's **live** state at a block the settler chooses inside the window. The curve metrics split in two: `graduated` and `curveComplete` are latches (safe — they never reverse), but `NET_RAISED_WEI`, `TOKENS_SOLD`, `PROGRESS_BPS` and `FDV` are **non-monotonic** — the library's own comment on net-raised says "not monotonic: selling back into the curve lowers it." If the subject curve is still trading during the settlement window, a losing-side staker buys the curve to shove the metric across the threshold, settles at that instant, then sells back.

Proven end to end. An attacker staked the false side, and after the deadline ran `curve.buy -> oracle.settle -> curve.sell` and claimed:

```
attacker start balance: 100.000 ETH
attacker end balance  : 104.984 ETH
attacker PROFIT       :   4.984 ETH   <- the honest, correct counterparty's stake
```

The honest staker predicted correctly and lost their stake anyway. Cost to the attacker is the curve's round-trip fee (~4% of the ETH moved, and they target markets sitting near the threshold so little is needed); reward is the whole opposing pool, uncut on native-ETH markets. **Fix:** do not settle a pushable metric on a single live read — either restrict settleable metrics to the latched set (`graduated`, `curveComplete`), or snapshot the value at the deadline (a permissionless capture that can only record at/after the deadline), or require the subject curve to be in a terminal state before a non-monotonic metric may settle.

Fixed. `LaunchOutcomeOracle.registerMarket` now rejects any metric that is not in the manipulation-resistant set (`LaunchPredicates.isSettleable`): only the latched lifecycle metrics (`graduated`, `complete`) and the monotonic creator counts can settle a market. The non-monotonic spot metrics (net raised, tokens sold, progress, FDV) and the tier derived from them revert at market creation with `MetricNotSettleable`. The exploit's market can no longer be created; the regression test asserts each rejected metric reverts, and that the safe metrics still work.

F-2 — `graduated` can be forced YES for gas (High)

`graduated` is a latch, so it cannot be flipped YES->NO — but it can be forced the other way almost for free. A curve that is `complete` but not yet `graduated` at the deadline honestly reads `graduated == 0`, and `graduate()` is permissionless. A YES staker calls `curve.graduate()` then `settle()` in one transaction and takes the NO pool for the price of gas. Proven: attacker profit was the entire 5 ETH NO pool. **Fix:** treat `complete` at the deadline as terminal for the `graduated` question, or snapshot at the deadline as in F-1.

Fixed. The `graduated` question now resolves from *completion* rather than the bare `graduated()` flag: `_readMetric` returns true when the curve is `complete` (or graduated). Completion requires the full real raise and never reverses, so it cannot be forced for gas inside the window the way the permissionless `graduate()` call could. The regression test shows an incomplete curve stays NO and a complete curve settles YES from completion alone, with the attacker's in-window `graduate()` a no-op.

F-3 — The anti-sybil census double-counts across sweep pages (High)

Listing a project token as a settlement currency is gated by a holder census: enough distinct addresses holding at least a floor, covering a minimum share of supply. The `sweep` that commits those two numbers reads each member's balance **live, per page**, and carries the running totals across pages in storage. The member set is frozen mid-sweep; the token is not. So the same floor of tokens can be walked from an already-counted wallet into a not-yet-counted one between page transactions, and counted again.

Proven: eight wallets holding **one floor between them** committed a census of **8 holders and 8x floor of covered supply**. It scales linearly — 2,000 such wallets clear the top tier's 20% covered-supply bar while holding 0.01% of supply, letting a creator self-list a token they wholly control. Companion weakness F-4: `submitHolders` checks balance only at submit time, so the wallets are cheap to create and the census is cheap to stuff (a griefing DoS in its own right). **Fix:** pin each member's balance for the duration of a sweep and count `min(live, submitted)` once per member, or require the sweep to complete atomically; cap counted supply at total supply as defence-in-depth; honour exclusions in the sweep (F-6).

Fixed. The sweep is now **atomic**: it counts the whole member set from one instant of chain state in a single transaction, and a partial page reverts (`SweepMustBeAtomic`). With no cross-transaction window, the same floor of tokens cannot be walked ahead of the cursor — a wallet counts only if it holds the floor at that instant, so faking N holders costs N×floor of genuinely distributed supply. The fix also skips excluded members in the sweep and lets `pruneHolder` remove them (F-6), and caps counted supply at total supply (Info-2). The regression test that once reported 8 holders for one floor of tokens now reports 1. F-5 (creator-tier manipulation) is closed by the F-1 gate, which makes tier a non-settleable metric. Cost: a census must sweep in one transaction, bounding member count to what a block's gas allows — several thousand on this chain, past the covered-supply bars.

The rest

F-5 through F-11 are the medium and low items in the table above — creator-tier flash-manipulation via the mandatory at-settle sync, exclusions not honoured on sweep/prune, the external graduation venue not auto-excluded, in-window monotonic count bumps, and two listing-ladder inconsistencies (monotonicity enforced on only one of three dimensions; no timelock where the creator-tier ladder has one). Each has a file-and-line reference and a fix in the working notes.

Status

F-1 through F-6 are fixed and regression-tested — every Critical, High and the two settlement/census mediums. The chosen census remediation is the atomic single-transaction sweep (member count bounded to a block's gas, past the covered-supply bars) rather than adding balance snapshots to the launch token. F-7 through F-11 remain open: the external graduation venue is not auto-excluded at census open (F-7, mitigated by the owner exclusion + the sweep now honouring it), in-window monotonic count bumps (F-8, largely inherent — they reflect genuine permanent activity), and two listing-ladder inconsistencies (F-9/F-10), plus the informational items (F-11). The five prediction-market contracts were deployed to Robinhood Chain mainnet on 15 August 2026, wired to the launch factory, after these fixes landed.

---

11. Centralization & privilege

Unchanged from the contract assessment and worth restating for a public reader. The factory owner can: change parameters for **future** launches only (bounded by hard ceilings); change the graduation

adapter for future launches **behind a two-day public timelock**; change the treasury for future launches; pause **new launches only**; create labelled test launches; and hand over ownership in two steps.

The owner **cannot** touch a live curve's terms, redirect fees already accrued, pause or freeze trading, move any user's funds or tokens, or remove liquidity from a graduated pool — no code path exists for any of those, for any caller. The residual trust is that a future adapter, once its timelock elapses, receives graduating launches' funds; the timelock makes that observable for two days before it can take effect. This is disclosed, not eliminated.

12. Controls that held

Across six contract passes, **65 adversarial tests** encode attacks that were run and failed. Highlights:

- **Contracts:** curve solvency holds under every reentrancy attempt; fee splits never exceed the fee; the locker's principal survives NFT-transfer, eight extraction-shaped selectors and a re-bind; token and ETH donations cannot move the price or complete the curve; launches are fully isolated; a foreign token cannot be sold into a curve; the funded-squat attack is unreachable because transfers are locked until graduation; graduation cannot run twice; the per-launch target holds on every rung.
- **Frontend:** no XSS in the report pipeline; client-side redaction is byte-identical to what is sent; auth headers structurally cannot be captured; every standing UI rule passes.
- **API:** no IDOR (no public read path at all), no enumeration, strong session auth, on-chain per-request admin authority that refuses rather than falls open, no SQL injection, CSRF-safe, secrets never logged or returned.
- **Database:** wei as `Decimal(78,0)`, insert-only ledgers with idempotency anchors, IP never persisted, the `problem_reports` migration purely additive.

13. Test coverage

Suite	Tests	Focus
`Pentest.t.sol`	10	Griefing, reentrancy, donation, access control, rounding
`Pentest2.t.sol`	22	Centralization, fund loss, manipulation, config integrity
`Pentest3.t.sol`	13	Orphaned pool, donation accounting, reentrancy, sequencing
`Pentest4.t.sol`	11	Boundaries, launch isolation, griefing, accounting identities
`Pentest6.t.sol`	9	The per-launch target ladder, fee derivation, dev exemption

Suite	Tests	Focus
Fork suites (live V4)	—	Pool squatting, callback entry, principal extraction, funded squat, sequencing
`predict/PentestOracle.t.sol`	7	Oracle manipulation, forced graduation, refunds, solvency
`predict/PentestCensus.t.sol`	2	Anti-sybil census double-count across sweep pages
Behavioural, fuzz, invariant	352	The pre-existing suite (launchpad + predict)
Total (this run)	**417 pass**	Zero failures, zero static-analysis warnings

14. Reproducing this assessment

```
cd contracts
export PATH="$HOME/.foundry/bin:$PATH"

# All security passes + the full suite.
forge test --no-match-path "test/fork/*"        # 408 pass
forge lint src/ script/                          # 0 warnings

# The exploits, against live Uniswap V4 on a mainnet fork.
FORK_TESTS=1 forge test --match-path "test/fork/*" -vv
```

Application findings were established by reading the named files; each carries a `file:line` anchor in the sections above. D-1 reproduces directly: `grep -rn "CREATE TABLE" packages/accounts/prisma/migrations` shows no indexer table, while `packages/indexer/src/persist.ts` calls `prisma.indexedLaunch.upsert`.