SMART CONTRACT SECURITY REVIEW · REVISION 2.5

Zxo Protocol Security Audit

REMEDIATION REVIEW COMPLETE

DATE	COMPILER	SCOPE	DEX MODEL
27 September 2026	`solc 0.8.36`	24 Solidity files · 2,831 lines	Uniswap V3 and forks (Pancake, Liberty)
METHOD	**REVIEWER**		
Manual line review + numeric tick/price verification	Claude (Anthropic), AI-assisted		

01

Executive summary

OVERALL SCORE

8.5/10

~~6.6~~ → 8.5 after remediation

Unweighted mean of ten categories. Conditional on successful compilation and the fork tests listed under Before mainnet.

Zxo is a single-contract liquidity farm. Users stake Uniswap V3 positions minted on a fixed tick range around parity, and earn a newly minted reward token (one ZxoToken per pool). Emissions follow a target APR on staked value. Each pool is anchored by a permanently locked, single-sided core LP that sells the reward token on the way up.

The original review found 2 High, 4 Medium, 7 Low and 10 informational issues. Both High issues and every Medium issue that could be fixed in code have been fixed. One Medium remains, the core-stake weighting (M-03), which the team accepted as a design trade-off. The spot-price stake bonus left over from M-02 is also accepted. Both are recorded in the accepted-risk register.

No path was found that lets anyone take user principal. Unstake has no price-range gate and no owner gate, so staked positions can always be withdrawn after the 1-day lock-up. The core LP has no withdrawal function.

20	8	0	0
Resolved	Accepted / intended	Open High / Medium	Open informational

Limits of this review

The review environment could not install dependencies. The team has since compiled Zxo with solc 0.8.36 and confirmed the explorer verification input reproduces the build bytecode, but no fuzzing, fork tests or live deployment were run as part of this review. This AI-assisted review does not replace an independent human audit. Contracts that will hold user savings should also get a professional re-audit of the diff before mainnet.

Revision 2.5 · 27 September 2026

All open items are closed. SepoliaTestLib now has a complete ZXO_ORACLE line, so the tree builds (I-12). ConfigUpdated now logs the value actually stored, including the capped target APR, from one shared log call (I-13). ZxoVerify has full NatSpec for the deploy order and both constructor flags. No change to runtime logic, funds flow or scores.

Revision 2.4 · 27 September 2026

ZxoVerify now checks ZxoOracle on-chain, behind a second constructor flag (verifyOracle): code present, WINDOW() == 900, and a fail-safe answer (no average, never in range) for an address that is not a pool. PoolInitialized now indexes the reward token, matching PoolCreated; I-13 is narrowed to the ConfigUpdated value. No change to runtime logic, funds flow or scores.

Revision 2.3 · 27 September 2026

The Zxo constructor now validates the full ChainLib configuration with readable error messages, at no runtime bytecode cost. New events record deployment, pool creation, pool initialization, core fee collection and config updates, and updatePoolConfig rejects unknown data types. All chain libraries now declare ZXO_ORACLE. I-11 resolved. Two informational items added (I-12 build, I-13 event data).

Revision 2.2 · 27 September 2026

Adds ZxoOracle: the health gate now also requires a 15-minute TWAP inside the stake range, which closes the same-transaction harvest bypass (M-02 resolved; the spot-based bonus remains accepted). Events now go through one shared function, redundant library checks and the write-only tokensIds_ mapping were removed, uint64 casts use a checked downcast, and the Zxo constructor validates the HOLDER settings. Revision 2.1 moved HOLDER_THRESHOLD into each chain library and added HOLDER checks to ZxoVerify. One informational item added (I-11, oracle setup).

02

Category scores

Original audit · After remediation · Scale 0–10

Reentrancy and CEI

~~9.5~~ **9.8**

Every mutating entry point is nonReentrant. Unstake now makes its external NFT transfer last, after all state is final.

Access control

~~7.0~~ **8.5**

Finalize flag and set-once ZXO_CONTRACT close the deployer path. Posman stays arbitrary until finalization, and the treasury uses single-step Ownable.

Token supply controls

~~6.0~~ **8.5**

burn(address) removed. Seeding is capped by finalization, and staking can't open until the token is finalized. v3StartAmnt remains uncapped but visible on-chain.

Reward accounting

~~6.0~~ **7.5**

TVL now counts every stake (M-01). The core stake still takes an unclaimed share of emissions (M-03, accepted).

V3 integration and LP handling

~~7.0~~ **9.0**

Fork-agnostic slot0 read, launch-price check, pair-decimals validation. All tick constants verified.

Price and oracle dependence

~~4.5~~ **7.0**

The health gate requires the spot price and a 15-minute TWAP to be in range (M-02). The bonus still uses the spot token mix.

External token compatibility

~~6.0~~ **8.5**

forceApprove supports USDT-style tokens. Fee-on-transfer and rebasing pair tokens are unsupported by policy.

DoS, gas and liveness

~~8.5~~ **9.0**

Unstake accepts a recipient. Fail-safe totals can't block exits. Loops are bounded.

Deployment and configuration

~~6.5~~ **9.0**

The Zxo constructor validates every ChainLib address, the DEX wiring, fee tier and tick spacing, the HOLDER token, and the treasury, deployer and oracle contracts before deployment can succeed. compile.py and deploy.py add build-time and chain-id checks

Code quality and docs

~~7.0~~ **8.5**

NatSpec matches the code. Dead code removed, out-of-range queries revert, downcasts are checked, and deployment settings are validated in the constructor. Some redundant guards are kept by choice.

03

Findings and status

SEVERITY	TOTAL	RESOLVED	ACCEPTED / INTENDED	OPEN
HIGH	2	2	0	0
MEDIUM	4	3	1	0
LOW	9	4	5	0
INFO	13	11	2	0

H-01 Owner could take pre-minted Zxo through an arbitrary position manager

HIGH RESOLVED

LOCATION	`ZxoV3Deployer.startV3pool, Zxo.createPool`
ISSUE	startV3pool approved any posman address for the seeding amount. A malicious posman could transferFrom the tokens during mint(). setZxoContract could also be repointed to a fake pool source.
RESOLUTION	Added isFinalized and finalizeToken, which burn the remainder and permanently block seeding for that token. setZxoContract can be set only once. initializePool(core stake) requires isFinalized, so staking can't open before seeding is closed.
RESIDUAL	Low. Posman is still arbitrary during the seeding window, before any user funds arrive. Users should confirm that every PoolStarted posman is canonical before staking.

H-02 slot0() decode reverted on PancakeSwap V3

HIGH RESOLVED

LOCATION	`Zxo._poolRatio`
ISSUE	The typed IUniswapV3Pool call decoded feeProtocol as uint8. Pancake stores a packed uint32 there, so decoding always reverted, which would have blocked stake and harvest on BNB.
RESOLUTION	Replaced with _readSlot0: a low-level staticcall that decodes only sqrtPriceX96 and tick, the fields every V3 fork shares. The Liberty branch and pool imports were removed. A failed read returns unhealthy.
RESIDUAL	None. Fork-test on each target chain.

M-01 TVL counted only stakes that qualified for a bonus

MEDIUM RESOLVED

LOCATION	`ZxoLib._calcBonusAndTotal`
ISSUE	Early returns sent back totalAmount = 0 for deposits at or below parity. Those stakes added reward weight but no TVL, which lowered emissions and inflated the displayed APR.
RESOLUTION	totalAmount is now computed before every early return.
RESIDUAL	Emissions for the same targetApr are higher than before. Recheck the caps per pool.

M-02 Health gate and bonus read the spot price

MEDIUM RESOLVED

LOCATION	`Zxo._poolRatio, ZxoOracle, ZxoLib._calcBonusAndTotal`
ISSUE	An atomic swap, then stake or harvest, then swap back could bypass the harvest gate or get the maximum bonus for about two 0.01% fees.
RESOLUTION	Revision 2.2: the health gate now requires both the spot tick and a 15-minute time-weighted average tick (ZxoOracle.inRange, read through observe()) to be inside the stake range. A same-transaction swap cannot move the average, so the harvest-gate bypass is closed. The oracle is only called when the spot check passes.
RESIDUAL	The stake bonus is still computed from the deposit's token mix at the spot price, so a same-transaction swap can still raise one stake's bonus (capped at 20%, 25% with HOLDER). Accepted: emissions stay capped and principal is not at risk. Optional next step: derive the bonus from the TWAP tick.

M-03 Core-stake position dilutes user rewards and its fees can't be collected

MEDIUM ACCEPTED

LOCATION	`Zxo.initializePool (core stake)`
ISSUE	The contract-owned core stake counts toward TVL and reward weight, with about 2.2× weight per token, but never claims. Users receive less than targetApr by roughly its share of totalStaked. Its trading fees are locked.
RESOLUTION	Accepted. Auto-adjust does not close this gap. The gap shrinks as user TVL grows, and targetAprBps can compensate.
RESIDUAL	Steady-state APR gap. Minor locked fees.

M-04 USDT-style approve reverted

MEDIUM RESOLVED

LOCATION	`Zxo.stake, ZxoV3Deployer.startV3pool`
ISSUE	A raw IERC20.approve expects a bool return value. USDT returns none, so staking a USDT pair was impossible.
RESOLUTION	Switched to SafeERC20.forceApprove after the transfer in, in both contracts.
RESIDUAL	None.

L-01 Launch price not verified on pool initialization

LOW RESOLVED

LOCATION	`Zxo.initializePool`
ISSUE	Anyone can pre-create the pool for the predictable token address at another price, which would put the core LP at the wrong price or block it.
RESOLUTION	After createAndInitializePoolIfNecessary, the pool's slot0 price must equal the launch sqrtPrice, or initialization reverts.
RESIDUAL	Recovery: a swap with sqrtPriceLimitX96 = launch price.

L-02 Pair decimals not validated

LOW RESOLVED

LOCATION	`Zxo.createPool`
ISSUE	The decimals parameter drives every tick and normalization calculation but wasn't checked against the pair token.
RESOLUTION	createPool reads IERC20Metadata.decimals() and requires a match with the parameter and a value of 6, 8 or 18.
RESIDUAL	None.

L-03 Bonus ramp reaches its cap about 20 ticks above parity

LOW INTENDED

LOCATION	`ZxoLib._calcBonusAndTotal`
ISSUE	The constant 59538 hits the 20% cap at a 53.36% paired ratio, so the bonus is close to all-or-nothing.
RESOLUTION	Intended. At the launch tick the organic bonus is about 10–12%.
RESIDUAL	Documented.

L-04 No user-supplied deadline

LOW ACCEPTED

LOCATION	`Zxo.stake, initializePool, ZxoV3Deployer.startV3pool`
ISSUE	block.timestamp + 20 minutes is evaluated at execution, so a delayed transaction still runs.
RESOLUTION	Accepted. Price protection comes from the slippage minimums.
RESIDUAL	Documented.

L-05 No rescue path for dust or stray NFTs

LOW ACCEPTED

LOCATION	`Zxo`
ISSUE	Refunds at or below the dust threshold, and NFTs sent directly to Zxo, can't be recovered.
RESOLUTION	Accepted by design. No rescue function exists, so the owner has no path to any asset Zxo holds.
RESIDUAL	Documented.

L-06 ZxoVerify checked the wrong fee tier

LOW RESOLVED

LOCATION	`ZxoVerify`
ISSUE	It checked feeAmountTickSpacing(500) while the protocol uses the 100 tier.
RESOLUTION	It now checks the 100 tier. The deployer-address check is conditional on verifyV3Deployer.
RESIDUAL	None.

L-07 Treasury uses single-step ownership

LOW ACCEPTED

LOCATION	`ZxoTreasury`
ISSUE	A mistaken transferOwnership can't be undone.
RESOLUTION	Accepted. Owner operations are handled carefully.
RESIDUAL	Documented.

L-08 Contract stakers without an NFT receiver could never unstake

LOW RESOLVED

LOCATION	`Zxo.unstake`
ISSUE	safeTransferFrom to msg.sender reverts for contracts, including some EIP-7702 wallets, that don't implement onERC721Received. That would lock their position permanently.
RESOLUTION	Added a to recipient (address(0) means msg.sender), a to == address(this) guard, and moved the transfer to the end. A failed transfer rolls back the whole unstake.
RESIDUAL	None.

L-09 Referrer can be set to the Zxo contract

LOW ACCEPTED

LOCATION	`Zxo._setRef`
ISSUE	The referral share would be minted to Zxo and be unrecoverable. Only that user's own share is affected.
RESOLUTION	Accepted. Any address may be a referrer, and address(0) defaults to the treasury.
RESIDUAL	Documented.

I-01 Counter overflow on stats

INFO RESOLVED

LOCATION	`Zxo`
ISSUE	totalGlobalStakes and referralsPerUser_ were uint16, and the referral counter wrapped silently.
RESOLUTION	Both widened to uint32.
RESIDUAL	None.

I-02 Silent uint16 truncation in updatePoolConfig

INFO RESOLVED

LOCATION	`Zxo.updatePoolConfig`
ISSUE	uint16(newValue) truncated oversized input, for example 70000 became 4464.
RESOLUTION	Added ZxoLib._safe16, which reverts on overflow.
RESIDUAL	None.

I-03 Redundant guards kept

INFO ACCEPTED

LOCATION	`Zxo, ZxoLib`
ISSUE	Some checks can't be reached or are no-ops: userReferrer_[this], the rewardDebt write when pending == 0, and several library guards. The unreachable queryData type-3 branch was removed in Revision 2.1 (verified: that branch can only see type 0 or 3).
RESOLUTION	Kept by choice as defense in depth. Unused enums, ownerOf checks and zero-token checks were removed.
RESIDUAL	None.

I-04 unstake truncated a uint256 token id

INFO RESOLVED

LOCATION	`Zxo.unstake`
ISSUE	The id was cast to uint64 to look up the position.
RESOLUTION	The parameter is now uint64, matching harvest.
RESIDUAL	None.

I-05 Dead owner burn on ZxoToken

INFO RESOLVED

LOCATION	`ZxoToken, IZxoToken`
ISSUE	burn(address, uint256) could never be called but looked like a rug vector to scanners.
RESOLUTION	Removed. The NatSpec was updated.
RESIDUAL	None.

I-06 HOLDER_TOKEN decimals not verified

INFO RESOLVED

LOCATION	`ZxoVerify`
ISSUE	HOLDER_THRESHOLD assumes 18 decimals.
RESOLUTION	Revision 2.1: ZxoVerify now requires HOLDER_TOKEN.decimals() == 18 and HOLDER_THRESHOLD ≥ 1 token. The threshold moved into each chain library, so every chain sets its own.
RESIDUAL	None, provided ZxoVerify is run with the same ChainLib as Zxo.

I-07 _unpackPoolConfig returned 0 for an index above 5

INFO RESOLVED

LOCATION	`ZxoLib._unpackPoolConfig`
ISSUE	An out-of-range index fell through and returned 0 from queryData(type 2), which could be misread as no cap or not launched.
RESOLUTION	Added revert() after index 5.
RESIDUAL	None.

I-08 Single enum-coded event

INFO ACCEPTED

LOCATION	`Zxo.ZxoEvent`
ISSUE	Only user is indexed, so indexers must filter on the action value.
RESOLUTION	Accepted. Each separate event adds bytecode, and one shared event keeps Zxo under the 24,576-byte limit while still emitting every action.
RESIDUAL	Indexer ergonomics only.

I-09 Build pipeline hardening

INFO RESOLVED

LOCATION	`compile.py, deploy.py, run.sh`
ISSUE	The contracts imported SepoliaTestLib, compile.py ignored standard-JSON errors, the verification input depended on local files, and run.sh continued after a failed step.
RESOLUTION	compile.py now builds deploy bytecode and explorer input from one standard-JSON compile with import-name source keys, stops on errors, checks EIP-170/3860 sizes, offers an EVM menu (Shanghai for PulseChain), and warns about test libraries or zero addresses. deploy.py checks the RPC chain id against the compiled ChainLib, probes PUSH0/TLOAD, estimates gas, checks receipt status and compares on-chain code with the build. run.sh stops at the first failure. The team confirmed with solc 0.8.36 that the input compiles with no local files and reproduces the build bytecode.
RESIDUAL	The ChainLib must still be switched to the target chain for each production build. The tooling enforces this.

I-10 Holder settings are not checked by Zxo itself

INFO RESOLVED

LOCATION	`Zxo.stake, ChainLib (per chain)`
ISSUE	Revision 2.1 moved HOLDER_THRESHOLD into each ChainLib. Zxo trusts both values without checking them. With HOLDER_THRESHOLD = 0 every staker passes balance ≥ 0 and gets the +25% bonus. With HOLDER_TOKEN = 0x0 the balanceOf call reverts and staking is impossible. Eight mainnet libraries currently have both set to zero; only Arbitrum and the two test libraries are configured.
RESOLUTION	The Zxo constructor now reverts when HOLDER_TOKEN is the zero address or HOLDER_THRESHOLD is below 1 token, so a misconfigured build cannot be deployed. Constructor code is not part of the runtime bytecode, so the check does not count toward the 24,576-byte limit. compile.py also warns at build time and ZxoVerify checks the same values on-chain.
RESIDUAL	None.

I-11 TWAP oracle needs setup and is not checked on-chain

INFO RESOLVED

LOCATION	`ZxoOracle, Zxo constructor, set-observations.py`
ISSUE	Each new V3 pool stores one price observation, so observe() over 15 minutes fails while the pool is active, which blocks stake and harvest. ZXO_ORACLE was also not checked on-chain.
RESOLUTION	Revision 2.3: the Zxo constructor now requires every ChainLib address (position manager, factory, HOLDER token, treasury, deployer, oracle) to be a deployed contract, checks the position manager's factory, requires fee tier 100 with tick spacing 1, HOLDER decimals of 18, the treasury and deployer markers, an unlinked deployer, and ZxoOracle.WINDOW() > 0. The per-pool history step is handled by set-observations.py, run once after initializePool(core).
RESIDUAL	Operational: run set-observations.py for every new pool. If missed, stake and harvest pause for that pool until it is run; unstake and funds are unaffected.

I-12 SepoliaTestLib.sol did not compile

INFO RESOLVED

LOCATION	`libraries/SepoliaTestLib.sol`
ISSUE	The ZXO_ORACLE line has no address ("ZXO_ORACLE = " with nothing after it). Zxo, ZxoV3Deployer and ZxoVerify currently import this library, so the uploaded tree does not build until it is fixed or the import is switched.
RESOLUTION	Revision 2.5: ZXO_ORACLE is now set in SepoliaTestLib, and update-chain-lib.py detects and repairs a line like this. The tree builds with the current import.
RESIDUAL	None. Switch the import to the target chain for each production build (enforced by compile.py warnings).

I-13 ConfigUpdated could log a value that differs from the stored one

INFO RESOLVED

LOCATION	`Zxo.updatePoolConfig`
ISSUE	ConfigUpdated logs the requested value, but a target APR above the pool's cap is stored clamped, so the logged value can differ from the applied one (the AutoAdjust event that follows shows the applied target). PoolInitialized uses token0 as its indexed address, which is the pair token when Zxo is token1, while PoolCreated indexes the reward token.
RESOLUTION	Revision 2.4: PoolInitialized now indexes tokens[2], the reward token, so it matches PoolCreated. Revision 2.5: updatePoolConfig now logs the value it stored (the capped targetApr, the referral rate or the new rewardRate) from a single _log call after the branches.
RESIDUAL	None.

04

Function flows

Each flow lists the order of operations in the current code. Tags mark each step as a check (reverts on failure), a state effect, an external interaction, or an event.

`CHECK` Reverts on failure `EFFECT` Writes state `EXTERNAL` Calls another contract `EVENT` Emits log

stake

`stake(uint8 poolId, uint256 amount0Desired, uint256 amount1Desired, uint256 slippageBps, address referrer)`

Caller: Any user

1	CHECK	Pool exists and is launched (stakes > 0). User has fewer than 21 positions
2	CHECK	Amounts, slippage ≤ 50% and minimum stake validated
3	CHECK	Pool is healthy: spot tick and 15-minute TWAP tick strictly inside the stake range
4	EFFECT	Set referrer if not already set (address(0) → treasury)
5	EXTERNAL	safeTransferFrom both tokens in, then forceApprove the position manager
6	EXTERNAL	Position manager mints the NFT to Zxo on the fixed range with slippage minimums
7	EXTERNAL	Refund unused amounts above the dust threshold
8	EFFECT	_updatePool, then compute bonus and 18-decimal total (HOLDER balance read)
9	EFFECT	Record position, add liquidity and bonus to totalStaked, add total to totalTokens
10	EVENT	Bonus (if any) and Stake events
11	EFFECT	Auto-adjust rewardRate if the 1-hour cooldown has passed

The NFT is always held by Zxo. The bonus is virtual liquidity added to reward weight only.

harvest

`harvest(uint64 tokenId, uint8 poolId, Action action)`

Caller: Position owner

1	EFFECT	_updatePool accrues rewards to now
2	CHECK	Valid pool and action. Pool is healthy
3	CHECK	Caller owns the position and liquidity > 0
4	EXTERNAL	Claim: mint the referral share to the referrer and the rest to the user, then reset rewardDebt
5	EXTERNAL	Collect: position manager collects fees to Zxo, which forwards them to the user
6	EVENT	Claim and/or CollectFees events
7	EFFECT	Auto-adjust rewardRate if due

Blocked while the pool is unhealthy. Rewards keep accruing and can be claimed once the price is back in range.

unstake

`unstake(uint64 tokenId, uint8 poolId, bool acceptForfeit, address to)`

Caller: Position owner

1	EFFECT	_updatePool accrues rewards to now
2	CHECK	Valid pool, caller owns the position, liquidity > 0, 1-day lock-up passed
3	CHECK	to defaults to msg.sender and can't be Zxo itself
4	CHECK	Unless acceptForfeit, pending must be ≤ MIN_UNSTAKE (1e12)
5	EFFECT	Decrement counters. Fail-safe subtraction from totalStaked and totalTokens
6	EFFECT	Delete the position record and remove the user reference
7	EVENT	Unstake event (records the forfeited pending amount)
8	EFFECT	Auto-adjust rewardRate if due
9	EXTERNAL	safeTransferFrom the NFT to `to` (last step)

No health gate and no owner gate. A failed transfer reverts the whole call with no state change.

createPool

`createPool(name, symbol, packedCore, packedParams, max30dRate, v3StartAmnt, pairToken, forceLower, withoutPool)`

Caller: Owner

1	CHECK	Pool count < 255. maxNormalAPR ≤ 320%. pairToken ≠ 0
2	CHECK	Pair decimals match the parameter and are 6, 8 or 18
3	EXTERNAL	Deploy ZxoToken (owner permanently = Zxo)
4	EFFECT	Push PoolInfo (target APR 10%, referral 3%) and pack config with a 365-day bonus window
5	EXTERNAL	Approve the position manager. Mint coreLP + coreStake to Zxo and v3StartAmnt to the deployer

withoutPool = true deploys the token only, with no mint.

initializePool

`initializePool(uint8 poolId, uint16 slippageBps, bool isCore)`

Caller: Owner

1	CHECK	stakes == 0. The core LP must not exist yet (core path) or must already exist (stake path)
2	CHECK	Stake path: IZxoV3Deployer.isFinalized(rewardToken)
3	EXTERNAL	Core path: create and initialize the pool, then require slot0 price == launch price
4	EXTERNAL	Mint the single-sided position to Zxo with the slippage minimum
5	EFFECT	Core path: store corePositionId in poolConfig
6	EFFECT	Stake path: record a Zxo-owned position, add to totals, stakes = 1 (opens staking)

Each path runs once per pool. Neither position can ever be withdrawn.

startV3pool / finalizeToken

`startV3pool(poolId, slippageBps, amount, posman) · finalizeToken(token)`

Caller: Deployer owner

1	CHECK	startV3pool: posman ≠ 0, amount > 0, token not finalized, balance ≥ amount
2	EXTERNAL	Read token0, token1 and rewardToken from Zxo.poolInfo. Read pair decimals
3	EXTERNAL	forceApprove posman. Create and initialize the pool. Mint single-sided to 0xdEaD
4	EVENT	PoolStarted(poolId, posman, tokenId, liquidity, amount0, amount1)
5	CHECK	finalizeToken: token ≠ 0 and not already finalized
6	EFFECT	Set isFinalized[token] = true
7	EXTERNAL	Burn the deployer's whole balance of the token
8	EVENT	TokenFinalized(token, burned)

Once finalized, seeding is closed for that token permanently.

updatePoolConfig

`updatePoolConfig(uint8 poolId, uint8 dataType, uint96 newValue)`

Caller: Owner

1	CHECK	Valid pool. Values checked with _safe16
2	EFFECT	Type 0: targetAprBps, clamped to maxNormalAPR (2× during the bonus year), then auto-adjust
3	EFFECT	Type 1: referral rate, 3%–30%
4	EFFECT	Type 2: lower rewardRate only. _updatePool runs first. Lasts until the next auto-adjust

No control over ranges, tokens or user positions.

updatePool / collectFeesFromCoreLP

`updatePool(uint8 poolId, bool updateAll) · collectFeesFromCoreLP(uint8 poolId)`

Caller: Anyone · Owner

1	CHECK	updatePool: validate poolId (single-pool path)
2	EFFECT	Accrue acc and auto-adjust the rate for one pool or all pools
3	CHECK	collectFeesFromCoreLP: owner only, valid pool
4	EXTERNAL	Collect the core LP's fees to Zxo, then transfer them to ZXO_TREASURY

Core LP liquidity is never touched. Only accrued fees move.

05

Pool launch sequence

The contracts enforce the order of steps 4 and 5. The core-stake mint reverts until the reward token is finalized on the deployer, and staking can't open until the core stake exists. Step 2 comes before any Zxo is in public hands, so no one can seed the main pool at a price of their choosing.

STEP 1

createPool

Deploys the ZxoToken. Checks pair decimals, mints coreLP + coreStake to Zxo and v3StartAmnt to the deployer.

STEP 2

initializePool · core

Creates and initializes the pool, requires price == launch price, and mints the single-sided core LP.

STEP 3

startV3pool ×n

Seeds external DEXes only. NFTs are sent to 0xdEaD. Never seed the main position manager here.

STEP 4

finalizeToken

Burns the deployer's remaining balance. Permanently blocks further seeding.

STEP 5

initializePool · stake

Requires isFinalized. Mints the core stake, which opens staking.

Seeding-window risk at production size

To push the tick from the launch tick (10) past the core-stake lower bound (31), buyers must absorb about 7% of the core LP. At the planned core sizes that is roughly $350k for a $5M core, $1.05M for $15M and $70M for $1B. Seeded DEX positions are too small to move the main pool through arbitrage. Before step 5, confirm that `queryData(0, poolId, 4, 0)` returns a tick below 31 (or above −31 when Zxo is token1).

Oracle setup (Revision 2.2): after step 2, run `set-observations.py` for the new pool (it calls `increaseObservationCardinalityNext` with a size suited to the chain's block time). Staking and harvest stay closed until the pool has 15 minutes of price history.

06

Farming logic (MasterChef on V3 liquidity)

Accounting model

```
acc += elapsed × rewardRate × 1e18 /
totalStaked
pending = liquidity × acc / 1e18 −
rewardDebt
rewardDebt = liquidity × acc / 1e18
(on stake, claim)
```

Rewards are minted on claim, not pre-funded. Forfeited or unclaimed rewards are never minted. Every user position shares one fixed tick range, so liquidity is proportional to position value at any price, and liquidity weighting is fair between users.

Emission controller

```
rewardRate = min( TVL × targetApr,
                  TVL × maxApr,
// 2× maxNormalAPR in year 1
                  max30dRate / 30 days
)
cooldown   = 1 hour after each change
```

TVL is `totalTokens`, the 18-decimal stake-time amounts summed at 1:1. Since M-01, every stake counts toward TVL. Lowering the rate by hand (updatePoolConfig type 2) only lasts until the next auto-adjust; lowering `targetAprBps` is the lasting control.

Invariants verified

INVARIANT	WHY IT HOLDS
Pending never underflows	`rewardDebt` is always set from the current `acc`, and `acc` only increases.
No retroactive rate changes	`_updatePool` runs before every change to `totalStaked` and before every rate change.
Bonus is symmetric	Bonus liquidity is added to `pos.liquidity` and `totalStaked` together and removed together.
Referral is not additive	The referral share comes out of the user's pending amount. A self-referral wallet earns nothing extra.

INVARIANT	WHY IT HOLDS
Record ⇒ custody	A record with owner ≠ 0 and liquidity > 0 exists only while Zxo holds the NFT. The one exit deletes the record before transferring.
Exit always available	Unstake has no health gate and no owner gate. The fail-safe subtractions stop accounting drift from blocking exits.
Pool cannot un-launch	The core stake keeps `stakes ≥ 1` permanently.

Stake range and health (18/18 pair)

ZXO PRICE	TICK	STATE
> ~1.0305	> +300	UNHEALTHY
~0.9704 – ~1.0305	(−300, +300)	HEALTHY
< ~0.9704	< −300	UNHEALTHY

When the pool is unhealthy, stake and harvest revert, unstake still works, and rewards keep accruing. Bounds are strict. For 6- and 8-decimal pairs the ranges center on ±276324 and ±230270.

Stake bonus curve

TICKS ABOVE PARITY	0	4	10	16	20+
Bonus	0%	4%	10%	16%	20%
With HOLDER	0%	5%	12.5%	20%	25%

The bonus reaches its cap at a 53.36% paired ratio. The core LP launches at tick 10, so the organic bonus is expected to be around 10–12%.

07

Owner powers

Zxo owner can

- Create up to 255 pools and tokens, setting coreLP, coreStake, maxNormalAPR (≤ 320%), decimals (validated), max30dRate and v3StartAmnt
- Initialize the core LP and core stake once per pool
- Set targetAprBps from 0 to the cap (turns emissions on or off)
- Set the referral rate between 3% and 30%
- Lower rewardRate temporarily
- Send core-LP trading fees to the treasury
- Transfer ownership (two-step) or renounce it

Zxo owner cannot

- Touch user NFTs, principal or unclaimed rewards
- Block or delay unstake
- Withdraw or reduce core LP or core-stake liquidity
- Mint the reward token outside pool creation and emissions
- Burn anyone's tokens (burn(address) removed)
- Change tick ranges, fee tier, position manager, factory or reward token
- Pause, upgrade, or rescue any asset held by Zxo

CONTRACT · ROLE	CAN	CANNOT
ZxoToken owner = Zxo, permanent	Mint	Transfer or renounce ownership. Burn others' tokens. Holders can burn their own tokens.
ZxoV3Deployer Ownable2Step	Seed external DEXes with any position manager until the token is finalized. Finalize (burn the remainder). Set ZXO_CONTRACT once.	Seed after finalization. Repoint ZXO_CONTRACT. Withdraw tokens on mainnet (testnet-only validator path).
ZxoTreasury Ownable (single-step)	Sweep any token to the destination. Change the destination after a 7-day delay. Lock the destination forever.	Receive ETH. Bypass the delay.

08

Core LP ownership

The core LP NFT is minted to `address(this)` and its id is stored in `poolConfig`. Zxo has no function that transfers it or reduces its liquidity. **The core liquidity, and all pair tokens it collects as it sells Zxo between ~1.001 and ~1.03, is permanently locked.** Trading fees are the only value that leaves, and they go to the fixed treasury through an owner-only call. If Zxo ownership is renounced, those fees are locked too.

Before minting, the launch price is now checked against the pool's actual price (L-01). All six launch prices were checked with a port of Uniswap's TickMath. Each sits exactly one tick outside its core range, on the correct side for a single-sided Zxo deposit.

PAIR	ZXO IS	LAUNCH TICK	CORE LP RANGE	CORE STAKE RANGE	USER STAKE RANGE
18 / 18	token0	10	[11, 300]	[31, 300]	[−300, 300]
18 / 18	token1	−10	[−300, −11]	[−300, −31]	[−300, 300]
18 / 6	token0	−276314	[−276313, −276024]	[−276293, −276024]	[−276624, −276024]
18 / 6	token1	276314	[276024, 276313]	[276024, 276293]	[276024, 276624]
18 / 8	token0	−230260	[−230259, −229970]	[−230239, −229970]	[−230570, −229970]
18 / 8	token1	230260	[229970, 230259]	[229970, 230239]	[229970, 230570]

09

Accepted-risk register

These items were reviewed with the development team and deliberately left as they are. Items the team excluded from scope before the audit are listed at the end.

ID	RISK	TEAM POSITION
M-03	**Core-stake position dilutes user rewards and its fees can't be collected** The contract-owned core stake counts toward TVL and reward weight, with about 2.2× weight per token, but never claims. Users receive less than targetApr by roughly its share of totalStaked. Its trading fees are locked.	Accepted. Auto-adjust does not close this gap. The gap shrinks as user TVL grows, and targetAprBps can compensate.
L-03	**Bonus ramp reaches its cap about 20 ticks above parity** The constant 59538 hits the 20% cap at a 53.36% paired ratio, so the bonus is close to all-or-nothing.	Intended. At the launch tick the organic bonus is about 10–12%.
L-04	**No user-supplied deadline** block.timestamp + 20 minutes is evaluated at execution, so a delayed transaction still runs.	Accepted. Price protection comes from the slippage minimums.
L-05	**No rescue path for dust or stray NFTs** Refunds at or below the dust threshold, and NFTs sent directly to Zxo, can't be recovered.	Accepted by design. No rescue function exists, so the owner has no path to any asset Zxo holds.

ID	RISK	TEAM POSITION
L-07	**Treasury uses single-step ownership** A mistaken transferOwnership can't be undone.	Accepted. Owner operations are handled carefully.
L-09	**Referrer can be set to the Zxo contract** The referral share would be minted to Zxo and be unrecoverable. Only that user's own share is affected.	Accepted. Any address may be a referrer, and address(0) defaults to the treasury.
I-03	**Redundant guards kept** Some checks can't be reached or are no-ops: userReferrer_[this], the rewardDebt write when pending == 0, and several library guards. The unreachable queryData type-3 branch was removed in Revision 2.1 (verified: that branch can only see type 0 or 3).	Kept by choice as defense in depth. Unused enums, ownerOf checks and zero-token checks were removed.
I-08	**Single enum-coded event** Only user is indexed, so indexers must filter on the action value.	Accepted. Each separate event adds bytecode, and one shared event keeps Zxo under the 24,576-byte limit while still emitting every action.

Out of scope by design (declared before the audit)

BEHAVIOR	RATIONALE
Bare `revert()` without reasons	Saves gas and bytecode. Frontends pre-check conditions.
Unstaking forfeits pending rewards	The protocol pays out only while the tick range is healthy. Users harvest first.
HOLDER bonus can be bought, used and sold	The HOLDER token is only available on public DEXes. The team treats this as volume and participation.
Tick-range requirement on stake	Covered by the user-set slippage parameter.

10

Before mainnet

Build and configuration

- [ ] Switch the ChainLib import from `SepoliaTestLib.sol` to the target chain with `update-chain-lib.py`, and set `ZXO_TREASURY`, `ZXO_V3_DEPLOYER` and `ZXO_ORACLE`
- [ ] Compile. Watch `createPool` and `initializePool` for "stack too deep"
- [ ] Confirm Zxo runtime bytecode is under 24,576 bytes (EIP-170)
- [x] compile.py stops on standard-JSON errors (done)
- [x] Verification input compiles with no local files and matches the build (confirmed for Zxo; repeat for the other contracts)
- [x] EVM version selectable in compile.py. Use Shanghai: PulseChain supports nothing newer (done)
- [ ] Set `HOLDER_TOKEN` and `HOLDER_THRESHOLD` in each chain library (currently zero on 8 mainnets; Arbitrum and test libraries are set)
- [ ] Run ZxoVerify with the same ChainLib as Zxo: once before ZxoV3Deployer exists, then again with verifyV3Deployer = true and verifyOracle = true
- [ ] Run Slither or Aderyn and triage the results against this register
- [ ] Move Zxo and deployer ownership to a multisig

Tests

- [ ] Fork: read slot0 on every target DEX. The old typed call reverts on PancakeSwap
- [ ] Fork: stake a USDT pair on Ethereum
- [ ] Fork: run the full launch order. The core stake reverts until finalizeToken
- [ ] Fork: `initializePool(core)` reverts on a pool pre-initialized at another price
- [ ] `createPool` reverts when decimals don't match the pair token
- [ ] Unstake to a contract without the NFT receiver reverts with no state change. A retry with an EOA as `to` succeeds
- [ ] Invariant: `totalStaked == Σ pos.liquidity` (core stake included)
- [ ] Invariant: `totalTokens == Σ pos.tokens` (core stake included)
- [ ] Invariant: the owner can always unstake after the 1-day lock-up
- [ ] Independent re-audit of the diff since Revision 1

Frontend and ABI changes

- `unstake(uint64 tokenId, uint8 poolId, bool acceptForfeit, address to)`. The selector changed. Pass `address(0)` to send the NFT to the caller.
- `queryData(…, 4)` returns `(false, 0, 0)` when a price can't be read, instead of reverting. Treat `sqrtPriceX96 == 0` as "not ready".
- `totalGlobalStakes()` and `referralsPerUser_()` now return `uint32`.
- New on ZxoV3Deployer: `isFinalized(address)`, `finalizeToken(address)` and the event `TokenFinalized(address indexed, uint256)`.
- New event action codes: 8 Deployed, 9 PoolCreated, 10 CoreFeesCollected, 11 ConfigUpdated, 12 PoolInitialized. Codes 0–7 are unchanged.
- `tokensIds_(address)` was removed. Map a reward token to its pool by scanning `poolInfo(i).rewardToken`.
- `queryData(…, 4)` "healthy" now also requires the 15-minute TWAP in range. Use `ZxoOracle.twapTick(pool)` to show the average price and explain a paused pool.
- Trust panel for each pool: `isFinalized` is true, the deployer balance is 0, every `PoolStarted` posman is canonical, and the seeded NFTs are owned by 0xdEaD.

11

Scope and file fingerprints

Revision 2.5 covers the contracts as uploaded on 27 September 2026 (14:19), with Zxo.sol as updated at 14:27. Hashes are the first 12 hex characters of each file's SHA-256. Any later change to these files is outside this review.

Compiler: solc 0.8.36, legacy pipeline (viaIR off)

Etherscan lists three known bugs for 0.8.36, all fixed in 0.8.37. None applies to Zxo. MisorderedNamedParametersInRequireWithCustomErrors and SpillSlotCollisionAcrossMutualRecursion only affect the via-IR pipeline, and Zxo has no custom errors in require and no recursion. MemoryByteArrayElementDeleteClearsWholeWord needs `delete` on an element of a memory `bytes` array; the only deletes in the build clear storage mappings. Keep viaIR off.

FILE	LINES	SHA-256 (PREFIX)	ROLE
Zxo.sol	1067	8c8c2947cfe6	Farm, staking, emissions, core LP
libraries/ZxoLib.sol	502	2af09f5c47af	Ticks, bonus, packing, validation
ZxoV3Deployer.sol	331	3c8fb36af7dd	External DEX seeding and finalization
ZxoTreasury.sol	231	aa47be5c2210	Fee and referral receiver

FILE	LINES	SHA-256 (PREFIX)	ROLE
ZxoVerify.sol	211	02ff34b11012	Deployment sanity check
tokens/ZxoToken.sol	103	87ba572691dd	Per-pool reward token
interfaces/Iv3PosManager.sol	56	0a8356439ff0	Position manager interface
interfaces/IZxo.sol	22	5fd91cc8d935	poolInfo reader
interfaces/IZxoToken.sol	12	856487dac631	mint, coreLP, coreStake
interfaces/IUniswapV3.sol	7	e5c8cd93d1d8	Factory interface
interfaces/IZxoV3Deployer.sol	5	d332e8f4212b	isFinalized
ZxoOracle.sol	26	6ac1f047bc61	15-minute TWAP tick reader
interfaces/IZxoOracle.sol	17	bb73eefaf56f	twapTick, inRange, WINDOW
libraries/*Lib.sol (11 chains)	21–22 each	various	Per-chain addresses and HOLDER threshold (all addresses EIP-55 valid)

This report is an AI-assisted security review, based on reading the source and on numeric checks of the tick and price constants. It isn't a guarantee that the code is free of defects, and it isn't financial or legal advice. Behavior depends on the external DEX contracts, chain conditions and deployment configuration, none of which were tested live. An independent professional audit is recommended before contracts hold user funds.