

Security Audit Report

Independent line-level review of the ZXO farming contracts prior to mainnet launch. Scope is theft of user principal, unauthorized minting outside the emission caps, permanent freeze of unstake, and owner reach into assets the protocol is not supposed to touch. A follow-up review will bind this source to creation tx hashes and deployed addresses after launch.

Protocol	Solidity	Review date
ZXO V3 farm	0.8.36	26 September 2026
Status	Compiler settings	Classification
Pre-mainnet	solc 0.8.36 · shanghai · 200 runs · no via-IR	No principal-theft finding

PASS – SOURCE

No permissionless path was found that seizes another user's staked V3 NFT or the tokens inside it. Unstake remains available after the one-day lock regardless of pool health. Reward tokens are minted by the farm contract against immutable per-pool caps; they are not a shared pot of user funds. Owner cannot collect user positions, cannot raise caps after `createPool1`, and cannot rescue anything held by `Zxo` (by design). Findings opened in the first pass (M-01, L-01, L-08, L-09, L-10) are accepted or intended per the team response of 26 September 2026. Remaining items are informational notes on operations and comments, not security defects.

CONTENTS

1. Scope and method
2. Architecture
3. Security invariants
4. Access control
5. Reward accounting
6. Uniswap V3 / fork integration
7. Multichain and ChainLib
8. Findings
9. Team-accepted risks
10. Attack paths that fail
11. Launch checklist
12. Conclusion

1. Scope and method

Sources were taken from the team archive `zxo-sol-audit.zip` and stored at `artifacts/zxo-sol-audit/`. This is a source-only review. No contracts in this tree are treated as canonical mainnet deployments. The team will commission a second audit after launch to match creation bytecode, constructor arguments, and Sourcify / explorer verification to the addresses that actually go live.

In scope

- `contracts/Zxo.sol` – pool factory, stake / harvest / unstake, emission accountant
- `contracts/libraries/ZxoLib.sol` – ticks, packing, bonus, caps, validations
- `contracts/ZxoTreasury.sol` – fee sink with delayed destination change
- `contracts/ZxoV3Deployer.sol` – satellite DEX seeding and finalize/burn
- `contracts/ZxoVerify.sol` – constructor-time ChainLib sanity checks
- `contracts/tokens/ZxoToken.sol` – per-pool reward token, mint locked to Zxo
- Per-chain `ChainLib` files and the interfaces under `contracts/interfaces/`

Out of scope

- Frontend, deploy keys, and RPC configuration in `deploy.py` / `run.sh`
- Uniswap, Pancake, Liberty, or other V3 fork internals beyond the calls ZXO makes
- Economic design of APR targets, the 20% / 25% bonus curve, and token distribution off-chain
- Formal verification, fuzzing, and invariant tests (none were supplied)
- Matching deployed bytecode – explicitly deferred until after mainnet launch

Method: line-level review of every external and internal path in the five contracts and the library, with particular attention to NFT custody, MasterChef-style accumulator math, the health gate, initialization order, mint authority, and owner reach. Plain `revert()` with no data is treated as intended (gas / size), not as a defect. Team-supplied accepted risks were re-checked against the source and recorded as accepted rather than re-litigated.

2. Architecture

ZXO is a concentrated-liquidity farm. Each pool is a Uniswap-V3-style pair of a newly deployed `ZxoToken` and an externally supplied pair token (6, 8, or 18 decimals). Users do not deposit raw tokens into a vault. They approve the farm, which mints a fee-tier-100 position NFT to itself, books the position as a stake, and later returns that same NFT on unstake.

Zxo

Ownable2Step + ReentrancyGuard. Holds user NFTs and the core LP / core-stake NFTs. Only minter of each pool's `ZxoToken`. Chain-locked at construction via `ChainLib.CHAIN_ID`.

ZxoToken

ERC-20 + burnable. Ownership is the Zxo contract and cannot be transferred or renounced. `coreLP` / `coreStake` / `pairing` are immutable. No supply cap; emissions are capped in Zxo, not in the token.

ZxoTreasury

Receives core-LP trading fees and default referral mints. Owner sweeps to `destination`. Destination change is 7-day delayed and can be locked forever. Ownable is single-step (accepted).

ZxoV3Deployer

Owner seeds single-sided positions on arbitrary V3 position managers and sends the NFT to `0x...dEaD`. `finalizeToken` burns leftovers and permanently closes seeding for that token. `setZxoContract` is one-shot.

ZxoVerify

Throwaway constructor probe: factory owner + fee-tier 100 tick spacing, position-manager factory pointer, holder token supply, treasury marker `== 1`, optional deployer marker `== 2`.

ChainLib

Compile-time constants: chain id, holder token, treasury, V3 deployer, position manager, factory. `Zxo.sol` and `ZxoV3Deployer.sol` currently import `SepoliaTestLib.sol`. Production builds must swap the import.

Lifecycle of a pool

1. Owner calls `createPool`. A `ZxoToken` is deployed. Unless `withoutPool`, a `PoolInfo` row is pushed, caps are packed, `coreLP + coreStake` is minted to

Zxo, and `v3StartAmnt` is minted to the deployer.

2. Owner (and/or deployer) seeds satellite DEX pools via `startV3pool`, then `finalizeToken`.
3. Owner calls `initializePool(poolId, slip, true)` – creates the canonical factory pool at the launch sqrt price if needed, requires slot0 to equal that price, mints the single-sided core LP, stores its tokenId in the packed config.
4. Owner calls `initializePool(poolId, slip, false)` – requires finalize, mints the narrower core-stake position, books it as a stake owned by Zxo itself. `stakes` becomes 1. User staking opens.
5. Users `stake` / `harvest` / `unstake`. Auto-adjust retargets `rewardRate` at most once per hour against `targetAprBps`, `maxNormalAPR` (2× during the first year), and `max30dRate`.

3. Security invariants

INVARIANT	STATUS	NOTES
User principal is the V3 NFT, not an internal balance the owner can write	HOLD S	Mint recipient is always <code>address(this)</code> . Unstake is the only path that transfers that NFT, and only to a non-Zxo address after the lock.
Unstake cannot be frozen by price, owner, or health gate	HOLD S	No pause. <code>_requireFairRatio</code> is not on unstake. Pending rewards are forfeited, not required, if the user passes <code>acceptForfeit</code> .
Only the position owner can harvest or unstake it	HOLD S	<code>pos.owner != msg.sender</code> reverts. Core stake is owned by Zxo and has no function that calls harvest/unstake as the contract.
ZxoToken mint authority is permanently the farm	HOLD S	<code>transferOwnership</code> / <code>renounceOwnership</code> revert with custom errors even when called by the owner.
Per-pool emission ceilings cannot be raised after createPool	HOLD S	<code>maxNormalAPR</code> and <code>max30dRate</code> live in packed <code>poolConfig</code> and have no setter. Type-2 config can only lower <code>rewardRate</code> .
Owner cannot sweep user NFTs, pair tokens, or reward tokens out of Zxo	HOLD S	No rescue. <code>collectFeesFromCoreLP</code> collects only the packed core tokenId and sends to treasury, not owner.
Referral mint is a split of the user's pending, not an extra uncapped mint	HOLD S	Total minted on claim equals <code>pending</code> . Referral share is 3–30% of that pending.
User staking cannot open before core LP + core stake + token finalize	HOLD S	<code>stake</code> requires <code>stakes != 0</code> . Only core-stake init increments <code>stakes</code> from zero. Core-stake init requires a non-zero core tokenId and <code>isFinalized(rewardToken)</code> .
Health gate and bonus use spot tick, not TWAP	ACCE PTED	M-02. Rewards still accrue while out of range; the gate only blocks claim/collect and the stake bonus.

4. Access control

Zxo — owner

- `createPool` — deploy token, optionally register pool, mint bootstrap supply. Pair decimals must match packed params and be 6, 8, or 18. `maxNormalAPR` capped at 32_000 bps.
- `initializePool` — one-shot core LP, then one-shot core stake. Reverts if already done or if users have already staked.
- `updatePoolConfig` — (0) set target APR, clipped to the immutable cap / 2× cap during bonus year; (1) referral rate in [300, 3000] bps; (2) strictly lower `rewardRate` and stamp cooldown.
- `collectFeesFromCoreLP` — core LP fees to `ChainLib.ZXO_TREASURY` only.

Owner cannot: raise caps, seize positions, pause unstake, change packed decimals / bonus end / max30dRate, mint except through `createPool` bootstrap and the public claim path, or recover dust / stray NFTs.

ZxoV3Deployer — owner

- `setZxoContract` once. Wrong address is unrecoverable.
- `startV3pool` until the deposit-side token is finalized. Position manager address is a free parameter.
- `burnTokens` / `finalizeToken` — require the token to implement OZ `burn(uint256)`.
- `renounceValidator` — available on any chain where validator is non-zero (constructor sets validator only when `CHAIN_ID == 943`).

`receiveTokens` is validator + testnet only. Mainnet constructor sets validator to zero and `IS_TESTNET = false`, so that path is dead after a production compile.

ZxoTreasury — owner

Sweep to current destination at any time. Schedule a new destination (not zero) with a 7-day delay, execute after the delay, or lock the destination forever and clear any pending change. Receive/fallback reject ETH.

Trust assumption

Owner is trusted for bootstrap (pair choice, core sizes, `v3StartAmnt`, satellite DEX list, launch bundling) and for ongoing target-APR / referral-rate policy inside the immutable caps. Owner is not trusted with user principal. A compromised owner after finalize can still open new pools and mint bootstrap supply for those new tokens, and can drain the treasury via `sweepToken` to the current destination. Production owner should be a multisig or timelock; treasury destination should be moved off the deployer EOA and then locked.

5. Reward accounting

The farm is a MasterChef accumulator. `accRewardPerShare` grows by $\frac{\text{elapsed} * \text{rewardRate} * 1e18}{\text{totalStaked}}$. A position's pending is $\frac{\text{liquidity} * \text{acc}}{1e18} - \text{rewardDebt}$. `rewardRate` is retargeted from TVL (`totalTokens`, 18-decimal normalized principal) $\times \frac{\text{targetAprBps}}{\text{seconds-per-year}}$, then clipped by the packed APR cap and the packed 30-day token cap.

`totalStaked` is Uniswap V3 liquidity, plus the stake bonus (up to 20%, 25% with holder). `totalTokens` is normalized token notionals. They are different units. That is intentional: emissions follow TVL, distribution follows liquidity weight. The core stake is included in both, never claims, and is minted on a narrower range, so it carries more weight per token than a user range (accepted M-03). Users therefore earn below headline `targetAprBps` until user TVL dominates.

Accrual uses `unchecked` with a comment that intermediates stay inside uint256 given the caps and the permanent core stake. Under those caps the product $\frac{\text{elapsed} * \text{rewardRate} * 1e18}{\text{totalStaked}}$ does not overflow uint256 on any realistic horizon. `rewardDebt` is stored as uint128 via `_safe128`; a pathological "tiny liquidity, enormous acc" state would revert on write rather than wrap.

Auto-adjust cooldown is one hour and lives inside `_performAutoAdjust`. First adjust after core-stake init happens on the first user stake or an explicit `updatePool`, because `lastAutoUpdate` starts at 0. Rate is not applied retroactively to the seconds between core-stake and that first adjust.

Claim mints the pending amount of that pool's ZxoToken. If the stored referrer is non-zero and not the claimer, `refRateBps` of pending is minted to the referrer and the rest to the user. Unstake never mints; pending is emitted in the event and discarded.

6. Uniswap V3 / fork integration

- Fee tier is fixed at 100 (0.01%). Tick spacing on official Uniswap / Pancake / Liberty V3 at that tier is 1, so the hardcoded ticks (11, 300, 276024, ...) are legal.
- Stake range for 18-dec pairs is `[-300, 300]`. 6- and 8-dec pairs use the documented parity tick ± 300 , flipped when ZXO is token0.
- Core LP is single-sided just off parity. Core stake is the same side, tightened by 20 ticks. Satellite seeds in the deployer widen the far tick to ± 887272 .
- `_readSlot0` staticcalls `slot0()` and decodes only the first two words so Algebra-style extra return fields do not break the health gate. Trailing ABI words are ignored by solc 0.8 `abi.decode`.
- Pool lookup is `ChainLib.FACTORY.getPool(token0, token1, 100)` not the position manager. That is the correct host for Uniswap, Pancake V3, and Liberty V3.
- Health is an open interval on the *stake* ticks, slightly stricter than Uniswap's `[lower, upper)` in-range rule at the exact boundary ticks.
- User slippage is capped at 5_000 bps. Owner slippage on init / seed is not capped; values above 10_000 revert by underflow, 10_000 means a zero minimum.
- Deadline is `block.timestamp + 20 minutes` at execution (accepted L-04). Price protection is the slippage minimum only.
- Approvals use `forceApprove` on user stake (USDT-class pair tokens) and a one-time max approve of the ZxoToken to the position manager at pool creation.
- Fee-on-transfer, rebasing, or fee-on-collect pair tokens are not supported. Collect amount is trusted and transferred 1:1.

7. Multichain and ChainLib

Each production binary is a different compile. `run.sh` runs `update-chain-lib.py` during the deployment process: the operator picks a chain and the script rewrites the `ChainLib` import in `Zxo.sol`, `ZxoV3Deployer.sol`, and `ZxoVerify.sol` to that chain's library (and fills treasury / deployer / holder once those addresses exist). The contract constructors then revert if `block.chainid != ChainLib.CHAIN_ID`, so a leftover test import cannot go live on the wrong network. `ZxoVerify` is the on-chain check that the constants point at live factory / NPM / holder / treasury / deployer contracts. It does not check that `HOLDER_TOKEN` has 18 decimals (accepted) and does not check Liberty- or Pancake-specific extras.

LIB	CHAIN ID	DEX	FACTORY / NPM IN SOURCE	NOTES
EthereumLib	1	Uniswap V3	canonical 0x1F98... / 0xC364...	Treasury / holder / deployer are placeholders until <code>update-chain-lib.py</code> writes them.
OptimismLib	10	Uniswap V3	same CREATE2 addresses as Ethereum	Placeholders until the deploy script fills them.
UnichainLib	130	Uniswap V3	0x1F9840...0003 / 0x943e6e07...	Confirm fee-tier 100 with ZxoVerify after the lib is selected.
PolygonLib	137	Uniswap V3	canonical CREATE2	Placeholders until the deploy script fills them.
PulseChainLib	369	Liberty V3	0x796fcbDC... / 0x19b13479...	Liberty is a Pancake/Uni V3 fork. Factory address matches public Liberty V3 factory.
BNBChainLib	56	Pancake V3	0x0BFbCF9f... / 0x46A15B0b...	Confirm 0.01% tier is enabled via ZxoVerify.
ArbitrumLib	42161	Uniswap V3	canonical CREATE2	Placeholders until the deploy script fills them.

LIB	CHAIN ID	DEX	FACTORY / NPM IN SOURCE	NOTES
BaseLib	8453	Uniswap V3	0x33128a8f... / 0x03a520b3...	NPM address is the documented Base Uniswap NPM.
RobinhoodLib	4663	Uniswap-style	0x1f7d7550... / 0x73991a25...	Chain id matches public Robinhood Chain listings. Re-verify addresses at launch.
SepoliaTestLib	9991	Uniswap (fork)	Ethereum Uniswap addresses	Default checkout import before <code>update-chain-lib.py</code> . Chain id 9991 is a Tenderly-style virtual mainnet, not public Sepolia (11155111).
PulseChainTestLib	943	Uniswap (ETH fork)	Ethereum Uniswap addresses	Correct: Pulse testnet/mainnet forked Ethereum with the existing V3 factory and NPM. Only lib that turns <code>IS_TESTNET</code> on.

Compiling the current tree as-is targets chain 9991. A mainnet deploy from this checkout without swapping ChainLib would revert in the constructor on every other chain. That is the correct fail-closed behavior.

8. Findings

Items opened in the first pass that the team has now answered (M-01, L-01, L-08, L-09, L-10) are recorded in [§9](#) as accepted or intended. I-03, I-05, I-06, and I-07 are intended or not defects (landed tick vs position edge; Pulse Uniswap-at-fork-addresses; address mining for DEXScreener buy-side ordering; ownership renounce when no further pools will be created). I-01, I-02, and I-04 remain informational notes. None of I-01 through I-07 is a theft, freeze, or unauthorized-mint path.

INFO

I-01 — updatePoolConfig types other than 0/1/2 silently succeed

`updatePoolConfig(poolId, dataType, newValue)` has three live branches: `0` writes `targetAprBps` (clipped to the packed cap), `1` writes `refRateBps` (must sit in [300, 3000]), `2` writes a strictly lower `rewardRate` and stamps `lastAutoUpdate`. Anything else (`3`, `255`, a typo) falls out of the if/else chain, emits nothing, changes nothing, and returns success.

That is not an exploit. An attacker cannot use a junk `dataType` to raise caps or mint. The only cost is operational: a mis-typed owner script looks like it worked. A frontend that always sends 0/1/2 never hits this. Adding a final `else revert()` would make a bad script fail closed; it is a convenience, not a security fix.

INFO

I-02 — tokensIds_ is written and never read inside Zxo

On `createPool` the farm stores `tokensIds_[zxoToken] = poolId`. Nothing in `Zxo.sol`, the lib, the deployer, or the token reads that map. It is leftover indexing, likely meant for a UI or an older helper.

Two footnotes, neither of which is a contract bug today:

- Solidity mappings default to `0`. An off-chain caller that treats “key missing” as “not a farm token” will confuse an unknown address with pool 0. The on-chain paths do not use the map that way.
- The slot costs gas on every `createPool` and can never be cleared. Harmless at 255 pools.

Safe to leave. Safe to delete in a future size-sensitive revision. Do not start using it as an existence check without a separate “this token was registered” flag.

INFO INTENDED

I-03 — 18-dec core comment is the landed tick, not the position edge

The 18-decimal comment says core range `[10, 300]`. The compiled position edges are `TICK_MIN_18_DEC = 11` and `TICK_MAX_18_DEC = 300` (mirrored `[-300, -11]` when ZXO is token1). That is not a mismatch to “fix.”

`SQRTPRICE_18_18_LOWER` initializes slot0 so the current tick *lands on 10* when ZXO is token0 (lower). `SQRTPRICE_18_18_UPPER` lands on *-11* when ZXO is token1 (higher). Uniswap in-range is `[tickLower, tickUpper)`, so the core NFT is minted at `[11, 300]` or `[-300, -11]` — one spacing-1 step off the landed tick, on the single-sided side of parity. The comment documents where price sits and how far the range runs, not the raw constant names.

Team disposition: intended. Leave the comment as written. Do not change `TICK_MIN_18_DEC` to 10 to “match” it, and do not rewrite the comment to 11.

INFO

I-04 — Production ChainLibs ship with zero protocol addresses

The checked-in `EthereumLib`, `BaseLib`, `ArbitrumLib`, and the other mainnet files hard-code factory and position-manager addresses, but leave `HOLDER_TOKEN`, `ZX0_TREASURY`, and `ZX0_V3_DEPLOYER` as `address(0)`. That is expected in this tree: those three addresses do not exist until this chain's treasury and deployer have been deployed.

The deploy path is `update-chain-lib.py` (operator picks the chain, the script rewrites the import and the three addresses) → compile → constructor `chainid` check → `ZxoVerify`. Skipping the script and compiling a mainnet lib as-is would produce a farm that calls `balanceOf(address(0))` on stake and sends core-LP fees to the zero address. The constructor would still pass on the matching chain, because it only checks chain id. `ZxoVerify` is the gate that refuses a zero treasury / holder / deployer. Do not skip it; do not compile a production Zxo from a lib the script has not finished writing.

INFO CORRECT / NOT A DEFECT

I-05 — PulseChainTestLib uses Ethereum Uniswap addresses on chain 943

`PulseChainTestLib` sets `CHAIN_ID = 943` and the Ethereum mainnet Uniswap V3 factory / NPM addresses. The same addresses appear conceptually on Pulse mainnet (Liberty is a separate Dex in `PulseChainLib`; the test lib is labeled Uniswap because the Pulse networks are Ethereum forks that retained the pre-fork Uniswap V3 factory and position manager).

Team disposition: expected. PulseChain testnet and mainnet forked Ethereum with those contracts already deployed. This is not a stale copy-paste. `IS_TESTNET` is still keyed only off chain id 943, which is the intended testnet flag.

I-06 — withoutPool tokens are not registered as farms

`createPool(..., withoutPool=true)` deploys a `ZxoToken` owned by Zxo, stores `coreLP` / `coreStake` / `pairing`, and enforces `forceLower`, but does not push `PoolInfo`, pack config, or mint bootstrap. Failed address probes stay at zero supply. The live farm is a later `createPool` with `withoutPool=false` once CREATE lands an address lower than the pair token.

Team disposition: intended. The flag burns CREATE nonces cheaply until the token address sorts *below* its pair. On DEXScreener-class UIs token0 is the base, so a lower hex makes buys render as buys rather than sells. Combined with `forceLower` on the live pool path, this is address mining, not a leftover stub.

I-07 — Renouncing Zxo ownership ossifies admin and leaves users live

`Ownable2Step` still exposes `renounceOwnership`. After it fires:

- Dead: `createPool`, `initializePool`, `updatePoolConfig`, `collectFeesFromCoreLP`.
- Still live: user `stake` / `harvest` / `unstake`, public `updatePool`, auto-adjust, claim mints, NFT return.
- Unchanged: packed caps, token ownership (already locked to Zxo), treasury destination (separate contract).

Team disposition: intended. If the protocol decides there will be no further pools, ownership is renounced. That is the explicit end-state switch, not an accidental footgun the report needs to talk the team out of. Do it after planned pools are initialized; users are not frozen either way.

9. Team-accepted risks (re-validated)

Original accepted list plus the 26 September 2026 dispositions for M-01, L-01, L-08, L-09, and L-10. Each item was re-checked against the source.

MEDIUM **ACCEPTED**

M-01 — Canonical pool can be pre-initialized at the wrong price

`createPool` and `initializePool(core)` are separate calls. After the token exists, anyone can call the factory's `createAndInitializePoolIfNecessary` at a non-launch sqrt price and cause the owner's core init to revert. No funds move to the griever.

Team disposition: accepted. Merging create + init into one function would push Zxo over the 24 576-byte runtime limit. A griever pays gas to pin the pool at a wrong price; the same pool can be left unused and a new pair / new pool opened, or the griever's own liquidity at that price can be competed. Not a principal-theft path. No atomic helper will be added.

LOW **INTENDED**

L-01 — Self-referral or third-party referral pays no treasury commission

First write of `userReferrer_` is permanent. `address(0)` is stored as the treasury. Any other address — the user themselves, a friend, or Zxo — receives the referral split on later claims. When the stored referrer equals the claimer, the contract mints 100% of pending to the user and 0% to anyone else.

Team disposition: intended. Treasury earns a commission only when the user leaves referrer at the default (zero → treasury). Choosing a self-referral or someone else's address is the user directing that share. There is no missing access-control check and no extra mint. First-pass wording that called this a "policy leak" is withdrawn; the policy is opt-in commission, not a mandatory protocol cut.

LOW ACCEPTED

L-08 — startV3pool takes an arbitrary position manager and does not check slot0

Owner-only. Approves the deposit token to `posman` and mints a single-sided seed NFT to `0x...dEaD` with no launch-price check. A compromised owner key can drain the deployer's balance through a fake manager. Seeding is permanently closed per token by `finalizeToken`.

Team disposition: accepted, mitigated by the existing finalize gate. User `stake` cannot open until `initializePool` (core-stake) sees `isFinalized(rewardToken) == true`. By then seeding is frozen, leftover deployer inventory is burned, and anyone can inspect which DEX managers were used and for how much before they deposit. No further on-chain allowlist of position managers will be added.

LOW ACCEPTED / NOT A USER RISK

L-09 — No pause switch

There is no `pause()` on stake, harvest, unstake, or createPool. Type-2 config can only lower an existing pool's `rewardRate`, including to zero. That does not stop a later `createPool` from minting a new token's bootstrap supply.

Team disposition: accepted. The absence of a pause is the feature that keeps unstake available in every state (see also L-05: owner has no rescue / freeze handle). A pause would be a user-facing freeze, which this review treats as worse than leaving admin live. Residual "risk" is only that a hypothetical future mint bug on a new pool cannot be globally stopped without the owner simply not calling `createPool`. Existing user principal does not depend on a pause button.

LOW **ACCEPTED**

L-10 — Owner slippage uncapped; core-stake init does not re-read slot0

Owner `slippageBps` on `initializePool` / `startV3pool` is not clipped to 5_000. Core-LP init already requires `slot0.sqrtPriceX96 == launch price`. Core-stake init does not repeat that check. TVL for the core stake is booked as the full `coreStake()` figure.

Team disposition: accepted. Owner slippage is set by the owner frontend. After core LP is in, the single-sided core is sized as a large sell wall (team figure: over \$1M notional) sitting at the launch tick. Core stake is placed on the same side, above the 1:1 tick, so a move large enough to desynchronize the second mint has to walk that wall. No second slot0 check will be added.

MEDIUM **ACCEPTED**

M-02 — Spot-price health gate and bonus composition

`_requireFairRatio` and `_calcBonusAndTotal` read the pool tick at execution. A user can atomically swap into range (or above parity), stake or harvest, and swap back, paying about two 0.01% fees. Rewards continue to accrue while the gate is closed. Principal is never taken. Caps still bound mint. No TWAP will be added.

MEDIUM **ACCEPTED**

M-03 — Core-stake position

Core stake sits in `totalTokens` and `totalStaked`, cannot claim, and is tighter than the user range (~2.2× weight per token). Users earn below `targetAprBps` by the core's share of weight. Auto-adjust does not close that gap. Core-stake NFT trading fees are uncollectable (no collect on that tokenId). Core LP fees are collectable to treasury via `collectFeesFromCoreLP`.

LOW **INTENDED**

L-03 — Bonus ramp constant 59538

20% cap at ~53.36% paired ratio, about 20 ticks above parity. With a large single-sided core the organic spot is expected near the launch tick (roughly 10–12% bonus). Higher values need flow against the core or M-02-style manipulation.

LOW **ACCEPTED**

L-04 — Deadline = block.timestamp + 20 minutes

Computed at execution, so it never rejects a delayed transaction. Stake / init / seed price protection is the slippage minimum only.

LOW **ACCEPTED (BY DESIGN)**

L-05 — No rescue path for dust or stray NFTs

`onERC721Received` accepts any NFT. Refunds at or below dust stay in Zxo. There is no `withdrawERC20` / `withdrawNFT`. Owner therefore has no path to user assets or to protocol dust. That is the stated reason the function does not exist.

LOW **ACCEPTED**

L-07 — ZxoTreasury uses Ownable (single-step)

A mistaken `transferOwnership` cannot be undone. Destination changes are two-step with a 7-day delay; ownership itself is not. Perform owner ops from a controlled key.

INFO **ACCEPTED**

Referrer may be set to any address, including Zxo

Referral share minted to Zxo is unrecoverable (L-05). Only that user's split is affected. A fresh address can start over. `address(0)` becomes the treasury.

INFO ACCEPTED

HOLDER_TOKEN decimals are not verified

`HOLDER_THRESHOLD` is `3_690_000_000 ether` (18 decimals). The team states the holder token is 18 decimals on every target chain. ZxoVerify only checks `totalSupply() > 0`.

10. Attack paths that fail

- **Steal another user's NFT.** Storage owner check + nonReentrant + transfer only on unstake to a non-Zxo recipient.
- **First-depositor acc donation.** Core stake is the first position, booked at acc = 0 with rewardRate still 0. Users cannot stake while `stakes == 0`.
- **Double unstake / harvest after unstake.** Position is deleted; liquidity-zero and owner checks fail.
- **Harvest out of range without a swap.** Gate reverts. (With a swap, see M-02.)
- **Unstake during the 24h lock.** Reverts. Harvest during the lock is allowed if healthy.
- **Owner mint on an existing pool's token after createPool.** Token owner is Zxo; Zxo only mints bootstrap once and then pending claims. No owner `mint` on Zxo.
- **Raise maxNormalAPR or max30dRate later.** No setter. Type 0 clips target APR to those packed values.
- **Collect user position fees to treasury.** `collectFeesFromCoreLP` uses only `unpackCorePositionId`.
- **Reenter through ERC-721 receiver on unstake.** External call is last; guard is on the external function.
- **Reenter through ERC-20 hooks on stake/harvest.** Same guard. Reward token is a vanilla OZ ERC-20.
- **Overflow poolId past 255.** `uint8(poolInfo.length) == 255` reverts. Maximum 255 pools (ids 0–254).
- **User stake before finalize.** Core-stake init requires `isFinalized`; user stake requires `stakes != 0`, which only core-stake init sets.
- **Core unstake / core harvest by owner EOA.** Position owner is `address(this)`, not the Ownable owner.
- **Deployer drain on mainnet via receiveTokens.** `IS_TESTNET` is false and validator is zero unless ChainLib.CHAIN_ID is 943.
- **Wrong-chain deploy.** Constructor reverts if `block.chainid != ChainLib.CHAIN_ID`.

- **ETH sent to treasury.** receive/fallback revert.

11. Launch checklist (pre-mainnet)

This review does not certify any live address. Before each chain's first user stake:

1. Fill `HOLDER_TOKEN`, `ZXO_TREASURY`, `ZXO_V3_DEPLOYER` in that chain's lib. Confirm holder decimals = 18 off-chain.
2. Run `update-chain-lib.py` so the operator-selected chain rewrites the `ChainLib` import and the three protocol addresses. Confirm the constructor chain-id check matches the RPC.
3. Compile with solc 0.8.36, shanghai (or the chain's actual fork), optimizer 200, via-IR off, matching `compile.py`.
4. Deploy treasury → verify (v3 flag false) → deployer → write deployer into the lib → recompile verify + Zxo → verify (v3 flag true) → deploy Zxo → `setZxoContract` once.
5. Confirm ZxoVerify `isVerified() == true` and factory `feeAmountTickSpacing(100) > 0` on that chain.
6. Move treasury destination to the long-term sink, wait 7 days, execute, consider `lockDestinationForever`.
7. Move Zxo / deployer / treasury ownership to the production multisig (2-step on Zxo and deployer).
8. Finalize the reward token on the deployer before core-stake init (already enforced). M-01 pool-init griefing is accepted; no atomic create+init helper.
9. Record creation tx hash, deployed addresses, compiler settings, and ChainLib hashes for the post-launch matching audit.

12. Conclusion

The ZXO contracts implement a constrained V3 farm rather than a custodial vault. User principal is a position NFT the farm cannot spend except to return it. Reward supply is an owner-gated bootstrap plus a public claim mint bounded by immutable per-pool caps. The owner surface after launch is policy (target APR, referral rate, lower-only rate override, new pools, core-fee sweep to treasury), not custody.

No high or critical finding was opened. First-pass items M-01, L-01, L-08, L-09, and L-10 are accepted or intended: contract-size limit prevents an atomic create+init, referral commission is opt-in, seeding is frozen by finalize before user stake, a pause would freeze users, and core-stake price is defended by the core-LP sell wall plus the owner frontend. Informational notes I-01–I-07 do not change those conclusions.

Prior accepted items M-02, M-03, L-03, L-04, L-05, L-07, free referrer (including Zxo), and unverified holder decimals remain as described. Plain `revert()` is used consistently on failure paths other than ZxoVerify and ZxoToken's ownership locks. Chain selection is performed by `update-chain-lib.py` and enforced again by the constructor chain-id check.

This report is a source opinion dated 26 September 2026. It is not a bytecode match against any deployed address. After mainnet launch, a second review should take each chain's creation transaction, runtime code (immutables masked), and Sourcify / explorer metadata and confirm they are this tree with the intended ChainLib.